Link to maintained File from OpenRefine for potential processing: https://drive.google.com/drive/folders/11bslcR4DTo47MZ6yEQIavp-9r0r7phj8?usp=sharing
Link to ID_Check output CSV File: https://drive.google.com/drive/folders/1y1hxgQkUXsGG13frBIUgWdO8fN7bOPrF?usp=sharing
Link to FD and other IC check Output CSV File: https://drive.google.com/drive/folders/1_2T0WrhTNVMVFaT_c0R1h5WJ1qPRCeRI?usp=sharing
Link to FINAL New_Data_Generation: https://drive.google.com/drive/folders/1RttzDcAqZ7Wj6cpTaRYlv3USO0x_CLB8?usp=sharing
The UIUC Box folder link is included to the endnote.

# What Shall Be on The Menu? - Data Cleaning Project on CS513

Hao Tang

*Department of Computer Science*
*University of Illinois at Urbana-Champaign*
*Champaign, IL*

*haot3@illinois.edu*

***Abstract* - From data quality assessment to loading cleaned data to relational database for application, the goal for this project is to simulate the real data cleaning process and apply the theories and methods from course CS513. The author worked on an open source dataset from New York Public Library (NYPL) and processed cleaning on missing data, informal input or inconsistent format. The post-processed data are used to input relational database for integrity constraint (IC) check. The data cleaning roadmap and data provenance will be documented as scientific workflow.**

## I. Introduction

This project aims to generate cleaned and ready-to-use data from the raw dataset to future application. The author would like to attain data file with plain format that can be applied in menu design and restaurant business assistance. These files shall be verified by lightweight and popular database SQLite so they can be used in app development or software design. For another, even though it's not the immediate goal, the data cleaning process will also maintain another set of data file that possess the post-cleaned data for more option in future data analysis.

## II. Initial Assessment

### *A. The Dataset*

The dataset in this project is a recent version of NYPL's restaurant menu collection[1] belonging to NYPL's Rare Book Division. The dataset itself has been thriving with on-going update since 1900. As the first contributor, MS. Frank E. Buttolph firstly transcribed more than 25,000 menus to the collections from 1850 to 1921[2]. After 110-year growth, the latest version has included approximately 45,000 menus in the collection.

Since 2011, the menu data transcribed and curated by volunteers as part of the *NYPL Digital Gallery* [3]. The dataset is open to download in nypl.org[4]. Researchers or menus users can conveniently access to the collection to search the answers to their questions. Simultaneously, its corresponding RESTful API is also developed and available on GitHub as advanced tool to explore the dataset[5]. And some interesting projects are triggered on top of the historical menu collection afterward[6].

### *B. Dataset Structure and Data Quality*

In this project, the data is downloaded directly from nypl.org as four csv files: "Dish.csv", "Menu.csv", "MenuItem.csv" and "MenuPage.csv". Most of these files come with less than 10 columns, linking by specific column that serves as foreign key in the child table or csv file. Figure 1. shows the structure to whole dataset with separated tables. The author would like to briefly introduce the four files below before we continue to dive into our data cleaning journey.

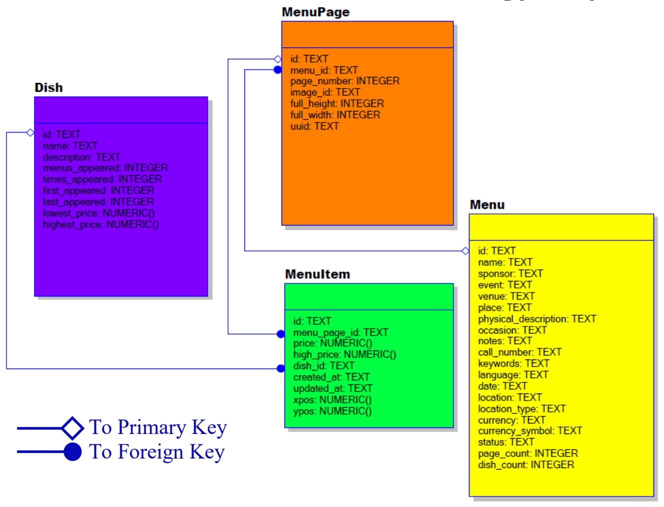


Fig. 1 ER Diagram to "What's on the Menu?" Dataset.

### Dish

As the most popular section at first glance, Dish may be the first part that people will turn to when they're interested in the collection's content. In the version applied in this project, the file contains more than 423,000 tuples with their most interesting information such as dishes' name, number of appearance and price range. Here are the attributes in this table:

1) *id – Dish id*
2) *name – Dish name*
3) *description – No content*
4) *menus_appeared – Dish occurrence in menus*
5) *times_appeared – Overall dish occurrence*
6) *first_appeared – Dish first time occurrence*

7) *last_appeared – Dish last time occurrence*
8) *lowest_price – Dish lowest price*
9) *highest_price – Dish highest price*

Immediately, we want to verify if the *first_appeared* is valid or reasonably earlier than the *last_appeared*, and if the *lowest_price* is lower than the *highest_price*. Also, it seems to be exciting to have more than four hundred thousand dishes' name and description in the past 170 years if people want to apply their data analysis technique. However, let's hold the horse in this section before we examine the table detail.

**MenuItem**

The MenuItem.csv is largest in the four files with 1.3 million tuples. Each tuple represents one instance of the dishes captured from one menu. And it has the following attributes:

1) *id – Item id*
2) *menu_page_id – Menu page id that has this item*
3) *price – Item price*
4) *high_price – Item high price comparing to normal price*
5) *dish_id – Dish id in the dish table.*
6) *created_at – Item creation time*
7) *updated_at – Item update time*
8) *xpos – x position of the transcription item*
9) *ypos - y position of the transcription item*

There is identification column in this table as well. The *menu_page_id* serves as foreign key if these tables are loaded into database. We also need to verify the number in *price* columns and timing order in *created_at*/*update_at* columns.

**MenuPage**

This file provides dimension information and assigned id to each individual page in all the menus. When these pages are digitized, their image id and unique identifier are also captured as attributes in this file. The *menu_id* is the foreign id in this table referring to the *id* column in the menu table in our relational database in this project. Here are attributes in this table:

1) *id – Menu page id*
2) *menu_id – Menu id in the menu table*
3) *page_number – The page number in the carrying menu*
4) *image_id – Image id to the page when it's digitalized.*
5) *full_height – height of the page*
6) *full_width – width of the page*
7) *uuid – Unique id to the digitalized image*

**Menu**

Menu table emphasizes on the factors other than the food itself. It introduces the event, sponsor, venue and place the menus are associated with, the occasion of the meals, date of the menu's call, currency on the menu item. This is the most informative file, meanwhile we have more cleaning work to this table. We want to check the inclusion dependency, the attribute format, some columns' integrity and remove the column with no content. Here are its 20 attributes:

1) *id – Menu id*
2) *name – Name to the menu application organization*
3) *sponsor – Sponsor organization*
4) *event – The event (dinner, lunch, breakfast, etc.) with this menu*
5) *venue – venue to the menu application*
6) *place – Physical address to the menu application*
7) *physical_description – The menu material or spec*
8) *occasion – The occasion to the menu application*
9) *notes – Notes to menu application*
10) *call_number – Call number associated the menu*
11) *keywords – No content*
12) *language – No content*
13) *date – Date to the menu application*
14) *location – Location to the menu application*
15) *location_type - No content*
16) *currency – Currency that's applied in the menu*
17) *currency_symbol – Currency symbol*
18) *status – Complete or under review*
19) *page_count – Page quantity to the menu*
20) *dish_count – Dish quantity to the menu*

Some of attributes in these files are very sparse, such as *high_price* in MenuItem and *occasion* in Menu. Some attributes don't have any content, such as *keywords* and *language* in Menu. Nevertheless, more than 1.3 million tuples in MenuItem and more than 420,000 tuples in Dish require efficient method in the data cleaning process. The volatile content and inconsistent input format make data wrangling more challenging.

### C. *Use Case*

Some direct application can work before data cleaning:

1) **Search in the collection** It's straightforward to search the historical record to some popular dishes if they have specific keyword, such as Blanquette de Veau or Kung Pao Chi Ding.
2) **Investigation to Menu application in last few decades** Even though there're flaws on some attributes, we can still achieve most frequent menu applying event, venue.
3) **Estimation to the collection history** Most of the date attributes in the tables can tell the input action corresponding to certain menu and dish instance.

However, since we are not interested in simple search and detail to the data collection itself, we need solution to data cleaning and IC check in future application. Because the dataset is generated from various menu with different verbal style, and the volunteer is expected to transcribe the menu with minimal change, the same item of wording can have various literal forms.

E.g. there exist DINNER, Dinner, [DINNER], ABENDESSEN and SUPPER in the *event* attribute of the Menu file. Therefore, it's not efficient to summarize the similar item with descriptive information.

According to the data review above, the most applicable target use case is to provide menu evolution and characteristics information as business assistance after cleaning the data. The output data files should be ready-to-use for any software or app design corresponding to business application.

If we can trace back to the price change to the menu items in the history, then tracking the price trend is likely another use case as helpful information to the business application.

There exist Latin letter in Dish name as New York City has numerous French or Spanish restaurant. We are also wondering if the dishes will be more expensive if their names have Latin letter. Also, if we can do machine learning to the dish name, we may explore gradual change of dish that's presented by the collection. But this is not the goal to this project and there seems to be no effective clustering or labeling solution to our dish name so far.

Finally, since all data is collected from menu, we cannot achieve food culture change that's not reflected on the menu in this dataset.

## III. Data Wrangling by OpenRefine

This project applies OpenRefine as major tool in data wrangling process due to its visualized operation and optimized algorithm. I will use OpenRefine to remove unnecessary space or punctuation mark, to transform certain attribute to consistent format, to cluster specific value as syntactic check. After this data wrangling process, the detail of workflow will be transformed into flow graph[7] by OR2YW python toolkit that's developed by one of the TAs, Lan Li in CS513 [8].

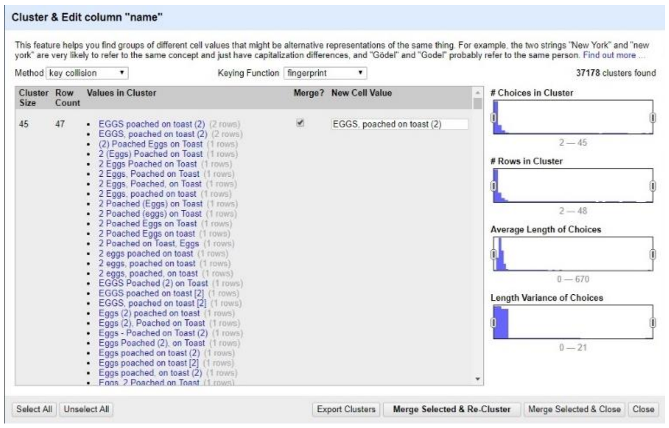
Detail to the OpenRefine procedure will be presented with different files in this section. Leading and trailing space removal is by default applied to the text column after filter function validation. The id uniqueness to all tables is partially verified in duplicates facet and text facet. None of them return positive or null value.

### Dish

Even though OpenRefine acts as data wrangling tool here, it really provides some great features to check the metadata or basic structure of the table. For example, we can check the uniqueness of the id attribute that will affect the dish table and other table referring to this column.

The applied dataset in the project covers material from 1850 to 2017, the facet feature can also help to check if there is year input that's out of range. An odd number '2928' is caught on *first_appeared* and *last_appeared* that seems to be typo in specific input batch so we can correct it with '1928'.

I also removed "*" and "!" since they barely carry any value like other punctuation marks. After this removal, I apply the clustering function to normalize the *name* column. Due to the heavy processing load from file's scale, I increase the applying memory in the OpenRefine initialization file before dealing with the *name* column. Figure. 2 is the cluster window at the normalization. There are 37178 clusters found by OpenRefine and most of clusters has less than 10 rows in it. Obviously, the *name* column normalization can't be effective. Here I used the default key collision method and fingerprint keying function. The key collision method will be most inexpensive comparing to nearest-neighbor method in OpenRefine. The Fingerprint key function is basically removing the leading/trailing space, punctuation or control characters, change English letter to lowercase and normalize letters from other language to English letter in ASCII[9]. I just applied this clustering once to the name attribute, you can see it works pretty good at combining the same dishes with various input style in Figure. 2. After the first clustering, OpenRefine will have issue in second clustering to this attribute and it started to cluster the dishes name that's very different from each other. I also tried to apply other clustering methods in dish name normalization, including other keying function but they are very expensive even though more memory is assigned to OpenRefine. Besides, we need to balance the efficiency and false positive clustering target value. There is no effective solution in other data cleaning software or python, R script either. After the first clustering, the distinct values decreased about 10%. Deeper normalization may heavily rely on human input and judgment.


Cluster & Edit column "name"

This feature helps you find groups of different cell values that might be alternative representations of the same thing. For example, the two strings "New York" and "new york" are very likely to refer to the same concept and just have capitalization differences, and "Gödel" and "Godel" probably refer to the same person. Find out more ...

Method: key collision    Keying Function: fingerprint    37178 clusters found

| Cluster Size | Row Count | Values in Cluster | Merge? | New Cell Value |
|---|---|---|---|---|
| 45 | 47 | EGGS poached on toast (2) (2 rows)<br>EGGS, poached on toast (2) (2 rows)<br>(2) Poached Eggs on Toast (1 rows)<br>2 (Eggs) Poached on Toast (1 rows)<br>2 Eggs Poached on Toast (1 rows)<br>2 Eggs, Poached on Toast (1 rows)<br>2 Eggs, Poached, on Toast (1 rows)<br>2 Eggs, poached on toast (1 rows)<br>2 Poached (Eggs) on Toast (1 rows)<br>2 Poached (eggs) on Toast (1 rows)<br>2 Poached Eggs on Toast (1 rows)<br>2 Poached Eggs on toast (1 rows)<br>2 Poached on Toast, Eggs (1 rows)<br>2 eggs poached on toast (1 rows)<br>2 eggs, poached on toast (1 rows)<br>2 eggs, poached, on toast (1 rows)<br>EGGS Poached (2) on Toast (1 rows)<br>EGGS poached on toast [2] (1 rows)<br>EGGS, poached on toast [2] (1 rows)<br>Eggs (2) poached on toast (1 rows)<br>Eggs (2), Poached on Toast (1 rows)<br>Eggs - Poached on Toast (2) (1 rows)<br>Eggs Poached (2), on Toast (1 rows)<br>Eggs poached on toast (2) (1 rows)<br>Eggs poached on toast [2] (1 rows)<br>Eggs poached, on toast (2) (1 rows)<br>Eggs, 2 Poached on Toast (1 rows) | ✓ | EGGS, poached on toast (2) |

Fig. 2 Clustering Feature to Name Attribute in Dish Table

Figure 3. shows the workflow to the Dish table cleaning process in OpenRefine. It's also included in the zip file. Table 1. present the detail to the OpenRefine provenance.

| Operation | Affected cells |
|---|---|
| Trim leading/trailing whitespace in *name* | 9045 |
| Collapse consecutive whitespace in *name* | 6415 |
| Cluster and merge in *name* | 110463 |
| Replace '!' with whitespace from *name* | 330 |
| Trim leading/trailing whitespace in *name* | 330 |
| Collapse consecutive whitespace in *name* | 329 |
| Replace '*' with whitespace from *name* | 2513 |
| Trim leading/trailing whitespace in *name* | 2247 |
| Collapse consecutive whitespace in *name* | 244 |
| Correct '2928' with '1928' in *first_appeared* | 11 |
| Correct '2928' with '1928' in *last_appeared* | 179 |

Table. 1 OpenRefine Operation in Dish File

## MenuItem

As for the Dish table, OpenRefine facet statistics is used to check the uniqueness of the id before IC check in section IV. After transforming the *xpos* and *ypos* attribute to numeric, I also verified both parameters are within [0,1].

In order to apply the SQLite query to the *created_at* and *update_at* columns in the next section, the UTC suffix is removed from the value by GREL pattern transformation. Figure 4. shows the YesWorkflow graph generated by or2yw tool. Table 2. shows the operation to MenuItem file.

| Operation | Affected cells |
|---|---|
| Transform *xpos* column to numeric type for value verification | 1332726 |
| Transform *ypos* column to numeric type for value verification | 1332726 |
| Remove UTC from *created_at* column | 1332726 |
| Remove UTC from *updated_at* column | 1332726 |

Table. 2 OpenRefine Operation in MenuItem File

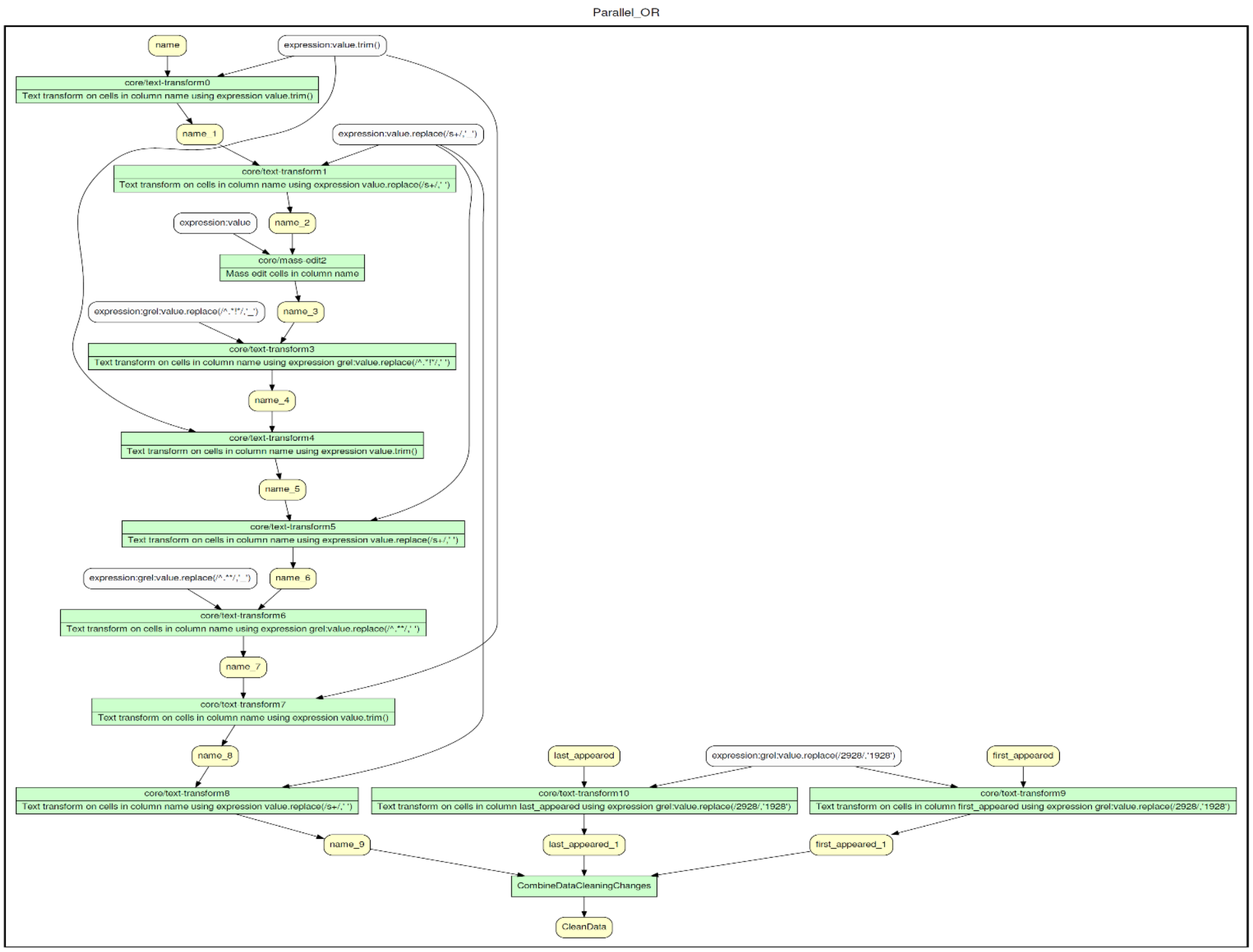

Parallel_OR
name
expression:value.trim()
core/text-transform0
Text transform on cells in column name using expression value.trim()
name_1
expression:value.replace(/s+/,'_')
core/text-transform1
Text transform on cells in column name using expression value.replace(/s+/,' ')
expression:value
name_2
core/mass-edit2
Mass edit cells in column name
expression:grel:value.replace(/^.*!*/,'_')
name_3
core/text-transform3
Text transform on cells in column name using expression grel:value.replace(/^.*!*/,' ')
name_4
core/text-transform4
Text transform on cells in column name using expression value.trim()
name_5
core/text-transform5
Text transform on cells in column name using expression value.replace(/s+/,' ')
expression:grel:value.replace(/^.**/,'_')
name_6
core/text-transform6
Text transform on cells in column name using expression grel:value.replace(/^.**/,' ')
name_7
core/text-transform7
Text transform on cells in column name using expression value.trim()
name_8
last_appeared
expression:grel:value.replace(/2928/,'1928')
first_appeared
core/text-transform8
Text transform on cells in column name using expression value.replace(/s+/,' ')
core/text-transform10
Text transform on cells in column last_appeared using expression grel:value.replace(/2928/,'1928')
core/text-transform9
Text transform on cells in column first_appeared using expression grel:value.replace(/2928/,'1928')
name_9
last_appeared_1
first_appeared_1
CombineDataCleaningChanges
CleanData

Fig. 3 Dish Table OpenRefine Workflow

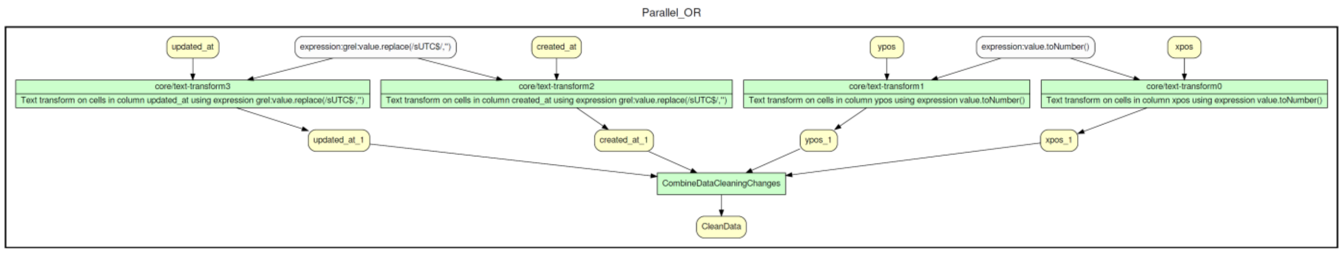

Parallel_OR
updated_at
expression:grel:value.replace(/sUTC$/,'')
created_at
ypos
expression:value.toNumber()
xpos
core/text-transform3
Text transform on cells in column updated_at using expression grel:value.replace(/sUTC$/,'')
core/text-transform2
Text transform on cells in column created_at using expression grel:value.replace(/sUTC$/,'')
core/text-transform1
Text transform on cells in column ypos using expression value.toNumber()
core/text-transform0
Text transform on cells in column xpos using expression value.toNumber()
updated_at_1
created_at_1
ypos_1
xpos_1
CombineDataCleaningChanges
CleanData

Fig. 4 MenuItem Table OpenRefine Workflow

**MenuPage**

OpenRefine can help to check the *id* uniqueness and estimate the value range of the *page_number* column. There is no value correction or format transformation to this table.

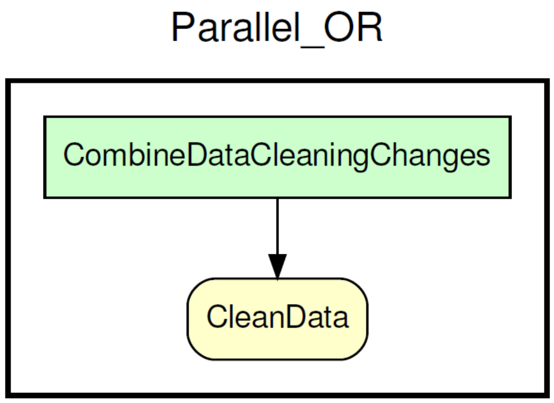


Fig. 5 MenuPage Table OpenRefine Workflow

**Menu**

Figure 6. shows the workflow to the menu cleaning process in OpenRefine. The punctuation mark that's removed in Dish table do not exist in the text column of this table. But I found quite a few of values of specific columns coming with the beginning parenthesis/bracket and ending parenthesis/bracket. They're removed by General Refine Expression Language (GREL) transformation in *sponsor*, *event*, *occasion*, *place*, *venue*, *location* columns. Before applying regex pattern (value.replace(/^\[|\]$/,'')) in transform function, I used filter function to make sure this action only take effect to the parenthesis/bracket occupying the leading or trailing position.

In order to enhance the potential analysis application to the dataset, the content of *physical_design* and *notes* column will not be changed.

The *date* column has been verified and maintained in the format of "yyyy-mm-dd" which is also standard date format to the SQLite query.

Based on the content similarity of *sponsor* and *event*, clustering with key-collision and fingerprint key function is applied to correct the syntactic issue. E.g. "lunch", "luncheon", "[LUNCH]", "MITTAGESSEN" are all merged to "LUNCH", however, some value with specific meaning like "LUNCH ROOM", "LUNCH SECOND CABIN" are kept without change for potential data analysis. After clustering, both *sponsor* and *event* attributes have more than 5000 normalized values.

| Operation | Affected cells |
|---|---|
| Cluster and merge in *event* | 5597 |
| Cluster and merge in *sponsor* | 5489 |
| Remove leading/trailing '[' and ']' from *sponsor* | 231 |
| Remove leading/trailing '(' and ')' from *sponsor* | 95 |
| Remove leading/trailing '[' and ']' from *event* | 20 |
| Remove leading/trailing '(' and ')' from *event* | 8 |
| Remove leading/trailing '[' and ']' from *venue* | 17 |
| Remove leading/trailing '(' and ')' from *venue* | 3 |
| Remove leading/trailing '[' and ']' from *place* | 200 |
| Remove leading/trailing '(' and ')' from *place* | 55 |
| Remove leading/trailing '[' and ']' from *occasion* | 42 |
| Remove leading/trailing '(' and ')' from *occasion* | 14 |
| Remove leading/trailing '[' and ']' from *location* | 203 |
| Remove leading/trailing '(' and ')' from *location* | 0 |
| Transform value to format "yyyy-MM-dd" in *date* | 4 |

Table. 3 OpenRefine Operation in Menu File except space trimming and consecutive space removal

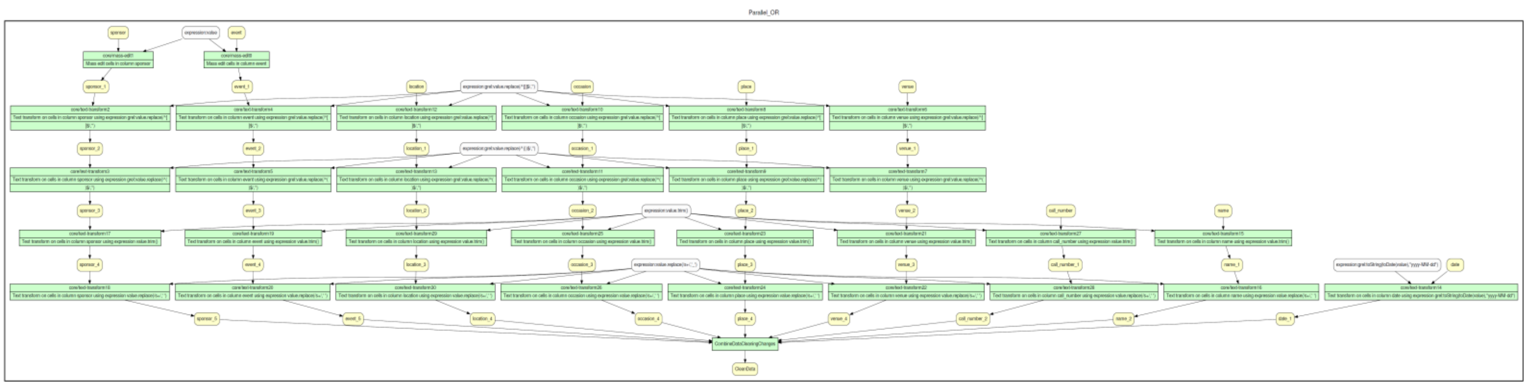


Fig. 6 Menu Table OpenRefine Workflow

Clustering is not applied to *venue*, *place*, *occasion* and *location*. This is because attributes like *venue*, *occasion* has acronym value which is prone to cause false positive candidate to the clustering. The other set of attributes like *place* and *location* has various values so clustering is not efficient to them, in fact, the clustering normalization can take effect on less than 3% of all the values in these columns.

Table 3. shows the detailed operation of Menu file.

The raw data generate a new set of cleaned csv files[10] and json files[10] after OpenRefine cleaning process. This new set is also the input to following sections.

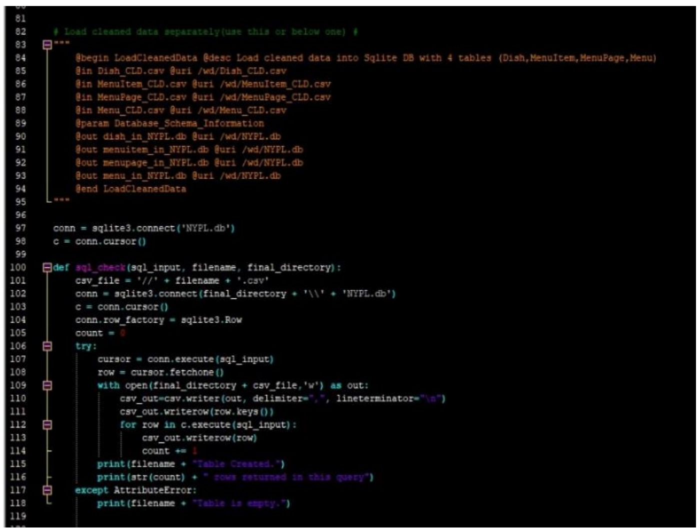
## IV. Develop Relational Database Schema and Semantic Integrity Check

### A. *Create Relational Database and Load Data*

The output of section III is cleaned file with csv format and provenance with json format. At the beginning of this section, csv files are loaded into new SQLite database with a few command lines. These command lines are included in the zip file SQLite folder [11]. The datatype of attributes is edited after loading. In order to modify the attribute smoothly, I used DB Browser GUI. It's not mandatory to modify the attribute data type for most of the SQLite utility, because SQLite's query provides handy methods to fulfill user's query requirement even though all the columns are with TEXT type. In this project, data type is modified, and we have the database and data type to each column as Fig 1.

### B. *Semantic Integrity Constraint Check*

The integrity constraint I emphasized in this section is the semantic constraints identified by SQL query. In order to process the IC check and display the result efficiently, I process query in python function by importing SQLite library into python script (Figure 7.). The YesWorkflow-style annotation is also placed into the same script for workflow generation purpose. This python script[12], the individual SQL command[13] are included in the zip file. The result view is stored in share folder,[14][15]. Some of the attribute, the standard format is not clearly defined, such as *uuid* in MenuPage and *call_number* in Menu. This project will NOT check their integrity.



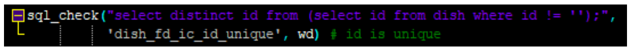
```
# Load cleaned data separately(use this or below one) #
"""
    @begin LoadCleanedData @desc Load cleaned data into Sqlite DB with 4 tables (Dish,MenuItem,MenuPage,Menu)
    @in Dish_CLD.csv @uri /wd/Dish_CLD.csv
    @in MenuItem_CLD.csv @uri /wd/MenuItem_CLD.csv
    @in MenuPage_CLD.csv @uri /wd/MenuPage_CLD.csv
    @in Menu_CLD.csv @uri /wd/Menu_CLD.csv
    @param Database_Schema_Information
    @out dish_in_NYPL.db @uri /wd/NYPL.db
    @out menuitem_in_NYPL.db @uri /wd/NYPL.db
    @out menupage_in_NYPL.db @uri /wd/NYPL.db
    @out menu_in_NYPL.db @uri /wd/NYPL.db
    @end LoadCleanedData
"""

conn = sqlite3.connect('NYPL.db')
c = conn.cursor()

def sql_check(sql_input, filename, final_directory):
    csv_file = '//' + filename + '.csv'
    conn = sqlite3.connect(final_directory + '\\' + 'NYPL.db')
    c = conn.cursor()
    conn.row_factory = sqlite3.Row
    count = 0
    try:
        cursor = conn.execute(sql_input)
        row = cursor.fetchone()
        with open(final_directory + csv_file,'w') as out:
            csv_out=csv.writer(out, delimiter=",", lineterminator="\n")
            csv_out.writerow(row.keys())
            for row in c.execute(sql_input):
                csv_out.writerow(row)
                count += 1
        print(filename + "Table Created.")
        print(str(count) + " rows returned in this query")
    except AttributeError:
        print(filename + "Table is empty.")
```

Fig. 7 Python script for SQLite query and IC check Workflow

### **Functional Dependency and Logical Constraints Query**

i) Dish

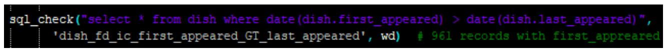
a. *id* uniqueness (functional dependency constraint)



```
sql_check("select distinct id from (select id from dish where id != '');",
            'dish_fd_ic_id_unique', wd) # id is unique
```

output: 423397 records (All match constraint)

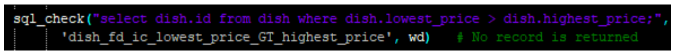
b. *first_appeared* is later than *last_appeared* (logical constraint violation)



```
sql_check("select * from dish where date(dish.first_appeared) > date(dish.last_appeared)",
        'dish_fd_ic_first_appeared_GT_last_appeared', wd)  # 961 records with first_appreared
```

output: 961 records (violated this logical constraint)

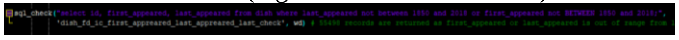
c. *lowest_price* is higher than *highest_price* (logical constraint violation)



```
sql_check("select dish.id from dish where dish.lowest_price > dish.highest_price;",
        'dish_fd_ic_lowest_price_GT_highest_price', wd)   # No record is returned
```

output: 0 record (violated this logical constraint)

d. *first_appeared* or *last_appeared* is NOT within the range from '1850' to '2018' (logical constraint violation)

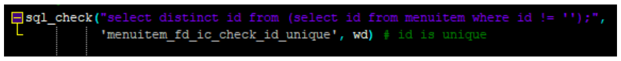


```
sql_check("select id, first_appeared, last_appeared from dish where last_appeared not between 1850 and 2018 or first_appeared not BETWEEN 1850 and 2018;",
        'dish_fd_ic_first_appreared_last_appreared_last_check', wd) # 55498 records are returned as first_appeared or last_appeared is out of range from
```

output: 55498 records (violated this logical constraint)

ii) MenuItem

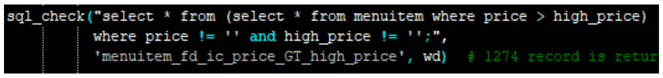
a. *id* uniqueness (functional dependency constraint)



```
sql_check("select distinct id from (select id from menuitem where id != '');",
            'menuitem_fd_ic_check_id_unique', wd) # id is unique
```

output: 1332726 records (All match constraint)

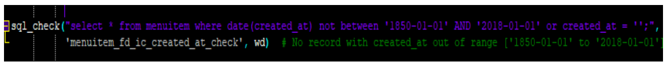
b. *price* is higher than *high_price* (logical constraint violation)



```
sql_check("select * from (select * from menuitem where price > high_price)
            where price != '' and high_price != '';",
            'menuitem_fd_ic_price_GT_high_price', wd)  # 1274 record is retur
```

output: 1274 records (violated this logical constraint)

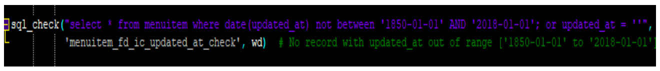
c. *created_at* is not from '1850-01-01' to '2018-01-01' (logical constraint violation)



```
sql_check("select * from menuitem where date(created_at) not between '1850-01-01' AND '2018-01-01' or created_at = '';",
            'menuitem_fd_ic_created_at_check', wd)  # No record with created_at out of range ['1850-01-01' to '2018-01-01']
```

output: 0 record (violated this logical constraint)

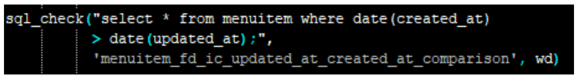
d. *updated_at* is not from '1850-01-01' to '2018-01-01' (logical constraint violation)



```
sql_check("select * from menuitem where date(updated_at) not between '1850-01-01' AND '2018-01-01'; or updated_at = ''",
            'menuitem_fd_ic_updated_at_check', wd)  # No record with updated_at out of range ['1850-01-01' to '2018-01-01']
```

output: 0 record (violated this logical constraint)

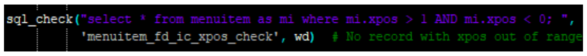
e. *created_at* is later than *updated_at*
(logical constraint violation)



```
sql_check("select * from menuitem where date(created_at)
        > date(updated_at);",
        'menuitem_fd_ic_updated_at_created_at_comparison', wd)
```

output: 3 records with created_at later than updated_at

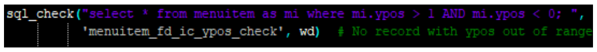
f. *xpos* is out of range [0,1] (logical constraint violation)



```
sql_check("select * from menuitem as mi where mi.xpos > 1 AND mi.xpos < 0; ",
        'menuitem_fd_ic_xpos_check', wd)  # No record with xpos out of range
```

output: 0 record (violated this logical constraint)

g. *ypos* is out of range [0,1] (logical constraint violation)

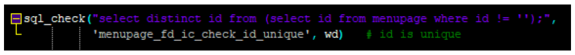


```
sql_check("select * from menuitem as mi where mi.ypos > 1 AND mi.ypos < 0; ",
        'menuitem_fd_ic_ypos_check', wd)  # No record with ypos out of range
```

output: 0 record (violated this logical constraint)

iii) MenuPage

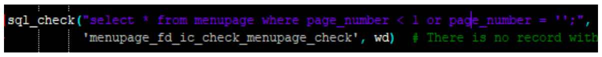
a. *id* uniqueness (functional dependency constraint)



```
sql_check("select distinct id from (select id from menupage where id != '');",
        'menupage_fd_ic_check_id_unique', wd)   # id is unique
```

output: 66937 records (All match constraint)

b. *page_number* is less than 1 or null (logical constraint violation)

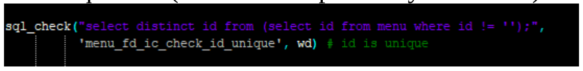


```
sql_check("select * from menupage where page_number < 1 or page_number = '';",
        'menupage_fd_ic_check_menupage_check', wd)  # There is no record with
```

output: 1202 records (violated this logical constraint)

iv) Menu

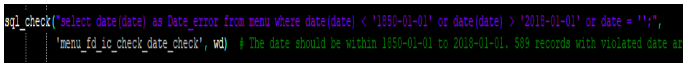
a. *id* uniqueness (functional dependency constraint)



```
sql_check("select distinct id from (select id from menu where id != '');",
        'menu_fd_ic_check_id_unique', wd) # id is unique
```

output: 17545 record (All match constraint)

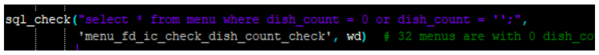
b. *date* should be from 1850-01-01 to 2018-01-01
(logical constraint violation)



```
sql_check("select date(date) as Date_error from menu where date(date) < '1850-01-01' or date(date) > '2018-01-01' or date = '';",
        'menu_fd_ic_check_date_check', wd)  # The date should be within 1850-01-01 to 2018-01-01. 589 records with violated date ar
```

output: 589 records (violated this logical constraint)

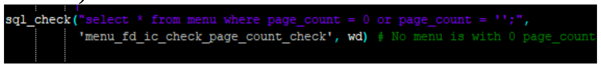
c. menus shouldn't carry 0 *dish_count* (logical constraint violation)



```
sql_check("select * from menu where dish_count = 0 or dish_count = '';",
        'menu_fd_ic_check_dish_count_check', wd)  # 32 menus are with 0 dish_co
```

output: 32 records (violated this logical constraint)

d. menus shouldn't carry 0 *page_count* (logical constraint violation)

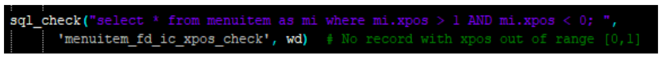


```
sql_check("select * from menu where page_count = 0 or page_count = '';",
        'menu_fd_ic_check_page_count_check', wd) # No menu is with 0 page_count
```

output: 0 record (violated this logical constraint)

**Inclusion Dependency Constraints Query**

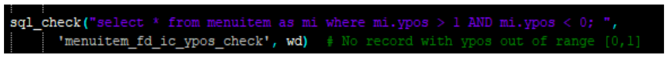
i) Verify *menu_page_id* in MenuItem is included as part of *id* in MenuPage (Inclusion Dependency Constraints)



```
sql_check("select * from menuitem as mi where mi.xpos > 1 AND mi.xpos < 0; ",
        'menuitem_fd_ic_xpos_check', wd)  # No record with xpos out of range [0,1]
```

output: 1332726 records (All records meet inclusion dependency constraint)

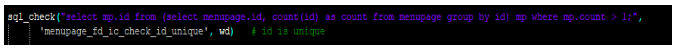
ii) Verify *dish_id* in MenuItem is included as part of *id* in Dish (Inclusion Dependency Constraints)



```
sql_check("select * from menuitem as mi where mi.ypos > 1 AND mi.ypos < 0; ",
        'menuitem_fd_ic_ypos_check', wd)  # No record with ypos out of range [0,1]
```

output: 1332482 records (meet inclusion dependency constraint)

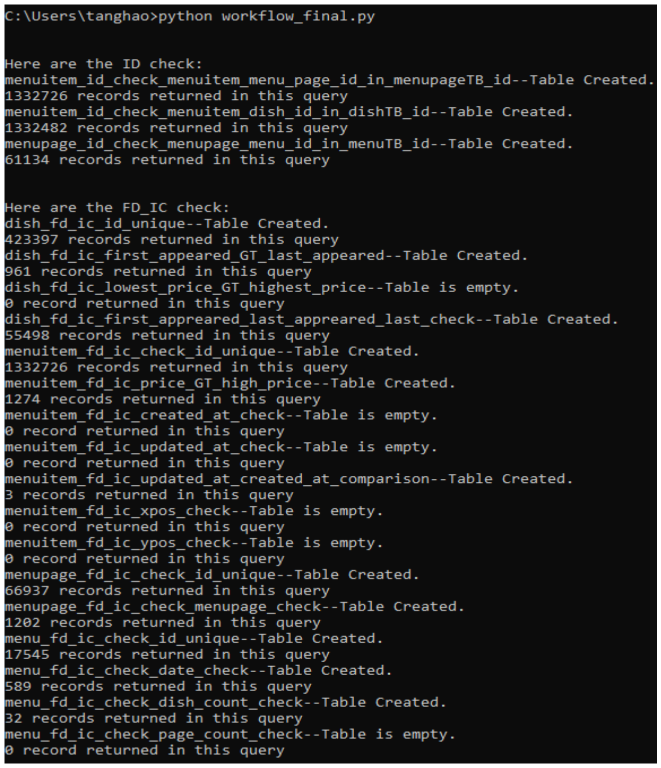
iii) Verify *menu_id* in MenuPage is included as part of *id* in Menu (Inclusion Dependency Constraints)



```
sql_check("select mp.id from (select menupage.id, count(id) as count from menupage group by id) mp where mp.count > 1;",
        'menupage_fd_ic_check_id_unique', wd)   # id is unique
```

output: 61134 records (meet inclusion dependency constraint)

Thus *dish_id* in MenuItem table and *menu_id* in MenuPage table violated the referential constraint according to their parent table Dish and Menu in the relational database we built. However, *menu_page_id* in MenuItem meets referential constraint as the foreign key. Figure 8. is the query result to the IC check.

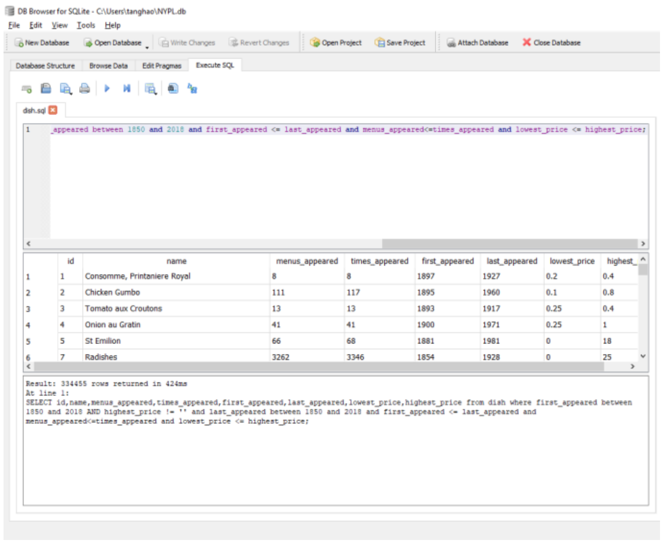


```
C:\Users\tanghao>python workflow_final.py


Here are the ID check:
menuitem_id_check_menuitem_menu_page_id_in_menupageTB_id--Table Created.
1332726 records returned in this query
menuitem_id_check_menuitem_dish_id_in_dishTB_id--Table Created.
1332482 records returned in this query
menupage_id_check_menupage_menu_id_in_menuTB_id--Table Created.
61134 records returned in this query


Here are the FD_IC check:
dish_fd_ic_id_unique--Table Created.
423397 records returned in this query
dish_fd_ic_first_appeared_GT_last_appeared--Table Created.
961 records returned in this query
dish_fd_ic_lowest_price_GT_highest_price--Table is empty.
0 record returned in this query
dish_fd_ic_first_appeared_last_appeared_last_check--Table Created.
55498 records returned in this query
menuitem_fd_ic_check_id_unique--Table Created.
1332726 records returned in this query
menuitem_fd_ic_price_GT_high_price--Table Created.
1274 records returned in this query
menuitem_fd_ic_created_at_check--Table is empty.
0 record returned in this query
menuitem_fd_ic_updated_at_check--Table is empty.
0 record returned in this query
menuitem_fd_ic_updated_at_created_at_comparison--Table Created.
3 records returned in this query
menuitem_fd_ic_xpos_check--Table is empty.
0 record returned in this query
menuitem_fd_ic_ypos_check--Table is empty.
0 record returned in this query
menupage_fd_ic_check_id_unique--Table Created.
66937 records returned in this query
menupage_fd_ic_check_menupage_check--Table Created.
1202 records returned in this query
menu_fd_ic_check_id_unique--Table Created.
17545 records returned in this query
menu_fd_ic_check_date_check--Table Created.
589 records returned in this query
menu_fd_ic_check_dish_count_check--Table Created.
32 records returned in this query
menu_fd_ic_check_page_count_check--Table is empty.
0 record returned in this query
```

Fig. 8 Query result to the IC check

## V. GENERATE CLEANED DATA FILE

The IC check in Section IV. doesn't alter the tables in the database so we can complete all the integrity constraint check. In this section, I will used SQLite to generate ready-to-use csv file by removing the empty column and attribute violated integrity constraints. We will have this new dataset with no disturbing attribute in business application but there is still another relatively original version[10] for advanced data analysis.



```
1   _appeared between 1850 and 2018 and first_appeared <= last_appeared and menus_appeared<=times_appeared and lowest_price <= highest_price;
```

|  | id | name | menus_appeared | times_appeared | first_appeared | last_appeared | lowest_price | highest_ |
|---|---|---|---|---|---|---|---|---|
| 1 | 1 | Consomme, Printaniere Royal | 8 | 8 | 1897 | 1927 | 0.2 | 0.4 |
| 2 | 2 | Chicken Gumbo | 111 | 117 | 1895 | 1960 | 0.1 | 0.8 |
| 3 | 3 | Tomato aux Croutons | 13 | 13 | 1893 | 1917 | 0.25 | 0.4 |
| 4 | 4 | Onion au Gratin | 41 | 41 | 1900 | 1971 | 0.25 | 1 |
| 5 | 5 | St Emilion | 66 | 68 | 1881 | 1981 | 0 | 18 |
| 6 | 7 | Radishes | 3262 | 3346 | 1854 | 1928 | 0 | 25 |

```
Result: 334455 rows returned in 424ms
At line 1:
SELECT id,name,menus_appeared,times_appeared,first_appeared,last_appeared,lowest_price,highest_price from dish where first_appeared between
1850 and 2018 AND highest_price != '' and last_appeared between 1850 and 2018 and first_appeared <= last_appeared and
menus_appeared<=times_appeared and lowest_price <= highest_price;
```

Fig. 9 Query to generate new tables in SQLite

It's efficient to apply DB browser for SQLite to generate the cleaned csv file by processing the SQL command (Figure 9.). The output is saved in share folder[16]. The SQL command that's used here is included in the project's zip file. Table 4 to Table 7 represent the query condition in new file generation.

| Constraints | Description |
|---|---|
| 1 | *Description* column is removed due to no content |
| 2 | Tuples with *first_appeared*/*last_appeared* out of 1850 to 2018 are removed. |
| 3 | Tuples with *first_appeared* later than *last_appeared* are removed. |
| 4 | Tuples with *menus_appeared* greater than *times_appeared* are removed. |
| 5 | Tuples with *lowest_price* higher than *highest_price* are removed. |

Table. 4 Procedure to generate new Dish csv file with IC
334455 records are generated.

| Constraints | Description |
|---|---|
| 1 | *keywords* column is removed due to no content. |
| 2 | *language* column is removed due to no content. |
| 3 | *location_type* columns is removed due to no content. |
| 4 | Tuples with *dish_count* = 0 or '' are removed. |
| 5 | Tuples with *page_count* = 0 or ''are removed. |

Table. 5 Procedure to generate new Menu csv file with IC
16942 records are generated.

| Constraints | Description |
|---|---|
| 1 | Tuples with *menu_id* violating the referential constraint are removed |
| 2 | Tuples with *page_number* = 0 or '' are removed |
| 3 | Tuples with *full_height*/*full_width* = '' are removed |

Table. 6 Procedure to generate new MenuPage csv file with IC
57485 records are generated.

## VI. WORKFLOW TO PROJECT

In the Section III. we have seen the YesWorkflow's power of workflow rendering. Now we use the YesWorkflow 0.2.0 [17] to generate flow graph directly from our python file where I have recorded the detail to all the steps as the comment. The .gv file and facts file to the YesWorkflow implementation are included in the project's zip file[18].

```
$ yw graph workflow_final.py | dot -Tpng -o workflow_final.png
```

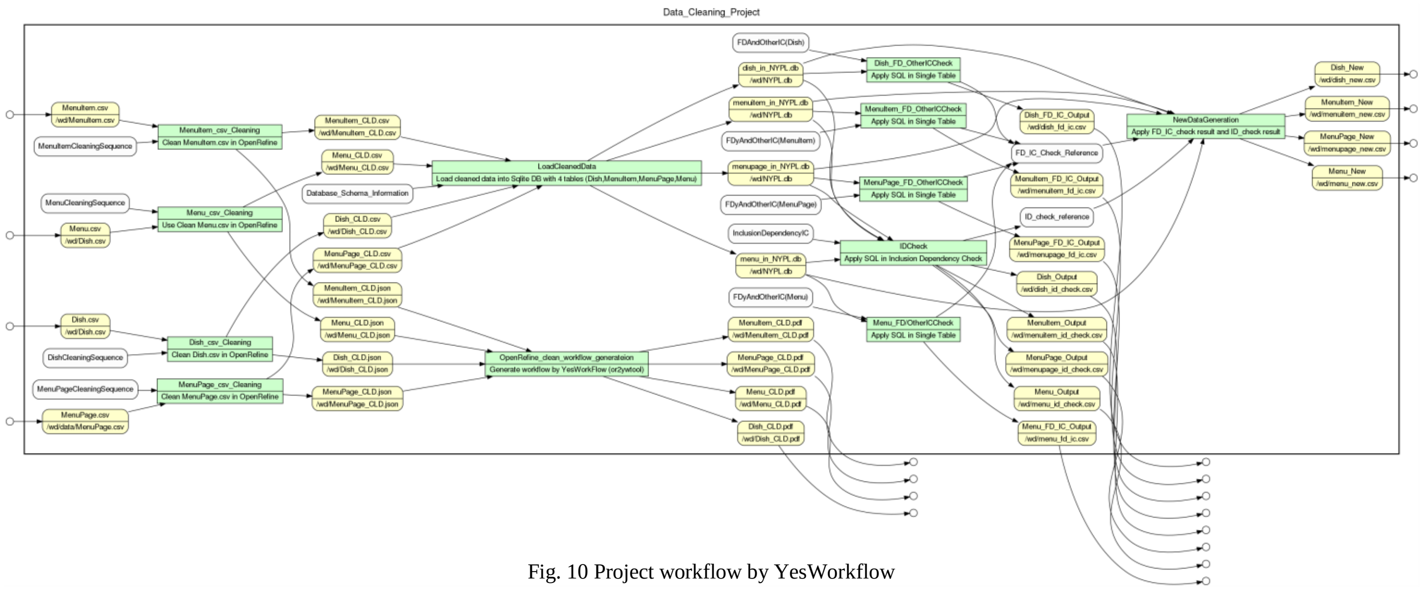


Fig. 10 Project workflow by YesWorkflow

However, we should consider implementing the YesWorkflow editor service interface in local machine to interactively edit the workflow script and rendering in the browser [19]. Figure 10. presents the data and steps in data cleaning project [20].

| Constraints | Description |
|---|---|
| 1 | If both *price* and *high_price* are not '', the tuples with *price* higher than *high_price* are removed |
| 2 | Tuples with *dish_id* violating the referential constraint are removed |
| 3 | Tuples with *menu_page_id* violating the referential constraint are removed |
| 4 | Tuples with *created_at* not in the date from '1850-01-01' to '2018-01-01' are removed |
| 5 | Tuples with *updated_at* not in the date from '1850-01-01' to '2018-01-01' are removed |
| 6 | Tuples with *created_at* later than *updated_at* are removed |
| 7 | Tuples with *xpos* out of range [0,1] are removed |
| 8 | Tuples with *ypos* out of range [0,1] are removed |

Table. 7 Procedure to generate new MenuItem csv file with IC
1099523 records are generated.

## VII. CONCLUSION

This project implements the data cleaning solution to the "What's on the menu?" dataset. The detail to this process is documented in the workflow graph. The workflow generation is one of the great features to this project. The whole flow chart display roadmap to the challenges, the solution applied to conquer them, and the output generated from the solution.

This project also applied one of the powerful visualized data wrangling tool OpenRefine in data cleaning, which dramatically increases the cleaning efficiency by its facet and clustering feature. Indeed, normalization of string value is always challenging even though OpenRefine has provided optimized clustering solution. This process is essentially an unsupervised learning. So far, human judgement is still the most significant input to the task.

The OpenRefine's output is imported into SQLite database and we build up the schema accordingly. Most of the query in this project is implemented in python SQLite library syntax and we can see SQLite can work efficiently with processing code script. The other database tool DB browser for SQLite is applied to modification and data export. Both the initially cleaned data [10] from OpenRefine and new data[16] generated by applying integrity constraint are retained because we want to have ready-to-use data as well as look forward to implementing advanced data analysis method. As expected at the beginning of the project, the new data generated by this project can be used in business assistance and menu design as solid basis.

## REFERENCES

[1] http://menus.nypl.org/about About in what's on the menu website.
[2] https://frankbuttolph.wordpress.com/ Menu Lady Mystery Solved
[3] http://menus.nypl.org/help Help in what's on the menu website.
[4] http://menus.nypl.org/data Data in what's on the menu website.
[5] http://nypl.github.io/menus-api/ Menu Beta API v1
[6] http://curatingmenus.org/
[7] /Tang-Hao/or2yw_json_flow
[8] https://pypi.org/project/or2ywtool/
[9] https://github.com/OpenRefine/OpenRefine/wiki/Clustering-In-Depth
[10]Google Drive Link:
https://drive.google.com/drive/folders/11bslcR4DTo47MZ6yEQIavp-9r0r7phj8?usp=sharing
UIUC Box Folder Link:
https://uofi.box.com/s/ahfrom1fl0espe0xn6g6yt8ne7ibprls
Cleaned data/json from OpenRefine is maintained for potential analysis
[11] /Tang-Hao/SQLite_SQL_Output/Build_Dataset_Dot_Command.txt
[12] /Tang-Hao/Workflow
[13] /Tang-Hao/Workflow/SQL_FD_OtherIC_Check
/Tang-Hao/Workflow/SQL_ID_Check
[14]Google Drive Link:
https://drive.google.com/drive/folders/1_2T0WrhTNVMVFaT_c0R1h5WJ1qPRCeRI?usp=sharing
UIUC Box Folder Link:
https://uofi.box.com/s/3fkszgb2jwaybm0pakt8v0p2ucve1aii
FD and other IC check output is stored in Google Drive.
[15]Google Drive Link:
https://drive.google.com/drive/folders/1y1hxgQkUXsGG13frBIUgWdO8fN7bOPrF?usp=sharing
UIUC Box Folder Link:
https://uofi.box.com/s/dhs3nrabbnat2hws8ep20xsqgmao3c0a
ID check output is stored in Google Drive.
[16]Google Drive Link:
https://drive.google.com/drive/folders/1RttzDcAqZ7Wj6cpTaRYlv3USO0x_CLB8?usp=sharing
UIUC Box Folder Link:
https://uofi.box.com/s/gkxlp6fwxrets2eyk67q34fgno2t93rl
New_Data_Generation is stored in Google Drive.
[17] https://github.com/yesworkflow-org/yw-prototypes
YesWorkflow 0.2.0 pre-release
[18] /Tang-Hao/Workflow/workflow_final.gv
/Tang-Hao/Workflow/workflow_final_facts
[19] https://github.com/yesworkflow-org/yw-editor-webapp
yw-editor-app-0.2.1.2 Release 0.2.1.2
[20] /Tang-Hao/Workflow/Rendering_of_Workflow.jpg