Project Report

Normalization Checker Tool For Relational Databases

Saranya Pandiaraj | Pooja Malage

Gangadhar Kavuri | Mercy Bose

14 December, 2020

Acknowledgment

We would like to thank the following people for helping with our research project: Authors from the research paper - *Du H and Wery L [1], Yazici A, Ziya K [2], Dongare, Y., Dhabe, P., & Deshmukh, S. [3], Yazici, A., and Karakaya [4], M. Demba [6], G.Sunitha, Dr.A.Jaya [10], Amir Bahmani, M. Naghibzadeh [11].* The information in the research paper helped us to understand the concepts involved in this project on a deeper level.

We would also like to express our gratitude to our professor, *Ming-Hwa Wang* for encouraging us to work on this project.

Table of Contents

1 Abstract

Normalization is the most applied technique for the consideration of relational databases. It aims at creating a set of relations with minimum data redundancy that preserve consistency and simplify the appropriate insertion, deletion, and modification. A normalized database does not show any anomalies in future updates. It's a very time-consuming process when it is done manually which affects the productivity of any organization. Thus, an automated Normalization tool for relational databases is proposed to aid in automating the relational database schema normalization up to BCNF form. This tool mainly concentrates on addressing the legacy system issues where we will be unaware of the physical design behind the system. It's always a challenging problem to identify the dependencies among the attributes when the system is outdated, yet still in use.

2 Introduction

In many software organizations, the activities involved in the design and implementation phase are mostly done manually which is a very time-consuming process and thus makes it less productive. One such process in the design of a relational database is known as Normalization which takes a lot of time to organize the data in the database. For any industry, productivity and quality matter a lot in order to improve the profitability of their organization. To attain this, it's necessary for them to automate the activities involved in the design and implementation phase.

Normalization is a very important phase in a database to ensure that only the related data is stored in the respective tables. It takes the attributes and the functional dependencies from any bigger relations to produce smaller relational schemas. Thus, minimizes the redundancy (duplicate data) and certain anomalies like insert, delete, and update. The stages of organizing the data are known as "Normal Forms" which helps in redesigning the database and ensuring to satisfy all the different types of normal forms. Normal forms are carried out in the following stages: First Normal Form (1NF), Second Normal Form (2NF), Third Normal Form (3NF), Boyce-Codd Normal Form (BCNF), Fourth Normal Form (4NF), Fifth Normal Form (5NF) & Sixth Normal Form (6NF). In most of the organization, Third Normal Form (3NF) is considered to be the most adequate relational database design which always ensures functional dependency preservation and

losslessness. 3NF is sufficient enough since most of the 3NF tables are free from the insert, update and delete anomalies.

A relation is considered to be unnormalized when it goes through various anomalies and no normalization rules have been applied. This only happens where the relation is designed using a bottom-up approach i.e.; obtaining the attributes to a universal relation from a manual document, report, etc.; A relation satisfies 1NF when there are no repeating groups into a new relation. A relation satisfies 2NF when there are no partial dependencies into a new relation. A relation satisfies 3NF when there are no transitive dependencies into a new relation.

2.1 Objective

The main objective is to propose a tool to check if the data in the relational databases are normalized by uploading any structured data and also implement a way to identify the functional dependencies in the data in the case of a legacy system or by getting the attributes and functional dependencies from the user via a web interface.

2.2 What is the problem

During application development which involves a relational database, normalizing the data is one of the key aspects of data modeling. So much manual effort is spent to validate whether the data has been normalized correctly. On another note, often project teams tend to overlook data normalization issues in legacy applications as it is very difficult to manually identify them. Unnormalized data and incorrect normalization of data will increase costs as you tend to store more numbers of records.

2.3 Why this is a project related to this class

The main purpose of a database in any organization is to systematize the data. Database normalization is a technique that helps in refining the data. It is the most important concept in the design of any relational database. Upon the creation of a database, the next key step is *Normalization*. This process will be useful in avoiding the Data Redundancy, Insertion, Update, and Deletion anomalies in your data. It is a vital consideration for an application developer since it is very difficult to store the objects in a relational database that preserves similar data in several places. This process makes the database more consistent and natural, thus reducing the size and simplifying the structure for easier to locate, compare, and retrieve the data. Below are the few advantages when data is normalized.

- Logical Map - Data is always organized and stored in an appropriate place
- Data Consistency – Reliability also increases
- Data Connections – How data from different relation relates to each other
- Increased security
- Cost savings

2.4 Why other approach is no good

Below are the few drawbacks we have seen during the research.

- 1 Normal Form (identifying the repeating groups) needs to be done manually by the user. For each composite attribute, GUI asks for the set of atomic attributes corresponding to the composite attribute

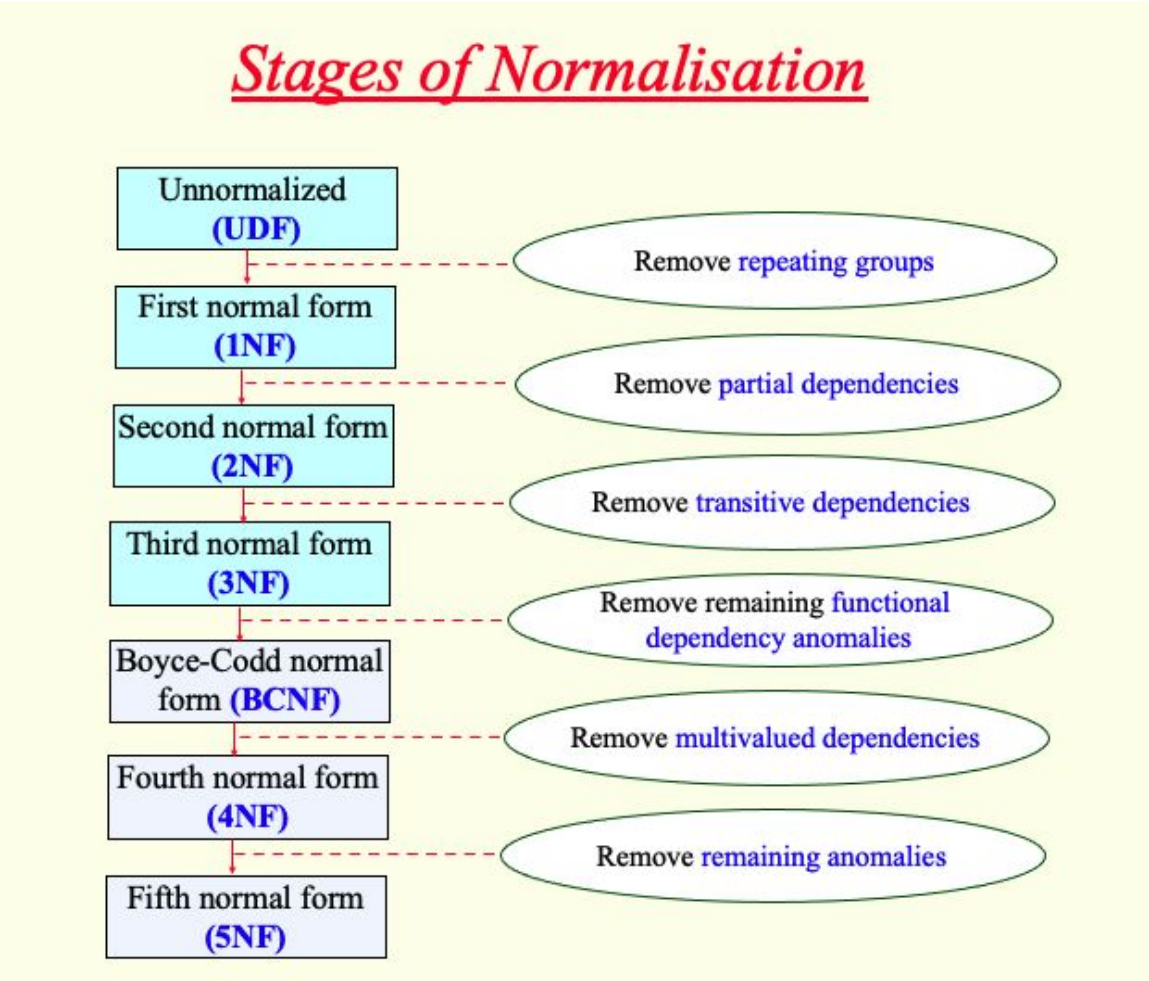
- Functional Dependency is entered manually. In the case of a legacy system where a user is not aware of the dependencies, this might be a difficult approach

2.5 Why we think our approach is better

Our approach is better because our tool can handle the legacy system constraints as well as the recent systems. There are few systems that still run in an outdated model, so if you want to improvise the data for a better performance by reducing the redundancy data; this approach will be useful in attaining it. Since the tool will handle to determine the functional dependencies for a given relation and also will establish the identifying groups.

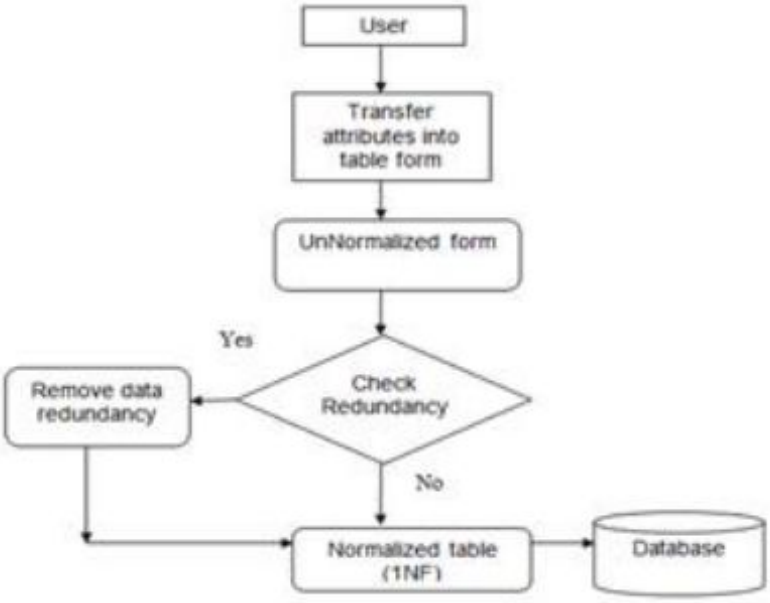
2.6 Area or scope of the investigation

Our main research was in analyzing the concepts of the normal forms and the issues faced by the organization when it's done manually. Below are the stages which help in attaining the process of database normalization.



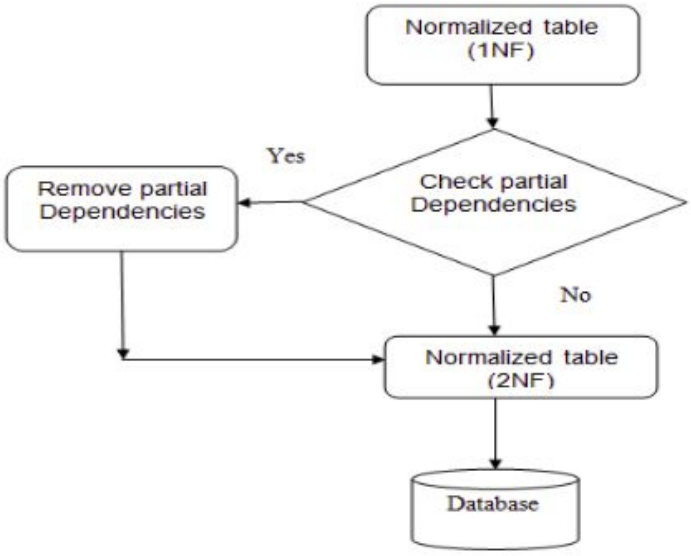
First Normal Form – 1NF:

- Examine every field whether it has atomic values or composite values
- Remove null values by entering corresponding data type value
- Replicas of records are removed



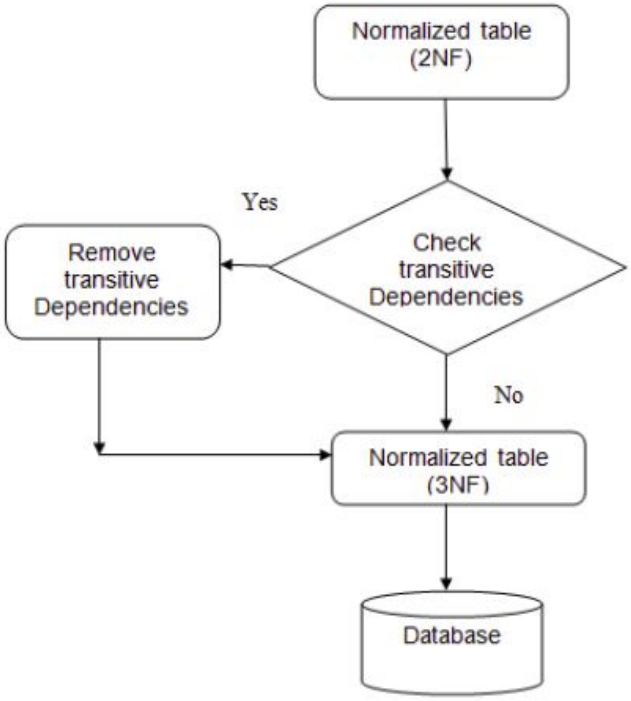
Second Normal Form – 2NF:

- Aim- Remove partial functional dependency
- Functional dependency-as every field in a table should be connected with a primary key or candidate key of that table
- Partial functional dependency- If non-key fields are connected or logically grouped with a PART of candidate key or a part of the primary key



Third Normal Form – 3NF:

- Aim: to satisfy the transitive dependency
- Achieved by extracting the independent columns in the table
- Checks the corresponding and all the required constraints in the flow of streams



3 Theoretical bases and Literature review

3.1 Definition of the problem

The Relational Database model is the most popular database model used today in almost all modern settings involving data management (Business, Research, Administration). Data is managed most efficiently when stored in a database. The relational database has the capability to manage large amounts of data because of its performance, reliability, and integrity. One important aspect of relational database theory is database design through normalization during application development.

Normalizing the data is one of the key aspects of data modeling which requires skilled expertise in databases and normalization. So much manual effort is spent to validate whether the data has been normalized correctly as the database grows.

3.2 Theoretical background of the problem

Normalization is carried out to eliminate the flaws of a database that is poorly designed. The database is inconsistent and creates issues while we add, delete, or update the information. It's very important that when a user uses the database, data integrity and scalability should be maintained. Normalization is carried out to reduce and eliminate data redundancy.

To model today's database which requires a large amount of data to be stored, the number of relations increases, each contains a large number of attributes and functional dependencies. To understand those relations, it requires skilled expertise in the manual process of normalization.

The following are drawbacks of normalization carried out manually.

1. It is time-consuming
2. less productive
3. It is prone to errors
4. It is costly

To eliminate these drawbacks several researchers already tried to automate normalization by proposing new tools/methods.

3.3 Related research to solve the problem

Algorithms for decomposing relation schemes up to 3NF and BCNF existed in the early literature on relational database design in the late 1970s and early 1980s and can be found in several reputable textbooks on databases today. Nevertheless, attempts at automating the normalization process have been continuing for more than three decades.

Some of the systems presently existing to cater database normalization is as follows:

1. RDBNorma
2. EDNA
3. Micro
4. NORMI
5. A Web-Based Relational Database Design Tool to Perform Normalization
6. A Web-Based Tool to Enhance Teaching/Learning Database Normalization
7. A Web-Based Environment for Learning Normalization of Relational Database Schemata
8. A Computer-Aided Learning Tool: Case of Normalization of Relational Schemata
9. Web-Based E-Learning System for Data Normalization

The majority of these systems have however been developed purely as academic research projects, with an emphasis on efficiently delivering accurate results over explaining the process involved.

3.4 Advantages/disadvantages of those research

Advantages:

1. Tools developed to tutor students in the about database normalization and test their knowledge in the subject
2. Provides the normalized forms of tables depending on user input
3. A relation can be represented with only one singly linked list along with its set of FDs' which save considerable space as compared with representing a relation using two linked list one for attributes and other for FD's
4. RDBNorma is better than Micro in terms of both the speed and memory space requirement

Disadvantages:

1. Assumes that 1NF exists by defaults
2. Does not connect to legacy systems and check for normalization
3. Does not provide SQL scripts to help in building a normalized database
4. Not available in Python- a rapid developing language
5. Functional Dependency is entered manually

3.5 Our solution to solve this problem

Our solution is to create an automated normalization checker tool for the user to solve the drawbacks of normalization when it is carried out manually. This tool will help you in normalizing the data by just uploading the data file or manually entering the attributes and functional dependencies via web interface.

Also, our tool will be able to connect to a legacy system and check the normalization forms for the existing tables and provide a way for it to be converted into other normal forms as required.

3.6 Where our solution different from others

Our Implementation of the tool is different from the others since our solution also takes into consideration the conversion of legacy database systems to normalized forms. In the legacy systems where the design details are unknown, our tool helps you in determining the repeating groups and also the functional dependencies by just connecting to the respective database schema.

3.7 Why our solution is better

We will be using Python to implement our tool. We will be developing a tool to connect to a legacy database system to be able to propose normalization or existing tables. Thus, satisfying both the legacy and modern systems.

4 Hypothesis

4.1 Single/Multiple hypotheses

We hold the below hypotheses in which we will be evaluating the normalization tool.

- Computing the closure of a given set of attributes w.r.t to its functional dependencies

- Identify and prompt the user to remove any implied irrelevant attributes
- Removing redundant dependencies
- Determining full and partial dependencies
- Decomposing into 2NF
- Decomposing into 3NF
- Decomposing into BCNF

4.2 Positive or Negative hypothesis

Positive Hypothesis:

Below are the positive hypotheses we will be evaluating:

- Identifying Functional Dependencies: In the case of a legacy system, the tool should be able to identify the functional dependencies for a given data in a relation
- Repeating groups in a relation: The tool should be able to identify the repeating groups in a given relation
- Determining the Duplicate Records: The tool should be able to identify the duplicate records for a given relation
- Redundant dependencies: The tool should be able to identify the redundant dependencies for a given set of functional dependencies
- Partial Dependencies: The tool should be able to identify the partial dependencies in a given relation for the set of FDs
- Transitive Dependencies: The tool should be able to identify the transitive dependencies in a given relation

- Removing remaining functional dependency anomalies: The tool should be able to remove any remaining functional dependency anomalies

Negative Hypothesis

Below are the Negative hypotheses we will be evaluating:

(For handling negative scenarios)

- Non-Repeating groups: The tool should not throw an error or process when the relation is already in 1NF and should display as “The table is already in 1NF Form”
- 1NF + No Partial Dependencies: The tool should not throw an error or process when the relation is already in 2NF and should display as “The table is already in 2NF Form”
- 2NF + No Transitive Dependencies: The tool should not throw an error or process when the relation is already in 3NF and should display as “The table is already in 3NF Form”
- No Duplicate Records: The tool should not display any duplicate records when a relation doesn’t have any duplicate records

5 Methodology

5.1 How to generate/collect input data

The primary source of input is provided by the business which is collected in the ‘Business Requirements’ phase which is referred to as ‘Business Rules’. These rules are

then transformed into a set of entities that comprise a set of ‘attributes’ and their ‘relationships’.

5.2 How to solve the problem

As a next step, a temporary key is assigned and FDs (functional dependencies) among attributes are established. This temporary key is referred to as ‘Primary Key’.

Each Ri (relation schema) is transformed into well-formed groupings such that one fact in one group is connected to other facts in other groups through relationships. We try to automate this process via the normalization algorithms mentioned below.

5.2.1 Algorithm design

Normal Form Steps:

Steps from UNF to 1NF:

- Remove the repeating groups and create a new relation
- Add a replica of the primary key of the relation immediately enclosing it to the new relation
- Name the new entity
- Determine the Primary Key of the new entity
- Repeat steps until no more repeating groups

Steps from 1NF to 2NF:

- Remove the partially functionally dependent attributes on the composite key and place them into a new relation

- Add a replica of the attributes which are the determinants of those attributes to the new relation which will become the primary key.
- Name the new entity
- Rename the original entity

Steps from 2NF to 3NF:

- Remove the transitively dependent attributes on the non-key attribute(s) and place them into a new relation.
- Add a replica of the attributes which are the determinants of those attributes to the new relation which will become the primary key.
- Name the new entity
- Rename the original entity

Algorithm Steps:

Algorithm 1: FullClosureX (X+): Determining X+, the Closure of X under F (FD set)

Algorithm 1 computes the closure of a given set of attributes w.r.t F:

Input: A set F of FDs on a relation schema R, and a set of attributes X, which is a subset of R.

Algorithmic steps:

X+: = X;

repeat

oldX+: = X+;

for each FD Y-->Z in F do

if X+ --> Y then X+: = X+ ∪ Z;

until (X+ = oldX+);

Logic explanation with examples:

s=('ABCDEFG','B->BE, B->DG, BF->AD, C->G DG->BA, G->F')

ABCDEFG

B->BDEG

BF->AD

C->G

DG->AB

G->F

closure (C, s) # cl (C, s)

CFG

Algorithm 2: Identify any implied extraneous attributes

Input: F, a set of FDs.

Output: F', a partially left-reduced set of FDs.

Algorithmic steps:

Let F':=F

For each fd X→A∈F' do

Let Y:=X

For each attribute B∈Y

If A∈(Y-B)+ w.r.t F' then

Y:=Y-B

End if

End for F':=F'-(X➔A) ∪ (Y➔A)

End for

Algorithm 3: Finding a Minimal Cover F for a Set of Functional Dependencies E

Input: A set of functional dependencies E.

Algorithmic steps:

1. Set F:= E.

2. Replace each functional dependency X-->{A1, A2, ..., An} in F by the n functional dependencies X-->A1, X-->A2, ..., X-->An.

3. For each functional dependency X-->A in F

for each attribute B that is an element of X

if { {F – {X-->A} } ∪ { (X – {B}) -->A} } is equivalent to F

then replace X-->A with (X – {B}) -->A in F.

4. For each remaining functional dependency X-->A in F

if {F – {X-->A} } is equivalent to F,

then remove X-->A from F.

Logic Explanation with an Example:

ABCDEFG

B->DEG

C->G

DG->AB

G->F

> minc(s,0)

ABCDEFG

B->DEG

C->G

DG->AB

G->F

> minc(s,1)

ABCDEFG

B->ADEG

C->G

DG->B

G->F

Algorithm 4: Identifying the Full and Partial Dependencies

Input: Fm, a minimal cover set of F.

Output: Fp, Ff, sets of partial and full dependencies in Fm respectively.

Algorithmic steps:

Let Fp:=φ and Ff:=Fm.

For each dependency X➝A∈Ff

If X is a proper subset of a candidate key and A is not a key attribute, then

Fp:=Fp ∪ {X→A} Ff:=Ff-(X→A)

While there is a fd Z→B∈Ff

s.t. Z∈A + do

Fp:=Fp ∪ {Z→B}

Ff:=Ff-(Z→ B)

End while

End if

End for

Algorithm 5: Decomposes into 2NF

Input: R, Ff, and Fp

Output: relations into 2NF. s

Algorithmic steps:

Let G:=Fp and Xf the set of attributes in Ff.

For each Y→A∈G do

If Y is a key attribute, then

- create RY and add any attribute in YG+.

- remove from G any dependency whose lhs is in YG +

- choose Y as the primary key of RY.

- if A∈Xf, Xf:=Xf-{A}

End if

End for

Let K∈Xf be the (chosen) primary key of R. Create a new relation schema RK(Xf).

Algorithm 6: Relational Synthesis into 3NF with Dependency Preservation

Input: A universal relation R and a set of functional dependencies F on the attributes of R.

Algorithmic steps:

1. Find a minimal cover G for F (use Algorithm 3);
2. For each left-hand-side X of a functional dependency that appears in G, create a relation schema in D with attributes {X U {A1} U {A2} ... U {Ak}}, where X-->A1, X-->A2, ..., X-->Ak are the only dependencies in G with X as the left-hand-side (X is the key to this relation);
3. Place any remaining attributes (that have not been placed in any relation) in a single relation schema to ensure the attribute preservation property.

Logic Explanation with an Example:

sets(s)

{'re': 'AE', 'ua': '', 'mi': 'BDFG', 'li': 'C'}

k=keys(s); pp k;

BCD

{BC, CD}

a = synthesis3NF (k, s);

one relation per dependency: [u'FG', u'ABDG', u'CG', u'BDEG']

union of relations with mutually dependent left side: [u'FG', u'ABDEG', u'CG']

no relation has a key; adding key CD (all keys: BC, CD)

relations including key relation: [u'FG', u'ABDEG', u'CG', u'CD']

all subset relations removed: [u'FG', u'ABDEG', u'CG', u'CD']

FG FG {G->F}

ABDEG ABDEFG {B->DEG, DG->AB}

CG CFG {C->G}

CD ABCDEFG {}

5.2.2 Language used

Below are the languages we will be using as part of this project.

- Python (Programming Language)
- eXtensible Markup Language (XML)
- Structured Query Language (SQL)
- Hypertext Markup Language (HTML)
- Java Scripting (JS)

5.2.3 Tools used

Below are the Tools we will be using as part of this project.

- JetBrains PyCharm Community Edition
- MYSQL

5.3 How to generate output

Onus on users to provide the relevant relational attributes and the functional dependencies as per their business requirements by keying in the appropriate sections

in the UI and then selecting the normalization form needed to generate the necessary output for their consumption.

5.4 How to test against hypotheses

Test Case ID	Test Case Description	Test Steps	Expected Result
TC01	Check the input for 1NF	Upload the data into a tool to check for 1NF	There should be No repetition groups. Where there are repetitions, users should be warned to remove them
TC02	Check for any duplicate data present in the relation	Retrieve the table data and check if the data is not being duplicated in a relation by executing the duplicate check query.	There should not be any duplicate data in a given relation. Where there are duplications, users should be warned.
TC03	Check the result if it satisfies minimal cover requirements	Enter the FD's, for the attributes to check if the result satisfies minimal cover.	There should be a minimal cover for the specified input.
TC04	Check if the result satisfies 2NF requirements	1. Specify the Relational attributes 2. Specify Functional dependencies 3. Run the code to check the output.	The resultant code should never have any partial functional dependencies
TC05	Check if the result satisfies 3NF properties	Compare the result with the business specified requirements.	There should not be any transitive dependencies. There should not be any loss of information.

TC06	Check if the result satisfies BCNF properties	1. The generated output from 3NF may have other dependencies which may not have been addressed in the run up to 3NF, all such dependencies have to be removed. 2. Generate the output and check if all such dependencies are removed.	All the dependencies other inclusive non-prime attributes have to be removed.

6 Implementation

6.1 Code

The Implementation code for this project is attached in the Appendix section. Before implementing the code, below are the prerequisites that need to be done.

Prerequisites

Tools to install: Python, MySQL, Google Chrome

Python modules to install from the Python terminal :

Using the pip command, install below modules [>> pip install <module_name>]

flask, sys, mysql.connector, tabulate, os, csv, io, pandas, numpy, IPython.display, HTML, IPython.display, Decomp,re,itertools, functools

Example: pip install flask

How to Run the Program

1. First Download the Project Code Folder in your system.
2. Navigate to Project Code Folder from your terminal
3. Go to the respective location where the app.py file is present

   Example : cd /Users/sara/Desktop/SJSU/Data 225/Project Code

4. Run the command in the terminal: **python app.py**

Once you run the above command, you should get something like below in the terminal :

Serving Flask app "app" (lazy loading)

** Environment: production*

WARNING: This is a development server. Do not use it in a production deployment.

Use a production WSGI server instead.

** Debug mode: on*

** Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)*

** Restarting with fsevents reloader*

** Debugger is active!*

** Debugger PIN: 156-589-807*

>> Navigate to the Google Chrome Browser and type the below server link and press enter

http://127.0.0.1:5000/

>> The Normalization Checker Tool will display in the browser and use the tool according to specifications in *readme.txt*

6.2 Design Document and Flow Chart

The Tool is designed using HTML and the formatting & presentation are done using CSS (Cascading style sheets). The normalization logics are handled through python scripts. We have used flask which is a web framework written in python. It helps to create HTML files and use them as templates for our web pages. The Normalization Automated tool for relational databases has the below sections :

- **Home**

The Home Section tells about the process of Normalization and Why we need to do Normalization.

- **About**

  About Section tells Why this automated tool for and What it does.

  Legacy System - Functional Dependency (FD)

  The functional Dependency section helps you to identify the functional dependencies for Legacy systems given a table/file.

  Steps to follow to identify the FD's

  - Select the Mode for retrieving the Data
  - If the Mode Selected is Database:
    - Enter the Database Credentials of Your Legacy System (Supports only Mysql)
    - Enter the User Name, Password, Database Name, Host Name & Table Name (you wish to retrieve the FD's)
    - After entering the Credentials, Click on Create Functional Dependency Button to retrieve the FD's
  - If the Mode Selected is Upload File:
    - Choose the required file and upload it
    - Click on Create Functional Dependency Button to retrieve the FD's

First Normal Form (1NF)

- The First Normal Form section helps you to identify if a given table/file is in 1NF or not.

  Steps to follow to identify if data is in 1NF

  - Select the Mode for retrieving the Data
  - If the Mode Selected is Database:
    - Enter the Database Credentials of Your Legacy System (Supports only Mysql)
    - Enter the User Name, Password, Database Name, Host Name & Table Name (you wish to retrieve the FD's)
    - After entering the Credentials, Click on **Check for 1NF** Button to check if the Data is in 1NF or not
  - If the Mode Selected is Upload File:
    - Choose the required file and upload it
    - Click on **Check for 1NF** Button to check if the Data is in 1NF or not

Second Normal Form (2NF)

The Second Normal Form section helps you to normalize the given data in 2NF.

Steps to follow to normalize the data in 2NF

- Enter the attributes of a relation/table/file (separated by commas : for ex. A,B,C,D,E)
- Enter each Functional Dependency in a single line and then press enter to input the next FD as given in the placeholder of the text area
- Click on **Normalize to 2NF** Button to retrieve the 2NF Proposal

Third Normal Form (3NF)

The Third Normal Form section helps you to normalize the given data in 3NF.

Steps to follow to normalize the data in 3NF

- Enter the attributes of a relation/table/file (separated by commas : for ex. A,B,C,D,E)
- Enter each Functional Dependency in a single line and then press enter to input the next FD as given in the placeholder of the text area
- Click on **Normalize to 3NF** Button to retrieve the 3NF Proposal

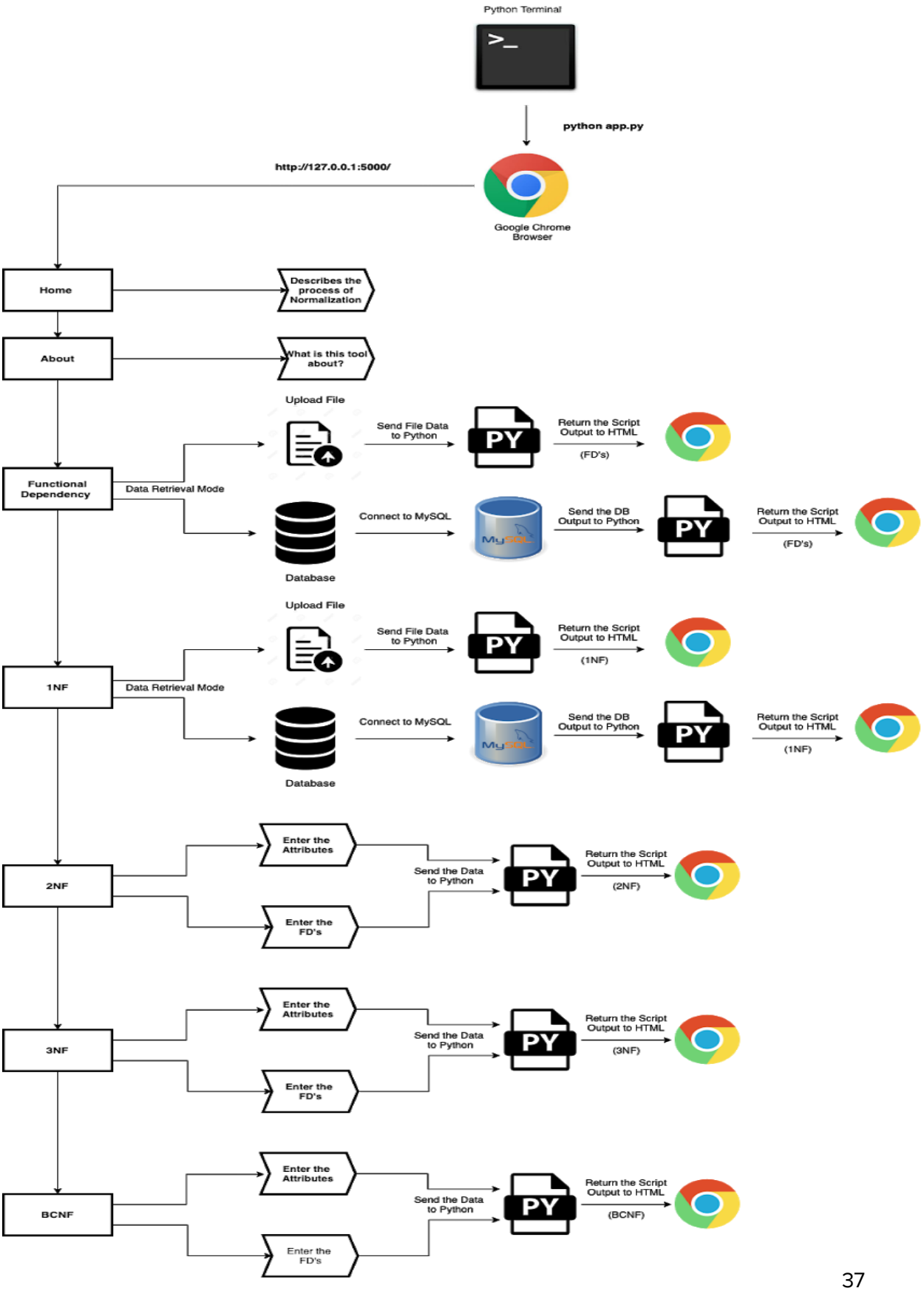
Boyce-Codd Normal Form (BCNF)

The BCNF section helps you to normalize the given data in BCNF.

Steps to follow to normalize the data in BCNF

- Enter the attributes of a relation/table/file (separated by commas : for eg. A,B,C,D,E)
- Enter each Functional Dependency in a single line and then press enter to input the next FD as given in the placeholder of the text area
- Click on **Normalize to BCNF** Button to retrieve the BCNF Proposal

Below is the flow chart of our normalization checker tool.

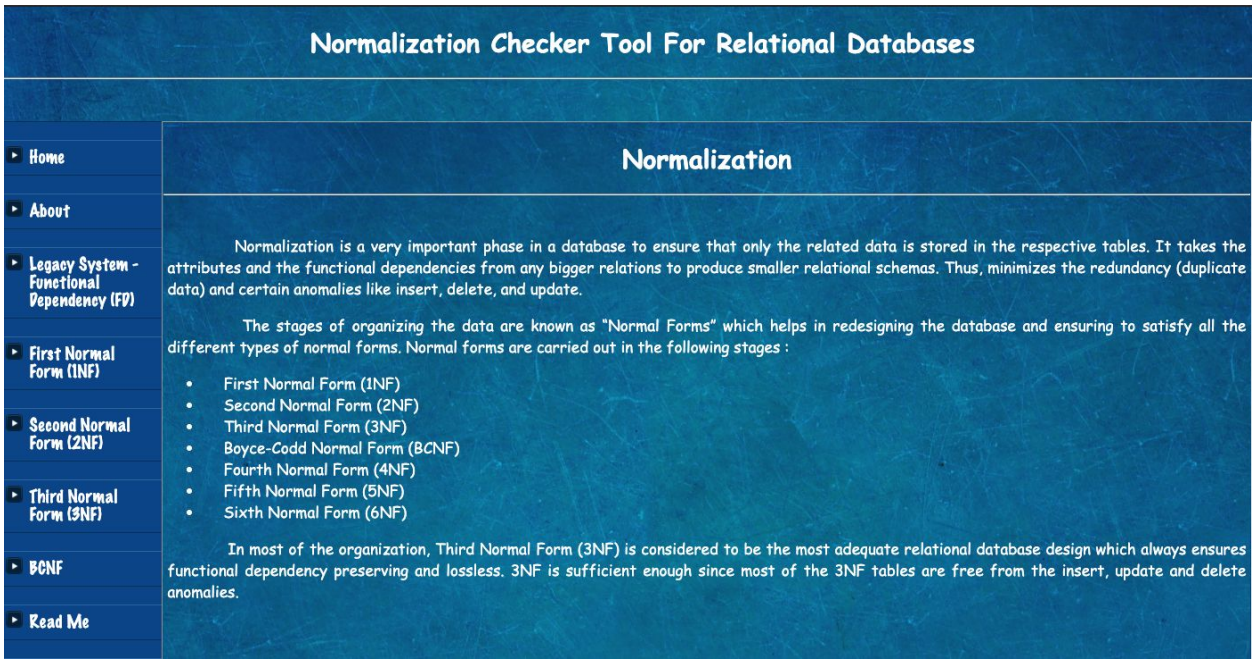


Normalization Checker Tool Flow Chart

7 Data Analysis and Discussion

7.1 Output generation

The tool is designed where a user is guided through stages of the Normalization design process till BCNF by the use of tabs as shown below.



The outputs are generated using the above-defined tabs where functional dependencies for legacy systems are identified by either connecting to a database or uploading a file and required functional dependencies are created. First Normal Form checks whether the uploaded file or table taken from the database is in 1NF or not. The output shows if there are any duplicate columns, rows, or multivalued attributes in the table. Users can check whether the table is in 2NF,3NF, and BCNF by giving a set of attributes separated by a comma and each functional dependency

of given attributes. The output generated shows whether the given set of attributes satisfy the normalization form individually and decompose the functional dependencies to 2NF,3NF, and BCNF respectively.

7.2 Output analysis

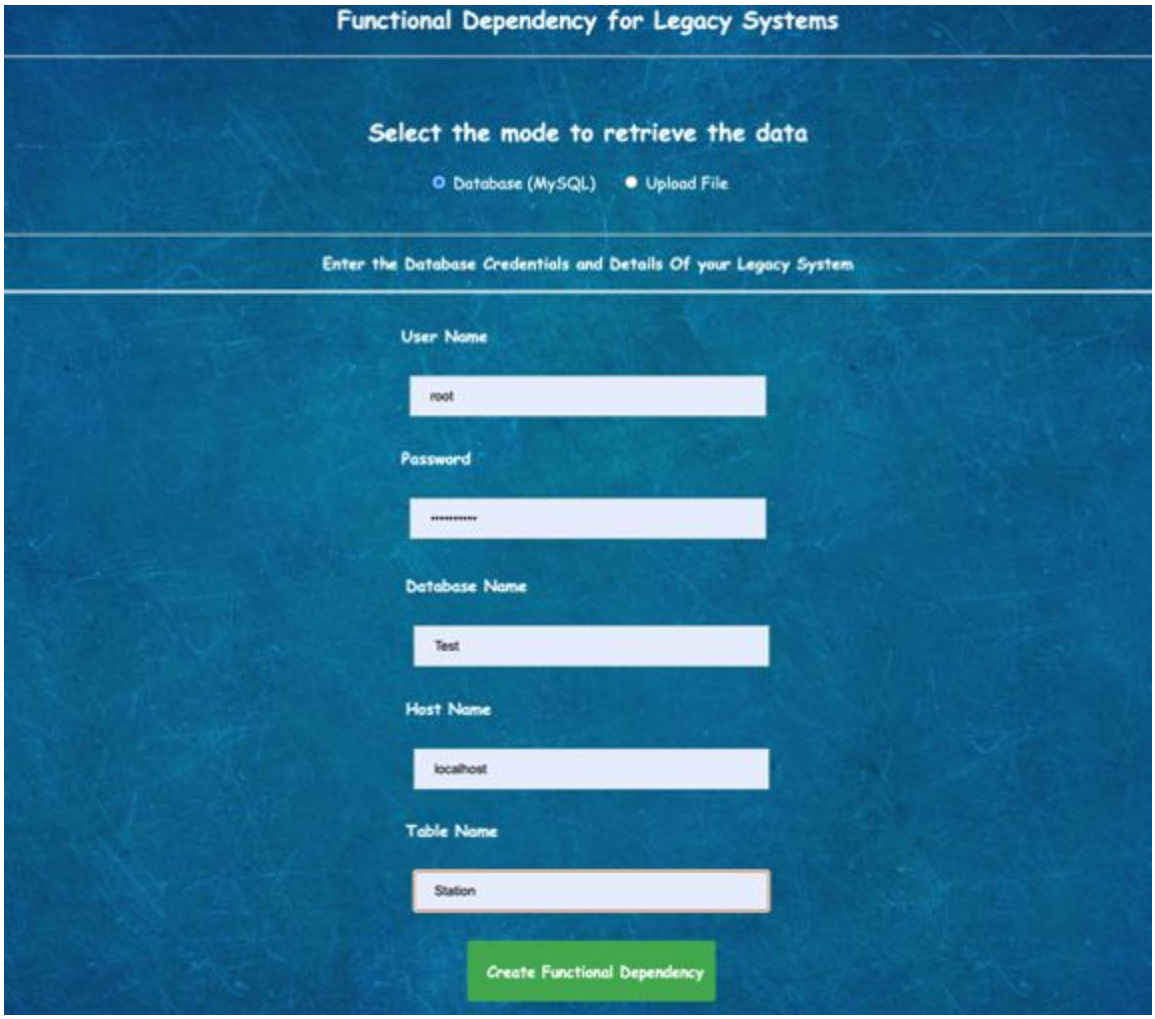
Analysis of output generated by the tool

- **Legacy System - Functional Dependency (FD)**

  *When the Data Retrieval Mode is selected as **Database**:*

  - The Database Credentials of the Legacy System is entered. User Name, Password, Database Name, Host Name & Table Name are entered as shown below.
  - After entering the Credentials, the "**Create Functional Dependency**" Button is clicked to display the respective FD's as shown below.

Database Credentials - FDs



Functional Dependency for the table Station

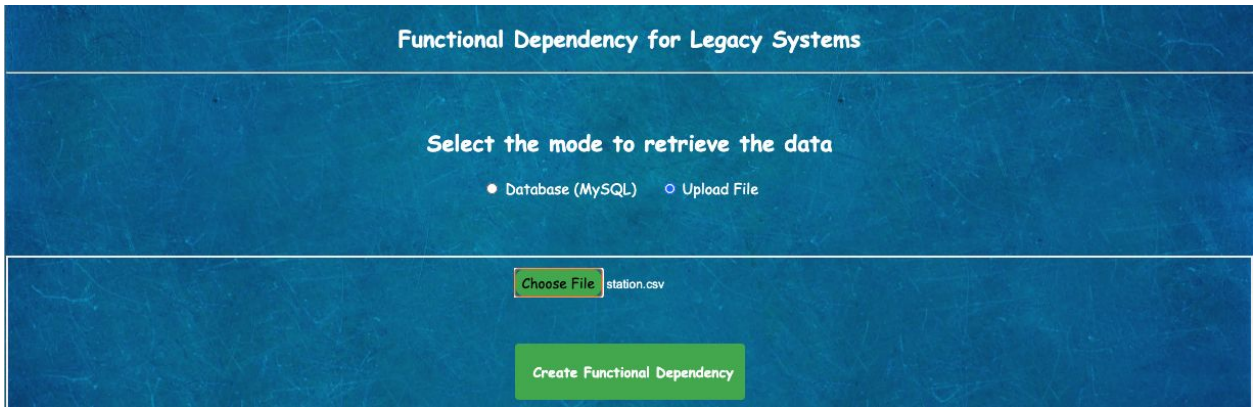
Analysis:

- Since the data values in the ID column is unique to all the columns : CITY, STATE, LAT_N & LONG_W, so ID->CITY, ID->STATE, ID->LAT_N, ID->LONG_W holds true
- CITY->STATE and CITY->LONG_W holds true since we can find the unique combination of values present for these two dependencies
- Similarly, for STATE->LONG_W, LAT_N->STATE, LAT_N->LONG_W & LONG_W->STATE also holds true because of the unique combination values.

When the Data Retrieval Mode is selected as Upload File :

- We uploaded a sample file (Station.csv) and the "**Create Functional Dependency**" Button is clicked to retrieve the FD's as shown below.
- We observe that the below FD's are created for the **station** file.

Upload File - FD's



Functional Dependency for the file Station

Analysis:

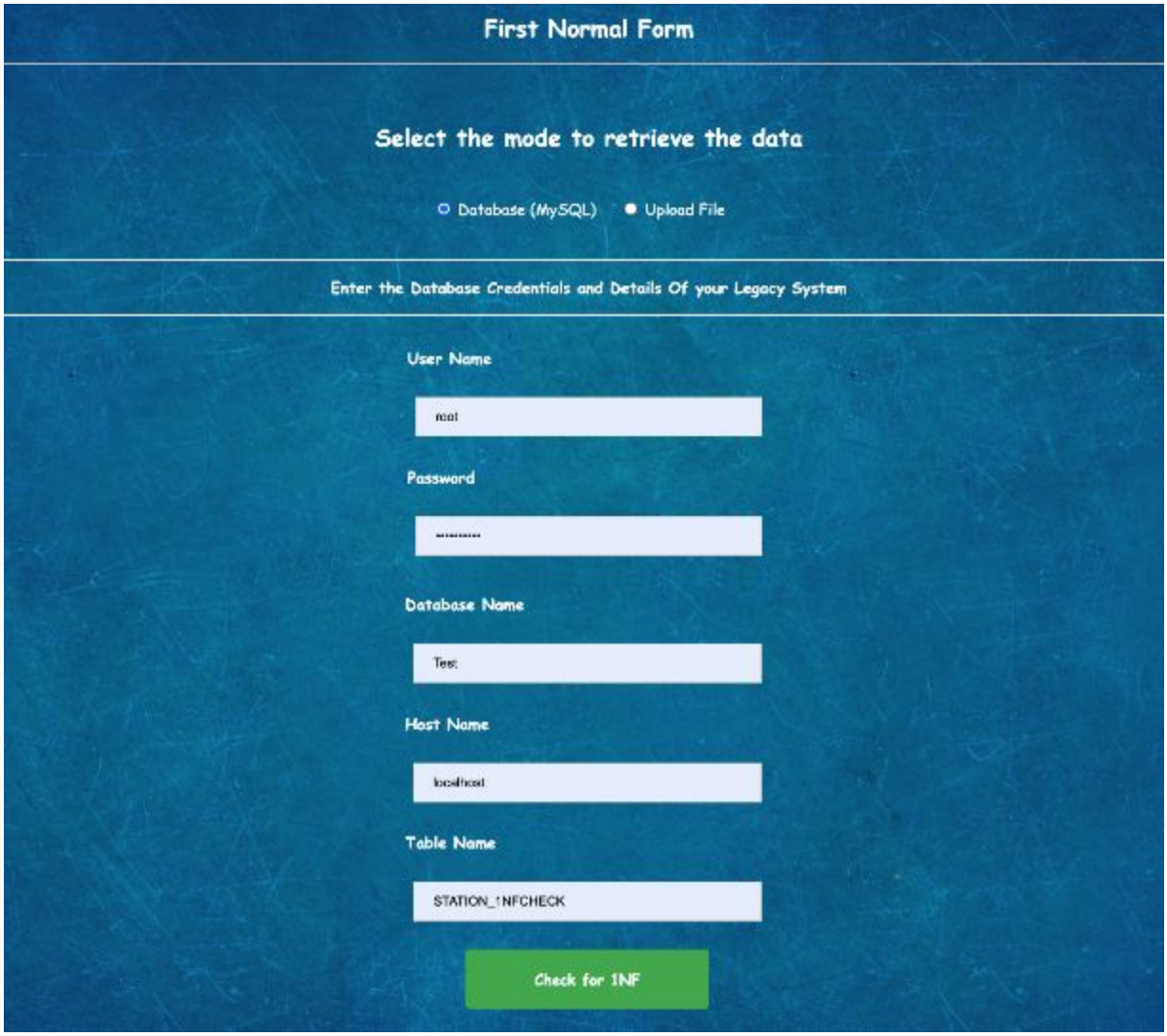
- Since the data values in the ID column are unique to all the columns : CITY, STATE, LAT_N & LONG_W, so ID->CITY, ID->STATE, ID->LAT_N, ID->LONG_W holds true.
- CITY->STATE and CITY->LONG_W holds true since we can find the unique combination of values present for these two dependencies.
- Similarly, for STATE->LONG_W, LAT_N->STATE, LAT_N->LONG_W & LONG_W->STATE also holds true because of the unique combination values.

- **First Normal Form (1NF)**

  When the Data Retrieval Mode is selected as Database:

  - The Database Credentials of the Legacy System is entered. User Name, Password, Database Name, Host Name & Table Name are entered as shown below.
  - After entering the Credentials, the "**Check for 1NF**" Button is clicked to check if the given table is in 1NF or not as shown below.

Database Credentials - 1NF



First Normal Form to Check if the Station_1NFCheck table is in 1NF

Table Name : 'STATION_1NFCHECK'

	ID	CITY	STATE	LAT_N	LONG_W
0	13	Phoenix,Mexico	AZ	33.0	112.0
1	44	Denver,California	CO	40.0	105.0
2	66	Caribou	ME	47.0	68.0
3	67	test	ME	47.0	68.0
4	100	Caribou	ME	11.0	68.0
5	67	test	ME	47.0	68.0
6	100	Caribou	ME	11.0	68.0

CHECKING FOR DUPLICATE COLUMNS

The Columns are unique for this Table since MYSQL doesnt allow duplicate columns while creating a table!

CHECKING FOR DUPLICATE ROWS

Below are the duplicate rows found for this Table :

	ID	CITY	STATE	LAT_N	LONG_W
5	67	test	ME	47.0	68.0
6	100	Caribou	ME	11.0	68.0

1NF PROPOSAL

Table Data is not in First Normal Form, since the below column has Multi-Valued Attributes for the given table

	CITY
0	Phoenix,Mexico
1	Denver,California

POSSIBLE PRIMARY KEYS CANDIDATES

There is no possible primary keys candidates found for this table!

Analysis:

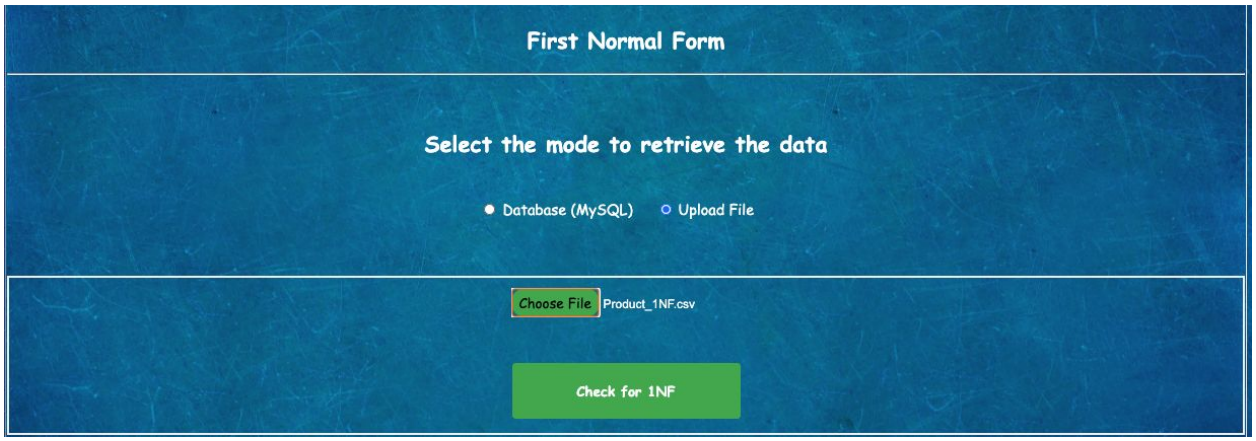
- We observed that the given table doesn't have duplicate columns since MySQL doesn’t allow duplicate columns while creating a table

- We also observed that there are duplicate rows present in the last two rows of the table Station_1NFCheck
- City column has multivalued attributes in row 0 and row 1 for the given table Station_1NFCheck

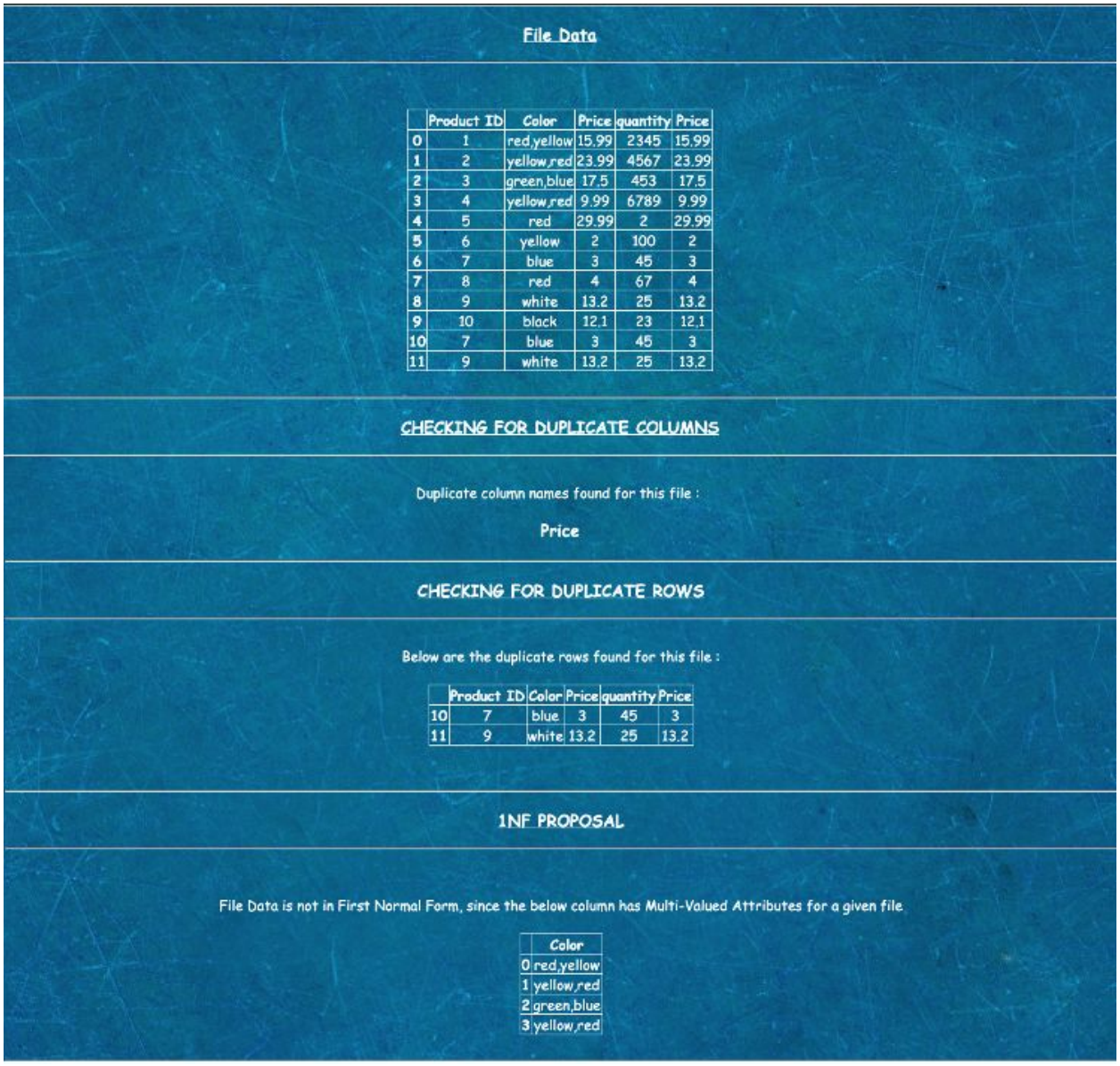
When the Data Retrieval Mode is selected as Upload File :

- We uploaded a sample file (**Product_1NF.csv**) and the "**Check for 1NF**" Button is clicked to check if the given file is in 1NF or not as shown below
- We observed that the given file has duplicate columns, duplicate rows, and multivalued attributes as shown below

Upload File - 1NF



First Normal Form to Check if the Product_1NFCheck File is in 1NF



File Data

	Product ID	Color	Price	quantity	Price
0	1	red,yellow	15.99	2345	15.99
1	2	yellow,red	23.99	4567	23.99
2	3	green,blue	17.5	453	17.5
3	4	yellow,red	9.99	6789	9.99
4	5	red	29.99	2	29.99
5	6	yellow	2	100	2
6	7	blue	3	45	3
7	8	red	4	67	4
8	9	white	13.2	25	13.2
9	10	black	12.1	23	12.1
10	7	blue	3	45	3
11	9	white	13.2	25	13.2

CHECKING FOR DUPLICATE COLUMNS

Duplicate column names found for this file :

Price

CHECKING FOR DUPLICATE ROWS

Below are the duplicate rows found for this file :

	Product ID	Color	Price	quantity	Price
10	7	blue	3	45	3
11	9	white	13.2	25	13.2

1NF PROPOSAL

File Data is not in First Normal Form, since the below column has Multi-Valued Attributes for a given file

	Color
0	red,yellow
1	yellow,red
2	green,blue
3	yellow,red

Analysis:

- We observed that for a given file “Product_1NFCheck”, the price column is duplicated
- We also observed that there are duplicate rows present in the last two rows of the file “Product_1NFCheck”
- The color column has multivalued attributes in the first four rows for the given file “Product_1NFCheck”

Second Normal Form (2NF)

We have entered a set of attributes and their functional dependencies in the 2NF tab.

Attributes given are: {beer,brewery,strength,city,region,warehouse,quantity} and Functional Dependencies is given as below.

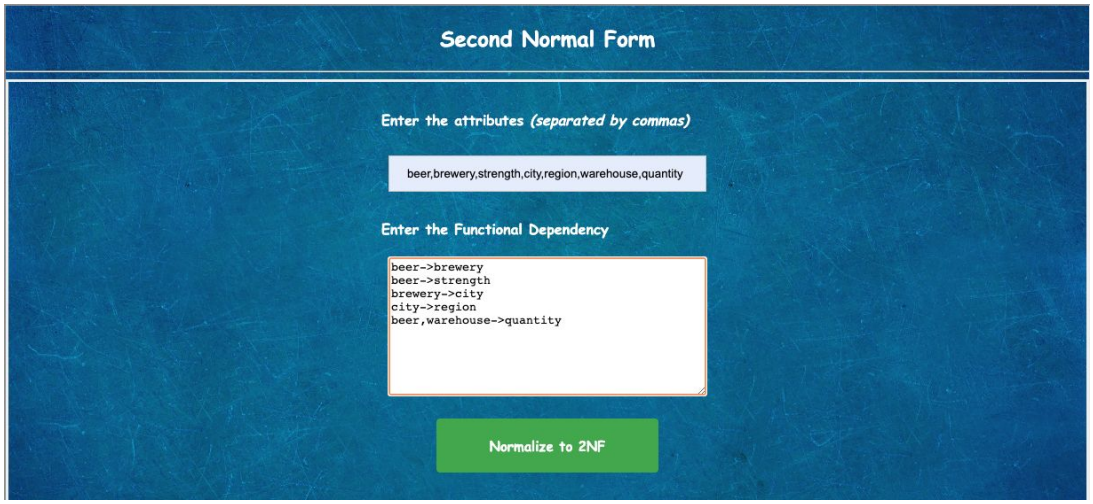
beer->brewery

brewery->city

city->region

beer,warehouse->quantity

Normalize to 2NF button is clicked and the normalized relation is displayed below.


Second Normal Form
Enter the attributes (separated by commas)
beer,brewery,strength,city,region,warehouse,quantity
Enter the Functional Dependency
beer->brewery
beer->strength
brewery->city
city->region
beer,warehouse->quantity
Normalize to 2NF


Given FD List
['beer'] -> ['brewery']
['beer'] -> ['strength']
['brewery'] -> ['city']
['city'] -> ['region']
['beer', 'warehouse'] -> ['quantity']
MinCover
['beer'] -> ['brewery']
['beer'] -> ['strength']
['brewery'] -> ['city']
['city'] -> ['region']
['warehouse', 'beer'] -> ['quantity']
Keys
[('warehouse', 'beer')]
2NF is in violation for
['beer'] -> ['brewery']
['beer'] -> ['strength']
2NF PROPOSAL
Merge the below functional dependencies to create a new relation
('brewery', 'beer') : ['beer'] -> ['brewery']
('beer', 'strength') : ['beer'] -> ['strength']
Create the below relations along with above 2NF proposals
('brewery', 'city') : ['brewery'] -> ['city']
('city', 'region') : ['city'] -> ['region']
('warehouse', 'beer', 'quantity') : ['beer', 'warehouse'] -> ['quantity']

Analysis:

- As part of 2NF decomposition based on the user's specifications, we first calculate the full closures, minimal cover and then determine the keys.
- As a next step, we identify the partial dependencies and propose the appropriate solution.
- In our example, we have two violations: **beer->brewery, beer->strength** for which the proposal is to merge both to create a new relation.
- Along with the above proposal, should the user only want normalization till 2NF, he needs to create the other relations which were not a violation of 2NF as seen in the attached screenshot.

Third Normal Form (3NF)

We have entered a set of attributes and their functional dependencies in the 3NF tab.

Attributes: {beer,brewery,strength,city,region,warehouse,quantity}

Functional Dependencies :

beer->brewery

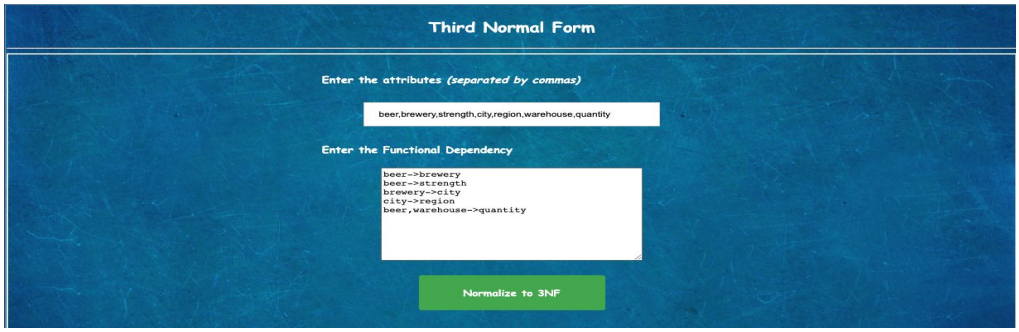
beer->strength

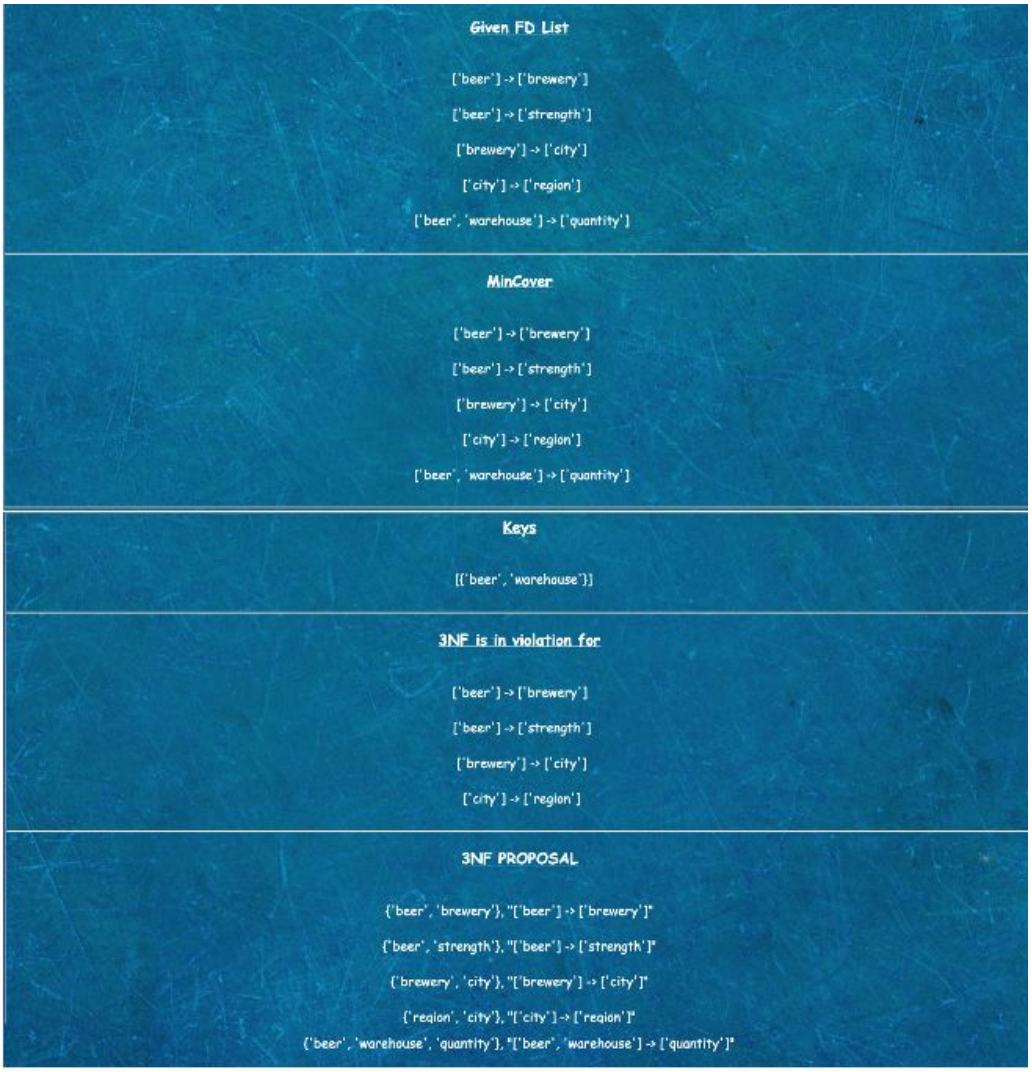
brewery->city

city->region

beer,warehouse->quantity

Normalize to 3NF button is clicked and the normalized relation is displayed as below.





Analysis:

- [‘beer’] -> [‘brewery’] is transitively dependent on [‘brewery’] -> [‘city’], hence a separate relation is created
- [‘brewery’] -> [‘city’] is transitively dependent on [‘city’] -> [‘region’], hence a separate relation is created
- ['beer'] -> ['brewery'] and ['beer'] -> ['strength'] are the remaining functional dependency anomalies which will be handled in BCNF

- **Boyce-Codd Normal Form (BCNF)**

  We have entered a set of attributes and their functional dependencies in the BCNF tab.

  Attributes given are: {beer,brewery,strength,city,region,warehouse,quantity} and Functional Dependencies is given as below.

  beer->brewery

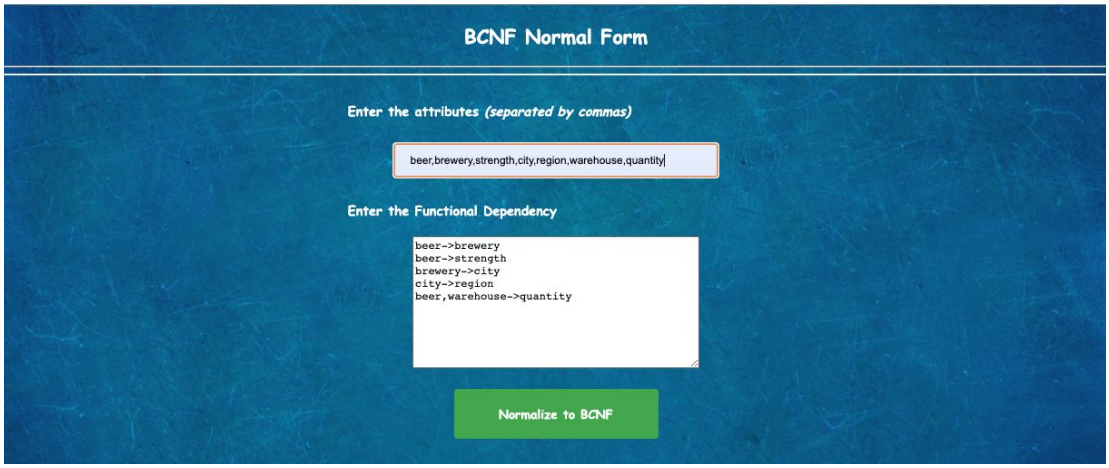
  beer->strength

  brewery->city

  city->region

  beer,warehouse->quantity

  Normalize to BCNF button is clicked and the normalized relation is displayed as below.


BCNF Normal Form
Enter the attributes (separated by commas)
beer,brewery,strength,city,region,warehouse,quantity
Enter the Functional Dependency
beer->brewery
beer->strength
brewery->city
city->region
beer,warehouse->quantity
Normalize to BCNF


Given FD List
['beer'] -> ['brewery']
['beer'] -> ['strength']
['brewery'] -> ['city']
['city'] -> ['region']
['beer', 'warehouse'] -> ['quantity']
MinCover
['beer'] -> ['brewery']
['beer'] -> ['strength']
['brewery'] -> ['city']
['city'] -> ['region']
['beer', 'warehouse'] -> ['quantity']
Keys
[{'beer', 'warehouse'}]
BCNF is in violation for
['beer'] -> ['brewery']
['beer'] -> ['strength']
['brewery'] -> ['city']
['city'] -> ['region']
BCNF PROPOSAL
{'region', 'city'}, "['city'] -> ['region']"
{'brewery', 'city'}, "['brewery'] -> ['city']"
{'beer', 'brewery', 'strength'}, "['beer'] -> ['brewery'], ['beer'] -> ['strength']"
{'beer', 'warehouse', 'quantity'}, "['beer', 'warehouse'] -> ['quantity']"

Analysis:

- Should a user directly want to normalise till BCNF, then with the inputs : FDs and relations, we first identify the violations and then proceed with decomposing the relations from 2NF through BCNF by iterating through all steps mentioned earlier for each of the normal forms
- In the above example, all the other violations are handled in a similar fashion as specified earlier which would leave us with ['beer'] -> ['brewery'] and ['beer'] -> ['strength'] as the remaining functional dependency anomalies, hence a separate relation is created for {'beer', 'brewery', 'strength'} which satisfies BCNF

7.3 Compare output against hypothesis

Test Case ID	Test Case Description	Test Steps	Expected Result	Actual Result
TC01	Check the input for 1NF	Upload the data into a tool to check for 1NF	There should be No repetition groups. Where there are users to be warned.	The resultant output from 1NF produced no repetitive groups. Where there are repetitive groups, warning messages will be displayed.
TC02	Check for any duplicate data present in the relation	Retrieve the table data and check if the data is not being duplicated in a relation by executing the duplicate check query.	There should not be any duplicate data in a given relation. Where there are users to be warned.	Where there were duplicates, warning messages have been displayed.
TC03	Check the result if it satisfies minimal cover requirements	Enter the FD's, for the attributes to check if the result satisfies minimal cover.	There should be a minimal cover for the specified input.	Generated output produced at least one cover as expected per minimal cover algorithm.

TC04	Check if the result satisfies 2NF requirements	A. Specify the Relational attributes B. Specify Functional dependencies C. Run the code to check the output.	The resultant code should never have any partial functional dependencies	Generated output matches
TC05	Check if the result satisfies 3NF properties	Compare the result with the business specified requirements.	A. There should not be any transitive dependencies. B. There should not be any loss of information.	Implementation of 3NF algorithm resulted in: a. Removal of Transitive dependencies b. It ensured lossless decomposition.
TC06	Check if the result satisfies BCNF properties		All the dependencies other inclusive of non-prime attributes have to be removed	BCNF decomposition is as expected and resulted in the removal of dependencies where determinants were not necessarily candidate keys.

When there are repetitive groups specified, we would check if there are any cases of repetitive /duplicate groups and warn users of any repetitive and duplicate groups. Using the minimal cover algorithm, we generated a minimal cover of FDs. Then, using the 2NF algorithmic implementation, we had generated the new proposals that suggested new relations where there were violations of the Second Normal Form.

As a natural progression into the 3NF, we've generated the output along with the proposals for new relations that address transitive dependencies where there were violations.

As a final step of implementation in our project, we generated output where the violations in relation to any other non-prime attributes having dependencies were addressed and the necessary proposals were presented to the users.

All the above steps/phases have produced expected results. Hence we conclude that the designed normalization model for checking and proposing appropriate normalization up to BCNF meets the expectations and therefore meets the hypothesis stated before designing the normalization model.

7.4 Abnormal case explanation (the most important task if you have it)

Where there are cases of attributes being mentioned in the functional dependencies(FDs) but not being a part of the relation set (R), the decomposition wouldn't proceed further as the attribute(s) in reference isn't part of the keyed-in relation set. Our normalization tool would validate such scenarios and abort the normalization process and the error handle would warn the user:

> *"Functional dependency has attributes that are not in relation set"*

7.5 Static regression (if you have a complex curve to fit)

Since our project involves automation of normalization and we do not have any quantitative variables to calculate or predict, this is not applicable for this project

7.6 Discussion

Our tool for automated normalization until BCNF is primarily designed to automate otherwise tedious, manual and often overlooked to a point where the anomalies are only found out at the later stages of implementation of a project. All the normal forms generated till BCNF have been validated and are supported by the tool. The system performs a complete decomposition where no extra attributes are added or removed. The separation of normalization functionality from the user interface helps us to understand each component of the normalization form.

The implementation is based on the use of existing algorithms which are widely acknowledged in the field of normalization of databases.

Through the implementation process, we have found various scenarios pertaining to the behaviour of the algorithms where there are multiple attributes on the right hand side of the specified functional dependencies, cases where there could be extraneous dependencies being specified while the original relation set does not include the attributes in discussion, cyclic dependencies being specified(while this is handled, it is

advisable to refrain from having cyclic dependencies to the possible extent especially in cases where there are a lot of attributes in view of performance and accuracy.

8 Conclusions and Recommendations

8.1 Summary and Conclusions

Only having a good relational database software is not enough to avoid data redundancy. Normalization as a process is required for evaluating and correcting table structures to minimize data redundancies, thereby reducing the likelihood of data anomalies.

The Normalization checker tool developed in Python can be used by anyone trying to analyze their entity-relationship model, to understand database normalization concepts, or to get normalization proposals for their existing system.

The tool can be used to check normal forms for a legacy database system and by providing functional dependency inputs through the web interface. The tool involves assigning attributes to tables based on the concept of determination.

Although we can use our tool to achieve high normal forms -up to BCNF, the user must be mindful of the fact that higher normal forms does not mean higher efficiency. Generally, the higher the normal form, the more joins are required by the database system to respond to end-user queries.

Thus to design any efficient system, an end-to-end performance on all connected systems must be considered to achieve overall fast performance.

8.2 Recommendations for Future Studies

- Currently, we have implemented normal forms from 2NF to BCNF. In the future, based on the use cases, 4NF and 5NF can be implemented
- Comprehensive testing with more use cases needs to be done
- Implementation of normalization checking for legacy databases has been done using MySQL
- Compatibility with other tools and versions needs to be verified
- Further enhancement for checking of normalization by uploading database schema for cases where we do not have access to the database
- In some systems where higher normal forms are not desired require denormalization to be done and that can be included in the scope to convert between any two normal forms

9 Bibliography

[1] Du H and Wery L (1999), “Micro: A normalization tool for relational database designers”, journal of network and computer application, Vol.22, pp.215-232.

[2] Yazici A, Ziya K (2007), “JMathNorm: A database normalization tool using mathematica”, In proc. international. conference on computational science, pp.186-193.

[3] Dongare, Y., Dhabe, P., & Deshmukh, S. (2011). RDBNorma: A semi-automated tool for relational database schema normalization up to the third normal form. International Journal of Database Management Systems, 3 (1), (pp. 133-153).

[4] Yazici, A. and Karakaya, Z.: Normalizing Relational Database Schemas Using Mathematica, LNCS, Springer-Verlag, Vol.3992 (2006) 375-382.

[5] Kung, H. and Case, T.: Traditional and Alternative Database Normalization Techniques: Their Impacts on IS/IT Students’ Perceptions and Performance, International Journal of Information Technology Education, Vol.1, No.1 (2004) 53-76.

[6] M. Demba - An Algorithmic Approach to Database Normalization - International Journal of Digital Information and Wireless Communications (IJDIWC) 3(2): 197-205 The Society of Digital Information and Wireless Communications, 2013 (ISSN: 2225-658X)

[7] Vangipuram, R., Velputa, R., Sravya, V.: A Web-Based Relational database design Tool to Perform Normalization, International Journal of Wisdom Based Computing, Vol.1(3) (2011).

[8] P.B. Alappanavar1, Radhika Grover2, Srishti Hunjan3, Dhiraj Patil4 Yuvraj Girnar5 - Automating the Normalization Process for Relational Database Mod, Vol. 3, Issue 1, January -February 2013, pp.1826-1831

[9] Adam Floyd, Hongbo Du - EDNA, a Software Tool for Verifying the Normalization of Relations during the Logical Database Design Process, Conference Paper · July 2014.

[10] G.Sunitha, Dr.A.Jaya - A KNOWLEDGE-BASED APPROACH FOR AUTOMATIC DATABASE NORMALIZATION, International Journal of Advanced Research in Computer Engineering & Technology (IJARCET) Volume 2, No 5, May 2013

[11] Amir Bahmani, M. Naghibzadeh - Automatic database normalization and primary key generation, June 2008, Canadian Conference on Electrical and Computer Engineering, DOI: 10.1109/CCECE.2008.4564486

10 Appendices

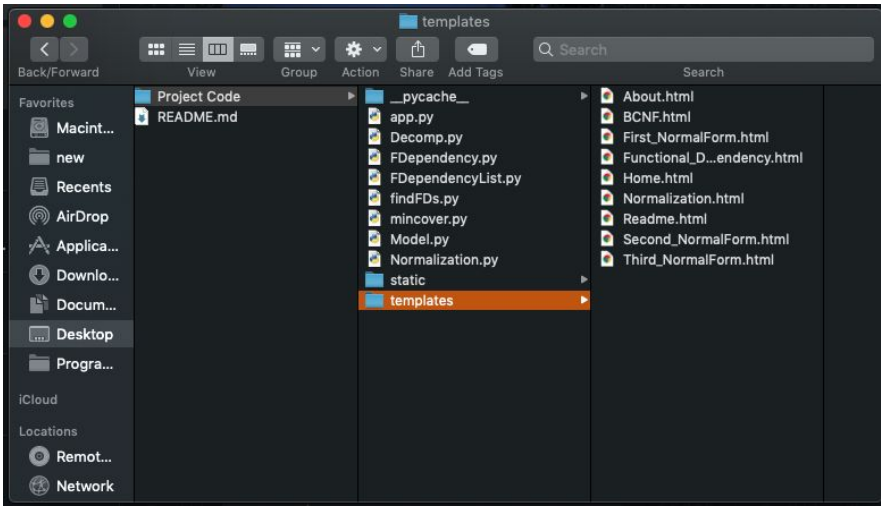
10.1 Program source code with documentation

Python Files

Python File Name	Description of the File
app.py	Main Python file to execute the automated normalization checker tool
Decomp.py	Decomposition file source code does the actual synthesis by creating new relations as needed.
FdepedencyList.py	Python file to return the FD's into lists
Fdependency.py	Python file to check the Functional Dependency
findFDs.py	Python file that finds all the full closures for the specified functional dependencies X->rhs with X subset of LHS
Mincover.py	Python file to return the Minimum cover of the attributes FD's
Normalization.py	Python file which determines the 2NF, 3NF & BCNF logics

Templates

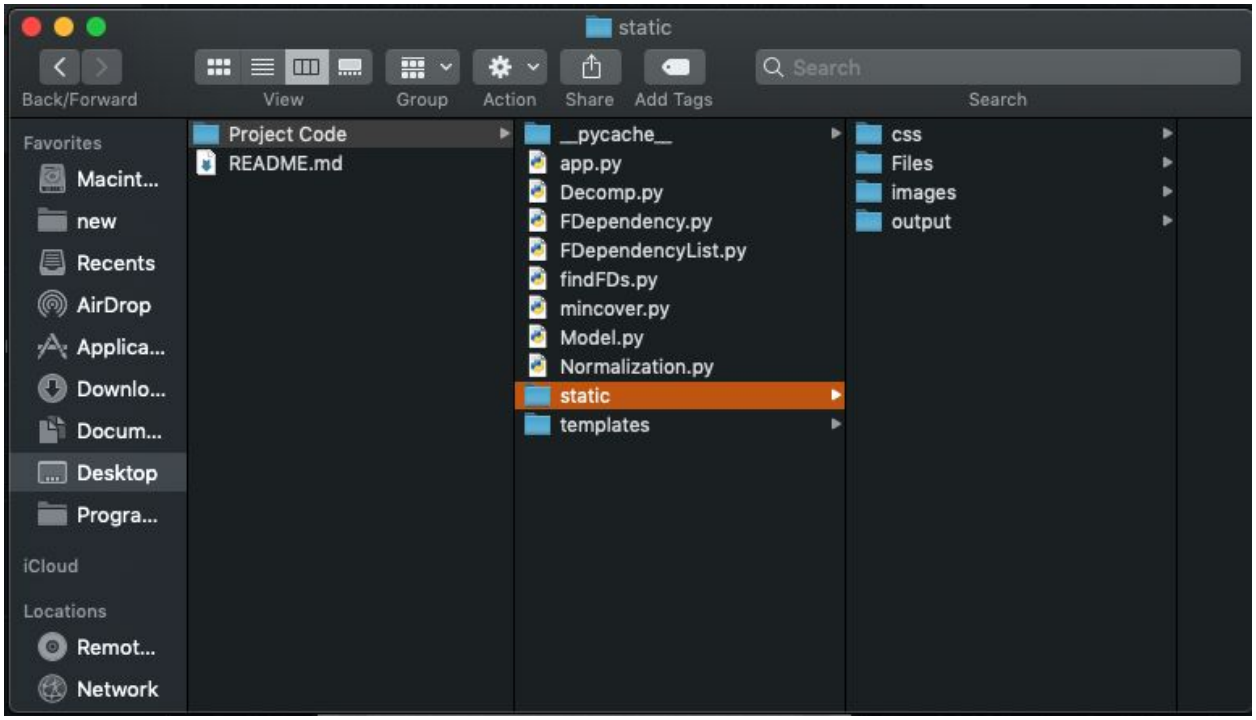
The below HTML files are placed in the templates folder. For each section in the tool, a separate HTML is created where the inputs are entered and outputs are generated based on the inputs provided.



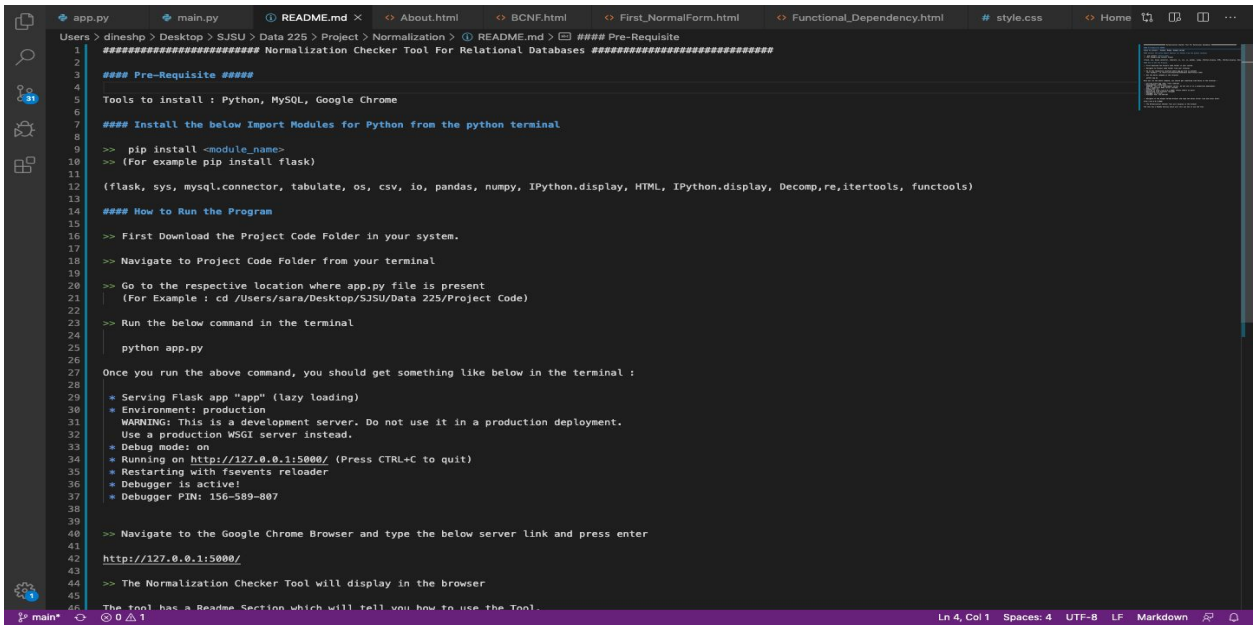
Static Files

Static Folder contains the below folders

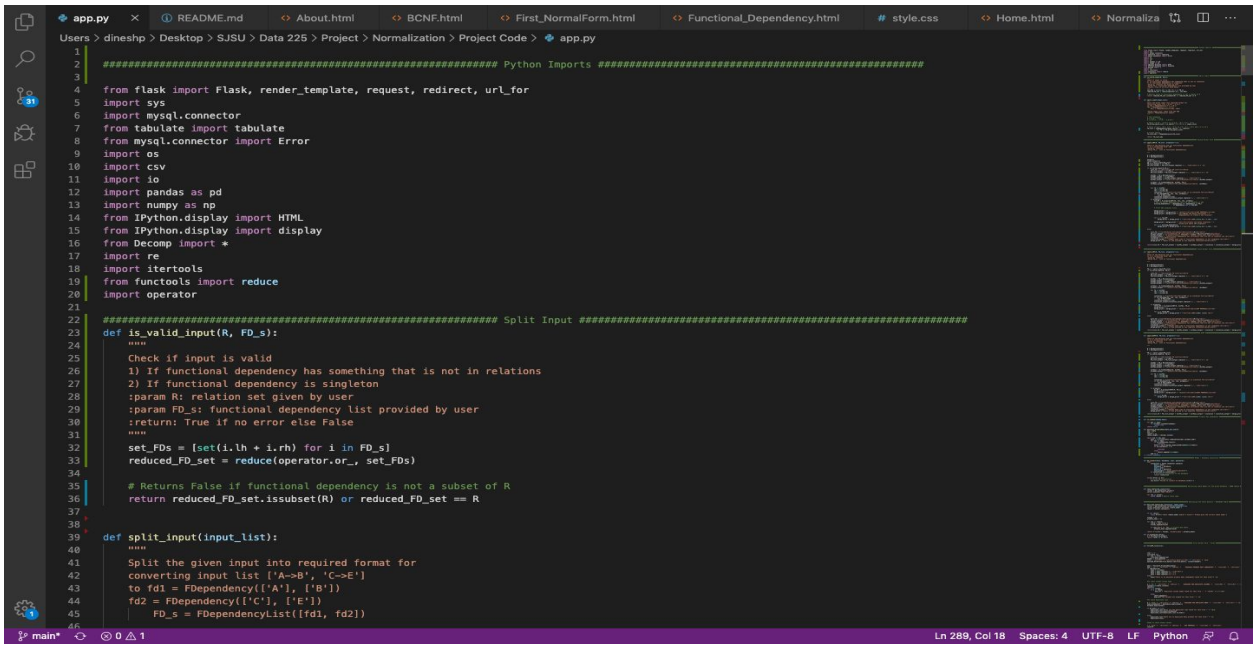
- CSS - contains cascading styling sheet files
- Files - Input Files
- Images
- Output Files


static
Back/Forward
View
Group
Action
Share
Add Tags
Search
Search
Favorites
Macint...
new
Recents
AirDrop
Applica...
Downlo...
Docum...
Desktop
Progra...
iCloud
Locations
Remot...
Network
Project Code
README.md
__pycache__
app.py
Decomp.py
FDependency.py
FDependencyList.py
findFDs.py
mincover.py
Model.py
Normalization.py
static
templates
css
Files
images
output

Read Me

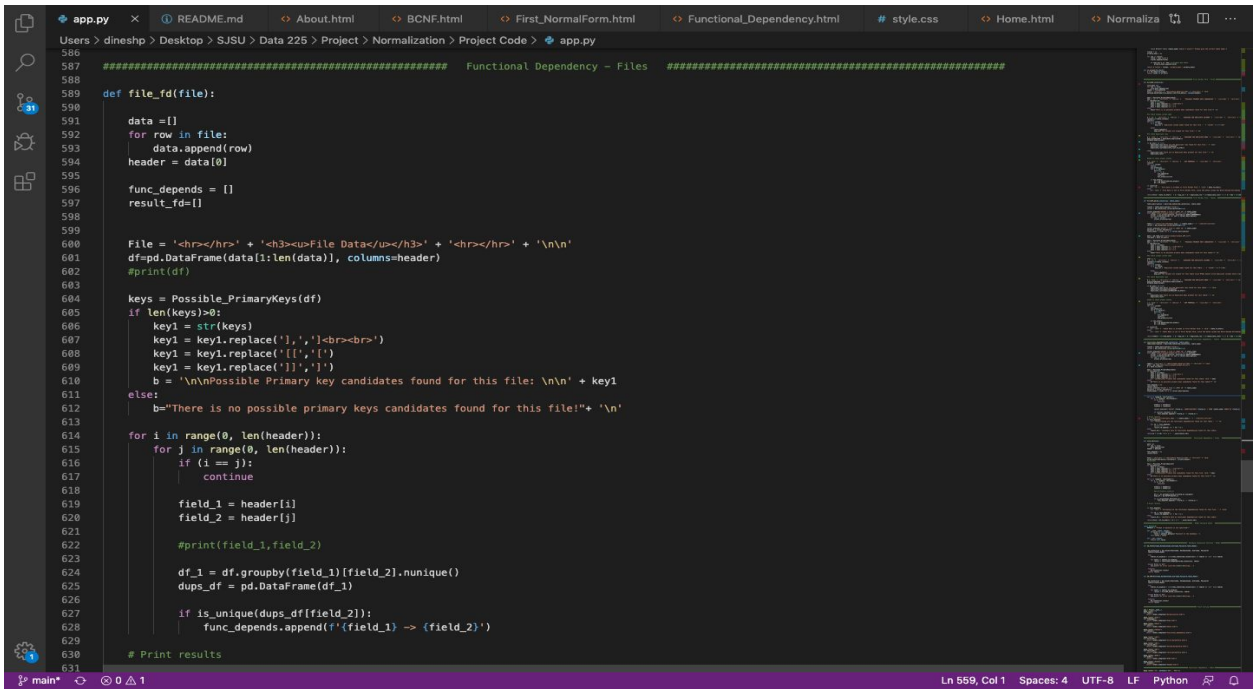

app.py
main.py
README.md
About.html
BCNF.html
First_NormalForm.html
Functional_Dependency.html
style.css
Home
Users > dineshp > Desktop > SJSU > Data 225 > Project > Normalization > README.md > #### Pre-Requisite
########################## Normalization Checker Tool For Relational Databases ##############################
Pre-Requisite
Tools to install : Python, MySQL, Google Chrome
Install the below Import Modules for Python from the python terminal
>> pip install <module_name>
>> (For example pip install flask)
(flask, sys, mysql.connector, tabulate, os, csv, io, pandas, numpy, IPython.display, HTML, IPython.display, Decomp,re,itertools, functools)
How to Run the Program
>> First Download the Project Code Folder in your system.
>> Navigate to Project Code Folder from your terminal
>> Go to the respective location where app.py file is present
(For Example : cd /Users/sara/Desktop/SJSU/Data 225/Project Code)
>> Run the below command in the terminal
python app.py
Once you run the above command, you should get something like below in the terminal :
* Serving Flask app "app" (lazy loading)
* Environment: production
WARNING: This is a development server. Do not use it in a production deployment.
Use a production WSGI server instead.
* Debug mode: on
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
* Restarting with fsevents reloader
* Debugger is active!
* Debugger PIN: 156-589-807
>> Navigate to the Google Chrome Browser and type the below server link and press enter
http://127.0.0.1:5000/
>> The Normalization Checker Tool will display in the browser
The tool has a Readme Section which will tell you how to use the Tool.
main*
Ln 4, Col 1
Spaces: 4
UTF-8
LF
Markdown

Final Code Snippet (app.py)



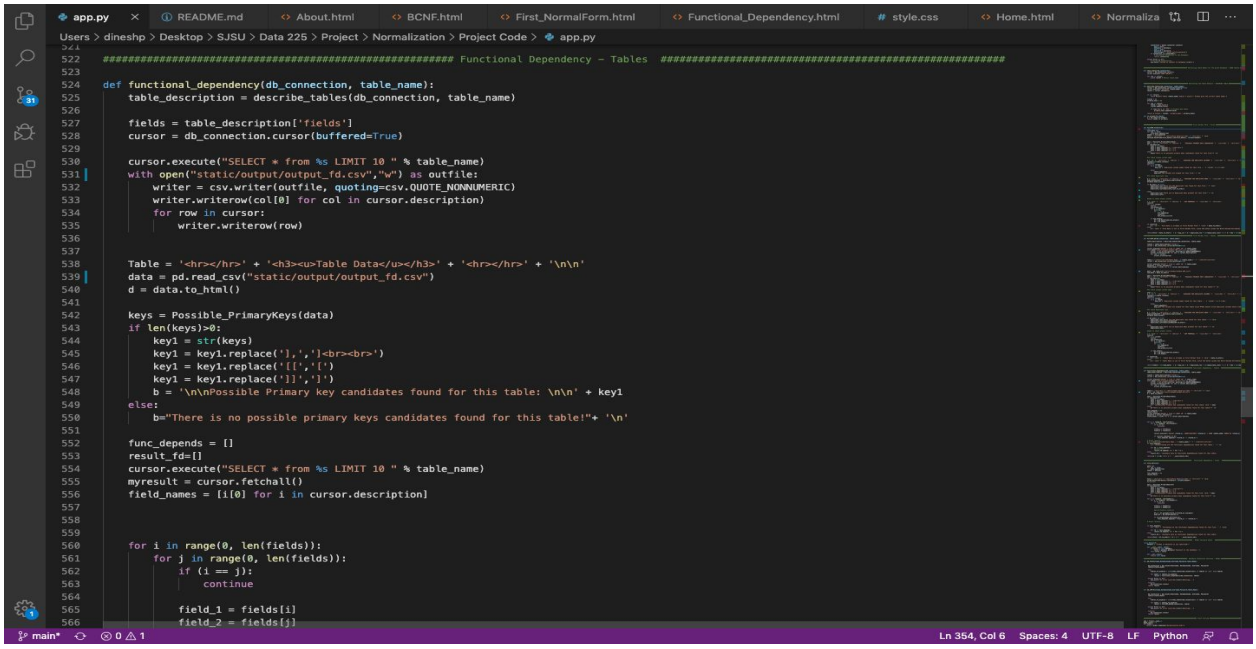
```
app.py | README.md | About.html | BCNF.html | First_NormalForm.html | Functional_Dependency.html | style.css | Home.html | Normaliza
Users > dineshp > Desktop > SJSU > Data 225 > Project > Normalization > Project Code > app.py

########################################################## Python Imports ##################################################

from flask import Flask, render_template, request, redirect, url_for
import sys
import mysql.connector
from tabulate import tabulate
from mysql.connector import Error
import os
import csv
import io
import pandas as pd
import numpy as np
from IPython.display import HTML
from IPython.display import display
from Decomp import *
import re
import itertools
from functools import reduce
import operator

########################################################## Split Input ##########################################################
def is_valid_input(R, FD_s):
    """
    Check if input is valid
    1) If functional dependency has something that is not in relations
    2) If functional dependency is singleton
    :param R: relation set given by user
    :param FD_s: functional dependency list provided by user
    :return: True if no error else False
    """
    set_FDs = [set(i.lh + i.rh) for i in FD_s]
    reduced_FD_set = reduce(operator.or_, set_FDs)

    # Returns False if functional dependency is not a subset of R
    return reduced_FD_set.issubset(R) or reduced_FD_set == R


def split_input(input_list):
    """
    Split the given input into required format for
    converting input list ['A->B', 'C->E']
    to fd1 = FDependency(['A'], ['B'])
    fd2 = FDependency(['C'], ['E'])
        FD_s = FDependencyList([fd1, fd2])

main*  0 ⚠ 1                                  Ln 289, Col 18  Spaces: 4  UTF-8  LF  Python
```

FDs: (Files)



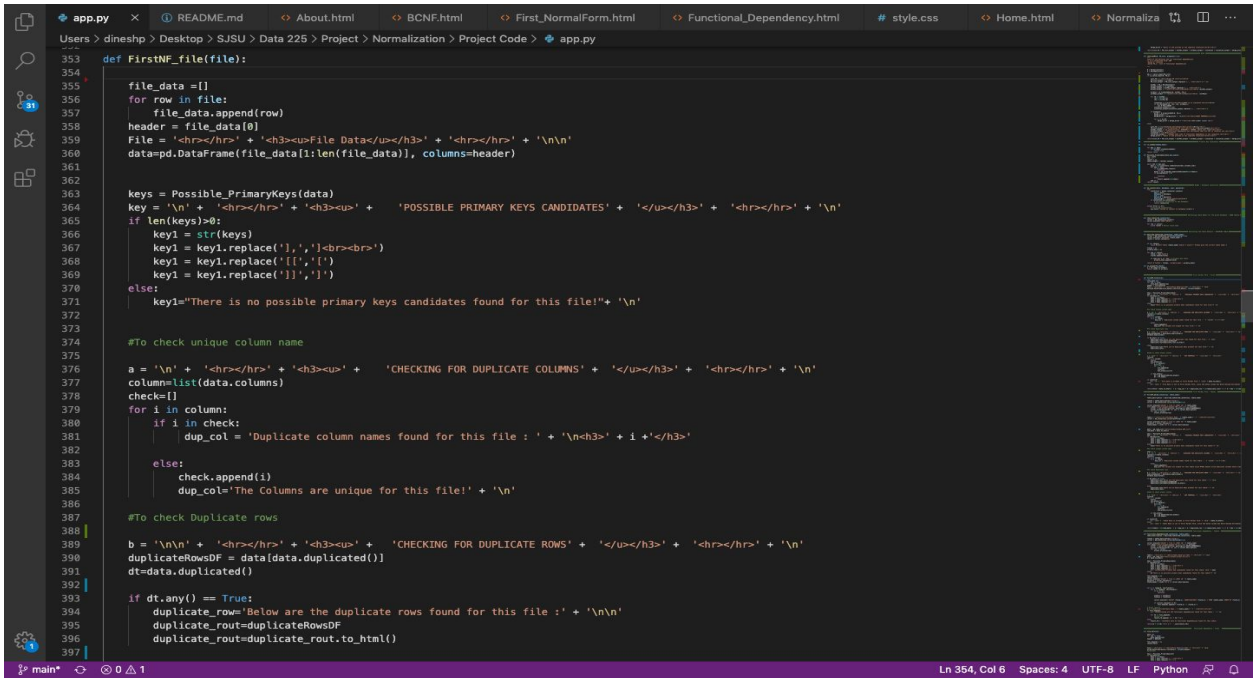
```
app.py | README.md | About.html | BCNF.html | First_NormalForm.html | Functional_Dependency.html | style.css | Home.html | Normaliza
Users > dineshp > Desktop > SJSU > Data 225 > Project > Normalization > Project Code > app.py

586
587 ##################################################   Functional Dependency - Files   ##################################################
588
589 def file_fd(file):
590
591     data =[]
592     for row in file:
593         data.append(row)
594     header = data[0]
595
596     func_depends = []
597     result_fd=[]
598
599
600     File = '<hr></hr>' + '<h3><u>File Data</u></h3>' + '<hr></hr>' + '\n\n'
601     df=pd.DataFrame(data[1:len(data)], columns=header)
602     #print(df)
603
604     keys = Possible_PrimaryKeys(df)
605     if len(keys)>0:
606         key1 = str(keys)
607         key1 = key1.replace('],',']<br><br>')
608         key1 = key1.replace('[[','[')
609         key1 = key1.replace(']]',']')
610         b = '\n\nPossible Primary key candidates found for this file: \n\n' + key1
611     else:
612         b="There is no possible primary keys candidates found for this file!"+ '\n'
613
614     for i in range(0, len(header)):
615         for j in range(0, len(header)):
616             if (i == j):
617                 continue
618
619             field_1 = header[i]
620             field_2 = header[j]
621
622             #print(field_1,field_2)
623
624             df_1 = df.groupby(field_1)[field_2].nunique()
625             dups_df = pd.DataFrame(df_1)
626
627             if is_unique(dups_df[field_2]):
628                 func_depends.append(f'{field_1} -> {field_2}')
629
630     # Print results
631

main*  0 ⚠ 1                                  Ln 559, Col 1  Spaces: 4  UTF-8  LF  Python
```

FDs: (Tables)



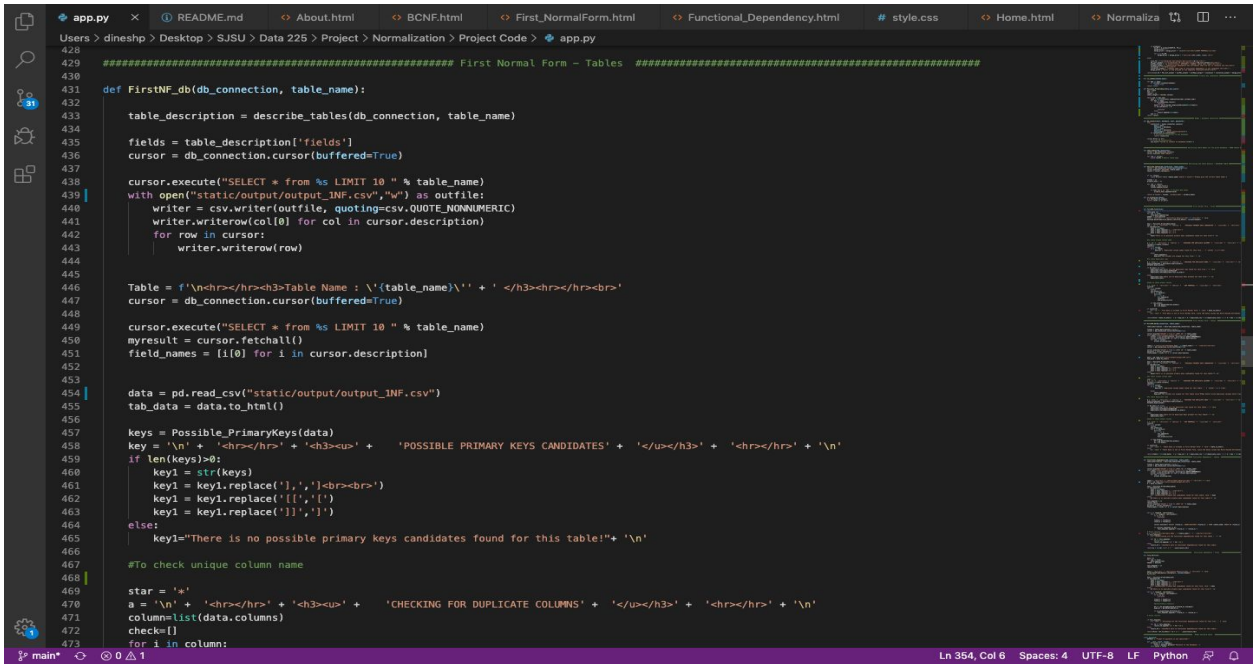
```
def functional_dependency(db_connection, table_name):
    table_description = describe_tables(db_connection, table_name)

    fields = table_description['fields']
    cursor = db_connection.cursor(buffered=True)

    cursor.execute("SELECT * from %s LIMIT 10 " % table_name)
    with open("static/output/output_fd.csv","w") as outfile:
        writer = csv.writer(outfile, quoting=csv.QUOTE_NONNUMERIC)
        writer.writerow(col[0] for col in cursor.description)
        for row in cursor:
            writer.writerow(row)


    Table = '<hr></hr>' + '<h3><u>Table Data</u></h3>' + '<hr></hr>' + '\n\n'
    data = pd.read_csv("static/output/output_fd.csv")
    d = data.to_html()

    keys = Possible_PrimaryKeys(data)
    if len(keys)>0:
        key1 = str(keys)
        key1 = key1.replace('],',']<br><br>')
        key1 = key1.replace('[[','[')
        key1 = key1.replace(']]',']')
        b = '\n\nPossible Primary key candidates found for this table: \n\n' + key1
    else:
        b="There is no possible primary keys candidates found for this table!"+ '\n'

    func_depends = []
    result_fd=[]
    cursor.execute("SELECT * from %s LIMIT 10 " % table_name)
    myresult = cursor.fetchall()
    field_names = [i[0] for i in cursor.description]



    for i in range(0, len(fields)):
        for j in range(0, len(fields)):
            if (i == j):
                continue

            field_1 = fields[i]
            field_2 = fields[j]
```

1NF: (Files)

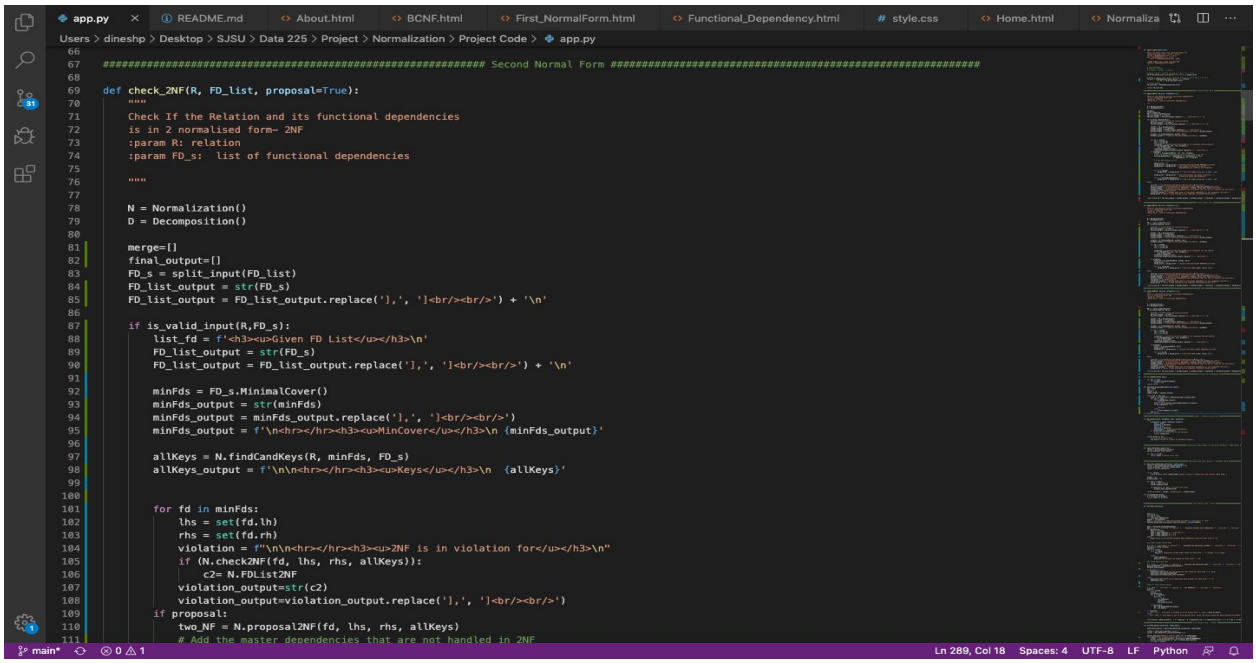


```
def FirstNF_file(file):

    file_data =[]
    for row in file:
        file_data.append(row)
    header = file_data[0]
    File = '<hr></hr>' + '<h3><u>File Data</u></h3>' + '<hr></hr>' + '\n\n'
    data=pd.DataFrame(file_data[1:len(file_data)], columns=header)


    keys = Possible_PrimaryKeys(data)
    key = '\n' +  '<hr></hr>' + '<h3><u>' +    'POSSIBLE PRIMARY KEYS CANDIDATES' +  '</u></h3>' +  '<hr></hr>' + '\n'
    if len(keys)>0:
        key1 = str(keys)
        key1 = key1.replace('],',']<br><br>')
        key1 = key1.replace('[[','[')
        key1 = key1.replace(']]',']')
    else:
        key1="There is no possible primary keys candidates found for this file!"+ '\n'


    #To check unique column name

    a = '\n' +  '<hr></hr>' + '<h3><u>' +    'CHECKING FOR DUPLICATE COLUMNS' +  '</u></h3>' +  '<hr></hr>' + '\n'
    column=list(data.columns)
    check=[]
    for i in column:
        if i in check:
            dup_col = 'Duplicate column names found for this file : ' + '\n<h3>' + i +'</h3>'

        else:
            check.append(i)
            dup_col='The Columns are unique for this file!' + '\n'

    #To check Duplicate rows

    b = '\n\n' +  '<hr></hr>' + '<h3><u>' +   'CHECKING FOR DUPLICATE ROWS' +  '</u></h3>' +  '<hr></hr>' + '\n'
    duplicateRowsDF = data[data.duplicated()]
    dt=data.duplicated()

    if dt.any() == True:
        duplicate_row='Below are the duplicate rows found for this file :' + '\n\n'
        duplicate_rout=duplicateRowsDF
        duplicate_rout=duplicate_rout.to_html()
```

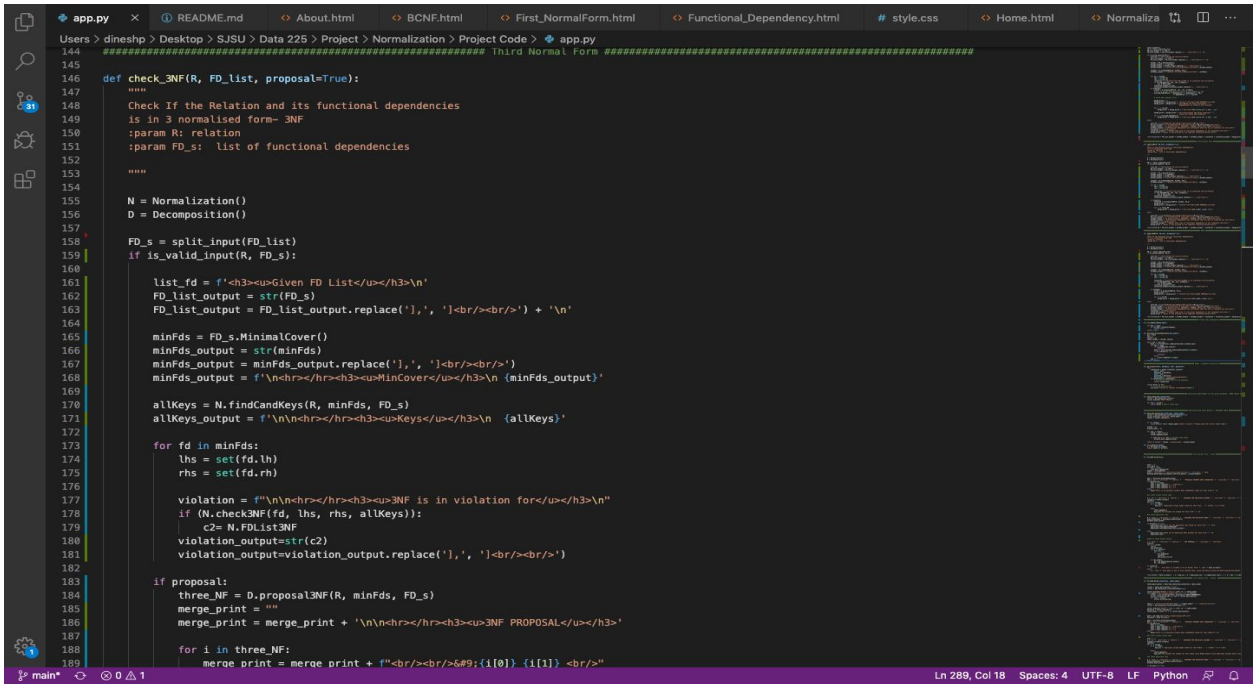
1NF (Tables)



2NF



3NF


app.py
README.md
About.html
BCNF.html
First_NormalForm.html
Functional_Dependency.html
style.css
Home.html
Users > dineshp > Desktop > SJSU > Data 225 > Project > Normalization > Project Code > app.py
144 ## Third Normal Form ##
145
146 def check_3NF(R, FD_list, proposal=True):
147 """
148 Check If the Relation and its functional dependencies
149 is in 3 normalised form- 3NF
150 :param R: relation
151 :param FD_s: list of functional dependencies
152
153 """
154
155 N = Normalization()
156 D = Decomposition()
157
158 FD_s = split_input(FD_list)
159 if is_valid_input(R, FD_s):
160
161 list_fd = f'<h3><u>Given FD List</u></h3>\n'
162 FD_list_output = str(FD_s)
163 FD_list_output = FD_list_output.replace('],', ']

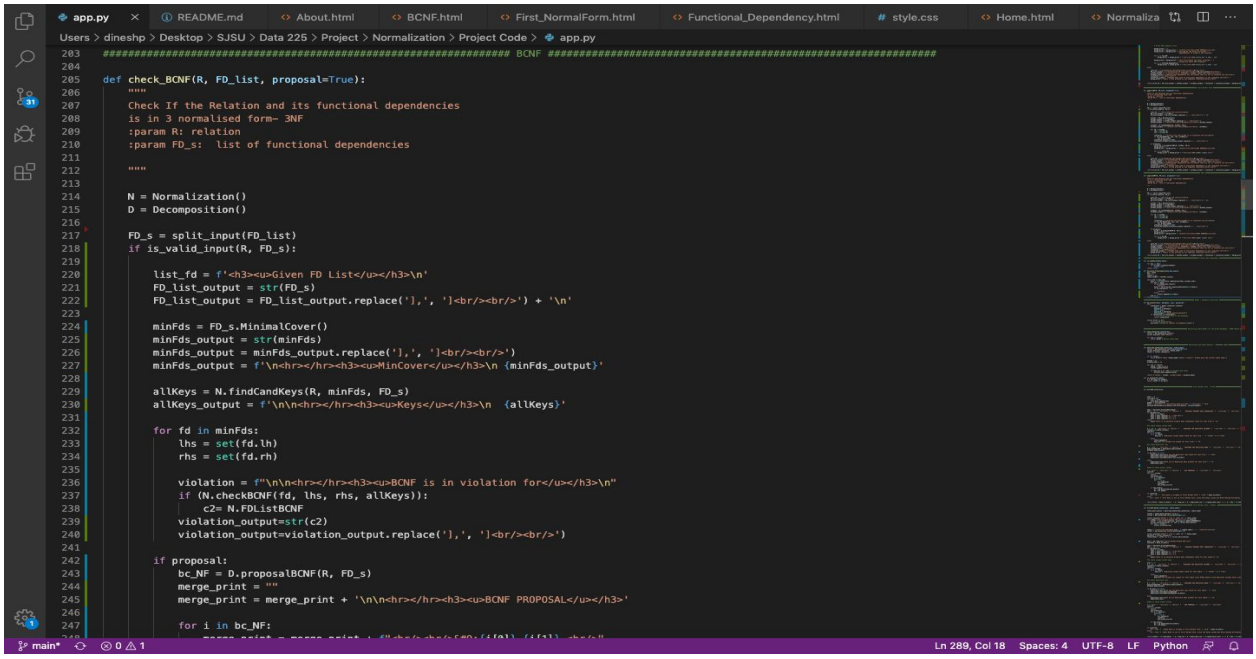
') + '\n'
164
165 minFds = FD_s.MinimalCover()
166 minFds_output = str(minFds)
167 minFds_output = minFds_output.replace('],', ']

')
168 minFds_output = f'\n<hr></hr><h3><u>MinCover</u></h3>\n {minFds_output}'
169
170 allKeys = N.findCandKeys(R, minFds, FD_s)
171 allKeys_output = f'\n\n<hr></hr><h3><u>Keys</u></h3>\n {allKeys}'
172
173 for fd in minFds:
174 lhs = set(fd.lh)
175 rhs = set(fd.rh)
176
177 violation = f"\n\n<hr></hr><h3><u>3NF is in violation for</u></h3>\n"
178 if (N.check3NF(fd, lhs, rhs, allKeys)):
179 c2= N.FDList3NF
180 violation_output=str(c2)
181 violation_output=violation_output.replace('],', ']

')
182
183 if proposal:
184 three_NF = D.proposal3NF(R, minFds, FD_s)
185 merge_print = ""
186 merge_print = merge_print + '\n\n<hr></hr><h3><u>3NF PROPOSAL</u></h3>'
187
188 for i in three_NF:
189 merge_print = merge_print + f"

	{i[0]} {i[1]}
"
main*
0 1
Ln 289, Col 18
Spaces: 4
UTF-8
LF
Python

BCNF


app.py
README.md
About.html
BCNF.html
First_NormalForm.html
Functional_Dependency.html
style.css
Home.html
Users > dineshp > Desktop > SJSU > Data 225 > Project > Normalization > Project Code > app.py
203 ## BCNF ##
204
205 def check_BCNF(R, FD_list, proposal=True):
206 """
207 Check If the Relation and its functional dependencies
208 is in 3 normalised form- 3NF
209 :param R: relation
210 :param FD_s: list of functional dependencies
211
212 """
213
214 N = Normalization()
215 D = Decomposition()
216
217 FD_s = split_input(FD_list)
218 if is_valid_input(R, FD_s):
219
220 list_fd = f'<h3><u>Given FD List</u></h3>\n'
221 FD_list_output = str(FD_s)
222 FD_list_output = FD_list_output.replace('],', ']

') + '\n'
223
224 minFds = FD_s.MinimalCover()
225 minFds_output = str(minFds)
226 minFds_output = minFds_output.replace('],', ']

')
227 minFds_output = f'\n<hr></hr><h3><u>MinCover</u></h3>\n {minFds_output}'
228
229 allKeys = N.findCandKeys(R, minFds, FD_s)
230 allKeys_output = f'\n\n<hr></hr><h3><u>Keys</u></h3>\n {allKeys}'
231
232 for fd in minFds:
233 lhs = set(fd.lh)
234 rhs = set(fd.rh)
235
236 violation = f"\n\n<hr></hr><h3><u>BCNF is in violation for</u></h3>\n"
237 if (N.checkBCNF(fd, lhs, rhs, allKeys)):
238 c2= N.FDListBCNF
239 violation_output=str(c2)
240 violation_output=violation_output.replace('],', ']

')
241
242 if proposal:
243 bc_NF = D.proposalBCNF(R, FD_s)
244 merge_print = ""
245 merge_print = merge_print + '\n\n<hr></hr><h3><u>BCNF PROPOSAL</u></h3>'
246
247 for i in bc_NF:
main*
0 1
Ln 289, Col 18
Spaces: 4
UTF-8
LF
Python

10.2 Input/output listing

Functional Dependency

Input : Table/File Data

Output : Set of FD's

1NF

Input : Table/File Data

Output : Checking if the table is in 1NF or not and giving warning to the user

2NF, 3NF & BCNF

Inputs

Set of Attributes {beer,brewery,strength,city,region,warehouse,quantity}

Functional Dependencies

beer->brewery

beer->strength

brewery->city

city->region

beer,warehouse->quantity

Outputs

2NF

Merge the below functional dependencies to create a new relation

{'brewery', 'beer'} : ['beer'] -> ['brewery']

{'strength', 'beer'} : ['beer'] -> ['strength']

Create the below relations along with above 2NF proposals

{'brewery', 'city'} : ['brewery'] -> ['city']

{'region', 'city'} : ['city'] -> ['region']

{'warehouse', 'quantity', 'beer'} : ['beer', 'warehouse'] -> ['quantity']

3NF

{'brewery', 'beer'} ['beer'] -> ['brewery']

{'strength', 'beer'} ['beer'] -> ['strength']

{'brewery', 'city'} ['brewery'] -> ['city']

{'region', 'city'} ['city'] -> ['region']

{'warehouse', 'quantity', 'beer'} ['warehouse', 'beer'] -> ['quantity']

BCNF

{'region', 'city'} ['city'] -> ['region']

{'brewery', 'city'} ['brewery'] -> ['city']

{'strength', 'brewery', 'beer'} ['beer'] -> ['brewery'], ['beer'] -> ['strength']

{'warehouse', 'quantity', 'beer'} ['warehouse', 'beer'] -> ['quantity']

10.3 Other related material

[1] https://www.geeksforgeeks.org/functional-dependency-and-attribute-closure/

[2] https://beginnersbook.com/2015/05/normalization-in-dbms/

[3] https://www.guru99.com/database-normalization.html

[4]https://stackoverflow.com/questions/50944844/the-fastest-way-to-find-primary-key-candidates-in-csv-file