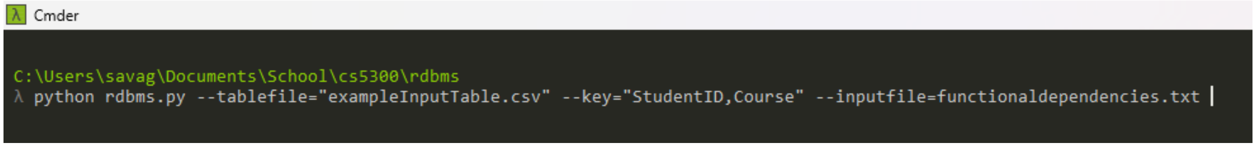
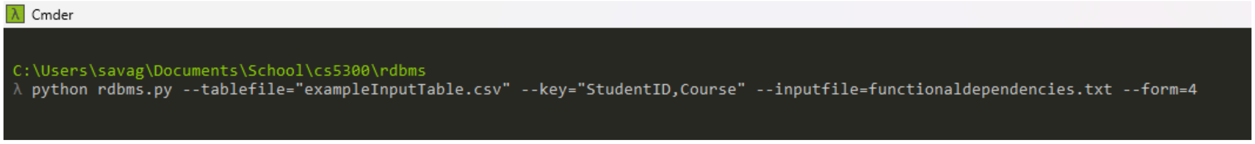
Tanner Armstrong
RDBMS Programming Project

The programming language chosen for the project is Python. The input table and corresponding functional dependencies are stored in a .csv and .txt file respectively. The names of the necessary files, and primary keys are inputted in the terminal when the program is run. The Python library Pandas is required to be installed for the program to work.



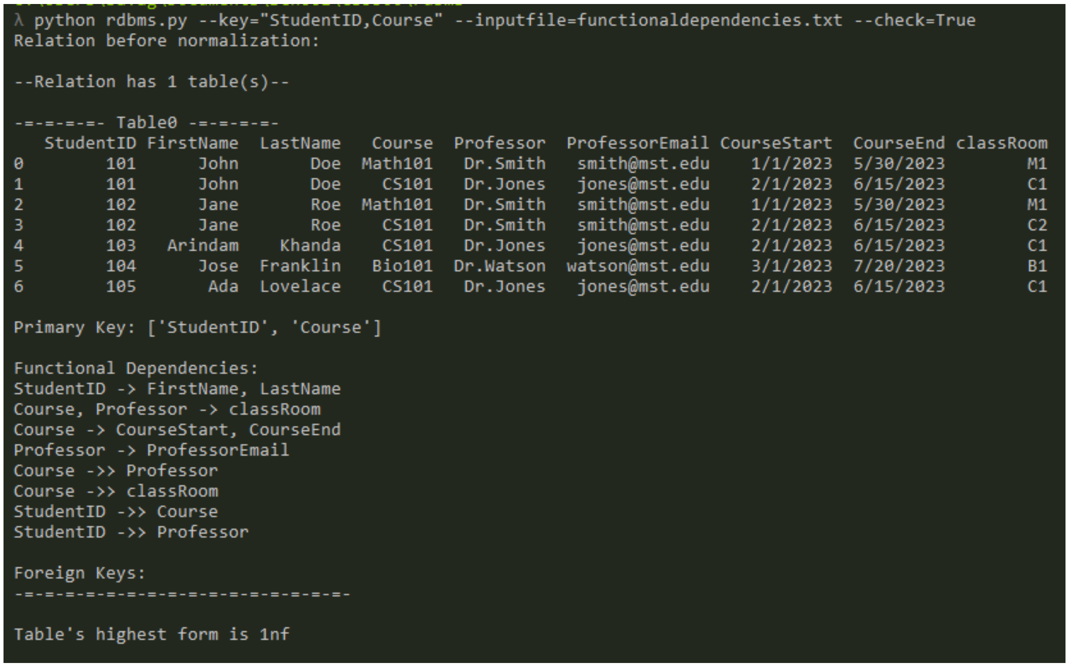
```
C:\Users\savag\Documents\School\cs5300\rdbms
λ python rdbms.py --tablefile="exampleInputTable.csv" --key="StudentID,Course" --inputfile=functionaldependencies.txt
```

Additionally, the user can input what highest normal form they would like to achieve using the --form=FORM input option. These need to be integer inputs, except for BCNF in which case the user will need to input --form=B. If this input is left out, the program will automatically normalize through 4NF.



```
C:\Users\savag\Documents\School\cs5300\rdbms
λ python rdbms.py --tablefile="exampleInputTable.csv" --key="StudentID,Course" --inputfile=functionaldependencies.txt --form=4
```

The user can also run the program with the --check=True argument. This will instead of running the standard normalization routine, simply check the highest form of the inputted table as it was inputted and output that information to the user.

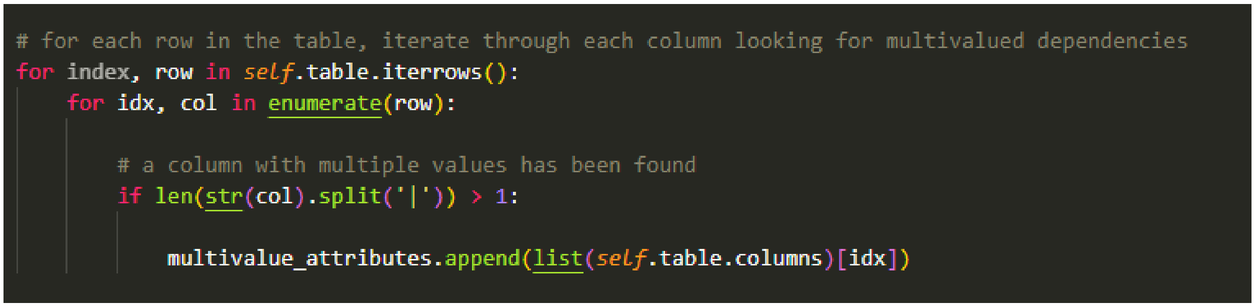


```
λ python rdbms.py --key="StudentID,Course" --inputfile=functionaldependencies.txt --check=True
Relation before normalization:

--Relation has 1 table(s)--

-=-=-=-=- Table0 -=-=-=-=-
   StudentID FirstName  LastName   Course  Professor  ProfessorEmail CourseStart  CourseEnd classRoom
0        101      John       Doe  Math101   Dr.Smith   smith@mst.edu    1/1/2023  5/30/2023        M1
1        101      John       Doe    CS101   Dr.Jones   jones@mst.edu    2/1/2023  6/15/2023        C1
2        102      Jane       Roe  Math101   Dr.Smith   smith@mst.edu    1/1/2023  5/30/2023        M1
3        102      Jane       Roe    CS101   Dr.Smith   smith@mst.edu    2/1/2023  6/15/2023        C2
4        103   Arindam    Khanda    CS101   Dr.Jones   jones@mst.edu    2/1/2023  6/15/2023        C1
5        104      Jose  Franklin   Bio101  Dr.Watson  watson@mst.edu    3/1/2023  7/20/2023        B1
6        105       Ada  Lovelace    CS101   Dr.Jones   jones@mst.edu    2/1/2023  6/15/2023        C1

Primary Key: ['StudentID', 'Course']

Functional Dependencies:
StudentID -> FirstName, LastName
Course, Professor -> classRoom
Course -> CourseStart, CourseEnd
Professor -> ProfessorEmail
Course ->> Professor
Course ->> classRoom
StudentID ->> Course
StudentID ->> Professor

Foreign Keys:
-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-

Table's highest form is 1nf
```

The main structure of the code uses two classes: RelationTable and Relation. RelationTable objects store the table itself as a pandas dataframe, primary key, foreign keys, and functional dependencies. The Relation objects simply store a list of all the RelationTable objects in the relation. The functions for determining if a table passes the criteria for a certain normal form are functions of the RelationTable class, and the functions for normalizing tables are of the Relation class.

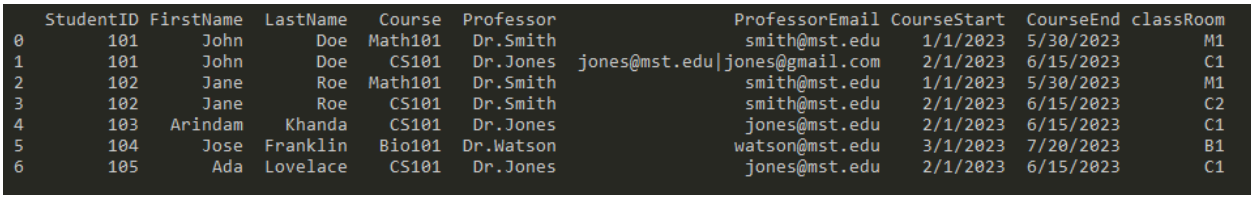
1st Normal Form

For this program, columns with multivalued attributes don't need to be specified by the user. The multivalued attributes just have to be delineated using a "|" character and the program will automatically detect this and normalize it.



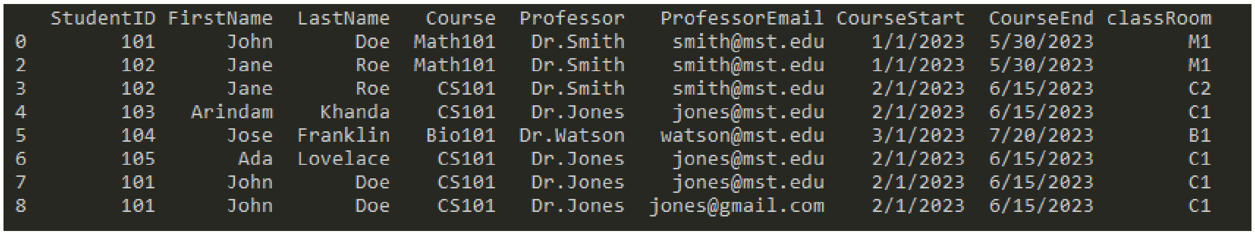
```
# for each row in the table, iterate through each column looking for multivalued dependencies
for index, row in self.table.iterrows():
    for idx, col in enumerate(row):

        # a column with multiple values has been found
        if len(str(col).split('|')) > 1:

            multivalue_attributes.append(list(self.table.columns)[idx])
```

In the given example there are no multivalued attributes, but if I added another ProfessorEmail, "jones@gmail.com" to one of the rows in the column like this:



```
  StudentID FirstName  LastName   Course  Professor                   ProfessorEmail CourseStart  CourseEnd classRoom
0       101      John       Doe  Math101   Dr.Smith                    smith@mst.edu    1/1/2023  5/30/2023        M1
1       101      John       Doe    CS101   Dr.Jones  jones@mst.edu|jones@gmail.com    2/1/2023  6/15/2023        C1
2       102      Jane       Roe  Math101   Dr.Smith                    smith@mst.edu    1/1/2023  5/30/2023        M1
3       102      Jane       Roe    CS101   Dr.Smith                    smith@mst.edu    2/1/2023  6/15/2023        C2
4       103   Arindam    Khanda    CS101   Dr.Jones                    jones@mst.edu    2/1/2023  6/15/2023        C1
5       104      Jose  Franklin   Bio101  Dr.Watson                   watson@mst.edu    3/1/2023  7/20/2023        B1
6       105       Ada  Lovelace    CS101   Dr.Jones                    jones@mst.edu    2/1/2023  6/15/2023        C1
```

The program will simply break the multivalued attribute across multiple rows. This works for any length of a multivalued attribute:

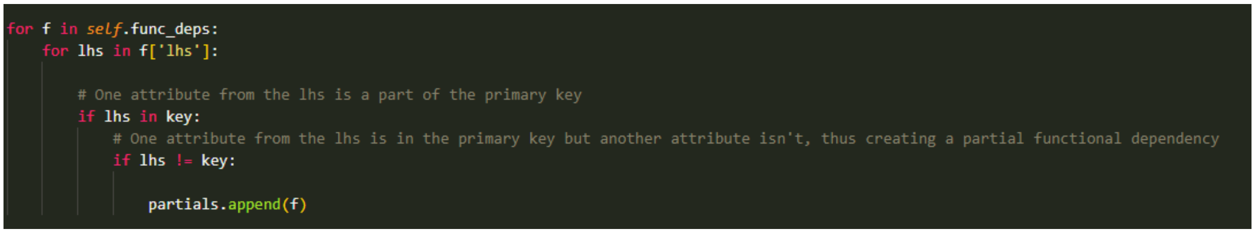


```
  StudentID FirstName  LastName   Course  Professor   ProfessorEmail CourseStart  CourseEnd classRoom
0       101      John       Doe  Math101   Dr.Smith    smith@mst.edu    1/1/2023  5/30/2023        M1
2       102      Jane       Roe  Math101   Dr.Smith    smith@mst.edu    1/1/2023  5/30/2023        M1
3       102      Jane       Roe    CS101   Dr.Smith    smith@mst.edu    2/1/2023  6/15/2023        C2
4       103   Arindam    Khanda    CS101   Dr.Jones    jones@mst.edu    2/1/2023  6/15/2023        C1
5       104      Jose  Franklin   Bio101  Dr.Watson   watson@mst.edu    3/1/2023  7/20/2023        B1
6       105       Ada  Lovelace    CS101   Dr.Jones    jones@mst.edu    2/1/2023  6/15/2023        C1
7       101      John       Doe    CS101   Dr.Jones    jones@mst.edu    2/1/2023  6/15/2023        C1
8       101      John       Doe    CS101   Dr.Jones  jones@gmail.com    2/1/2023  6/15/2023        C1
```

2nd Normal Form

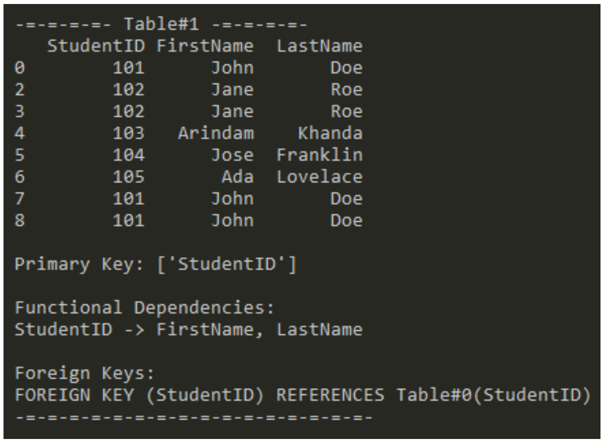
After the 1NF normalization, the table has the following partial functional dependencies:
StudentID -> FirstName, LastName,
Course, Professor -> classRoom,
Course -> CourseStart, CourseEnd

To find these dependencies the program iterates through all functional dependencies in a given table, and then if any attribute on the left hand side of the dependency is in the primary key and there's any other attributes in the left hand side not part of the key, the program determines this is a partial functional dependency.

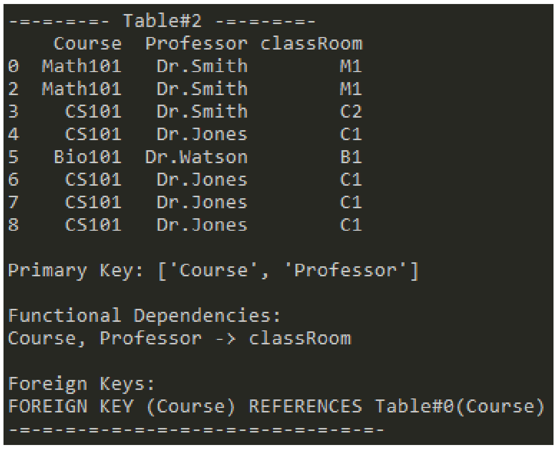


```
for f in self.func_deps:
    for lhs in f['lhs']:

        # One attribute from the lhs is a part of the primary key
        if lhs in key:
            # One attribute from the lhs is in the primary key but another attribute isn't, thus creating a partial functional dependency
            if lhs != key:

                partials.append(f)
```

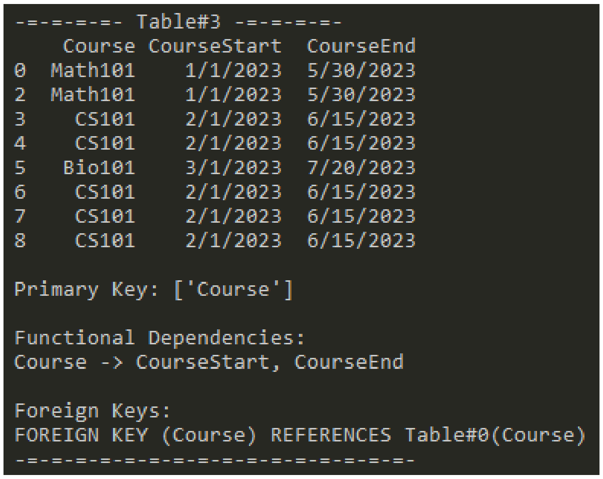
Normalizing out these dependencies yields the following four tables:



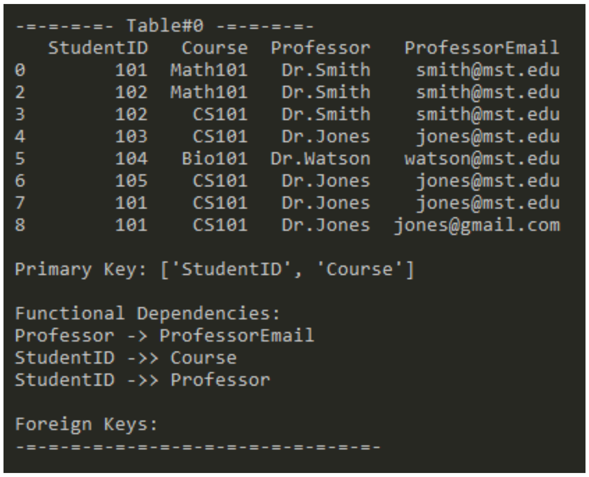
```
-=-=-=-=- Table#1 -=-=-=-=-
   StudentID FirstName  LastName
0        101      John       Doe
2        102      Jane       Roe
3        102      Jane       Roe
4        103   Arindam    Khanda
5        104      Jose  Franklin
6        105       Ada  Lovelace
7        101      John       Doe
8        101      John       Doe

Primary Key: ['StudentID']

Functional Dependencies:
StudentID -> FirstName, LastName

Foreign Keys:
FOREIGN KEY (StudentID) REFERENCES Table#0(StudentID)
-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
```



```
-=-=-=-=- Table#2 -=-=-=-=-
    Course  Professor classRoom
0  Math101   Dr.Smith        M1
2  Math101   Dr.Smith        M1
3    CS101   Dr.Smith        C2
4    CS101   Dr.Jones        C1
5   Bio101  Dr.Watson        B1
6    CS101   Dr.Jones        C1
7    CS101   Dr.Jones        C1
8    CS101   Dr.Jones        C1

Primary Key: ['Course', 'Professor']

Functional Dependencies:
Course, Professor -> classRoom

Foreign Keys:
FOREIGN KEY (Course) REFERENCES Table#0(Course)
-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
```



```
-=-=-=-=- Table#3 -=-=-=-=-
    Course CourseStart  CourseEnd
0  Math101    1/1/2023  5/30/2023
2  Math101    1/1/2023  5/30/2023
3    CS101    2/1/2023  6/15/2023
4    CS101    2/1/2023  6/15/2023
5   Bio101    3/1/2023  7/20/2023
6    CS101    2/1/2023  6/15/2023
7    CS101    2/1/2023  6/15/2023
8    CS101    2/1/2023  6/15/2023

Primary Key: ['Course']

Functional Dependencies:
Course -> CourseStart, CourseEnd

Foreign Keys:
FOREIGN KEY (Course) REFERENCES Table#0(Course)
-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
```

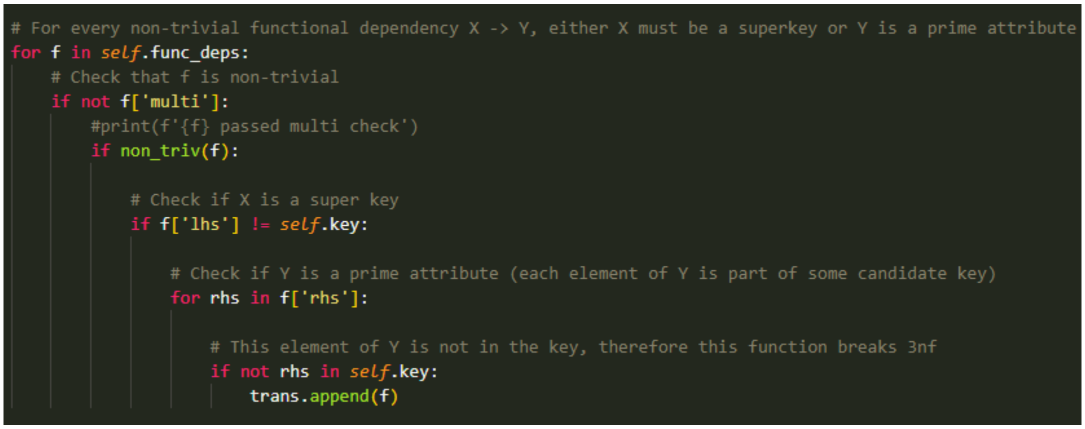


```
-=-=-=-=- Table#0 -=-=-=-=-
   StudentID   Course  Professor    ProfessorEmail
0        101  Math101   Dr.Smith    smith@mst.edu
2        102  Math101   Dr.Smith    smith@mst.edu
3        102    CS101   Dr.Smith    smith@mst.edu
4        103    CS101   Dr.Jones    jones@mst.edu
5        104   Bio101  Dr.Watson   watson@mst.edu
6        105    CS101   Dr.Jones    jones@mst.edu
7        101    CS101   Dr.Jones    jones@mst.edu
8        101    CS101   Dr.Jones  jones@gmail.com

Primary Key: ['StudentID', 'Course']

Functional Dependencies:
Professor -> ProfessorEmail
StudentID ->> Course
StudentID ->> Professor

Foreign Keys:
-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
```

All of the partial functional dependencies have been normalized into their own tables and have the appropriate foreign keys referencing back to the original table which is Table#0. Performing this normalization step also removes the multivalued functional dependency Course ->> Professor, classRoom from Table#0 since the classRoom column has been moved to a different table.

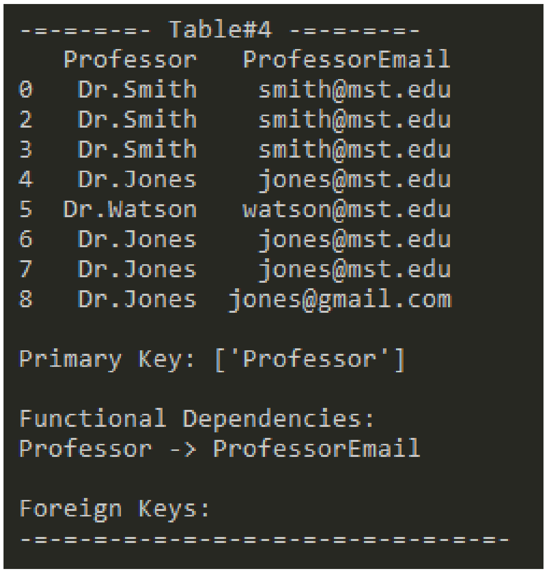
3rd Normal Form:

To find transitive dependencies, the program iterates through all functional dependencies, and if a functional dependency is determined to be non trivial, and the left hand side of the dependency is not equal to the primary key, and the right hand side is not a subset of the key, this functional dependency is determined to be transitive.

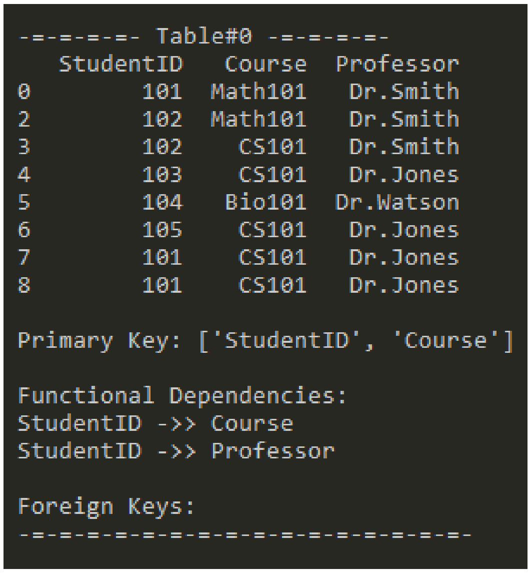


```
# For every non-trivial functional dependency X -> Y, either X must be a superkey or Y is a prime attribute
for f in self.func_deps:
    # Check that f is non-trivial
    if not f['multi']:
        #print(f'{f} passed multi check')
        if non_triv(f):

            # Check if X is a super key
            if f['lhs'] != self.key:

                # Check if Y is a prime attribute (each element of Y is part of some candidate key)
                for rhs in f['rhs']:

                    # This element of Y is not in the key, therefore this function breaks 3nf
                    if not rhs in self.key:
                        trans.append(f)
```

In the table's current format after the 2nd normal form normalization process, there is only a single transitive dependency that requires normalization - Professor -> ProfessorEmail in Table#0. This normalization breaks Table#0 into the following two tables:



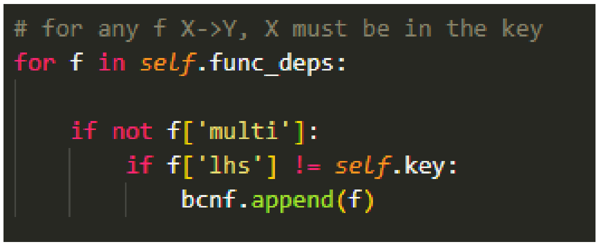
```
-=-=-=-=- Table#4 -=-=-=-=-
   Professor   ProfessorEmail
0   Dr.Smith    smith@mst.edu
2   Dr.Smith    smith@mst.edu
3   Dr.Smith    smith@mst.edu
4   Dr.Jones    jones@mst.edu
5  Dr.Watson   watson@mst.edu
6   Dr.Jones    jones@mst.edu
7   Dr.Jones    jones@mst.edu
8   Dr.Jones  jones@gmail.com

Primary Key: ['Professor']

Functional Dependencies:
Professor -> ProfessorEmail

Foreign Keys:
-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
```



```
-=-=-=-=- Table#0 -=-=-=-=-
   StudentID   Course  Professor
0        101  Math101   Dr.Smith
2        102  Math101   Dr.Smith
3        102    CS101   Dr.Smith
4        103    CS101   Dr.Jones
5        104   Bio101  Dr.Watson
6        105    CS101   Dr.Jones
7        101    CS101   Dr.Jones
8        101    CS101   Dr.Jones

Primary Key: ['StudentID', 'Course']

Functional Dependencies:
StudentID ->> Course
StudentID ->> Professor

Foreign Keys:
-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
```

Boyce-Codd Normal Form:

To determine if any functional dependency in a table does not conform to BCNF criteria, the program simply checks if the left hand side of the dependency is the primary key.



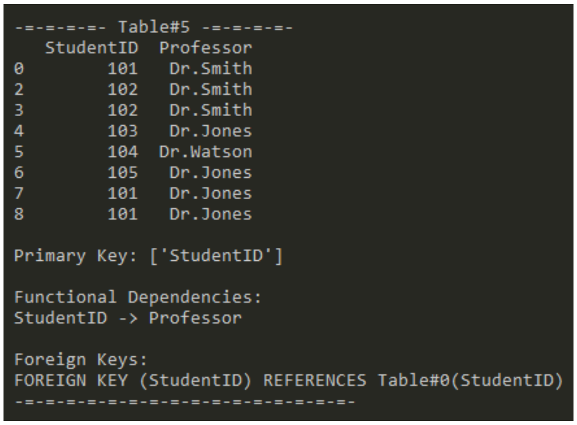
```python
# for any f X->Y, X must be in the key
for f in self.func_deps:

    if not f['multi']:
        if f['lhs'] != self.key:
            bcnf.append(f)
```

With the normalization steps already taken, all the tables are already in BCNF.

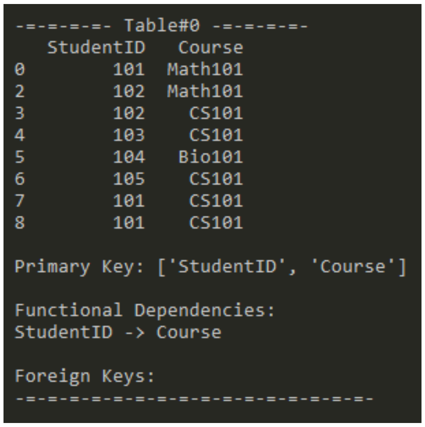
4th Normal Form:

When the functional dependencies are parsed from the user input, each one is assigned a boolean value for if it's a multivalued dependency or not based on if it contains a '->>' or a '->'. When performing 4NF normalization, the program simply checks this boolean value for each functional dependency.

The only remaining multivalued dependency left in the relation is StudentID ->> Course, Professor in Table#0. Normalizing this MVD breaks Table#0 into the following tables:



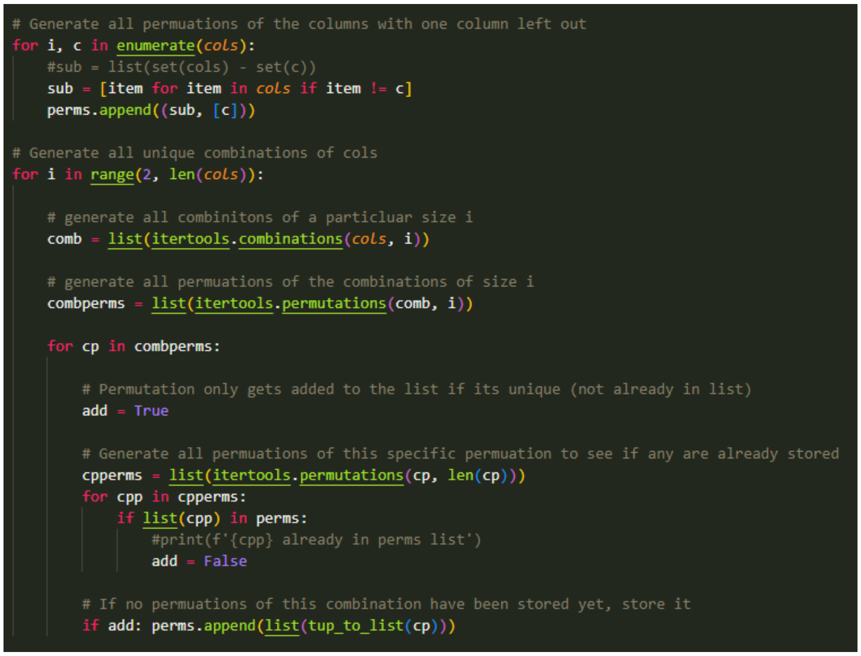
```
-=-=-=-=- Table#5 -=-=-=-=-
   StudentID  Professor
0        101   Dr.Smith
2        102   Dr.Smith
3        102   Dr.Smith
4        103   Dr.Jones
5        104  Dr.Watson
6        105   Dr.Jones
7        101   Dr.Jones
8        101   Dr.Jones

Primary Key: ['StudentID']

Functional Dependencies:
StudentID -> Professor

Foreign Keys:
FOREIGN KEY (StudentID) REFERENCES Table#0(StudentID)
-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
```



```
-=-=-=-=- Table#0 -=-=-=-=-
   StudentID   Course
0        101  Math101
2        102  Math101
3        102    CS101
4        103    CS101
5        104   Bio101
6        105    CS101
7        101    CS101
8        101    CS101

Primary Key: ['StudentID', 'Course']

Functional Dependencies:
StudentID -> Course

Foreign Keys:
-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
```

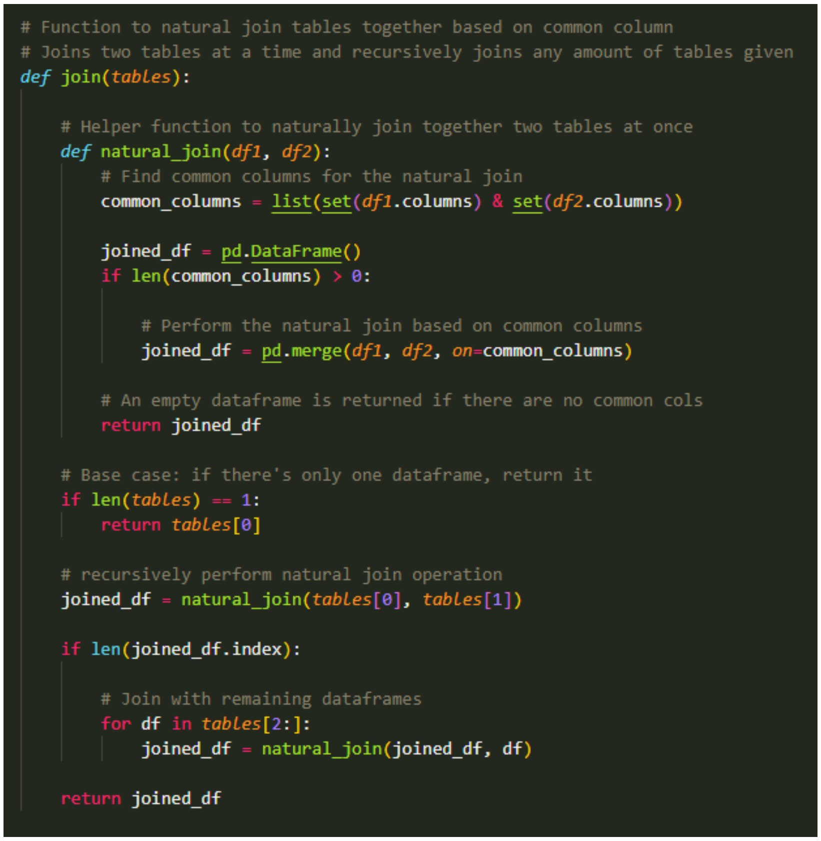
5th Normal Form:

To determine if a table is in 5nf, first the program generates all possible projections of the table columns.



```python
# Generate all permuations of the columns with one column left out
for i, c in enumerate(cols):
    #sub = list(set(cols) - set(c))
    sub = [item for item in cols if item != c]
    perms.append((sub, [c]))

# Generate all unique combinations of cols
for i in range(2, len(cols)):

    # generate all combinitons of a particluar size i
    comb = list(itertools.combinations(cols, i))

    # generate all permuations of the combinations of size i
    combperms = list(itertools.permutations(comb, i))

    for cp in combperms:

        # Permutation only gets added to the list if its unique (not already in list)
        add = True

        # Generate all permuations of this specific permuation to see if any are already stored
        cpperms = list(itertools.permutations(cp, len(cp)))
        for cpp in cpperms:
            if list(cpp) in perms:
                #print(f'{cpp} already in perms list')
                add = False

        # If no permuations of this combination have been stored yet, store it
        if add: perms.append(list(tup_to_list(cp)))
```

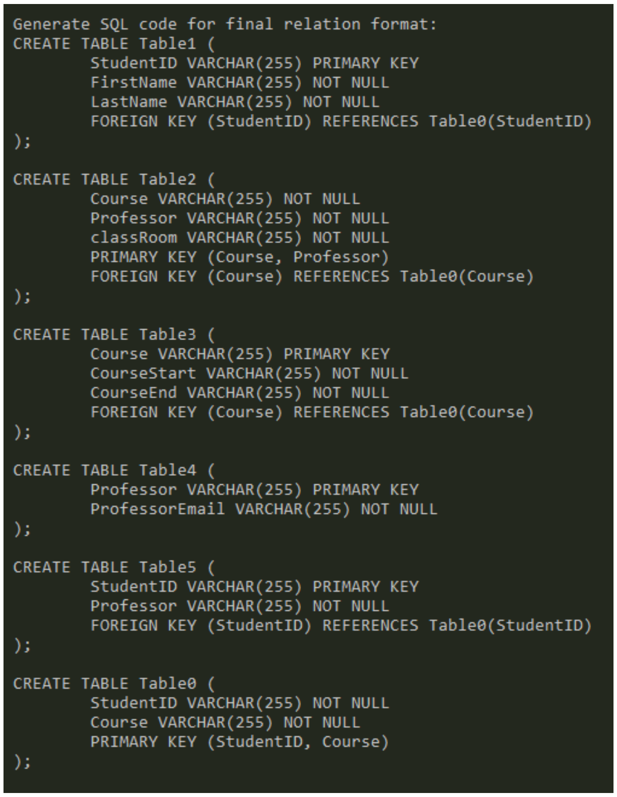
Then the program iterates through all generated projections, natural joins them together, and compares them to the original table for equality.



```python
# Function to natural join tables together based on common column
# Joins two tables at a time and recursively joins any amount of tables given
def join(tables):

    # Helper function to naturally join together two tables at once
    def natural_join(df1, df2):
        # Find common columns for the natural join
        common_columns = list(set(df1.columns) & set(df2.columns))

        joined_df = pd.DataFrame()
        if len(common_columns) > 0:

            # Perform the natural join based on common columns
            joined_df = pd.merge(df1, df2, on=common_columns)

        # An empty dataframe is returned if there are no common cols
        return joined_df

    # Base case: if there's only one dataframe, return it
    if len(tables) == 1:
        return tables[0]

    # recursively perform natural join operation
    joined_df = natural_join(tables[0], tables[1])

    if len(joined_df.index):

        # Join with remaining dataframes
        for df in tables[2:]:
            joined_df = natural_join(joined_df, df)

    return joined_df
```

This relation does not have any valid join dependencies.

SQL Output:
The program will output the SQL statements required to build all of the tables in the relation list after the normalization routine is finished. These statements are both outputted to the terminal and to a file named sqloutfile.txt.



```
Generate SQL code for final relation format:
CREATE TABLE Table1 (
        StudentID VARCHAR(255) PRIMARY KEY
        FirstName VARCHAR(255) NOT NULL
        LastName VARCHAR(255) NOT NULL
        FOREIGN KEY (StudentID) REFERENCES Table0(StudentID)
);

CREATE TABLE Table2 (
        Course VARCHAR(255) NOT NULL
        Professor VARCHAR(255) NOT NULL
        classRoom VARCHAR(255) NOT NULL
        PRIMARY KEY (Course, Professor)
        FOREIGN KEY (Course) REFERENCES Table0(Course)
);

CREATE TABLE Table3 (
        Course VARCHAR(255) PRIMARY KEY
        CourseStart VARCHAR(255) NOT NULL
        CourseEnd VARCHAR(255) NOT NULL
        FOREIGN KEY (Course) REFERENCES Table0(Course)
);

CREATE TABLE Table4 (
        Professor VARCHAR(255) PRIMARY KEY
        ProfessorEmail VARCHAR(255) NOT NULL
);

CREATE TABLE Table5 (
        StudentID VARCHAR(255) PRIMARY KEY
        Professor VARCHAR(255) NOT NULL
        FOREIGN KEY (StudentID) REFERENCES Table0(StudentID)
);

CREATE TABLE Table0 (
        StudentID VARCHAR(255) NOT NULL
        Course VARCHAR(255) NOT NULL
        PRIMARY KEY (StudentID, Course)
);
```

```
CREATE TABLE Table1 (
   StudentID VARCHAR(255) PRIMARY KEY,
   FirstName VARCHAR(255) NOT NULL,
   LastName VARCHAR(255) NOT NULL,
   FOREIGN KEY (StudentID) REFERENCES Table0(StudentID)
);

CREATE TABLE Table2 (
   Course VARCHAR(255) NOT NULL,
   Professor VARCHAR(255) NOT NULL,
   classRoom VARCHAR(255) NOT NULL,
   PRIMARY KEY (Course, Professor),
   FOREIGN KEY (Course) REFERENCES Table0(Course)
);

CREATE TABLE Table3 (
   Course VARCHAR(255) PRIMARY KEY,
   CourseStart VARCHAR(255) NOT NULL,
   CourseEnd VARCHAR(255) NOT NULL,
   FOREIGN KEY (Course) REFERENCES Table0(Course)
);

CREATE TABLE Table4 (
   Professor VARCHAR(255) PRIMARY KEY,
   ProfessorEmail VARCHAR(255) NOT NULL
);

CREATE TABLE Table5 (
            StudentID VARCHAR(255) PRIMARY KEY,
            Professor VARCHAR(255) NOT NULL,
            FOREIGN KEY (StudentID) REFERENCES Table0(StudentID)
);

CREATE TABLE Table0 (
            StudentID VARCHAR(255) NOT NULL,
            Course VARCHAR(255) NOT NULL,
            PRIMARY KEY (StudentID, Course)
);
```