**DBS Project Documentation**

**Step 1: Inputs**

We have created a python code that can normalize until the 3NF.

First we have taken inputs.

Read the table using pandas using:

table = pd.read\_csv(r"C:\Users\mprav\Downloads\exampleInputTable.csv")

then we took the functional and multi valued dependencies as input.

We have modified the functional dependencies based on the given assumptions.

So, please input the following Functional dependencies:

1. StudentID -> FirstName, LastName
2. Course, Professor -> classRoom
3. Course -> CourseStart, CourseEnd
4. Professor -> ProfessorEmail
5. StudentID, Course -> Professor

Now enter “exit” to continue with multi valued dependencies

please input the following multi valued dependencies:

1. Course ->> Professor
2. Course ->> classroom
3. StudentID ->> Course
4. StudentID ->> Professor

Now enter “exit”.

Enter the options between 1,2or 3 for the required normal form tables.

Then enter 1 to check the current table’s normal form level or else enter 2.

Enter the key: StudentID, Course

**Step 2: 1NF check**

Check for atomicity using following code

atomicity = True

for i in range(len(table)):

for column in table.columns:

if ', ' in str(table[column][i]):

atomicity = False

Then we need to check if more than 1 primary key exists for checking 2NF

if current == 1:

if atomicity:

print('\nThe given table satisfies 1NF')

else:

print('\nThe table fails 1NF')

if current == 1:

if len(key) == 1:

print('\nThe given table satisfies 2NF')

if len(key) > 1 and atomicity:

print('\nThe given table is in 1NF')

The code checks if its in 2NF or not. If not in 2NF, the following will be run

**Step 3: 2NF**

if step > 1:

if len(key) > 1:

num = 1

x = len(key)

tables = {}

for i in range(1 << x):

lst = [key[j] for j in range(x) if (i & (1 << j))]

if lst != []:

if len(lst) == 1:

unq\_vals = table[lst[0]].unique()

tables[num] = pd.DataFrame(unq\_vals, columns=lst)

else:

tables[num] = table[lst]

num += 1

cols = table.columns

remain = [i for i in cols if i not in key]

remain\_copy = remain

for val in range(1,num-1):

main\_col = list(tables[val].columns)

for col in [k for k, v in Functional\_dependencies.items() if v == main\_col]:

tables[val][col] = ''

remain.remove(col)

for i in range(len(tables[val])):

temp = table[table[main\_col[0]]==tables[val][main\_col[0]][i]][col]

temp = temp.reset\_index()

tables[val][col][i] = temp[col][0]

for col in remain:

main\_col = list(tables[num-1].columns)

tables[num-1][col] = ''

for i in range(len(tables[num-1])):

temp = table.loc[(table[main\_col[0]]==tables[num-1][main\_col[0]][i]) & (table[main\_col[1]]==tables[num-1][main\_col[1]][i])][col]

temp = temp.reset\_index()

tables[num-1][col][i] = temp[col][0]

The above code converts the given table into 2NF by creating the composite key into subset of primary keys.

**Step 4: 3NF**

if step > 2:

if len(key) == 2:

for col in remain\_copy:

try:

if Functional\_dependencies[col][0] in remain\_copy:

tables[3] = tables[3].drop([col], axis=1)

tables[num] = table[Functional\_dependencies[col]]

tables[num][col] = ''

remain\_copy.remove(col)

for i in range(len(tables[num])):

tables[num][col][i] = table[col][i]

tables[num] = tables[num].drop\_duplicates()

tables[num] = tables[num].reset\_index()

tables[num] = tables[num].drop(columns = ['index'])

num += 1

except:

x = 1

The above code converts the 2NF tables into 3NF tables by breaking the transitive functional dependencies.

**Step 5: BCNF**

if step > 3:

if len(key) == 2:

for col in remain\_copy:

try:

if Functional\_dependencies[col] != key:

tables[3] = tables[3].drop([col], axis=1)

tables[num] = table[Functional\_dependencies[col]]

tables[num][col] = ''

remain\_copy.remove(col)

for i in range(len(tables[num])):

tables[num][col][i] = table[col][i]

tables[num] = tables[num].drop\_duplicates()

tables[num] = tables[num].reset\_index()

tables[num] = tables[num].drop(columns = ['index'])

num += 1

except:

x = 1

The above code converts the 3NF tables into BCNF tables by breaking the functional dependencies further.

**Step 6: 4NF**

if step > 4:

for i in range(1,len(tables)+1):

tables[i].drop\_duplicates()

tables[i] = tables[i].reset\_index()

tables[i] = tables[i].drop(columns = ['index'])

print('\n\tTable ', i, ' is:')

print(tables[i])

The above code checks for existence of redundant multiple values using the current tables for multi valued dependencies

**Step 7: output**

Printing the query for creating tables based on the current set of tables:

if step == 1 and atomicity:

print('CREATE TABLE table1 (')

for col in table.columns:

print(col, end='')

try:

if table[col][0].is\_integer():

print(' INT', end ='')

except:

x = 1

if isinstance(table[col][0], str):

if '/' in table[col][0]:

print(' DATE', end ='')

else:

print(' VARCHAR(255)', end ='')

if col == tables[i].columns[0] and table[col].is\_unique:

print(' PRIMAR KEY', end ='')

print(',')

print(');')

if step > 1:

print('\nThe queries to create normalized tables are: ')

for i in range(1,len(tables)+1):

print('CREATE TABLE table',i, ' (')

for col in tables[i].columns:

print(col, end='')

try:

if tables[i][col][0].is\_integer():

print(' INT', end ='')

except:

x = 1

if isinstance(tables[i][col][0], str):

if '/' in tables[i][col][0]:

print(' DATE', end ='')

else:

print(' VARCHAR(255)', end ='')

if col == tables[i].columns[0] and tables[i][col].is\_unique:

print(' PRIMAR KEY', end ='')

print(',')

print(');')

This code creates SQL query for creating those tables, which is our output.