CS 513 - Theory and Practice of Data Cleaning

University of Illinois Urbana-Champaign

Prof. Bertram Ludäscher

Final Project - Phase 2

Team 194

Steve McHenry <mchenry7@illinois.edu>

Fabricio Brigagao <fb8@illinois.edu>

Roberto Godoy <ard8@illinois.edu>

July 31st, 2022.

Overview of Report

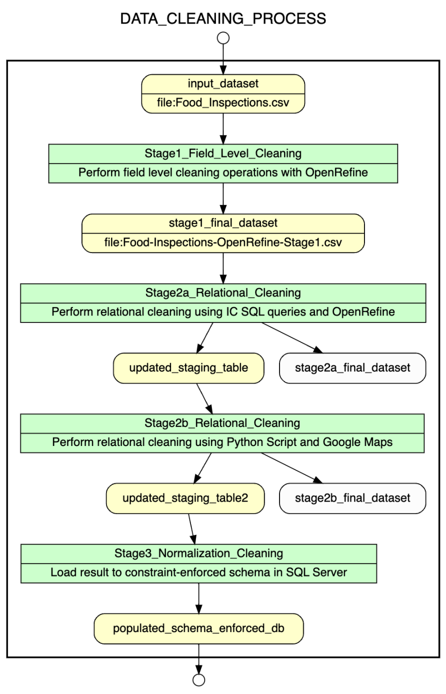
1. Data Cleaning Workflow

While developing our plan during Phase-I, we performed extensive analysis on the Chicago Department of Public Health's Food Inspections[1] dataset, *D*, to determine a main use case U_1 for the data contained within *D* which was appropriately challenging yet realistically possible (we discuss U_1 in Section 2). With the insight that we gained during Phase-I analysis of the 153,810 records in *D*, we were able to devise an informed and detailed data cleaning workflow, *W*, which we were able to execute successfully without deviation.

Our efforts were focused on moving the original dirty dataset, *D*, into a new state, *D'*, which is sufficiently cleaned to satisfy U_1 where the *D* is unable to do so. During our development of *W*, we imposed two supplementary directives upon our efforts so that *D'* would still be semantically true to *D*. In other words, that *D'* is a cleaned version of *D* and fully maintains the original meaning of *D*. We define both directives below, and later discuss how they were referenced throughout *W*.

The first directive was: *do not manufacture data to populate empty elements*. This dataset contained many empty data elements. It may be tempting to attempt to populate these fields with a reasonable, assumed value to obtain a "complete" dataset, however attempting to do so is inappropriate as the true value cannot be known with certainty – perhaps some empty elements are intentionally so. Therefore, manufacturing data would damage the integrity of the dataset and would cause U_1's use of such a dataset to produce incorrect and inaccurate results.

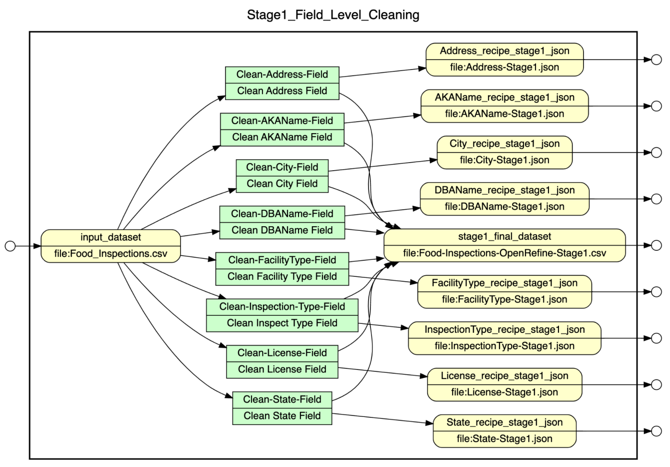
The second directive was: *do not falsify existing data for the sake of convenience*. This dataset contained many data elements which may – at face value – appear to be accidental duplicates due to slight differences between them (e.g., minor name spelling differences, slightly different street address numbers, etc.). It may be tempting to attempt to coerce and merge such data elements to assist in dataset normalization. Often, however, inspection in both the context of the element's full record and the whole dataset revealed that such values, though apparently similar in isolation, are truly *different*. Care must be taken to not overly-coerce the dataset such that we leave the realm of data cleaning and enter the realm of data falsification. Careless coercion attempts could potentially result in false meaning among some of the updated values and damage the integrity of the dataset. This would, again, cause U_1's use of such a dataset to produce incorrect and inaccurate results.


DATA_CLEANING_PROCESS
input_dataset
file:Food_Inspections.csv
Stage1_Field_Level_Cleaning
Perform field level cleaning operations with OpenRefine
stage1_final_dataset
file:Food-Inspections-OpenRefine-Stage1.csv
Stage2a_Relational_Cleaning
Perform relational cleaning using IC SQL queries and OpenRefine
updated_staging_table
stage2a_final_dataset
Stage2b_Relational_Cleaning
Perform relational cleaning using Python Script and Google Maps
updated_staging_table2
stage2b_final_dataset
Stage3_Normalization_Cleaning
Load result to constraint-enforced schema in SQL Server
populated_schema_enforced_db

The workflow *W*, reproduced in the image above, consists of three sequential stages, as described in our original plan in Phase-I. The first stage consists of field-level syntactic cleaning, with light semantic cleaning of fields whose values are drawn from a closed set. The second stage consists of record and dataset-level semantic cleaning from a relational perspective. The third stage consists of loading the cleaned dataset from the flat staging table into the normalized schema.

As in Phase-I, we use the Chicago Data Portal's specification[2] for this dataset as a guide throughout the data cleaning process. *W* is described in the remainder of this section, and *W*'s relation to U_1 is described in Section 2.

1.1 Stage 1: Field Level Cleaning

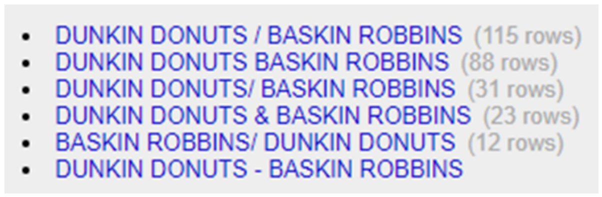


The first stage of *W* is field level cleaning tasks, as shown in the diagram. Here, we focus on syntactical corrections on each field in isolation. For fields whose values the specification defines as being drawn from a closed data set, we make only light semantic changes to merge values which are presented differently in the dataset, but clearly have identical meaning, while still taking care to not fall into the trap of data falsification.

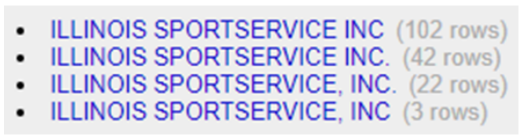
In this stage, OpenRefine is our primary tool of data cleaning and performs all operations upon the dataset. Additionally, we use SQL as a supplementary tool to perform more complex aggregations for analysis and validation activities against *D*, which are easier and more natural to perform in SQL than in OpenRefine.

For each field, we first perform the basic string operations of trimming whitespace from the beginning and end of values and collapsing multiple consecutive whitespaces into a single whitespace. We then normalize the case of text fields. Fields whose values are open-ended, such as names and addresses, are majority uppercase in *D*, so we convert those fields to all uppercase. Fields whose values are drawn from a closed set are majority title case in *D*, so we convert those fields to all title case.

For all text-based fields, we then make extensive use of OpenRefine's clustering and merging feature to identify and normalize values which are clearly semantically identical but differ only in the trivial ways. Trivial differences include injected or absent spaces, punctuation, or legal suffixes. Consider the two clustered sets below. The elements in each set clearly represent the same entity, differing only minorly in spacing and punctuation. We consider these values safely mergeable. A record-level inspection of these clusters confirms this to be true. These clusterings can be merged to "DUNKIN DONUTS / BASKIN ROBBINS" and "ILLINOIS SPORTSERVICE, INC.", respectively.

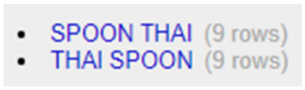


- DUNKIN DONUTS / BASKIN ROBBINS (115 rows)
- DUNKIN DONUTS BASKIN ROBBINS (88 rows)
- DUNKIN DONUTS/ BASKIN ROBBINS (31 rows)
- DUNKIN DONUTS & BASKIN ROBBINS (23 rows)
- BASKIN ROBBINS/ DUNKIN DONUTS (12 rows)
- DUNKIN DONUTS - BASKIN ROBBINS

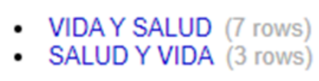


- ILLINOIS SPORTSERVICE INC (102 rows)
- ILLINOIS SPORTSERVICE INC. (42 rows)
- ILLINOIS SPORTSERVICE, INC. (22 rows)
- ILLINOIS SPORTSERVICE, INC (3 rows)

Now, consider the following proposed clusters. Perhaps these are the same business, simply with word ordering accidentally flipped? This is a somewhat reasonable assumption. However, inspecting both clusters at the record level shows, in reality, that these are separate businesses with distinct license numbers and addresses located across the city from each other. This underscores the degree of diligence required to avoid falsification of data during cleaning.



- SPOON THAI (9 rows)
- THAI SPOON (9 rows)

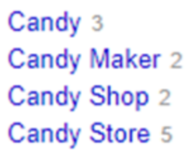


- VIDA Y SALUD (7 rows)
- SALUD Y VIDA (3 rows)

For these clustering and merging operations, we examine the values through each of OpenRefine's clustering methods and keying functions. We approximate that around 60% of

OpenRefine's proposed clusters for text-based fields in *D* are accurate; by no means a silver bullet – but still a great tool for expediting this process.

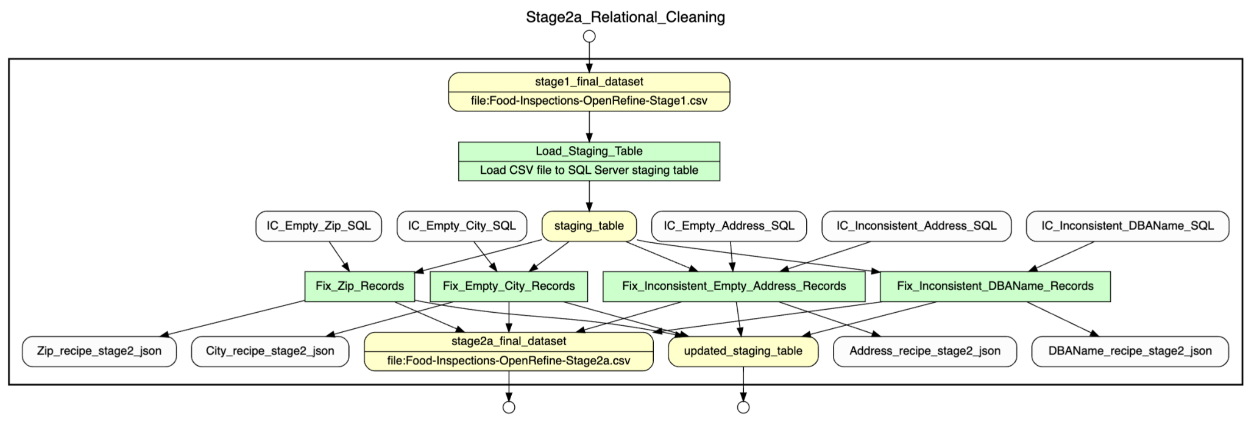
For text-based fields which the specification defines as having their values drawn from a closed set of values, we perform light semantic cleaning in addition to syntactic cleaning. The closed value set nature of these fields allows us to take this greater liberty. For example, given the four facility types below from *D*, and given the specification's list of defined facility types, specifically, "Candy Store", we can safely merge all four similar values from *D* into "Candy Store".


Candy 3
Candy Maker 2
Candy Shop 2
Candy Store 5

For the *License #* field, we update the small subset records containing a NULL value to *D*'s license placeholder value of "0" used by another small subset of other records in *D*. By doing so, we only have a single placeholder value in this field rather than two. There are 454 records (0.3%) in the dataset containing a default license value.

Throughout this stage, we maintain our OpenRefine-based operation history using a *separate* OpenRefine recipe file for each field. This way, we can work at the field level much more easily by not having to worry about accidentally rolling back (or rolling forward) operations on other unrelated columns which would otherwise be interleaved with the operations history of the column of interest. This has proven to be a very advantageous decision, given that accidents and realizations occur often during the cleaning process. For example, a batch of operations may need to be reverted completely, or reverted and then applied in a different way.

1.2 Stage 2: Relational Level Cleaning


Stage2a_Relational_Cleaning
stage1_final_dataset
file:Food-Inspections-OpenRefine-Stage1.csv
Load_Staging_Table
Load CSV file to SQL Server staging table
IC_Empty_Zip_SQL
IC_Empty_City_SQL
staging_table
IC_Empty_Address_SQL
IC_Inconsistent_Address_SQL
IC_Inconsistent_DBAName_SQL
Fix_Zip_Records
Fix_Empty_City_Records
Fix_Inconsistent_Empty_Address_Records
Fix_Inconsistent_DBAName_Records
Zip_recipe_stage2_json
City_recipe_stage2_json
stage2a_final_dataset
file:Food-Inspections-OpenRefine-Stage2a.csv
updated_staging_table
Address_recipe_stage2_json
DBAName_recipe_stage2_json

The second major stage of *W* is the relational level semantic cleaning tasks. The focus of this stage is to view the fields of *D* within the contexts of both their full record and among other records within the dataset. When fields are viewed in full context, we can make accurate semantic decisions that were impossible when viewing a single field in isolation. These relationally focused cleaning tasks set the stage for the subsequent schema normalization task. To begin relational level cleaning, the dataset updated by Stage 1 is exported from OpenRefine and imported into a staging table in a SQL database. We use Microsoft SQL Server as our database management system for this project.

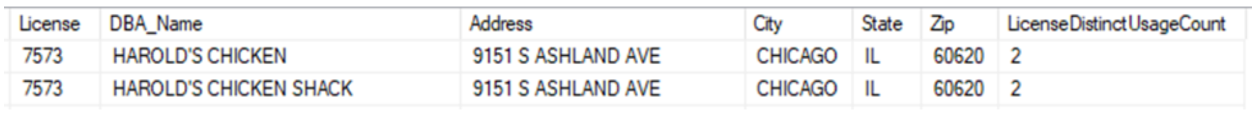
In this stage, we aim to clean and normalize the field values within each record such that we can identify all unique businesses from among the 153,810 inspection records. We determined a unique business to be represented by a composite key containing three elements: The license number (*License #* field), the business' legal name (the *DBA Name* field within *D*), and a concatenation of the address elements (the *Address* (i.e., street number), *City, State*, and *Zip* fields within *D*). Note that for reasons discussed in Section 5, a license number alone does not necessarily uniquely identify a business, as we discovered during Phase-I analysis.

Again, in this stage, we use OpenRefine to perform further field-level operations upon *D* but rely on SQL for schema-wide aggregations to determine which operations to perform. All fields involved in the business composite key must be non-NULL. Several fields, including *Address, City,* and *Zip* contain sparse NULL values throughout *D*. Although impossible to resolve at the field level (without manufacturing data), this issue can be resolved at the relational level. We resolve the composite key NULL values with a series of sequential cleaning steps.

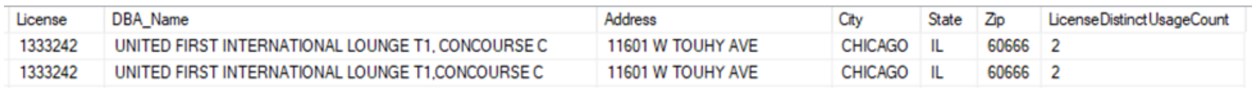
1.2.1 Resolving Duplicate License-Business Name pairs

We perform an aggregation to determine if there are any inconsistently represented duplicate business names that are unresolved after deduplicating business names during the field-level cleaning using OpenRefine's clustering and merging functionality. We determine one business name to be a duplicate of another when both names are nearly phonetically identical, share the same license number, and share a semantically identical address (e.g., differing only in internal spacing, street name suffixes, etc.).

During our first execution of Stage 2 on *D*, 1,228 candidate duplicates were returned. Although most candidates are licenses which relate to multiple distinct business names (therefore not duplicates), there are some duplicate business names that went undetected by OpenRefine's clustering operations. Consider the examples below.

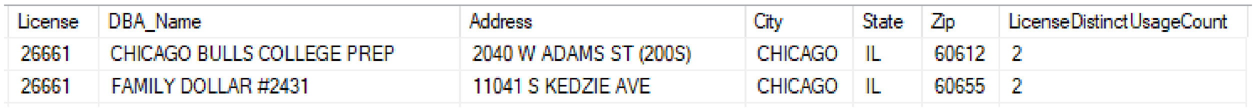


License	DBA_Name	Address	City	State	Zip	LicenseDistinctUsageCount
7573	HAROLD'S CHICKEN	9151 S ASHLAND AVE	CHICAGO	IL	60620	2
7573	HAROLD'S CHICKEN SHACK	9151 S ASHLAND AVE	CHICAGO	IL	60620	2



License	DBA_Name	Address	City	State	Zip	LicenseDistinctUsageCount
1333242	UNITED FIRST INTERNATIONAL LOUNGE T1, CONCOURSE C	11601 W TOUHY AVE	CHICAGO	IL	60666	2
1333242	UNITED FIRST INTERNATIONAL LOUNGE T1,CONCOURSE C	11601 W TOUHY AVE	CHICAGO	IL	60666	2

Of the 1,228 candidates, approximately 241 are determined to be duplicates. We eliminate duplicates by selecting the business name which has the greatest representation among the duplicates as the “winner”. We update these individually using OpenRefine, given the small occurrence of true duplicates remaining in the dataset. The SQL script used to generate these candidates is available as the file named “SQL Relational Integrity Constraint Check - Inconsistent DBA_Name.sql”. The 987 rejected candidates consist of businesses which share the same license number, but have clearly different business names or addresses and, therefore, can't be considered to be the same business. Consider the example below.

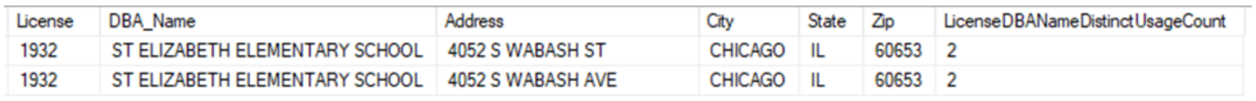


License	DBA_Name	Address	City	State	Zip	LicenseDistinctUsageCount
26661	CHICAGO BULLS COLLEGE PREP	2040 W ADAMS ST (200S)	CHICAGO	IL	60612	2
26661	FAMILY DOLLAR #2431	11041 S KEDZIE AVE	CHICAGO	IL	60655	2

1.2.2. Resolving Duplicate License-Business Name-Address tuples

We perform an aggregation to determine if there are any inconsistently represented duplicate addresses corresponding to unique license-business name pairs. We determine one address to be a duplicate of another when both addresses are semantically identical (e.g., differing only in internal spacing, street name suffixes, etc.) and share the same license number and business name.

During our first execution of Stage 2, approximately 211 candidates were returned. Although most of the candidates seem to be businesses which have moved to different locations over time, 85 addresses, such as the example below, are misspellings. In the case of the example below, the issue is a street name suffix mismatch.



License	DBA_Name	Address	City	State	Zip	LicenseDBANameDistinctUsageCount
1932	ST ELIZABETH ELEMENTARY SCHOOL	4052 S WABASH ST	CHICAGO	IL	60653	2
1932	ST ELIZABETH ELEMENTARY SCHOOL	4052 S WABASH AVE	CHICAGO	IL	60653	2

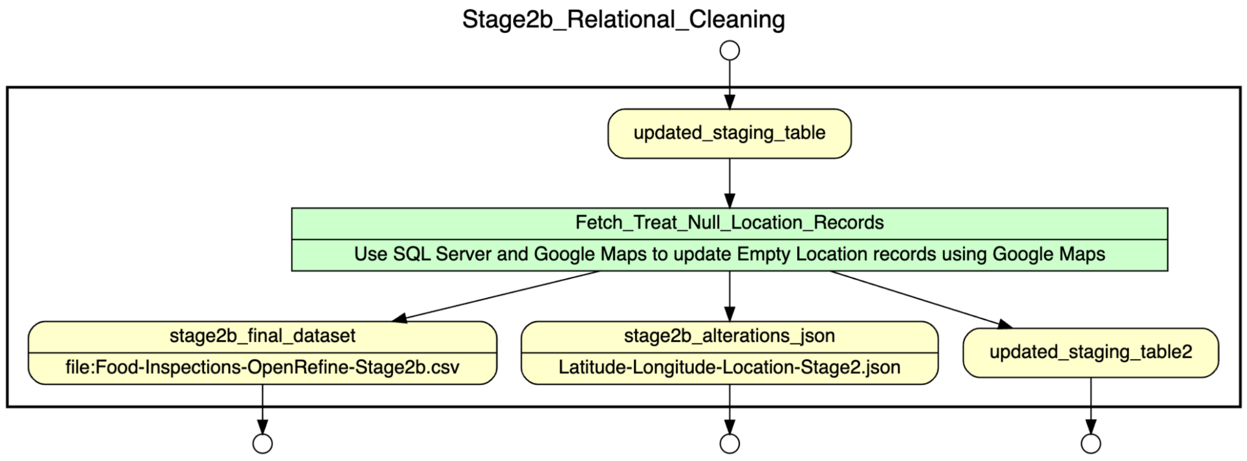
We update these individually in OpenRefine, using Google Maps’ user interface to manually confirm the correct spelling from the 85 identified duplicates. The SQL script used to generate these candidates is available as the file named “SQL Relational Integrity Constraint Check - Inconsistent Business Address.sql”.

1.2.3. Resolving Empty (NULL) Address and Location Components

We update all address and location components (*Address, City, Zip, Longitude, Latitude,* and *Location* fields) containing a NULL value. There are 174 records containing a NULL in one or more of the *Address*, *City*, and *Zip* fields. There are 544 records containing a NULL in the *Longitude*, *Latitude*, and *Location* fields. For this small subset of cases, there are two scenarios.

In the first scenario, the record contains insufficient *non*-NULL populated address fields (*Address, City*, and *Zip*), but does contain a *Location* (coordinates). We resolve this by submitting each of these records’ geographic coordinate address to Google Maps to retrieve a result. We update these individually in OpenRefine. The SQL scripts used to generate these candidates is available as the files named “SQL Relational Integrity Constraint Check - Empty

{Address|City|Zip}.sql”. After this process is complete, our use of OpenRefine in *W* is concluded. We export the current dataset in its current state, truncate the old contents of the staging table, and then load the current dataset into the staging table.

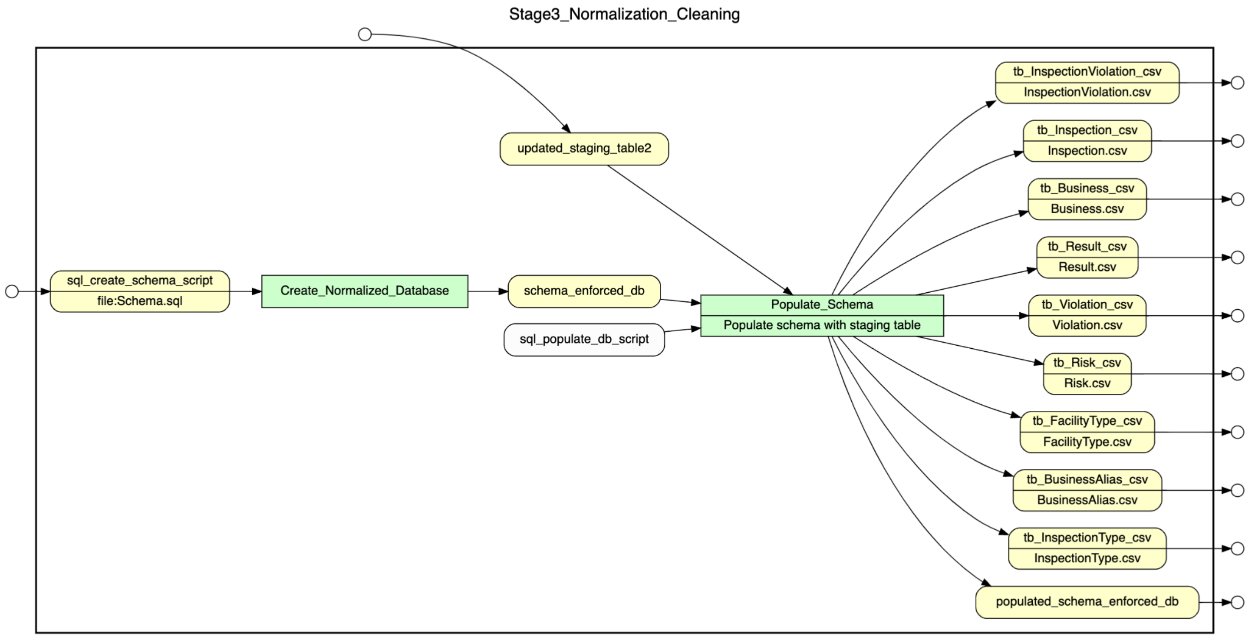

Stage2b_Relational_Cleaning
updated_staging_table
Fetch_Treat_Null_Location_Records
Use SQL Server and Google Maps to update Empty Location records using Google Maps
stage2b_final_dataset
file:Food-Inspections-OpenRefine-Stage2b.csv
stage2b_alterations_json
Latitude-Longitude-Location-Stage2.json
updated_staging_table2

In the second scenario, the record contains *non*-NULL populated address fields, but NULL-populated location coordinates. To resolve this, we implemented an API call from a Python script to the Geoapify API[3] to convert an address into geographic coordinates and directly update our SQL database with the result. This use of an API is the only deviation from our original Phase-I plan. We had planned to manually submit the addresses to Google Maps for conversion to geographic coordinates, but instead decided to take advantage of automation through an available API. After this process is complete, Stage 2 is concluded. The Python script used to perform this task is available as the file named “address-to-coordinates.py”.

Note that the Geoapify API has functionality for “reverse lookup”, i.e., translating coordinates to their respective address, but the syntactic and semantic quality of the results returned by the API were not as desired and would have required additional, manual cleaning efforts. We discuss this issue further in Section 5. For the sake of posterity, however, we provide a sample Python script which calls the API in this way as the file named “coordinates-to-address.py”.

During our first execution of Stage 2, we only encountered a single record which was missing both an address and geographical coordinates, and therefore had a completely unresolvable location. A placeholder address of “[NONE ON RECORD]” which identifies the record as having no known address nor coordinates was given to this record. Use of an explicit placeholder value to flag this record rather than a simulated value avoids data manufacturing concerns.

1.3 Stage 3: Constraint-Enforced Schema Normalization

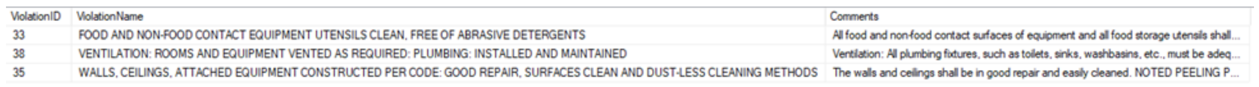


After Stages 1 and 2 are performed, the data in the flat staging table may be loaded into the normalized, constraint-enforced schema by executing the schema population SQL script. If *W* was executed properly, all integrity violations within the staging table are now resolved and schema-enforced constraints are now satisfied. If there are remaining integrity violations, the staging table will fail to be loaded into the schema and an error will be raised. We discuss the schema in Sections 3 and 4. The schema DDL is available in the file named “Schema.sql”, and the SQL script which loads the staging table into the schema is named “Populate Normalized Schema.sql”.

One final significant automated cleaning operation occurs during this stage: the normalization of the concatenated *Violations* field into a mapping table such that each inspection corresponds to zero or more health department defined violation codes with optional text comments provided by the inspector. Consider the following *Violations* field value below from the raw dataset. It contains three violation codes – each with an inspector-provided comment.

```
33. FOOD AND NON-FOOD CONTACT EQUIPMENT UTENSILS CLEAN, FREE OF ABRASIVE DETERGENTS - Comments: All food and non-food contact
surfaces of equipment and all food storage utensils shall be thoroughly cleaned and sanitized daily. INSTRUCTED TO CLEAN AND
MAINTAIN INTERIOR OF MICROWAVE OVEN IN TEACHERS LOUNGE. NOTED DRIED FOOD DEBRIS ACCUMULATION. | 38. VENTILATION: ROOMS AND
EQUIPMENT VENTED AS REQUIRED: PLUMBING: INSTALLED AND MAINTAINED - Comments: Ventilation: All plumbing fixtures, such as
toilets, sinks, washbasins, etc., must be adequately trapped, vented, and re-vented and properly connected to the sewer in
accordance with the plumbing chapter of the Municipal Code of Chicago and the Rules and Regulations of the Board of Health.
INSTRUCTED TO REPLACE MISSING FAUCET AT ONE OF 4 HANDSINK IN BOYS WASHROOM IN ANNEX. | 35. WALLS, CEILINGS, ATTACHED EQUIPMENT
CONSTRUCTED PER CODE: GOOD REPAIR, SURFACES CLEAN AND DUST-LESS CLEANING METHODS - Comments: The walls and ceilings shall be
in good repair and easily cleaned. NOTED PEELING PAINT ON CEILING AT ENTRANCE #3. INSTRUCTED TO REMOVE.
```

The normalization step splits this text string into records in the inspections-to-violations mapping table, as sampled below:



ViolationID	ViolationName	Comments
33	FOOD AND NON-FOOD CONTACT EQUIPMENT UTENSILS CLEAN, FREE OF ABRASIVE DETERGENTS	All food and non-food contact surfaces of equipment and all food storage utensils shall...
38	VENTILATION: ROOMS AND EQUIPMENT VENTED AS REQUIRED: PLUMBING: INSTALLED AND MAINTAINED	Ventilation: All plumbing fixtures, such as toilets, sinks, washbasins, etc., must be adeq...
35	WALLS, CEILINGS, ATTACHED EQUIPMENT CONSTRUCTED PER CODE: GOOD REPAIR, SURFACES CLEAN AND DUST-LESS CLEANING METHODS	The walls and ceilings shall be in good repair and easily cleaned. NOTED PEELING P...

With the conclusion of this step, *W* has now been fully executed, and *D* has been transitioned into state *D'* such that it satisfies the requirements of U$_1$. Data cleaning has been completed in accordance with our plan as described in Phase-I.

2. Relation of Data Cleaning Activities to U$_1$

The cleaning workflow, *W*, was designed specifically to clean the dataset *D* such that it was suitable for our use case, U$_1$. We begin our discussion of *W* in the context of U$_1$ by restating U$_1$ from our Phase-I report:

> *Provide the ability to query inspections based upon a logical disjunction of the following fields: Inspection date; Business name (both the business' legal name as well as its "also-known-as" name, if any); Business license number; Inspection result; Specific codes of violations.*

In dissecting U$_1$, we identify the elements of interest for query input as:

- Inspection date;
- Business name (both its legal and "also-known-as" names);
- Business license number;
- Inspection result;
- Violation codes.

Within *D*, these elements correspond to the *Inspection Date*, *DBA Name/AKA Name*, *License #*, *Results*, and *Violations* fields, respectively. We also include the Inspection ID field as a unique identifier to reference specific inspection instances.

Additional elements that are not explicitly specified as input criteria for U$_1$ are also required. As discovered during analysis, license numbers and business names alone are not sufficient to uniquely identify a business. License numbers may be reissued, entered erroneously, or even entered with a default "0" value in rare cases. Different businesses may have the same name, such as with franchised operations. Given these issues, additional fields are required to uniquely identify a business. For this, we include the business's address to complete the unique identifier for businesses. Within *D*, these address elements correspond with the *Address, City, State*, and *Zip* fields.

To aid software mapping and geolocation features, we also include the geographic location fields provided in *D.* These fields are complementary to the address fields by providing the geographic coordinates of the specified address. Within *D*, these are the *Longitude*, *Latitude*, and *Location* fields. The *Location* field is the longitude and latitude coordinates expressed as an ordered pair string.

D contains a remaining set of fields which, though interesting, are not necessary to fulfill U_1 and are only supplementary to our purposes. These fields are *Facility Type*, *Inspection Type*, and *Risk*.

Having completed this analysis of U_1, we determine the subset of fields from *D* requiring data cleaning to satisfy U_1 consists of:

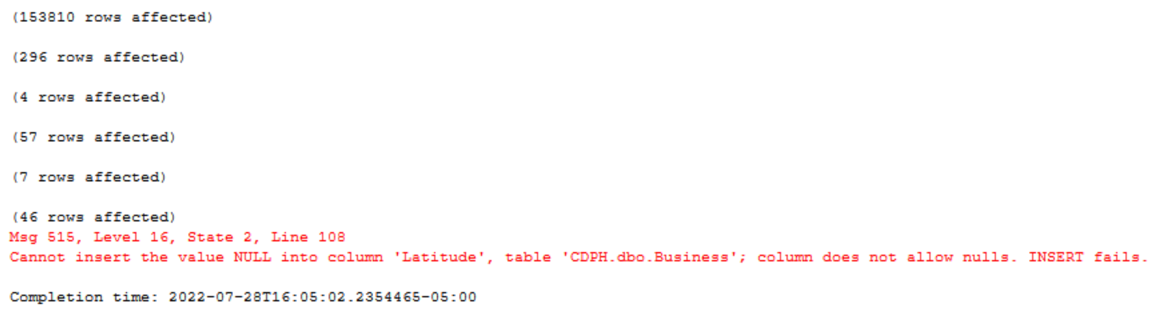
- Inspection ID;
- DBA Name;
- AKA Name;
- License #;
- Address;
- City;
- State;
- Zip;
- Inspection Date;
- Results;
- Violations;
- Latitude;
- Longitude;
- Location.

Recalling the discussion of *W* from Section 1, we see that these listed fields are the ones included in the data cleaning. Each of these fields must be cleaned to the point that they satisfy U_1. Three exceptions are the *Inspection ID*, *Inspection Date*, and *Results* fields which contain no anomalies and therefore do not require cleaning.

3. Demonstration of Data Quality Improvement

Data quality was greatly improved by our cleaning efforts. First, we discuss the integrity constraints enforced upon our clean dataset. Then, we provide a demonstration by executing several pairs of queries, where each pair consists of a query against the clean dataset, $Q_U(D')$, and its equivalent query against the dirty dataset, $Q_U(D)$. $Q_U(D')$ is available in the file “Use Case U1.sql”. $Q_U(D)$ is available in the file “Use Case U1 Dirty Data.sql”.

Stage 3, the constraint-enforced schema normalization stage discussed in Section 1.3, automatically tests for integrity constraint violations (ICVs) as a side effect of the schema instance loading process. In the schema, every table is defined with one or more constraints. As the staging table is loaded into the schema, if any single data element violates an integrity constraint, the transaction is rolled back and an error indicating the nature of the ICV is raised to the user. For example, if a user attempts to load the schema with a partially cleaned version of the dataset where the geographic location fields have not been populated, the database engine will abort and rollback the transaction. Consider the screenshot below.



```
(153810 rows affected)

(296 rows affected)

(4 rows affected)

(57 rows affected)

(7 rows affected)

(46 rows affected)
Msg 515, Level 16, State 2, Line 108
Cannot insert the value NULL into column 'Latitude', table 'CDPH.dbo.Business'; column does not allow nulls. INSERT fails.

Completion time: 2022-07-28T16:05:02.2354465-05:00
```

Such an implementation is preferred to manually executed ICV check scripts as it can be known with certainty that any data entering the database satisfies all integrity constraints; it is not possible to forget a cleaning step or make typos after executing ICV check scripts.

The integrity constraints enforced on each field of the dataset by the schema are:

- *Inspection ID* must be *non*-NULL and unique in the Inspection table (this field requires no cleaning in the context of U_1);
- *DBA Name* must be *non*-NULL;
- *AKA Name* must be unique in association with a record in the Business table;
- *License #* must be *non*-NULL;
- *Facility Type* must be unique in the FacilityType table (this field requires no cleaning in the context of U_1);
- *Risk* must be unique in Risk table (this field requires no cleaning in the context of U_1);
- *Address* must be *non*-NULL;
- *City* must be *non*-NULL;
- *State* must be *non*-NULL;
- *Zip* must be *non*-NULL;
- *Inspection Date* must be *non*-NULL (this field requires no cleaning in the context of U_1);
- *Inspection Type* must be unique in the InspectionType table (this field requires no cleaning in the context of U_1);
- *Results* must be *non*-NULL and unique in the Risk table (this field requires no cleaning in the context of U_1);
- *Violations* must be in a parsable structure containing a violation ID and name, both unique in the Violation table;
- *Latitude* must be *non*-NULL;
- *Longitude* must be *non*-NULL;
- *Location* must be *non*-NULL.

For the purpose of out-of-process validation, we provide the SQL script “queries.txt” which executes integrity constraint violation checks on dataset loaded into the staging table. The expected output of the script is also provided as “queries-expected-output.txt.”

For reference in our query demonstrations and discussion to follow, we include the contents of $Q_U(D')$ and $Q_U(D)$ below. First, consider $Q_U(D')$:

```sql
USE CDPH;

-- Usage: Set the parameter values below to filter results as desired.
--
-- Two parameters, Results (@results) and Violations (@violations), are
-- multi-valued parameters. Multi-valued parameters are provided as a string of
-- one or more candidate values delimited by pipes "|".
--
-- Each parameter may individually be set to NULL so that the query does not
-- include that parameter among its filtering criteria.

DECLARE @dateBegin DATE = '2012-01-01';  -- Demonstration value: From inspections performed on January
01, 2012...
DECLARE @dateEnd DATE = '2017-01-01';    -- Demonstration value: ...Until inspections performed on
January 01, 2017
DECLARE @businessName NVARCHAR(250) = N'MCDONALD''S';    -- Demonstration value: All businesses whose
name (or alias) contains "MCDONALD'S"
DECLARE @license INT = NULL      -- Demonstration value: NULL; not populated by user - no filter on
license
DECLARE @results NVARCHAR(MAX) = N'Pass|Pass w/ Conditions'      -- Demonstration value: "Pass" or
"Pass w/ Conditions" - a passing result
DECLARE @violations NVARCHAR(MAX) = N'1|2|3|5|8|9|11|12|22|33';  -- Demonstration value: An arbitrary
selection of multiple codes

-- There are no user-modifiable parameters beyond this point

SELECT Inspection.InspectionID
        ,CASE WHEN ((BusinessAlias.[Name] IS NOT NULL) AND (Business.[Name] <> BusinessAlias.[Name]))
                THEN CONCAT(Business.[Name], N' (', BusinessAlias.[Name], N')')
                ELSE Business.[Name] END AS BusinessName
        ,Business.License
        ,Business.[Address]
        ,Business.City
        ,Business.[State]
        ,Business.Zip
        ,Business.Latitude
        ,Business.Longitude
        ,Inspection.DatePerformed
        ,Result.[Name] AS Result
        ,Violation.ViolationID
        ,Violation.[Name]
        ,InspectionViolation.Comments
FROM Violation
        INNER JOIN InspectionViolation
                INNER JOIN Inspection
                        INNER JOIN Business ON Inspection.BusinessID = Business.BusinessID
                        LEFT JOIN BusinessAlias ON Inspection.BusinessAliasID =
BusinessAlias.BusinessAliasID
                        INNER JOIN Result ON Inspection.ResultID = Result.ResultID
                        ON InspectionViolation.InspectionID = Inspection.InspectionID
                ON Violation.ViolationID = InspectionViolation.ViolationID
WHERE (Inspection.DatePerformed >= @dateBegin OR @dateBegin IS NULL)
        AND (Inspection.DatePerformed < @dateEnd OR @dateEnd IS NULL)
        AND
        (
                (Business.[Name] LIKE CONCAT(N'%', @businessName, N'%'))
                OR
                (BusinessAlias.[Name] LIKE CONCAT(N'%', @businessName, N'%'))
        )
        AND (Business.License = @license OR @license IS NULL)
        AND ((Result.[Name] IN (SELECT * FROM STRING_SPLIT(@results, '|'))) OR @results IS NULL)
        AND ((Violation.ViolationID IN (SELECT * FROM STRING_SPLIT(@violations, '|'))) OR @violations
IS NULL)
ORDER BY Inspection.InspectionID
        ,Violation.ViolationID
        ,InspectionViolation.InspectionViolationID;
```

Now, consider $Q_U(D)$:

```
USE CDPH;

-- Usage: Set the parameter values below to filter results as desired.
--
-- Two parameters, Results (@results) and Violations (@violations), are
-- multi-valued parameters. Multi-valued parameters are provided as a string of
-- one or more candidate values delimited by pipes "|".
--
-- Each parameter may individually be set to NULL so that the query does not
-- include that parameter among its filtering criteria.

DECLARE @dateBegin DATE = '2012-01-01';  -- Demonstration value: From inspections performed on January
01, 2012...
DECLARE @dateEnd DATE = '2017-01-01';    -- Demonstration value: ...Until inspections performed on
January 01, 2017
DECLARE @businessName NVARCHAR(250) = N'MCDONALD''S';    -- Demonstration value: All businesses whose
name (or alias) contains "MCDONALD'S"
DECLARE @license INT = NULL      -- Demonstration value: NULL; not populated by user - no filter on
license
DECLARE @results NVARCHAR(MAX) = N'Pass|Pass w/ Conditions'      -- Demonstration value: "Pass" or
"Pass w/ Conditions" - a passing result
DECLARE @violations NVARCHAR(MAX) = N'1|2|3|5|8|9|11|12|22|33';   -- Demonstration value: An arbitrary
selection of multiple codes

-- There are no user-modifiable parameters beyond this point

WITH DataRaw_InspectionViolation AS
(
        SELECT Inspection_ID
                ,ViolationID
                ,ViolationText
                ,CASE WHEN LEN(ViolationComment) > 0 THEN ViolationComment ELSE NULL END AS
ViolationComment
        FROM
        (
                SELECT Inspection_ID
                        ,CAST(SUBSTRING(RawViolationText, 0, ViolationPosition) AS INT) AS ViolationID
                        ,SUBSTRING(RawViolationText, (ViolationPosition + 2), (ViolationCommentPosition
- (ViolationPosition + 2))) AS ViolationText
                        ,TRIM(SUBSTRING(RawViolationText, ViolationCommentPosition + 12,
LEN(RawViolationText))) AS ViolationComment
                        ,RawViolationText
                FROM
                (
                        SELECT Inspection_ID
                                ,PATINDEX('% - Comments:%', RawViolationText) AS
ViolationCommentPosition
```

```
                                ,PATINDEX('%.%', RawViolationText) AS ViolationPosition
                                ,RawViolationText
                        FROM
                        (
                                SELECT Inspection_ID
                                        ,TRIM(value) AS RawViolationText
                                FROM DataRaw
                                        CROSS APPLY STRING_SPLIT(Violations, '|')
                        ) AS RawViolation
                ) AS InspectedRawViolation
        ) AS ExtractedViolation
)

SELECT DataRaw.Inspection_ID AS InspectionID
        ,CASE WHEN ((DataRaw.AKA_Name IS NOT NULL) AND (DataRaw.DBA_Name <> DataRaw.AKA_Name))
                THEN CONCAT(DataRaw.DBA_Name, N' (', DataRaw.AKA_Name, N')')
                ELSE DataRaw.DBA_Name END AS BusinessName
        ,DataRaw.License
        ,DataRaw.[Address]
        ,DataRaw.City
        ,DataRaw.[State]
        ,DataRaw.Zip
        ,DataRaw.Latitude
        ,DataRaw.Longitude
        ,DataRaw.Inspection_Date AS DatePerformed
        ,DataRaw.Results AS Result
        ,DataRaw_InspectionViolation.ViolationID
        ,DataRaw_InspectionViolation.ViolationText
        ,DataRaw_InspectionViolation.ViolationComment
FROM DataRaw
        INNER JOIN DataRaw_InspectionViolation ON DataRaw.Inspection_ID =
DataRaw_InspectionViolation.Inspection_ID
WHERE (DataRaw.Inspection_Date >= @dateBegin OR @dateBegin IS NULL)
        AND (DataRaw.Inspection_Date < @dateEnd OR @dateEnd IS NULL)
        AND
        (
                (DataRaw.DBA_Name COLLATE Latin1_General_CS_AS LIKE CONCAT(N'%', @businessName, N'%'))
                OR
                (DataRaw.AKA_Name COLLATE Latin1_General_CS_AS LIKE CONCAT(N'%', @businessName, N'%'))
        )
        AND (DataRaw.License = @license OR @license IS NULL)
        AND ((DataRaw.Results IN (SELECT * FROM STRING_SPLIT(@results, '|'))) OR @results IS NULL)
        AND ((DataRaw_InspectionViolation.ViolationID IN (SELECT * FROM STRING_SPLIT(@violations,
'|'))) OR @violations IS NULL)
ORDER BY DataRaw.Inspection_ID
        ,DataRaw_InspectionViolation.ViolationID;
```

Even before performing either query a noticeable improvement can be seen, since executing the same query on *D* requires 60% more lines of code than *D'*.

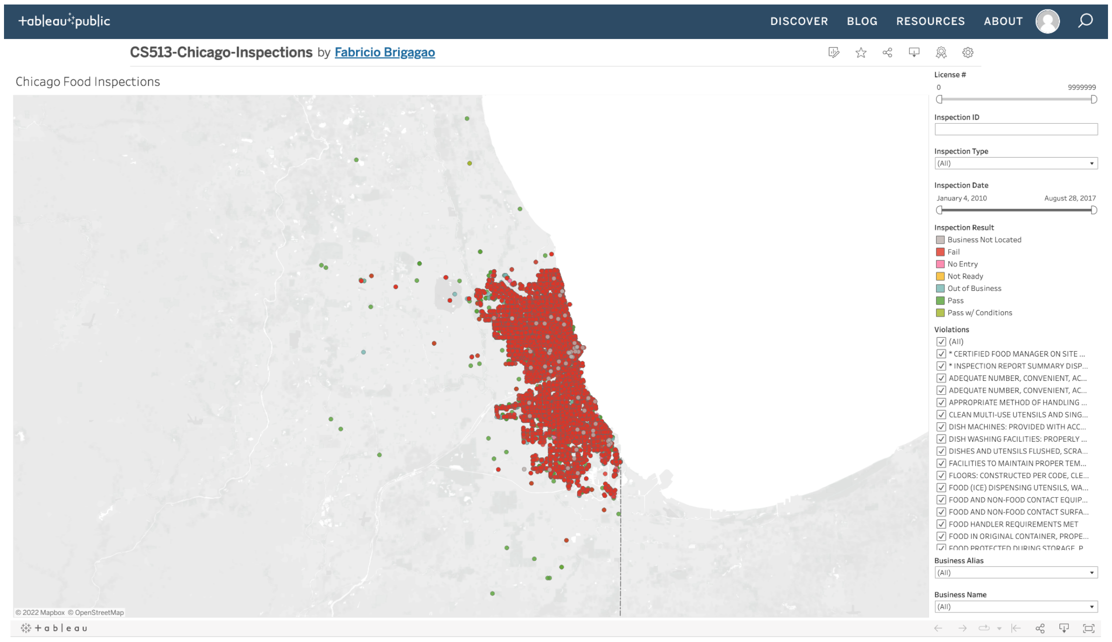
Now, consider the results of the queries when executed with the demonstration parameters provided. The query against *D'* returns 531 records. The query against *D* returns only 253 records. We see that the same query against *D* returns less than half of the records of *D'*. In this specific query, the culprit is the business name parameter. In *D,* we see many variations of “MCDONALD’S”, including: “McDonald’s”, “MC DONALD’S”, and “MCDONALD ‘ S”. In *D'*, these have been cleaned to “MCDONALD’S”, and a complete result set is returned.

The results set for $Q_U(D')$ is available in “mcdonalds-clean.csv”. The results set for $Q_U(D)$ is available in “mcdonalds-dirty.csv”.

Consider another example. Update the business name query parameter in both queries to “O’HARE”, such that we are now querying all business names containing that term. Both queries return 71 records, which may initially seem like a better showing for *D*. However, upon inspection, we can see that some of the records returned from *D* contain NULL-populated location fields. Even though all the desired records were retrieved from *D*, the data contained within them is incomplete. This would be an issue for a mapping application which plots records’ locations on a geographic map of the region. The results set for $Q_U(D')$ is available in “ohare-clean.csv”. The results set for $Q_U(D)$ is available in “ohare-dirty.csv”.

The previous two examples contained relatively tight filters restricting their result sets to a few hundred rows at most. At such a small scale, differences in time and space complexity are often dismissible. As a final example, consider an execution of both queries with all query parameters set to NULL – that is, apply no filters; return U_1’s specified subset of fields for the entire dataset. On a machine with an Intel Core i7-6850K 3.60 GHz CPU, 64 GB of RAM, and flushing the SQL Server clean page buffers prior to each execution, $Q_U(D)$ completed in 61.1 seconds of CPU time and loaded approximately 440 MB of data from disk into memory. $Q_U(D')$ completed in 5.9 seconds of CPU time and loaded approximately 205 MB of data from disk into memory. Clearly, *D’* is a drastic improvement to *D* in the case of U_1.

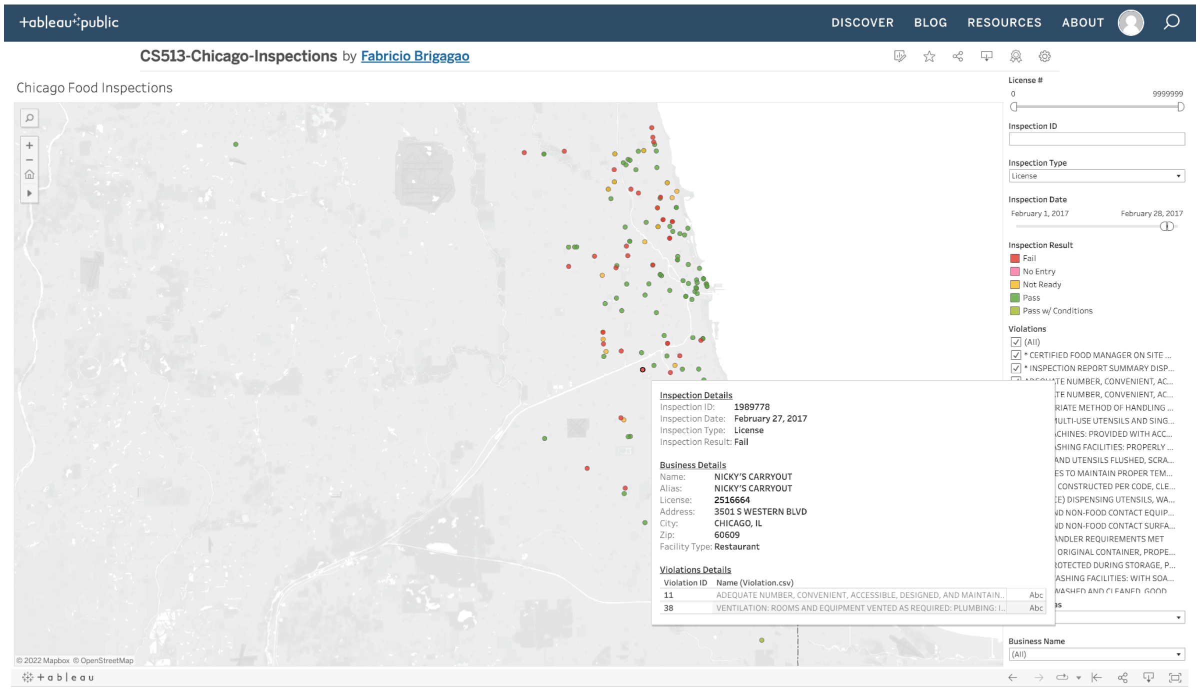
We were also able to use the resulting dataset to create a dashboard published to Tableau Public[1] [4] in accordance with U_1 that enables filtering of inspections based on criteria defined in U_1, such as by *Inspection Date*, *Results*, specific violations, and *Business Name*/Alias, among other fields.



+ableau public
DISCOVER
BLOG
RESOURCES
ABOUT
CS513-Chicago-Inspections by Fabricio Brigagao
Chicago Food Inspections
License #
0
9999999
Inspection ID
Inspection Type
(All)
Inspection Date
January 4, 2010
August 28, 2017
Inspection Result
Business Not Located
Fail
No Entry
Not Ready
Out of Business
Pass
Pass w/ Conditions
Violations
(All)
* CERTIFIED FOOD MANAGER ON SITE ...
* INSPECTION REPORT SUMMARY DISP...
ADEQUATE NUMBER, CONVENIENT, AC...
ADEQUATE NUMBER, CONVENIENT, AC...
APPROPRIATE METHOD OF HANDLING ...
CLEAN MULTI-USE UTENSILS AND SING...
DISH MACHINES: PROVIDED WITH ACC...
DISH WASHING FACILITIES: PROPERLY ...
DISHES AND UTENSILS FLUSHED, SCRA...
FACILITIES TO MAINTAIN PROPER TEM...
FLOORS: CONSTRUCTED PER CODE, CLE...
FOOD (ICE) DISPENSING UTENSILS, WA...
FOOD AND NON-FOOD CONTACT EQUIP...
FOOD AND NON-FOOD CONTACT SURFA...
FOOD HANDLER REQUIREMENTS MET
FOOD IN ORIGINAL CONTAINER, PROPE...
Business Alias
(All)
Business Name
(All)
© 2022 Mapbox © OpenStreetMap
+ a b l e a u

[1] https://public.tableau.com/app/profile/fabricio.brigagao/viz/CS513-Chicago-Inspections/ChicagoInspections

For instance, the image below features a query filtering inspections of the type "License" conducted from February 1st to 28th, 2017. The inspections are color-coded by result and upon hovering over each data point the viewer gets a summary of the inspection, with details about the business and corresponding code violations. Therefore, the creation of this dashboard further proves that the data cleaning steps performed were sufficient to satisfy use case U_1.



License #: 0 – 9999999

Inspection ID:

Inspection Type: License

Inspection Date: February 1, 2017 – February 28, 2017

Inspection Result
- 🟥 Fail
- 🟥 No Entry
- 🟨 Not Ready
- 🟩 Pass
- 🟩 Pass w/ Conditions

Violations
- ✓ (All)
- ✓ * CERTIFIED FOOD MANAGER ON SITE ...
- ✓ * INSPECTION REPORT SUMMARY DISP...
- ADEQUATE NUMBER, CONVENIENT, AC...
- TE NUMBER, CONVENIENT, AC...
- RIATE METHOD OF HANDLING ...
- MULTI-USE UTENSILS AND SING...
- CHINES: PROVIDED WITH ACC...
- SHING FACILITIES: PROPERLY ...
- AND UTENSILS FLUSHED, SCRA...
- ES TO MAINTAIN PROPER TEM...
- CONSTRUCTED PER CODE, CLE...
- E) DISPENSING UTENSILS, WA...
- D NON-FOOD CONTACT EQUIP...
- D NON-FOOD CONTACT SURFA...
- ANDLER REQUIREMENTS MET
- ORIGINAL CONTAINER, PROPE...
- ROTECTED DURING STORAGE, P...
- ASHING FACILITIES: WITH SOA...
- WASHED AND CLEANED, GOOD

Business Name: (All)

Inspection Details
Inspection ID: 1989778
Inspection Date: February 27, 2017
Inspection Type: License
Inspection Result: Fail

Business Details
Name: NICKY'S CARRYOUT
Alias: NICKY'S CARRYOUT
License: 2516664
Address: 3501 S WESTERN BLVD
City: CHICAGO, IL
Zip: 60609
Facility Type: Restaurant

Violations Details

Violation ID	Name (Violation.csv)	
11	ADEQUATE NUMBER, CONVENIENT, ACCESSIBLE, DESIGNED, AND MAINTAIN..	Abc
38	VENTILATION: ROOMS AND EQUIPMENT VENTED AS REQUIRED: PLUMBING: I..	Abc

4. Data Changes

Many changes to *D* were made during the execution of *W* to generate *D'*. We will summarize the data changes ΔD that were made both at the field level and the schema level. The number of changes stated throughout this section were gathered from both the OpenRefine operations summary pane and by running the SQL script named "Delta D Analysis.sql".

4.1. Field Level Changes

In this section, we list the sequence of operations applied to each field and the number of field cells modified by each operation. At the end of each sequence, we list the total number of unique field cells modified during *W*. Note that a single cell often participates in multiple operations (or even multiple times within the same operation), so *the total number of unique field cells modified is typically not a summation of the field cell count of the preceding operations.*

4.1.1. DBA Name

In the *DBA Name* field, all leading and trailing spaces were trimmed if any existed, consecutive whitespace collapsed, text was converted to uppercase and, lastly, distinct values were merged after being determined to be duplicates with inconsistencies.

Table 1
DBA Name field cleaning operations

Cleaning Operations	Affected Records
Remove leading and trailing spaces	0 (zero)
Collapse consecutive whitespaces	3,222
Convert text to uppercase	10,443
Merge values determined to be duplicates with inconsistencies	23,918
Total number of unique DBA Name cells modified	**17,977**

4.1.2. AKA Name

The same cleaning operations performed for the *DBA Name* field were performed for the *AKA Name* field. The results of these operations are listed in Table 2.

Table 2
AKA Name field cleaning operations

Cleaning Operations	Affected Records
Remove leading and trailing spaces	0 (zero)
Collapse consecutive whitespaces	3,560
Convert text to uppercase	9,585
Merge values determined to be duplicates with inconsistencies	15,964
Total number of unique AKA Name cells modified	**17,046**

4.1.3. License

In the *License #* field, few elements which had a NULL value were converted to the dataset's "default" license number value of "0", which itself was also rare with 439 elements.

Table 3
License # field cleaning operations

Cleaning Operations	Affected Records
Convert values from text to number	153,795
Convert NULL values to default license number value of "0"	15
Total number of unique License # cells modified	**153,810**

4.1.4. Address

The same operations performed for the *DBA Name* and *AKA Name* fields were also performed for the *Address* field. The results are listed on Table 4.

Table 4
Address field cleaning operations

Cleaning Operations	Affected Records
Remove leading and trailing spaces	153,366
Collapse consecutive whitespaces	618
Convert text to uppercase	9,488
Remove period (".") characters	231
Merge values determined to be duplicates with inconsistencies	188
Total number of unique Address cells modified	**153,467**

4.1.5. City

The same operations performed for the *DBA Name*, *AKA Name* and *Address* fields were also performed for the *City* field. The results are listed on Table 5.

Table 5
City field cleaning operations

Cleaning Operations	Affected Records
Remove leading and trailing spaces	0 (zero)
Collapse consecutive whitespaces	0 (zero)
Convert text to uppercase	348
Merge values determined to be duplicates with inconsistencies	244
Convert NULL values to the correct value by translating the Location	167

field into the city name using Google Maps	
Total number of unique City cells modified	**581**

4.1.6. State

In the *State* field, few elements which had filler whitespace values were converted to the known, static dataset-wide value of “IL”, as displayed in Table 6.

Table 6
State field cleaning operations

Cleaning Operations	Affected Records
Convert elements with filler whitespaces to "IL"	8
Total number of unique State cells modified	**8**

4.1.7. Zip

In the *Zip* field, elements which had NULL values were resolved by address or geographic coordinate lookup as described in Section 1.2.3.

Table 7
Zip field stage 1 cleaning operations

Cleaning Operations	Affected Records
Merge inconsistent values to the correct value by translating the Location field into the Zip code using Google Maps	4
Resolve NULL values by using address or geographic coordinates	98
Total number of unique Zip cells modified	**102**

4.1.8. Latitude, Longitude, Location

In the *Latitude*, *Longitude*, and *Location* fields, elements which had NULL values were resolved by translating the record’s associated address into geographic coordinates.

Table 8
Latitude, Longitude, and Location fields cleaning operations

Cleaning Operations	Affected Records
Resolve NULL values by translating associated address to geographic	544

coordinates	
Total number of unique Latitude, Longitude, Location cells modified	**544**

4.1.8. Facility Type, Inspection Type

As discussed previously, the *Facility Type* and *Inspection Type* fields are not necessary to satisfy U_1. However, knowing the type of business and the type of inspection performed may be valuable information to users in a future expansion of U_1. Because cleaning these fields was not time consuming - taking less than half an hour of effort - we decided to clean them as a supplementary exercise.

Table 9
Facility Type field cleaning operations

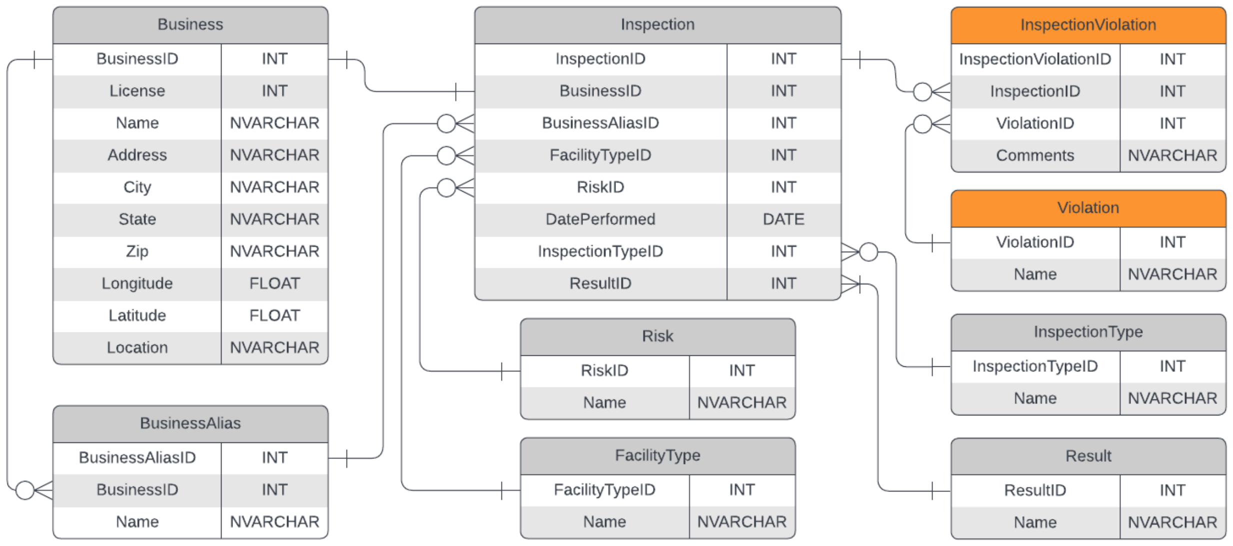
Cleaning Operations	Affected Records
Remove leading and trailing spaces	0 (zero)
Collapse consecutive whitespaces	13
Convert text to title case	4,984
Merge distinct values determined to be duplicates with inconsistencies	5,972
Total number of unique Facility Type cells modified	**4,849**

Table 10
Inspection Type field cleaning operations

Cleaning Operations	Affected Records
Remove leading and trailing spaces	0 (zero)
Collapse consecutive whitespaces	0 (zero)
Merge values determined to be duplicates with inconsistencies	1,520
Total number of unique Inspection Type cells modified	**1,520**

4.2 Schema Level Changes

Perhaps the most noticeable difference between *D* and *D'* is the physical arrangement of the data after *W*'s normalization step. This step loaded the modified dataset from existing as a single flat file of 17 fields and 153,810 records into a normalized, constraint-enforced database schema as shown in the ERD below.



The *Violations* field from the original dataset underwent its respective operations during this step. Field level cleaning tasks as performed on the fields in Section 4.1 were unnecessary for the *Violations* field, which was already syntactically and semantically sound.

However, this field violated the first normal form, as its elements' values were non-atomic. Therefore, during the normalization phase, the *Violations* field was normalized across two tables:

- a Violation table: containing the set of city-defined violations;
- a InspectionViolation table: containing mappings of each inspection to zero or more violations with associated comments for each instance of a violation.

After these optimizations, it was found that the dataset contains 568,654 recorded violations across all 153,810 inspection attempts.

5. Findings, Problems, Lessons Learned, and Next Steps

5.1. Findings and Problems

The primary problem discovered during this data cleaning project was the low quality of data involved. This dataset contained what appeared to be many fields into which hand-typed data was entered. Misspellings, inconsistencies, duplications, missing values, and apparently incorrect values abound throughout this dataset. It seems that there are few - if any - quality control measures in place at the point of data entry. To make matters worse, no provenance is available for the dataset. The input data cannot be validated or corrected in many cases as the truth and accuracy of recorded values are not provided and, in many cases, are undiscoverable.

A system of identifying suspected misspellings, duplications, etc. should be implemented at the point of data entry to identify these errors and allow the data entry clerk the opportunity to review and accept the system's suggestions. There isn't a way to validate or reconstruct suspected incorrect data with certainty after it has been entered into the dataset, especially if the original inspection forms are unavailable for review.

5.2. Lessons Learned

The effort required by this project underscored the importance of quality data collection and data entry processes. As previously discussed, the dataset was of a relatively low quality. Furthermore, it could be argued that much of the work that was required would have been unnecessary if these errors were identified and corrected at the point of data entry rather than allowed into the dataset. Providing a rich provenance to downstream users of a dataset is paramount so that the data can be verified and used with a high degree of confidence.

In addition, we discovered that OpenRefine was limited to resource allocations allowed to the browser. As a result, during large operations, such as clustered merges, the browser, either Google Chrome or Mozilla Firefox, would frequently crash and throw out-of-memory errors. To circumvent this, merging tasks were partitioned into subsets, around 50 merges at a time. After reducing the size of merges, there were no further issues. For larger datasets, a non-browser-based software solution may be preferable if large operations are to be performed.

5.3. Next Steps

Next steps of this project may involve refining our initial Tableau dashboard to provide easy access to additional information, such as the history of inspections for the same business.

In addition, we believe that the data cleaning workflow can be partially automated using tools such as Apache Airflow[5], that can extract new datasets, clean them and load them to a public database that, finally, could be used by a visual dashboard to display up to date information about inspections. This tool also provides provenance, since every executed action on the dataset is tracked and logged, allowing a post execution review (retrospective provenance). Moreover, the plugins available can enable the execution of additional tools, such as YesWorkflow, via the command line in order to document the developed scripts.

6. Team Members Contributions

Following the completion of the project, members' contributions were as follows:

Activities	**Member**
1. Perform an analysis of the field-level data of the dataset Tools: OpenRefine, SQL.	Steve Revision: Fabricio
2. Perform syntactic and semantic/integrity constraint corrections at column level related to U_1 Tools: OpenRefine, SQL, Python, Google Maps.	Steve Revision: Fabricio
3. Load output of OpenRefine into a staging table in SQL and correct schema-level integrity constraints Tools: OpenRefine, Microsoft SQL Server, SQL.	Steve
4. Load the staging table into the integrity constraint-enforced SQL schema Tools: Microsoft SQL Server, SQL.	Steve
5. Implement query(s) demonstrating resulting data set can successfully achieve use case U_1 Tools: Microsoft SQL Server, SQL.	Steve Revision: Fabricio
5.1 Integrate the resulting dataset with Tableau Public to create a dashboard visualization that exemplifies U_1. Tools: Tableau Desktop, Tableau Public.	Fabricio
6. Document changes to dataset and steps during data cleaning process (continuous process) Tools: Google Docs, Github, OpenRefine recipes, Tableau project, CSV, JSON, and SQL files.	Steve Revision: Fabricio
7. Develop illustrations and workflow diagrams Tool: YesWorkflow.	Fabricio
8. Write phase 2 project report Tools: Google Docs.	Steve Fabricio
9. Revision of phase 2 project report	Roberto

References

[1] Chicago Restaurant Inspections. (2017, August 30). Kaggle. Retrieved June 22, 2022, from https://www.kaggle.com/datasets/chicago/chi-restaurant-inspections

[2] Chicago Data Portal. (n.d.). Food Inspections. Retrieved June 28, 2022, from https://data.cityofchicago.org/api/assets/BAD5301B-681A-4202-9D25-51B2CAE672FF

[3] Maps, apis and components: Geoapify location platform. Geoapify. (2021, October 7). Retrieved July 13, 2022, from https://www.geoapify.com/

[4] CS513-Chicago-Inspections (2022, July 29). Tableau. Retrieved July 29,2022 from https://public.tableau.com/app/profile/fabricio.brigagao/viz/CS513-Chicago-Inspections/ChicagoInspections

[5] Documentation. (n.d.). Apache Airflow. Retrieved July 29, 2022, from https://airflow.apache.org/docs/