Relational Database Management System For Claryty

- BUS 895 Research Project in Business

San Francisco State University

Prof. Minu Kumar
Prof. Leyla Ozsen

Table of Contents

1. Executive Summary:

Claryty is looking to provide transparency between patients and doctors regarding the drugs that are prescribed. By aggregating multiple sources of risk factors, adverse events, and demographic data we inform the patient about the risks presented by a prescription drug.

The main purpose of this project is to build an effective database management system for 'Claryty' to fulfill all business requirements along with obtaining useful demographic and drug insights. Data for the project is obtained from the FDA FEARS website. The quarterly data files, which are available in ASCII or SGML formats, include:

- demographic and administrative information and the initial report image ID number (if available);
- drug information from the case reports;
- reaction information from the reports;
- patient outcome information from the reports;
- information on the source of the reports;

The main objectives of this project would be:

- Create a database management system having all the 7 entities of the FDA FEARS dataset.
- Check if Sql workbench is the right software(robust) to make the database.
- Build a script which extracts demographic and adverse event reports with respect to a specific drug robustly.
- The output of the information extracted should be 95% accurate.
- This dataset would be a small part of a huge data warehouse, and so add a RxCUI code to the database which links this database to the rest.
- One of the main business requirement is to return the extracted information in json output.

The data schema is prepared in MySQL workbench. The text files were first converted to individual CSV files and then uploaded to the MySQL Workbench. Data schema includes 7 entities with multiple attributes to capture reports, drug, demographics, symptoms, and reactions related information. For the scope of the project, we narrowed down our research to 1 year(4 quarters) worth data is used and analyzed.

This report also discusses the business rules, entity-relationship diagram to describe referential integrity between different entities.

2. Introduction:

In the transactions among doctors, patients, and pharmaceutical firms each party has different amounts of information regarding the product quality of the drug prescribed and the risk entailed in the transactions. When the Food and Drug Administration (FDA) allowed Direct To Consumer Advertising (DTCA) in 1998, these advertisements were intended to reduce the level of information.

Figure 2(a) shows the number of data points for the 3 pivotal tables made and analyzed for this project. As seen, the drug, demographic, and reaction table contains 1.7 mil, 0.5 mil, and 4.97 mil rows respectively. The demographic table has cases reported by 199 different countries, for 1,26,899 different drugs and contains information of 12,052,68 reports of adverse effects.

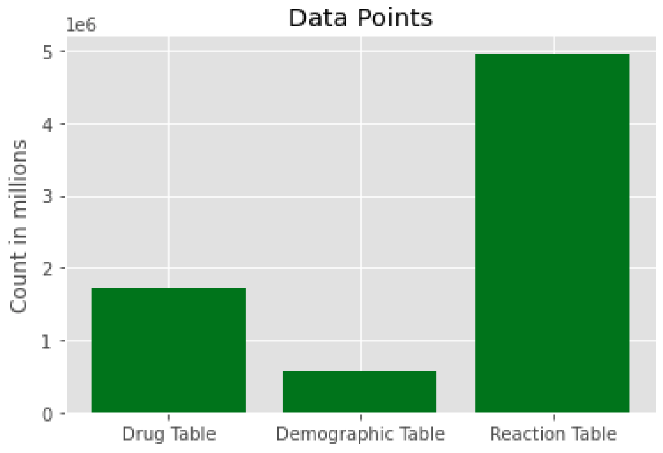


Figure 2(a)

We supply all prescription drug data with a risk score that is calculated with the adverse effect events/data that we receive. This risk score allows for any patient of any ethnicity to generalize the risk factor of a drug.

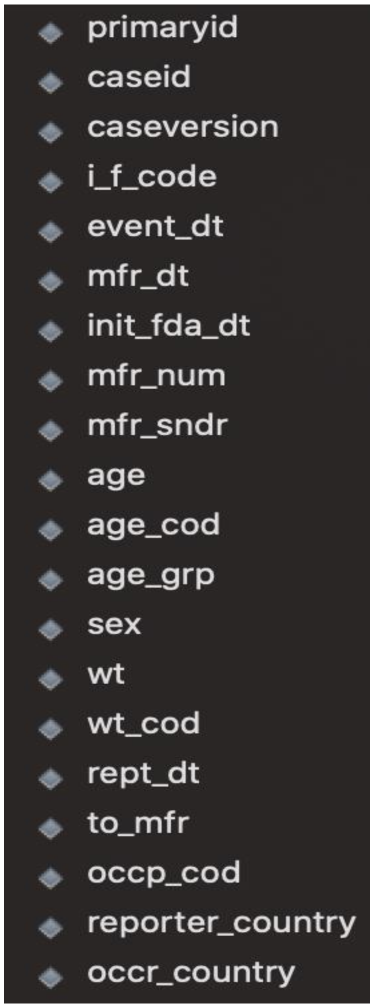
3. Use Case:

- Adam was recently diagnosed with depression and prescribed by his doctor with Abilify. He noticed an increase in suicidal ideation when taking Abilify and does not suspect the drug to be the cause due to the manner aspects of his life and that drugs are supposed to be helping his emotions. Adam uses Claryty to reduce asymmetries between what he and the drug manufacturer know about the risks of the product.

- Dr. Smith has been prescribing Metformin to his patients with diabetes throughout his whole career. He observes that a noticeable amount of them have been experiencing muscle pain/weakness. Dr. Smith mentions to his future patients that are prescribed with Metformin of the known adverse effects after viewing all the data on Claryty.
- Marie is getting ready for chemotherapy and has a list of drugs she has to take during her therapy. She wants to know the adverse effects that others have experienced in the past 2 years. She uses Claryty to search for her drugs for the last 2 years and finds similar adverse effects for each drug.

4. Entities and Attributes:

- Demographic Table



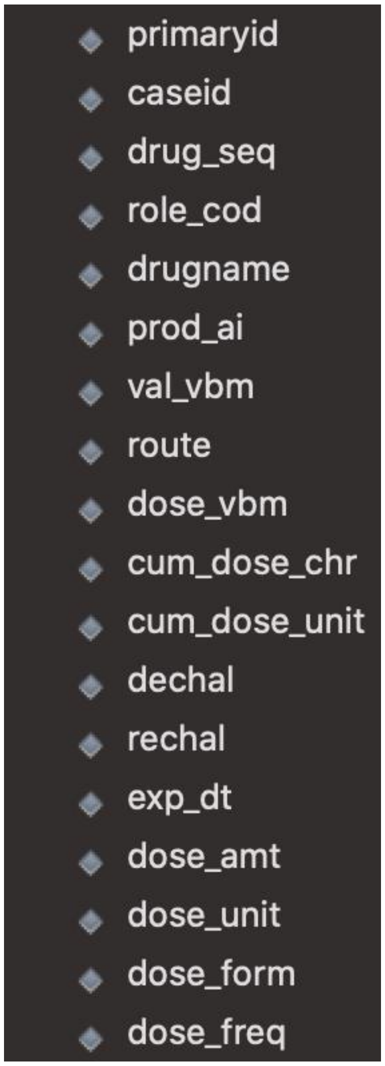
primaryid
caseid
caseversion
i_f_code
event_dt
mfr_dt
init_fda_dt
mfr_num
mfr_sndr
age
age_cod
age_grp
sex
wt
wt_cod
rept_dt
to_mfr
occp_cod
reporter_country
occr_country

Figure 4(a)

The demographic table contains information about demographic and administrative information. It records the patient's age, sex, and weight when the adverse effect occurred. It also has information about the country where the event occurred and the country where the event was reported. It also has some important information about the dates when the adverse event was reported to the FDA. Some columns like weight, age, and report are paired with weight code, age code, and report code respectively. The code columns represent the units in which the attributes are reported. An example of age and age_cod is shown below.

CODE	MEANING_TEXT
N	Neonate
I	Infant
C	Child
T	Adolescent
A	Adult
E	Elderly

- <u>Drug Table</u>



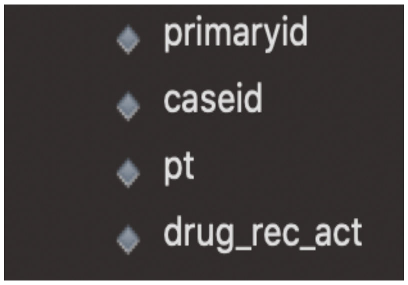
- primaryid
- caseid
- drug_seq
- role_cod
- drugname
- prod_ai
- val_vbm
- route
- dose_vbm
- cum_dose_chr
- cum_dose_unit
- dechal
- rechal
- exp_dt
- dose_amt
- dose_unit
- dose_form
- dose_freq

Drug entity contains all the information about the drugs reported for adverse effects.

drugname: Name of medicinal product.
prod_ai: Product Active Ingredient.
val_vbm: Code for source of drugname.
route: The route of drug administration.
dose_vbm: Verbatim text for dose, frequency, and route, exactly as entered on report.
cum_dose_chr: Cumulative dose to first reaction.
cum_dose_unit: Cumulative dose to first reaction unit.
dechal: Dechallenge code, indicating if reaction abated when drug therapy was stopped.
rechal: Rechallenge code, indicating if reaction recurred when drug therapy was restarted.
exp_dt: Expiration date of the drug.
dose_amt: Amount of drug reported.
dose_form: Form of dose reported.
dose_freq: Frequency of dose intake.
dose_unit: Unit of drug dose.

Figure 4(b)

- <u>Reaction Table</u>



- primaryid
- caseid
- pt
- drug_rec_act

pt: "Preferred Term"-level medical terminology describing the event, using the Medical Dictionary for Regulatory Activities (MedDRA).
drug_rec_act: Drug Recur Action data - populated with reaction/event information (PT) if/when the event reappears upon readministration of the drug.

Figure 4(c)

- Outcome Table

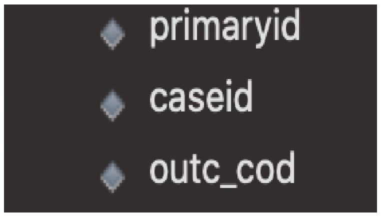

primaryid
caseid
outc_cod

Figure 4(d)

outc_cod: Code for patient outcome.

CODE	MEANING_TEXT
DE	Death
LT	Life-Threatening
HO	Hospitalization - Initial or Prolonged
DS	Disability
CA	Congenital Anomaly
RI	Required Intervention to Prevent Permanent Impairment/Damage
OT	Other Serious (Important Medical Event)

- Report Source Table

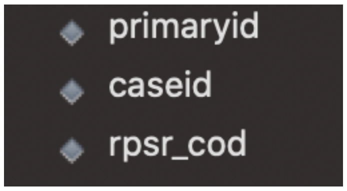

primaryid
caseid
rpsr_cod

Figure 4(e)

rspr_cod: Code for the source of the report.

CODE	MEANING_TEXT
FGN	Foreign
SDY	Study
LIT	Literature
CSM	Consumer
HP	Health Professional
UF	User Facility
CR	Company Representative
DT	Distributor
OTH	Other

- Therapy Dates Table

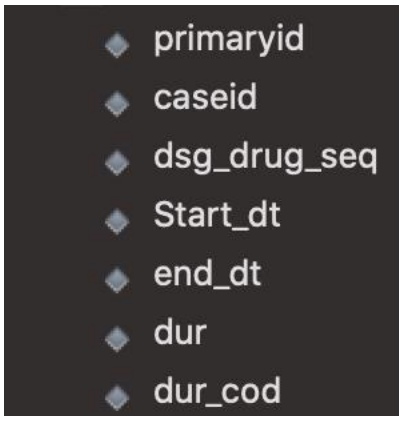

◆ primaryid
◆ caseid
◆ dsg_drug_seq
◆ Start_dt
◆ end_dt
◆ dur
◆ dur_cod

Figure 4(f)

dsg_drug_seq: Drug sequence number for identifying a drug for a Case.
start_date: Date the therapy was started (or re-started) for this drug (YYYYMMDD) – If a complete date not available, a partial date is provided.
end_dt: A date therapy was stopped for this drug. (YYYYMMDD).
dur: Numeric value of the duration of the therapy.
dur_cod: Unit abbreviation for duration of therapy.

- Indications Table

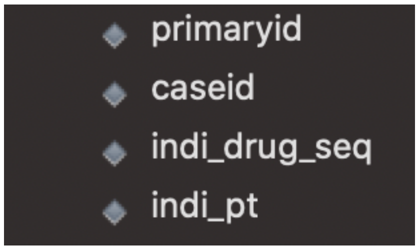
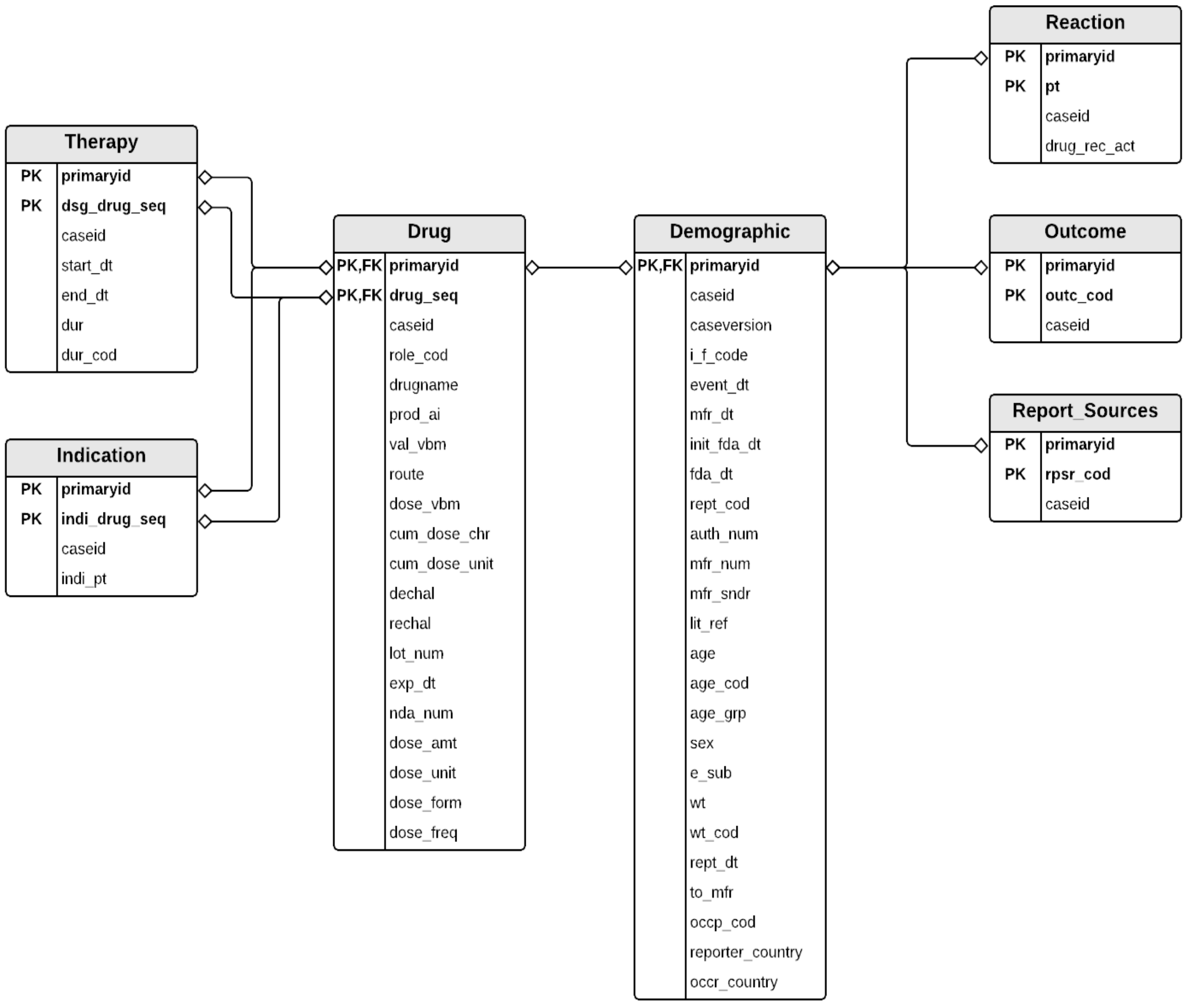

◆ primaryid
◆ caseid
◆ indi_drug_seq
◆ indi_pt

Figure 4(g)

indi_drug_seq: Drug sequence number for identifying a drug for a case.
indi_pt: "Preferred Term"-level medical terminology describing the Indication for use, using the Medical Dictionary for Regulatory Activities MedDRA).

5. Methodology:

5.1 ER Diagram and Business rules



ER DIAGRAM

1. For every report, there is one row in the "demographic" table (file). Each row in the demographic table can be linked to none, one, or more than one row in the "Reaction", "Outcome" and "Report_Sources" tables. Also for every one row in the "demographic" table, you can have one or more rows in the "Drug" table.

2. For every drug, you can have one or more rows in the "Therapy" table that shows when the drug was started and stopped. Also for every drug you can have none, one or more than one indication in the "Indication" table.

Preprocessing data

The research is based on an FDA Adverse Event Reporting System(FEARS). The data is available quarterly for every year, but for the scope of this project, we are working with 4 quarters, i.e.1 year. For analyzing the data and extract information, it is crucial to build a well-structured database. Prior to building a database, it is necessary to preprocess and understand the data. Especially, in such a huge dataset, in order to create productive queries and extract useful information, it is pivotal to understand the data.

As this is a dataset related to drugs and medicines, it is a very sensitive dataset to preprocess. The Cleaning process involves dropping the following columns from the demographic and drug tables, keeping in mind the goal of the project.

Demographic Table: fda_dt, rept_code, auth_num, lit_ref, e_sub

Drug Table: role_cod, lot_num, nda_num

5.2 Data Schema creation in MySQL workbench

Create Tables in Database

The first step in building a database is creating a data schema containing the various entities and attributes. As mentioned before, there are 7 entities (tables) used for database management of ‘Fears_Test’. To build the database, these entity tables should be created individually and the relationship between the tables has to be established as per the ERD using the primary keys and foreign keys.

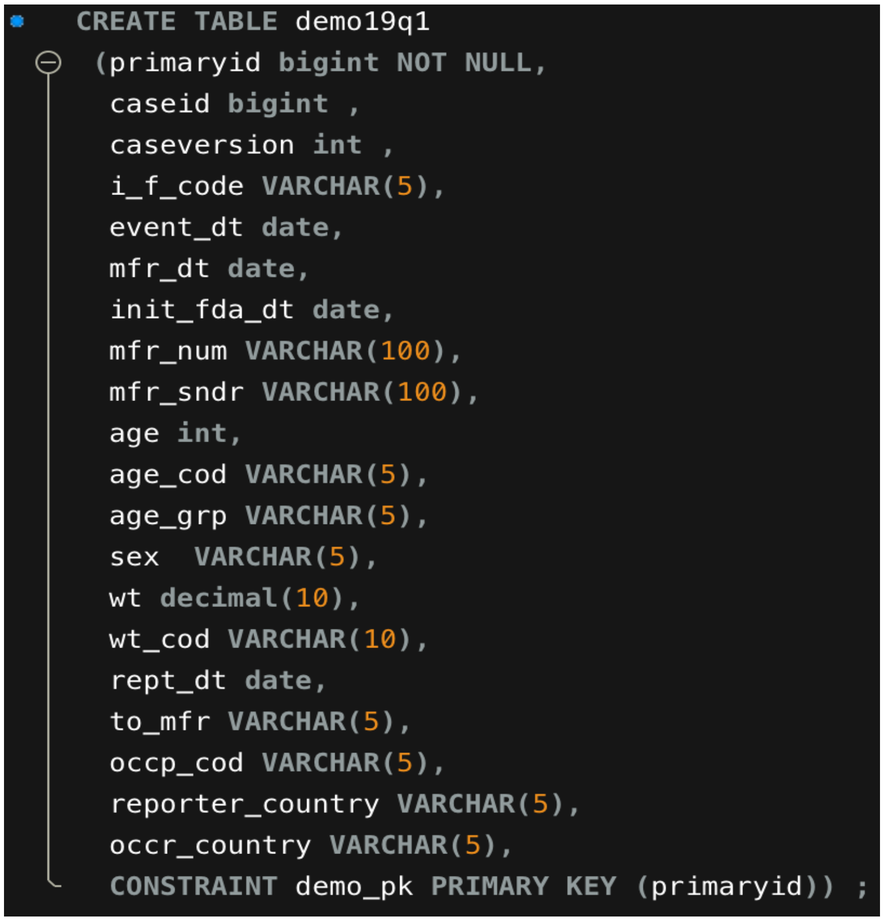
The ERD lays out the relationship rules between the tables. For example, the ‘demographic’ entity has a relationship with ‘report sources’, ‘outcome’ and ‘reaction’. Hence, it is essential to create the parent tables first and go about creating the tables with dependencies.

Keeping this in mind, independent entity tables such as ‘demo19q1’and ‘drug19q1’ are created first followed by the dependent tables such as ‘reac19q1’, ‘outc19q1’, ‘ther19q1’, ‘indi19q1’ and ‘rspr19q1’.

To build the tables, a space or a schema is required. Hence, the initial step is to build the schema namely ‘Fears_Test’, using the below create SQL command:

```
CREATE SCHEMA Fears_Test;
```

The entity tables can now be created in the schema using ‘Fears_Test’ where all the analysis will also take place. To create the tables using SQL commands, it is quintessential to mention the data types as well as the constraints for each attribute. Below is an example of the SQL query to create the independent entity table ‘demo19q1’.

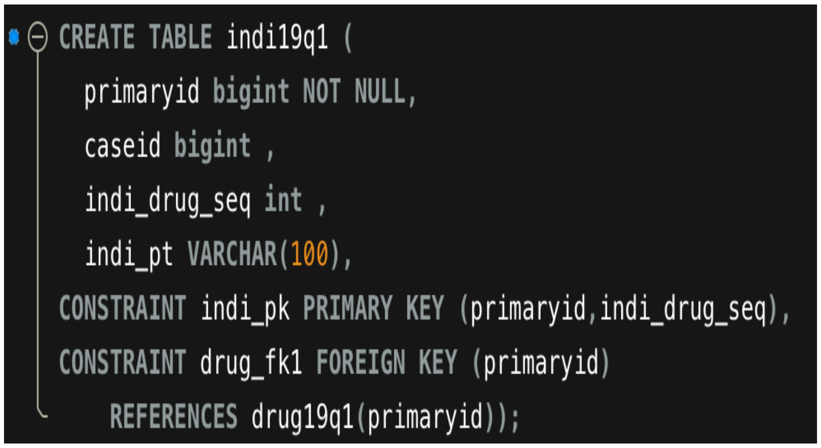


```
CREATE TABLE demo19q1
 (primaryid bigint NOT NULL,
  caseid bigint ,
  caseversion int ,
  i_f_code VARCHAR(5),
  event_dt date,
  mfr_dt date,
  init_fda_dt date,
  mfr_num VARCHAR(100),
  mfr_sndr VARCHAR(100),
  age int,
  age_cod VARCHAR(5),
  age_grp VARCHAR(5),
  sex  VARCHAR(5),
  wt decimal(10),
  wt_cod VARCHAR(10),
  rept_dt date,
  to_mfr VARCHAR(5),
  occp_cod VARCHAR(5),
  reporter_country VARCHAR(5),
  occr_country VARCHAR(5),
  CONSTRAINT demo_pk PRIMARY KEY (primaryid)) ;
```

Figure 5.2(a)

As seen in Section 2, ‘demo19q1’ entity has 20 attributes. Figure 5.2(a) shows how an independent entity table can be created. ‘primaryid’ is the primary key for ‘every report which is listed.

Similarly, a dependent entity table such as ‘indi19q1’ can be created using the SQL query as shown below:



```
CREATE TABLE indi19q1 (
  primaryid bigint NOT NULL,
  caseid bigint ,
  indi_drug_seq int ,
  indi_pt VARCHAR(100),
CONSTRAINT indi_pk PRIMARY KEY (primaryid,indi_drug_seq),
CONSTRAINT drug_fk1 FOREIGN KEY (primaryid)
   REFERENCES drug19q1(primaryid));
```

Figure 5.2(b)

Figure 5.2(b) shows the creation of the ‘indi19q1’ entity table along with defining all the attributes as discussed in Section 2. As seen in the ERD, there is a relationship established between ‘indi19q1’ and ‘drug19q1’. These relationships can be established by incorporating the foreign keys of the independent table in the constraints.

For the create queries of all the tables, Refer to Appendix A.

5.3 Load Data

Once the entity tables are created, data has to be loaded into each of these entity tables. While loading data, similar to creating the tables, the order in which the data is loaded is important; first the independent tables have to be populated followed by dependent tables. There are multiple ways to load data into the tables and for the purpose of this project, we have used the first method.

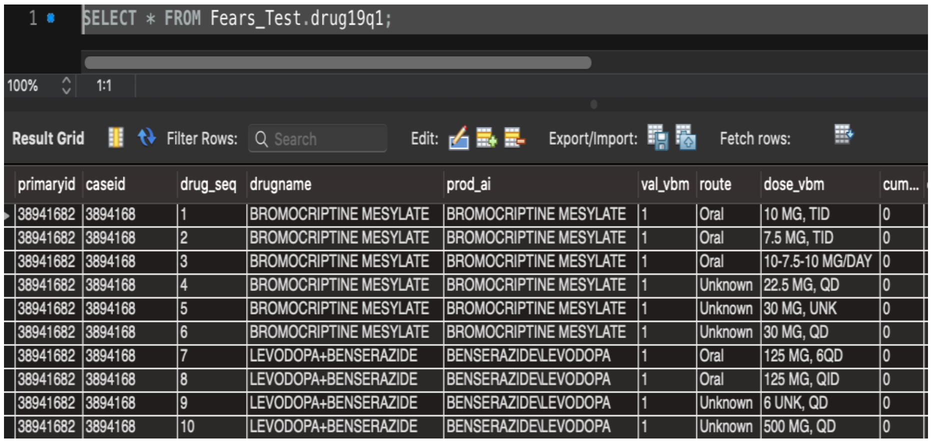
1. Table Data Import Wizard: Import CSV files containing data rows into the respective tables.
2. 'insert' SQL query: Insert data row by row into each of the columns.

To use the option of 'Table Import Wizard' in order to import CSV files, it is essential to maintain individual CSV files for each of the tables, and not in the form of multiple sheets in the same workbook. Another important point to keep in mind while importing the CSV files is to select the right column. There can be additional empty columns generated in the CSV which have to be unchecked. By following these steps (Appendix B), the entity tables can be populated.

On the contrary, 'insert' SQL queries can also be used to populate the entity tables row by row, but it is just not feasible for this project.

5.4 Retrieve Information from Tables

Now that the entity tables are created and loaded with data, meaningful insights in compliance with the business rules defined in Section 5.1 can be obtained. Using a simple select query as shown below in Figure 5.4(a) on each of the entity tables lists all the data available in the table. Refer to Appendix C for detailed data of each entity table.



```sql
SELECT * FROM Fears_Test.drug19q1;
```

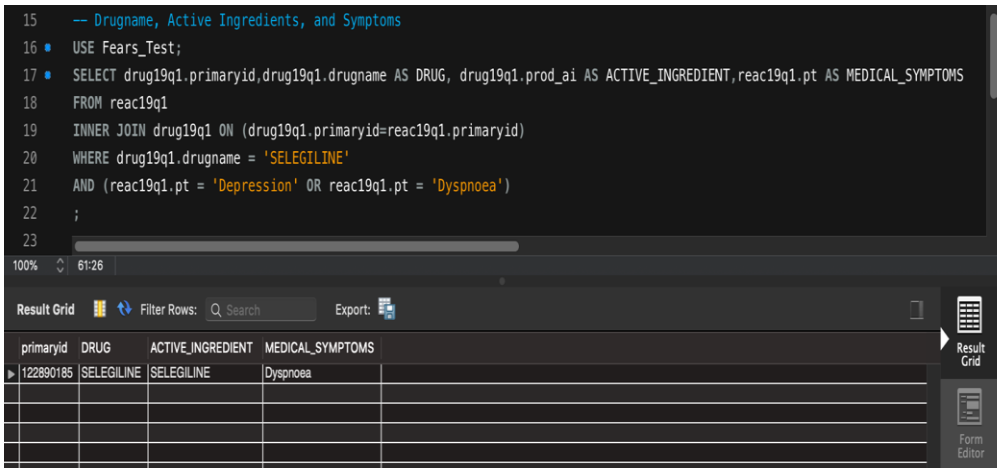
primaryid	caseid	drug_seq	drugname	prod_ai	val_vbm	route	dose_vbm	cum...
38941682	3894168	1	BROMOCRIPTINE MESYLATE	BROMOCRIPTINE MESYLATE	1	Oral	10 MG, TID	0
38941682	3894168	2	BROMOCRIPTINE MESYLATE	BROMOCRIPTINE MESYLATE	1	Oral	7.5 MG, TID	0
38941682	3894168	3	BROMOCRIPTINE MESYLATE	BROMOCRIPTINE MESYLATE	1	Oral	10-7.5-10 MG/DAY	0
38941682	3894168	4	BROMOCRIPTINE MESYLATE	BROMOCRIPTINE MESYLATE	1	Unknown	22.5 MG, QD	0
38941682	3894168	5	BROMOCRIPTINE MESYLATE	BROMOCRIPTINE MESYLATE	1	Unknown	30 MG, UNK	0
38941682	3894168	6	BROMOCRIPTINE MESYLATE	BROMOCRIPTINE MESYLATE	1	Unknown	30 MG, QD	0
38941682	3894168	7	LEVODOPA+BENSERAZIDE	BENSERAZIDE\LEVODOPA	1	Oral	125 MG, 6QD	0
38941682	3894168	8	LEVODOPA+BENSERAZIDE	BENSERAZIDE\LEVODOPA	1	Oral	125 MG, QID	0
38941682	3894168	9	LEVODOPA+BENSERAZIDE	BENSERAZIDE\LEVODOPA	1	Unknown	6 UNK, QD	0
38941682	3894168	10	LEVODOPA+BENSERAZIDE	BENSERAZIDE\LEVODOPA	1	Unknown	500 MG, QD	0

Figure 5.4(a)

A single entity by itself at times may not be sufficient to provide the whole picture. This calls for the need to join various entity tables using inner join, left join, unions, nested select queries, etc. For instance, business rule number 1, says that for every report in the demo table, there can be none, one, or multiple records in the reaction table.

From the data in the database developed, many questions can be answered that can be vital in discovering important patterns and information from the database.

- Additionally, by joining 'drug19q1 and 'reac19q1' entity tables, it is possible to retrieve information pertaining to a specific drug and their reactions. As shown in the figure 5.4(b), where drugname = 'SELEGLINE' and their symptoms as Depression or Dyspnoea, we single out a record from millions of data points. Similarly, we can filter the dataset on various drugs and their symptoms.



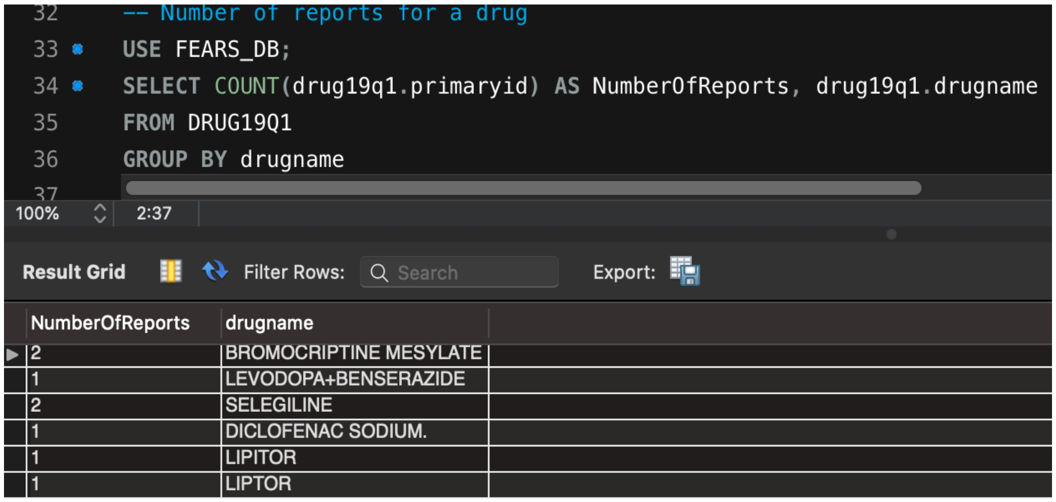
```sql
-- Drugname, Active Ingredients, and Symptoms
USE Fears_Test;
SELECT drug19q1.primaryid,drug19q1.drugname AS DRUG, drug19q1.prod_ai AS ACTIVE_INGREDIENT,reac19q1.pt AS MEDICAL_SYMPTOMS
FROM reac19q1
INNER JOIN drug19q1 ON (drug19q1.primaryid=reac19q1.primaryid)
WHERE drug19q1.drugname = 'SELEGILINE'
AND (reac19q1.pt = 'Depression' OR reac19q1.pt = 'Dyspnoea')
;
```

primaryid	DRUG	ACTIVE_INGREDIENT	MEDICAL_SYMPTOMS
122890185	SELEGILINE	SELEGILINE	Dyspnoea

Figure 5.4(b)

- For such a huge database, it is really important to test the integrity of the queries. For this purpose, the queries are tested on a dummy database and then subsequently applied to the larger dataset after satisfactory results.

- Number of reports generated for a particular drug is one of the pivotal pieces of information we can assess using the dataset. It can also help to identify the drugs with more number of reports. To find the number of reports generated, we add a group by query on drugname.



```
-- Number of reports for a drug
USE FEARS_DB;
SELECT COUNT(drug19q1.primaryid) AS NumberOfReports, drug19q1.drugname
FROM DRUG19Q1
GROUP BY drugname
```

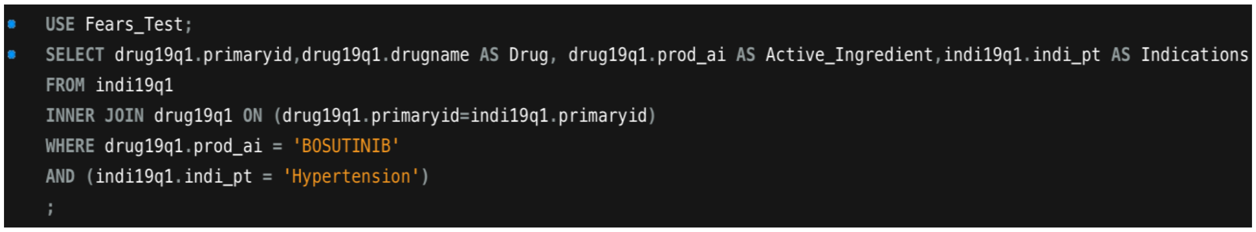
100% 2:37

Result Grid Filter Rows: Search Export:

NumberOfReports	drugname
2	BROMOCRIPTINE MESYLATE
1	LEVODOPA+BENSERAZIDE
2	SELEGILINE
1	DICLOFENAC SODIUM.
1	LIPITOR
1	LIPTOR

Figure 5.4(c)

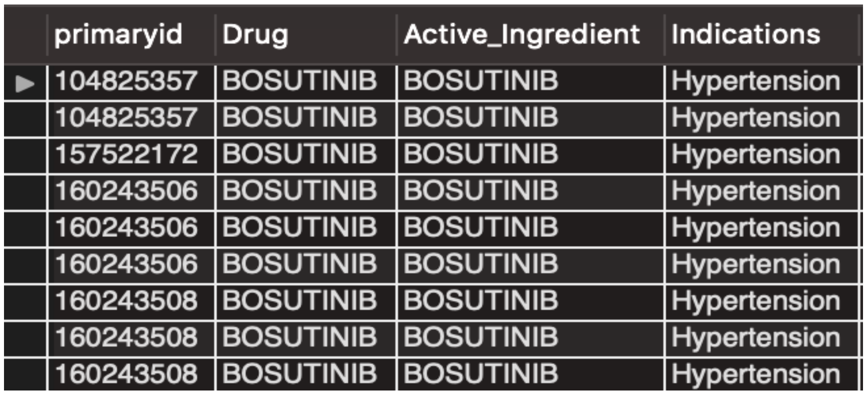
- We can also filter results based on Indications and their respective drug. Figure 5.4(d) performs a join on the drug and the indications table to match the Indications to their respective drug used.



```
USE Fears_Test;
SELECT drug19q1.primaryid,drug19q1.drugname AS Drug, drug19q1.prod_ai AS Active_Ingredient,indi19q1.indi_pt AS Indications
FROM indi19q1
INNER JOIN drug19q1 ON (drug19q1.primaryid=indi19q1.primaryid)
WHERE drug19q1.prod_ai = 'BOSUTINIB'
AND (indi19q1.indi_pt = 'Hypertension')
;
```

Figure 5.4(d)

- The query results returns all the data points where the active ingredient of a drug is BOSTNIB and the indication to use that drug is Hypertension. For runtime purposes, we have limited the output to 10 rows.



primaryid	Drug	Active_Ingredient	Indications
104825357	BOSUTINIB	BOSUTINIB	Hypertension
104825357	BOSUTINIB	BOSUTINIB	Hypertension
157522172	BOSUTINIB	BOSUTINIB	Hypertension
160243506	BOSUTINIB	BOSUTINIB	Hypertension
160243506	BOSUTINIB	BOSUTINIB	Hypertension
160243506	BOSUTINIB	BOSUTINIB	Hypertension
160243508	BOSUTINIB	BOSUTINIB	Hypertension
160243508	BOSUTINIB	BOSUTINIB	Hypertension
160243508	BOSUTINIB	BOSUTINIB	Hypertension

Figure 5.4(e)

Similarly, we can have a filter query for single and multiple drugs on single or multiple medical indications.

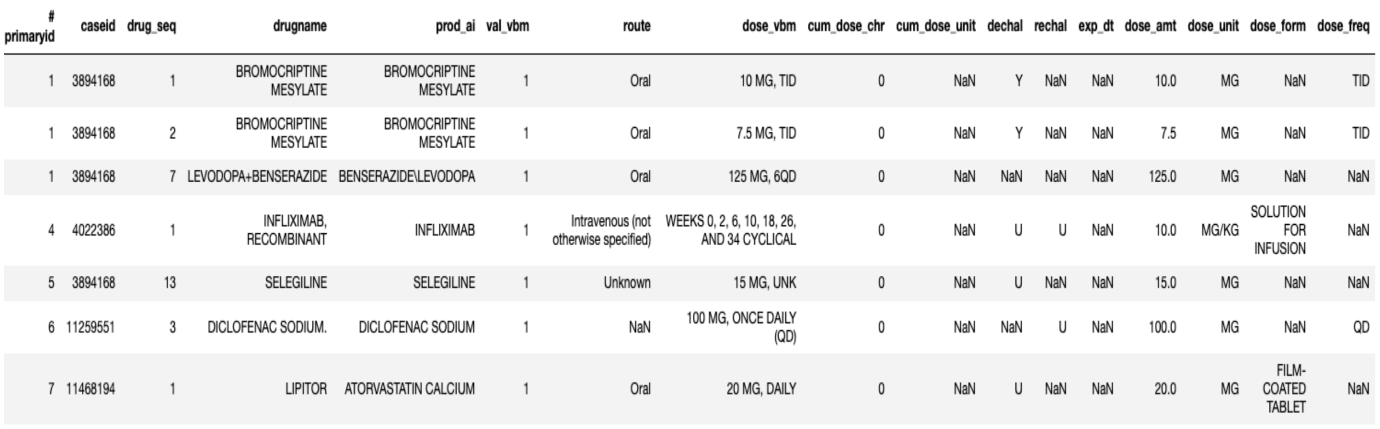
As we can see that there are multiple records returned with the same primaryid. Also, the runtime for the query is 12-15 seconds which does not comply with the project needs. For this, we exported the whole data for individual tables to individual CSV files, for analysis using the python IDE. Python IDE is not traditionally a database management tool, but for this project purposes, it fulfills the requirements.

6. Python Analysis:

6.1 Dummy database to check query integrity

Similar to the previous method, generating a dummy database similar to our original data frames would be our first step. This step is pivotal to check the integrity of the data as well as helps reduce the runtime for queries. The values used in this dummy database are randomly picked from the original database. The dummy database was made according to the entity-relationship mentioned in section 5.1.

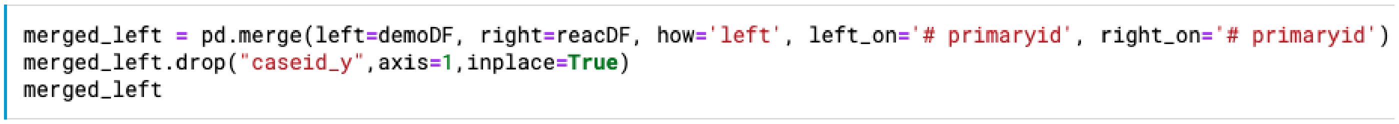
Figure 6.1(a) shows an example of the dummy drug data. As seen, there are multiple records with the same primaryid, as in the real database. Dummy datasets for the rest of the tables are in Appendix C.



# primaryid	caseid	drug_seq	drugname	prod_ai	val_vbm	route	dose_vbm	cum_dose_chr	cum_dose_unit	dechal	rechal	exp_dt	dose_amt	dose_unit	dose_form	dose_freq
1	3894168	1	BROMOCRIPTINE MESYLATE	BROMOCRIPTINE MESYLATE	1	Oral	10 MG, TID	0	NaN	Y	NaN	NaN	10.0	MG	NaN	TID
1	3894168	2	BROMOCRIPTINE MESYLATE	BROMOCRIPTINE MESYLATE	1	Oral	7.5 MG, TID	0	NaN	Y	NaN	NaN	7.5	MG	NaN	TID
1	3894168	7	LEVODOPA+BENSERAZIDE	BENSERAZIDE\LEVODOPA	1	Oral	125 MG, 6QD	0	NaN	NaN	NaN	NaN	125.0	MG	NaN	NaN
4	4022386	1	INFLIXIMAB, RECOMBINANT	INFLIXIMAB	1	Intravenous (not otherwise specified)	WEEKS 0, 2, 6, 10, 18, 26, AND 34 CYCLICAL	0	NaN	U	U	NaN	10.0	MG/KG	SOLUTION FOR INFUSION	NaN
5	3894168	13	SELEGILINE	SELEGILINE	1	Unknown	15 MG, UNK	0	NaN	U	NaN	NaN	15.0	MG	NaN	NaN
6	11259551	3	DICLOFENAC SODIUM.	DICLOFENAC SODIUM	1	NaN	100 MG, ONCE DAILY (QD)	0	NaN	NaN	U	NaN	100.0	MG	NaN	QD
7	11468194	1	LIPITOR	ATORVASTATIN CALCIUM	1	Oral	20 MG, DAILY	0	NaN	U	NaN	NaN	20.0	MG	FILM-COATED TABLET	NaN

Figure 6.1(a)

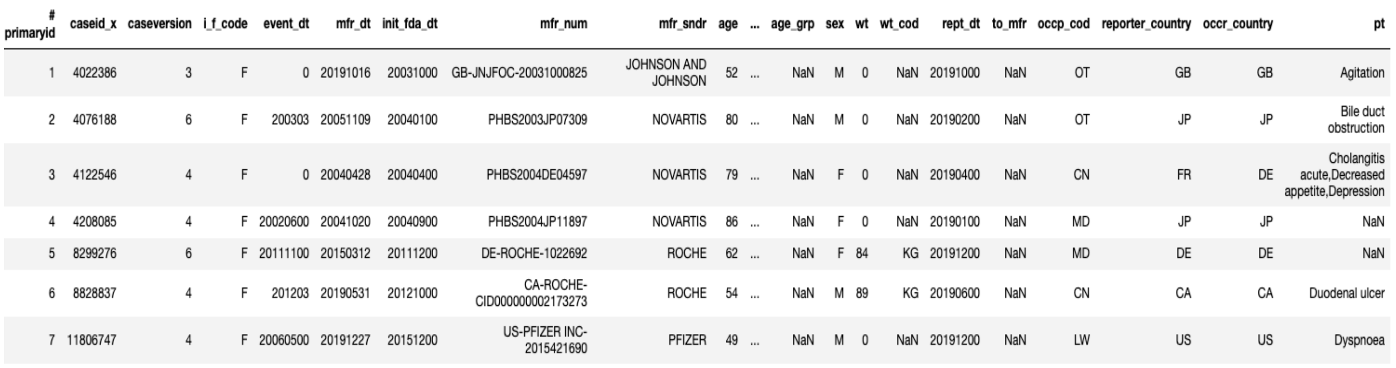
Using the query shown in figure 6.1(b), we performed an inner join using the pandas library on primaryid of demographic and the reaction table to match the reactions with the respective demographics.



```
merged_left = pd.merge(left=demoDF, right=reacDF, how='left', left_on='# primaryid', right_on='# primaryid')
merged_left.drop("caseid_y",axis=1,inplace=True)
merged_left
```

Figure 6.1(b)

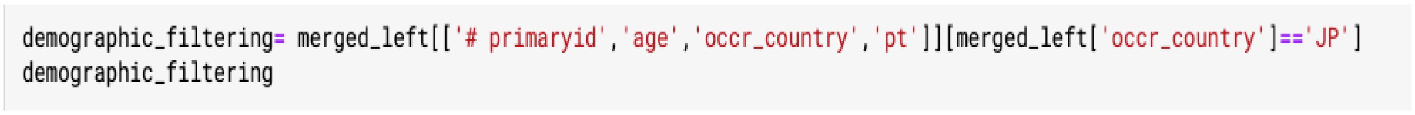
As seen in the results in figure 6.1(c), the reactions (pt) of the table is merged with the demographic table. Notice, only unique primary id's are returned. In the reaction table, there were 3 records with primary id 3, containing 3 different reactions. All of those reactions namely Cholangitis acute, Decreased appetite and Depression are merged to a single primary id 3.



# primaryid	caseid_x	caseversion	i_f_code	event_dt	mfr_dt	init_fda_dt	mfr_num	mfr_sndr	age	...	age_grp	sex	wt	wt_cod	rept_dt	to_mfr	occp_cod	reporter_country	occr_country	pt
1	4022386	3	F	0	20191016	20031000	GB-JNJFOC-20031000825	JOHNSON AND JOHNSON	52	...	NaN	M	0	NaN	20191000	NaN	OT	GB	GB	Agitation
2	4076188	6	F	200303	20051109	20040100	PHBS2003JP07309	NOVARTIS	80	...	NaN	M	0	NaN	20190200	NaN	OT	JP	JP	Bile duct obstruction
3	4122546	4	F	0	20040428	20040400	PHBS2004DE04597	NOVARTIS	79	...	NaN	F	0	NaN	20190400	NaN	CN	FR	DE	Cholangitis acute,Decreased appetite,Depression
4	4208085	4	F	20020600	20041020	20040900	PHBS2004JP11897	NOVARTIS	86	...	NaN	F	0	NaN	20190100	NaN	MD	JP	JP	NaN
5	8299276	6	F	20111100	20150312	20111200	DE-ROCHE-1022692	ROCHE	62	...	NaN	F	84	KG	20191200	NaN	MD	DE	DE	NaN
6	8828837	4	F	201203	20190531	20121000	CA-ROCHE-CID000000002173273	ROCHE	54	...	NaN	M	89	KG	20190600	NaN	CN	CA	CA	Duodenal ulcer
7	11806747	4	F	20060500	20191227	20151200	US-PFIZER INC-2015421690	PFIZER	49	...	NaN	M	0	NaN	20191200	NaN	LW	US	US	Dyspnoea

Figure 6.1(c)

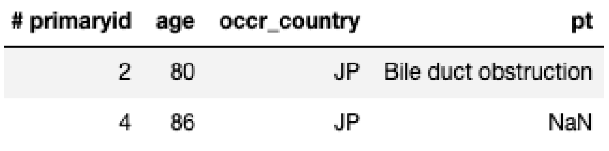
Using the merged data frame, we can now run filter queries on the resultant merged dataset. Running the query shown in figure 6.1(d), on the merged data-frame we can filter out records on any specific demographics. The query below filters out records reported in Japan(JP).



```
demographic_filtering= merged_left[['# primaryid','age','occr_country','pt']][merged_left['occr_country']=='JP']
demographic_filtering
```

Figure 6.1(d)

The results shown in figure 6.1(e) are filtered records where the occurred country is JP. According to our dummy database, 2 records should be filtered for occurrence country JP. The age, demographic, and reaction for the particular patients are returned.



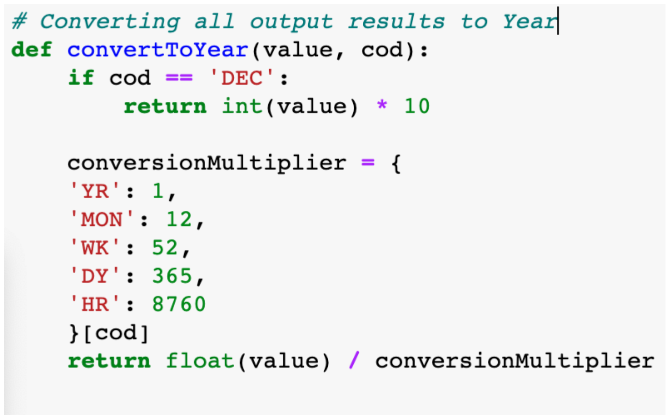
# primaryid	age	occr_country	pt
2	80	JP	Bile duct obstruction
4	86	JP	NaN

Figure 6.1(e)

6.2 <u>Data Preprocessing and knowledge extraction</u>

Now, knowing that we have a relational database set up, we can start creating a script that can extract the demographics and adverse event reports of a particular drug. The following preprocessing steps were performed so that we get a consistent output for our query.

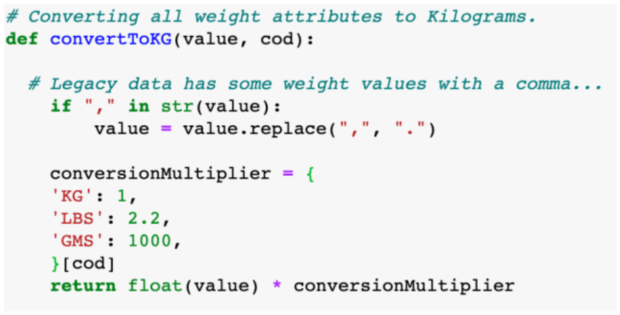
- The year attribute is reported in multiple formats around the world. As shown in Figure 6.2(a) for consistent output purposes, all observations were converted to year.



```python
# Converting all output results to Year
def convertToYear(value, cod):
    if cod == 'DEC':
        return int(value) * 10

    conversionMultiplier = {
    'YR': 1,
    'MON': 12,
    'WK': 52,
    'DY': 365,
    'HR': 8760
    }[cod]
    return float(value) / conversionMultiplier
```

Figure 6.2(a)

- Similarly, the weight of the patients are reported in multiple formats. Also, there are some ','(comma) values in the weight attribute. As seen in Figure 6.2(b), we will handle them all.



```python
# Converting all weight attributes to Kilograms.
def convertToKG(value, cod):

  # Legacy data has some weight values with a comma...
    if "," in str(value):
        value = value.replace(",", ".")

    conversionMultiplier = {
    'KG': 1,
    'LBS': 2.2,
    'GMS': 1000,
    }[cod]
    return float(value) * conversionMultiplier
```

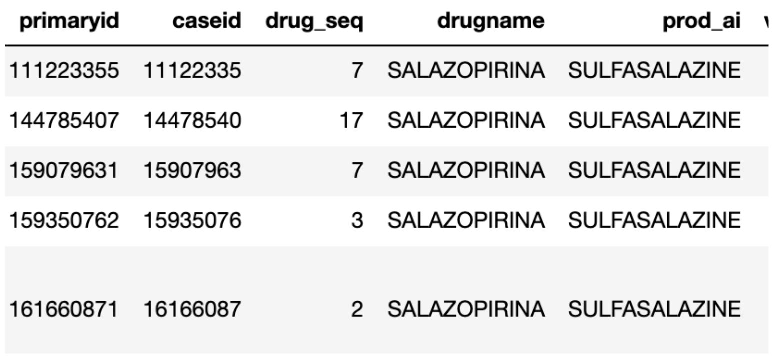
Figure 6.2(b)

There are some challenging key points to keep in mind before extracting information.

- Only the demographic table contains the unique adverse reports, in every other table there are multiple rows with the same primary id's.
- While dealing with such a huge database, we have to make sure that there are no duplicate records in our output file.

- Also, there are a large number of ‘null’ or ‘NaN’ values in the dataset. Handling missing values becomes very pivotal as we do not want to delete these rows and undershoot our results. This will decrease our accuracy.
- Merging tables according to the Entity-Relationship diagram is a must for increasing our accuracy.

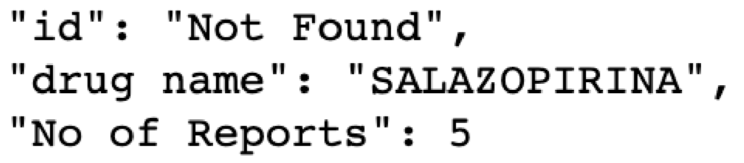
Considering the sensitivity of the dataset, it is very important to check the integrity of the results obtained. Figure 6.2(c) shows the records which have ‘SALAZOPIRINA’ as its drug name. As seen, there are 5 records extracted. When we run our script, we should get 5 as the number of reports, a rxCUI id, if there, and the demographics of the particular drug. The script is shown in Appendix C.



primaryid	caseid	drug_seq	drugname	prod_ai
111223355	11122335	7	SALAZOPIRINA	SULFASALAZINE
144785407	14478540	17	SALAZOPIRINA	SULFASALAZINE
159079631	15907963	7	SALAZOPIRINA	SULFASALAZINE
159350762	15935076	3	SALAZOPIRINA	SULFASALAZINE
161660871	16166087	2	SALAZOPIRINA	SULFASALAZINE

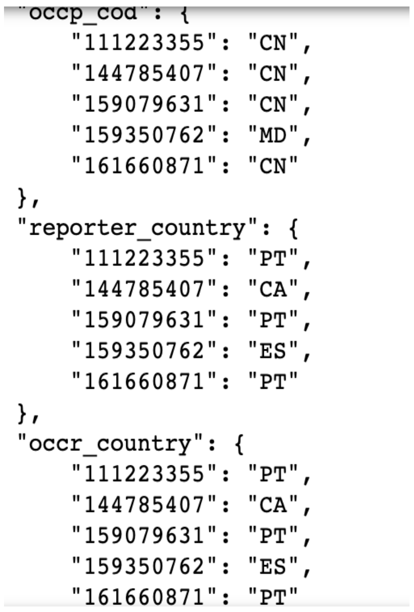
Figure 6.2(c)

As seen in Figure 6.2(d) and 6.2(e), our query results are accurate. All the attributes pertaining to the particular drug from the demographics table are returned. The rxCUI code(id) is called using the requests library. The request method is shown in the script in Appendix C. This alone is a ton of information. The results also help us to filter out all the information in the database related to the drug.



```
"id": "Not Found",
"drug name": "SALAZOPIRINA",
"No of Reports": 5
```

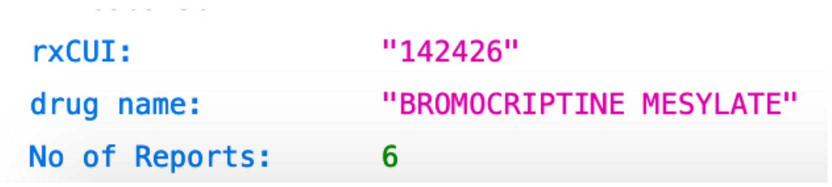
Figure 6.2(d)



```
"occp_cod": {
    "111223355": "CN",
    "144785407": "CN",
    "159079631": "CN",
    "159350762": "MD",
    "161660871": "CN"
},
"reporter_country": {
    "111223355": "PT",
    "144785407": "CA",
    "159079631": "PT",
    "159350762": "ES",
    "161660871": "PT"
},
"occr_country": {
    "111223355": "PT",
    "144785407": "CA",
    "159079631": "PT",
    "159350762": "ES",
    "161660871": "PT"
```

Figure 6.2(e)

As seein in figure 6.2(d), this particular drug does not have a rxCUI code. So, for output verification purposes, we filtered out results for a drug that had a rxCUI code. Figure 6.2(f) shows the reports reported for “BROMOCRIPTINE MESYLATE”. Using the rxCUI code we can merge the existing dataset with the rest of the data warehouse.



```
rxCUI:              "142426"
drug name:          "BROMOCRIPTINE MESYLATE"
No of Reports:      6
```

Figure 6.2(f)

Appendix D shows the rest of the results extracted. Similarly, we can return all the rest of the information like adverse effects, symptoms, and the output of a particular patient.

7. Conclusion and Future Work

Some important insights are gained from this project which can further help extend the use cases of the dataset. Some main objectives defined previously are analyzed and discussed. There are two methodologies discussed to achieve the objectives of this project. Both methodologies have their advantages and limitations discussed below.

The dataset is too large for a local SQL Workbench server. The output obtained by using the SQL server is accurate and the queries generated are efficient, but, the query processing time is too large making the results non-robust. On the positive side, database management is quite effective with a better user interface. This is one of the most important factors as it eases handling and querying a large-scaled database. Also, data manipulation and data altering become easy and efficient.
Moving to Python IDE, checking the integrity of the originally provided script is one of the most tasks. The results extracted via the original script undershoots our results, which decreases our accuracy. It is important to merge the tables according to the E-R diagram so that we handle the duplicates while maintaining high accuracy of our results. By iterating over every row of the drug and demographic table, the demographics and the number of unique reports for a specific drug are returned. These results are verified by first running the queries on a dummy database. The rxCUI code is called using the request library and returned with the appropriate drug query result. This helps us in expanding the database further. More information can be extracted by merging the tables via their appropriate relationship. The queries here are quite robust compared to the SQL Server. Python is not a database management tool, and so it is not user friendly while handling data schemas. Data alteration would be a challenge while using this environment.

The dataset is rich with information but equally noisy. Important patterns and visualizations can be created using different tables to extract valuable information from the database. For this, it is important to apply more preprocessing steps and further cleanse the data. Also, the data scale is huge and the FDA publishes the data for each quarter which makes this a dynamic growing dataset. To handle the scale of the data, the data warehousing step should be shifted to a big data software like SPARK. This step will make the data gathering process efficient and robust.

8. References:

- FDA Adverse Event Reporting System (FAERS) Quarterly Data Extract Files

  https://fis.fda.gov/extensions/FPD-QDE-FAERS/FPD-QDE-FAERS.html

- Sujay B., et al., " Adverse Drug Reaction Detection System On the basis of Clinical Data", International Journal of Advanced Research in Computer Engineering & Technology (IJARCET) Volume 5, Issue 4, 2016.

- Pandas GroupBy: Your Guide to Grouping Data in Python

  https://realpython.com/pandas-groupby/

- Naresh Poluju, Purushotham Muniganti "Adverse Drug Reaction Detection Using Data Mining Approaches: A Survey", International Journal of Recent Trends in Engineering and Research

- Combining Pandas DataFrames: The easy way

  https://towardsdatascience.com/combining-pandas-dataframes-the-easy-way-41eb0f2c1ebf

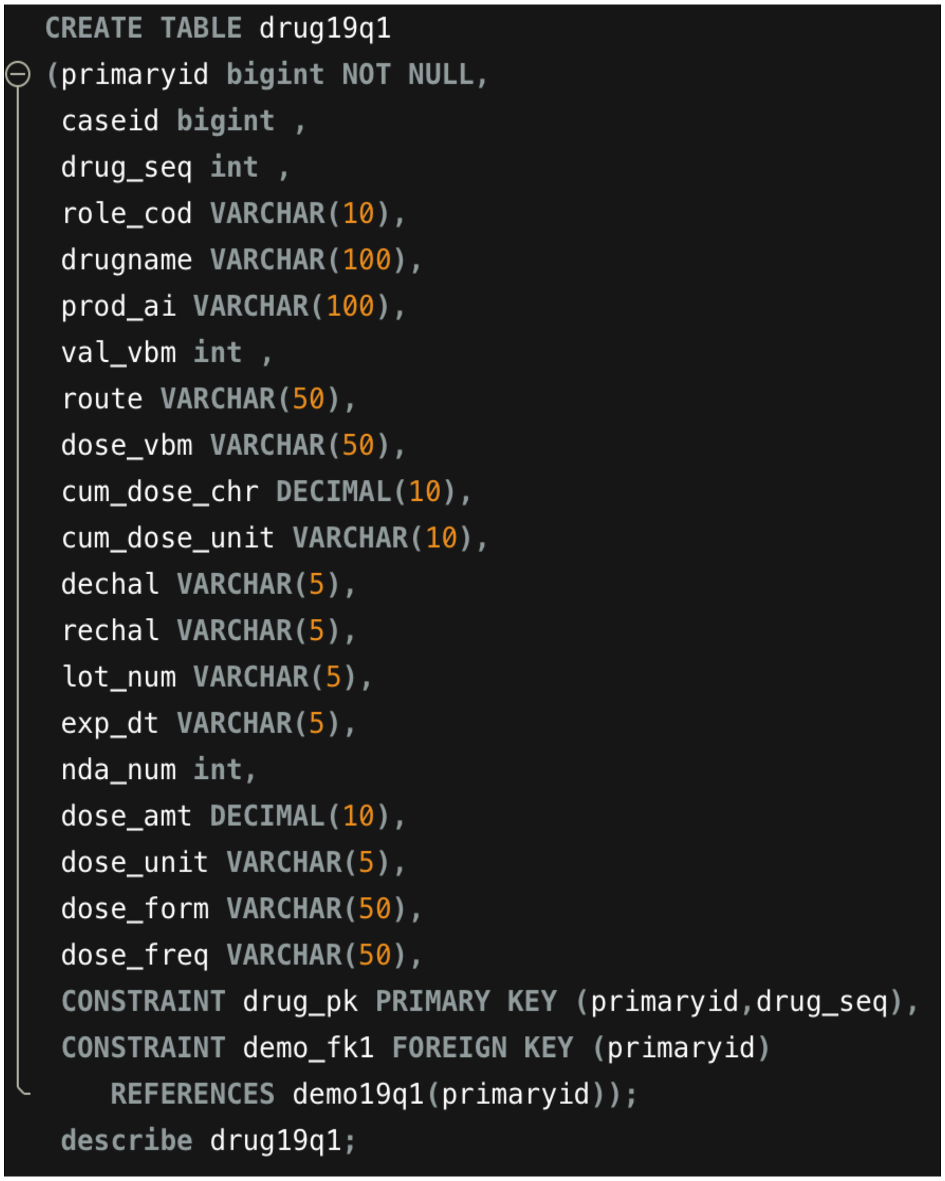
- Y. Ji, F. Shen and J. Tran, "A Multi-relational Association Mining Algorithm for Screening Suspected Adverse Drug Reactions," 2014 11th International Conference on Information Technology: New Generations, Las Vegas, NV, 2014, pp. 407-412, doi: 10.1109/ITNG.2014.96.

- Daniel Foley. "Let's Build a Streaming Data Pipeline", towards data science, May 7, 2019

  https://towardsdatascience.com/lets-build-a-streaming-data-pipeline-e873d671fc57

1. <u>Appendix A</u>

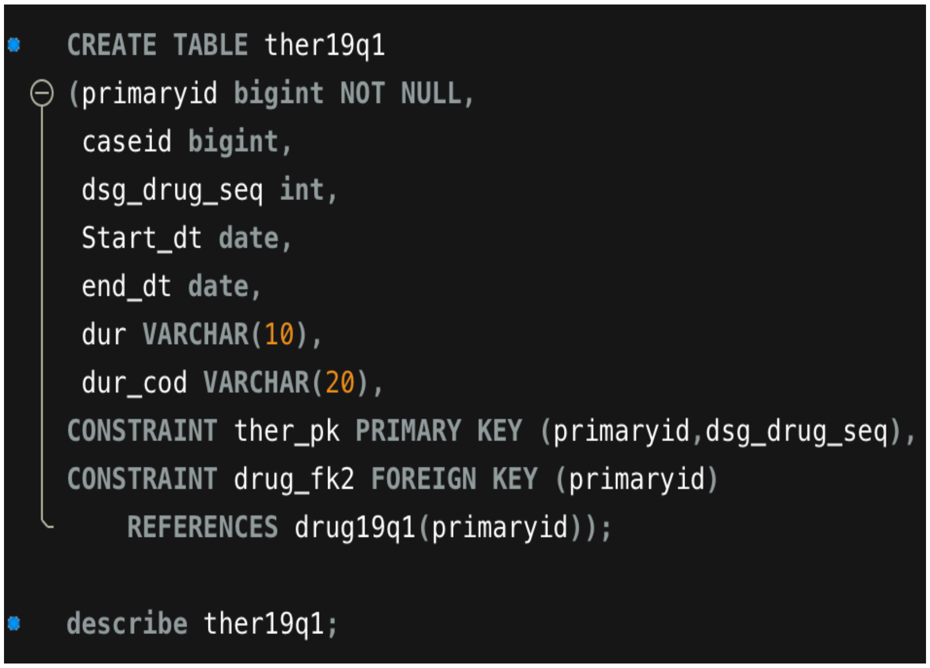
1. Drug Table



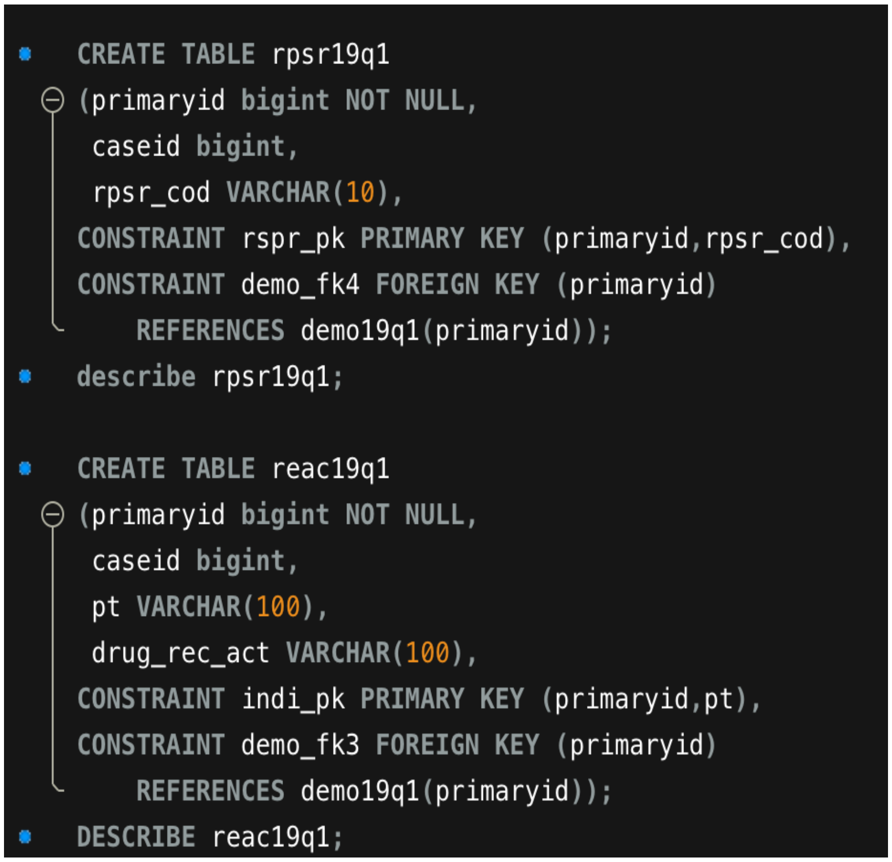
```
CREATE TABLE drug19q1
(primaryid bigint NOT NULL,
 caseid bigint ,
 drug_seq int ,
 role_cod VARCHAR(10),
 drugname VARCHAR(100),
 prod_ai VARCHAR(100),
 val_vbm int ,
 route VARCHAR(50),
 dose_vbm VARCHAR(50),
 cum_dose_chr DECIMAL(10),
 cum_dose_unit VARCHAR(10),
 dechal VARCHAR(5),
 rechal VARCHAR(5),
 lot_num VARCHAR(5),
 exp_dt VARCHAR(5),
 nda_num int,
 dose_amt DECIMAL(10),
 dose_unit VARCHAR(5),
 dose_form VARCHAR(50),
 dose_freq VARCHAR(50),
 CONSTRAINT drug_pk PRIMARY KEY (primaryid,drug_seq),
 CONSTRAINT demo_fk1 FOREIGN KEY (primaryid)
    REFERENCES demo19q1(primaryid));
 describe drug19q1;
```

- Using the describe query, we can have an overall look at the attributes and their respective datatypes. This helps us to assure that our table is created as we intended.

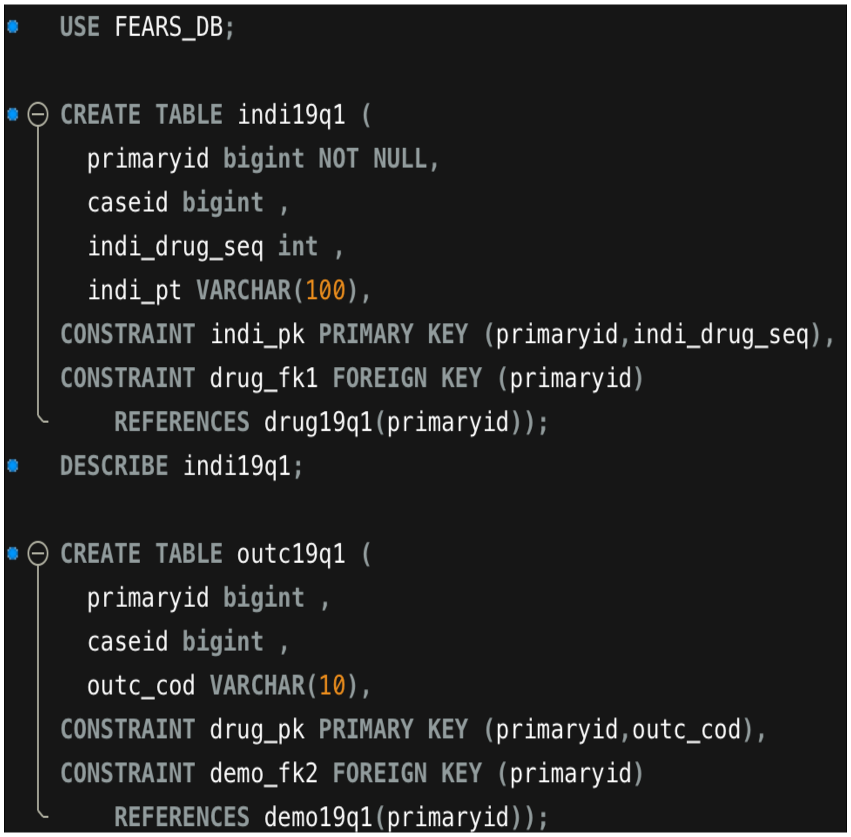
2. Therapy Table


```
CREATE TABLE ther19q1
(primaryid bigint NOT NULL,
 caseid bigint,
 dsg_drug_seq int,
 Start_dt date,
 end_dt date,
 dur VARCHAR(10),
 dur_cod VARCHAR(20),
CONSTRAINT ther_pk PRIMARY KEY (primaryid,dsg_drug_seq),
CONSTRAINT drug_fk2 FOREIGN KEY (primaryid)
    REFERENCES drug19q1(primaryid));

describe ther19q1;
```

3. Report Sources and Reaction Table

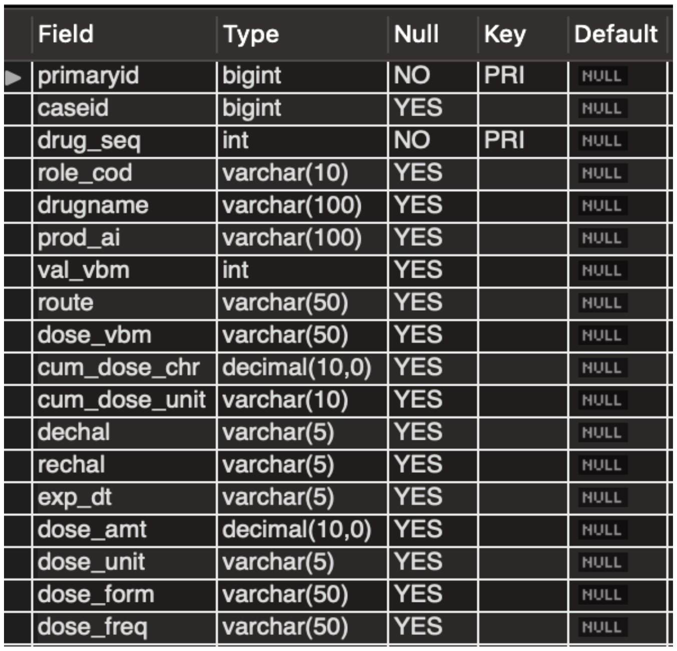

```
CREATE TABLE rpsr19q1
(primaryid bigint NOT NULL,
 caseid bigint,
 rpsr_cod VARCHAR(10),
CONSTRAINT rspr_pk PRIMARY KEY (primaryid,rpsr_cod),
CONSTRAINT demo_fk4 FOREIGN KEY (primaryid)
    REFERENCES demo19q1(primaryid));
describe rpsr19q1;

CREATE TABLE reac19q1
(primaryid bigint NOT NULL,
 caseid bigint,
 pt VARCHAR(100),
 drug_rec_act VARCHAR(100),
CONSTRAINT indi_pk PRIMARY KEY (primaryid,pt),
CONSTRAINT demo_fk3 FOREIGN KEY (primaryid)
    REFERENCES demo19q1(primaryid));
DESCRIBE reac19q1;
```

4. Indication and Outcome Table



```
USE FEARS_DB;

CREATE TABLE indi19q1 (
  primaryid bigint NOT NULL,
  caseid bigint ,
  indi_drug_seq int ,
  indi_pt VARCHAR(100),
CONSTRAINT indi_pk PRIMARY KEY (primaryid,indi_drug_seq),
CONSTRAINT drug_fk1 FOREIGN KEY (primaryid)
    REFERENCES drug19q1(primaryid));
DESCRIBE indi19q1;

CREATE TABLE outc19q1 (
  primaryid bigint ,
  caseid bigint ,
  outc_cod VARCHAR(10),
CONSTRAINT drug_pk PRIMARY KEY (primaryid,outc_cod),
CONSTRAINT demo_fk2 FOREIGN KEY (primaryid)
    REFERENCES demo19q1(primaryid));
```

5. Describe query for Drug Table

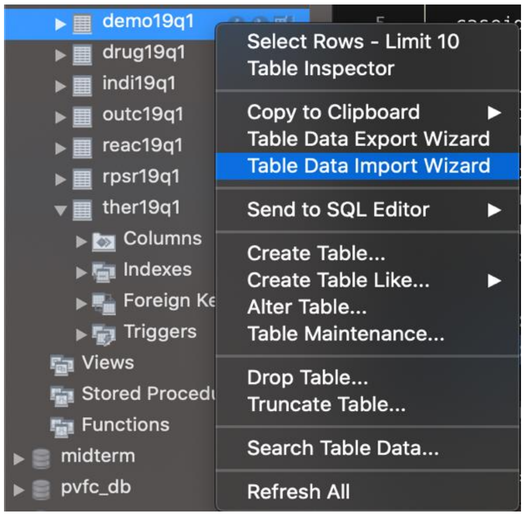


Field	Type	Null	Key	Default
primaryid	bigint	NO	PRI	NULL
caseid	bigint	YES		NULL
drug_seq	int	NO	PRI	NULL
role_cod	varchar(10)	YES		NULL
drugname	varchar(100)	YES		NULL
prod_ai	varchar(100)	YES		NULL
val_vbm	int	YES		NULL
route	varchar(50)	YES		NULL
dose_vbm	varchar(50)	YES		NULL
cum_dose_chr	decimal(10,0)	YES		NULL
cum_dose_unit	varchar(10)	YES		NULL
dechal	varchar(5)	YES		NULL
rechal	varchar(5)	YES		NULL
exp_dt	varchar(5)	YES		NULL
dose_amt	decimal(10,0)	YES		NULL
dose_unit	varchar(5)	YES		NULL
dose_form	varchar(50)	YES		NULL
dose_freq	varchar(50)	YES		NULL

2. Appendix B

Load Data using Table Import Wizard

Step 1: Select 'Table Data Import Wizard' for the entity table which is to be populated. Let's start with 'demo19q1'.



Step 2: Browse and select the correct CSV file and click Next.

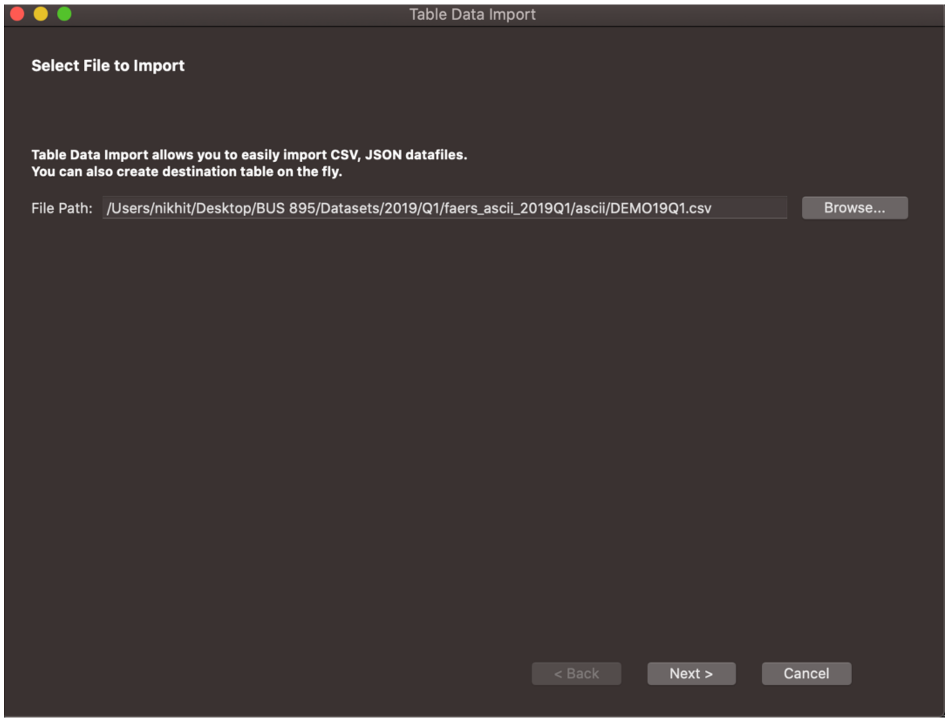


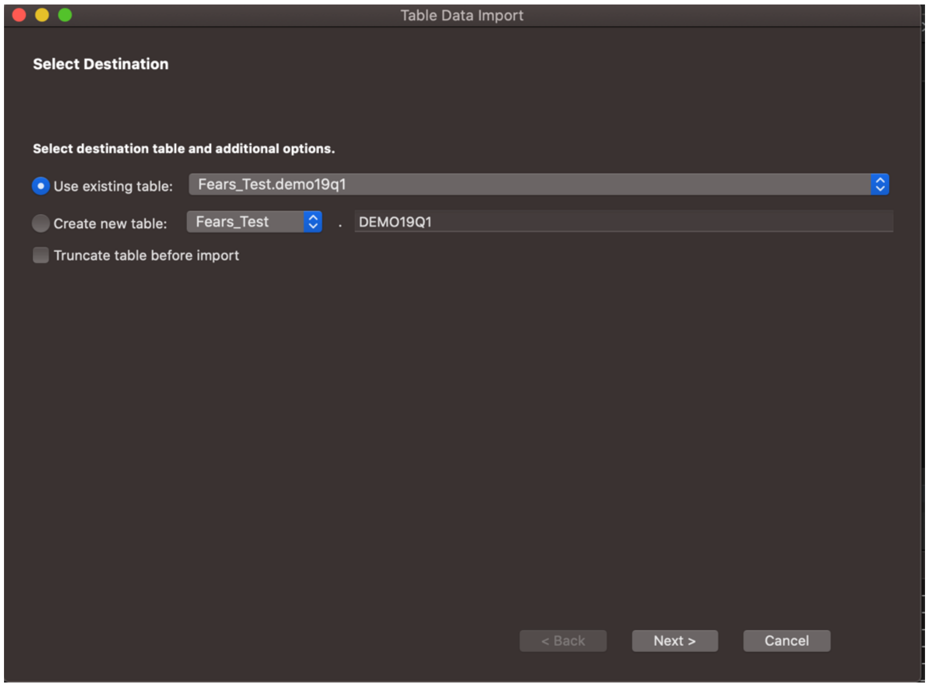
Table Data Import

Select File to Import

Table Data Import allows you to easily import CSV, JSON datafiles.
You can also create destination table on the fly.

File Path: /Users/nikhit/Desktop/BUS 895/Datasets/2019/Q1/faers_ascii_2019Q1/ascii/DEMO19Q1.csv

Step 3: Select the destination entity table. If the CSV file is too long, it is possible to truncate it. For this project, though the csv is large, we need all the records and so we won't truncate it. Click Next.



Select Destination

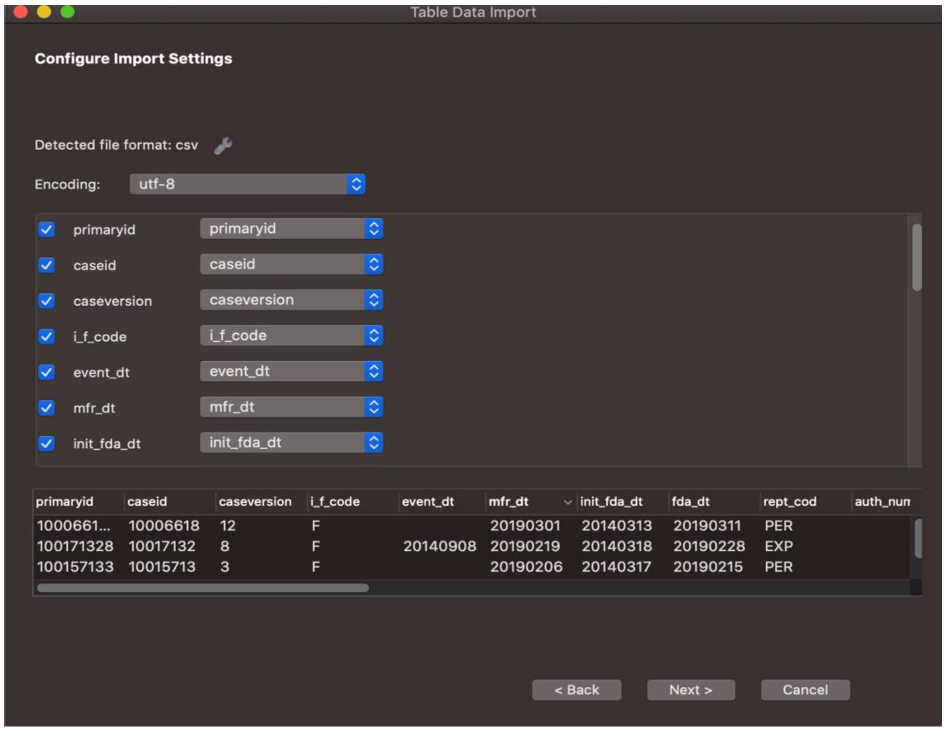
Select destination table and additional options.

[✓ Use existing table:] Fears_Test.demo19q1

Create new table: Fears_Test . DEMO19Q1

Truncate table before import

Step 4: Select the columns to be imported from the CSV file. Click Next



Configure Import Settings

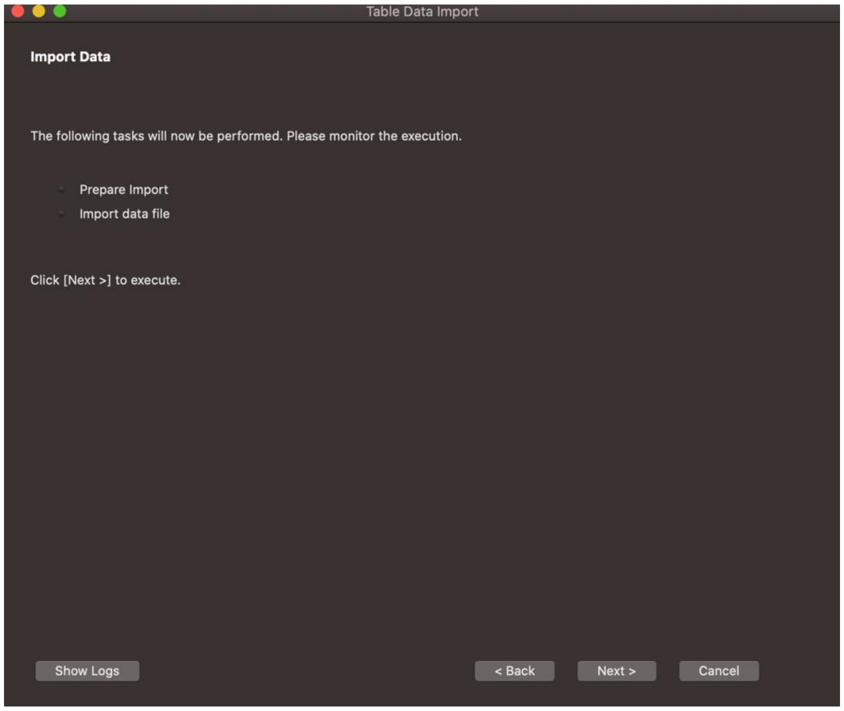
Detected file format: csv

Encoding: utf-8

✓	primaryid	primaryid
✓	caseid	caseid
✓	caseversion	caseversion
✓	i_f_code	i_f_code
✓	event_dt	event_dt
✓	mfr_dt	mfr_dt
✓	init_fda_dt	init_fda_dt

primaryid	caseid	caseversion	i_f_code	event_dt	mfr_dt	init_fda_dt	fda_dt	rept_cod	auth_nun
1000661...	10006618	12	F		20190301	20140313	20190311	PER	
100171328	10017132	8	F	20140908	20190219	20140318	20190228	EXP	
100157133	10015713	3	F		20190206	20140317	20190215	PER	

Step 5: Import the data from the CSV file by clicking on Next. Once the data is imported currently, ‘Prepare Import’ and ‘Import data files’ will turn green. If there are errors, they will turn red and the corresponding error will be displayed. Click Next after the data is successfully imported.



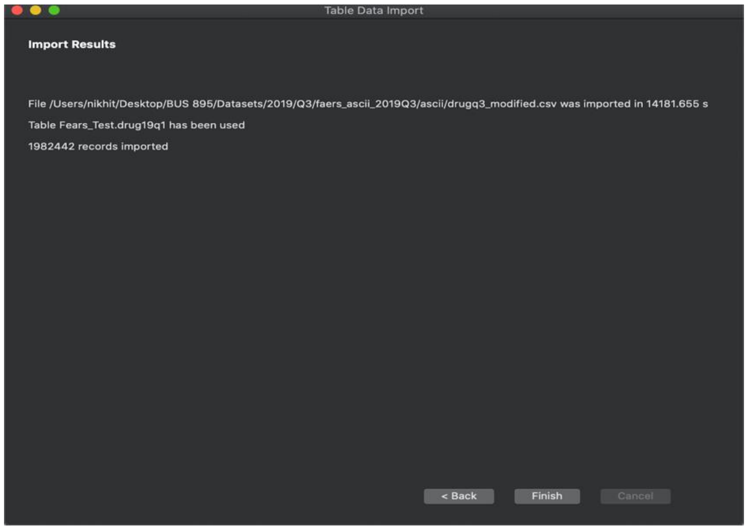
Import Data

The following tasks will now be performed. Please monitor the execution.

- Prepare Import
- Import data file

Click [Next >] to execute.

Step 6: Now the data is successfully imported into the ‘demo19q1’ table. Click on Finish to exit Table Data Import Wizard. Below is the screenshot of data successfully imported into the drug table.

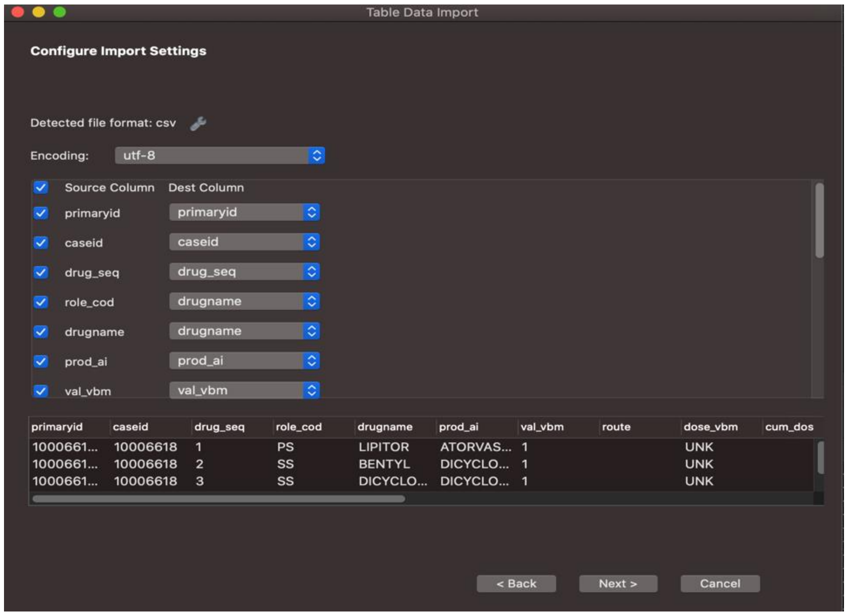


Import Results

```
File /Users/nikhit/Desktop/BUS 895/Datasets/2019/Q3/faers_ascii_2019Q3/ascii/drugq3_modified.csv was imported in 14181.655 s
Table Fears_Test.drug19q1 has been used
1982442 records imported
```

All the 7 tables are huge in size, and so it takes a lot of time to load these tables on the local computer. The above steps can be applied to populate other entity tables as well. Below are the configurations of import settings (Step5) for ‘drug19q1, ‘indi19q1’, ‘outc19q1’, ‘reac19q1’, ‘rpsr19q1‘, and ‘ther19q1’.

Drug Entity



Configure Import Settings

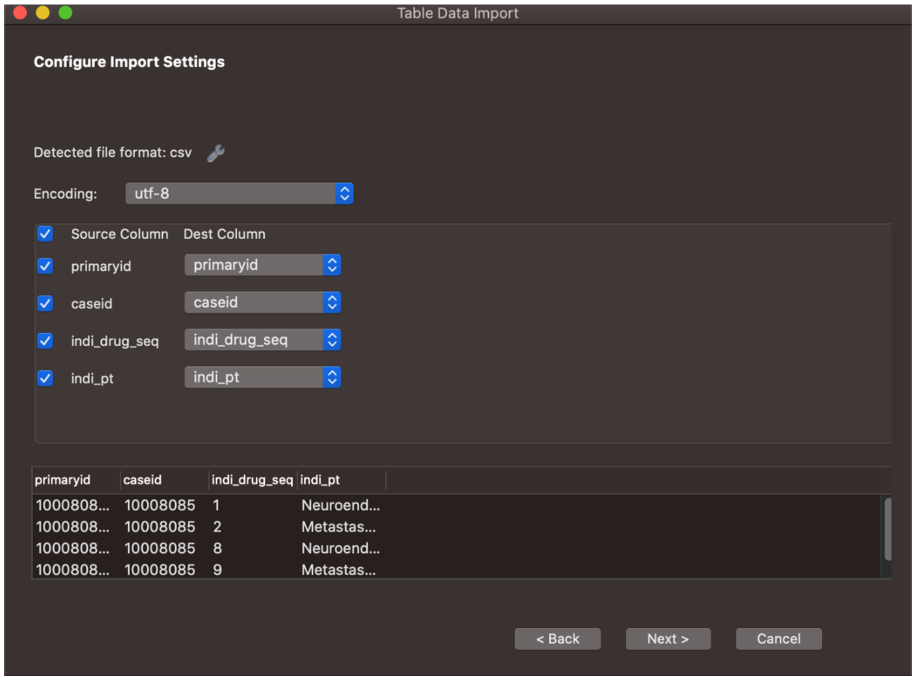
Detected file format: csv

Encoding: utf-8

Source Column	Dest Column
primaryid	primaryid
caseid	caseid
drug_seq	drug_seq
role_cod	drugname
drugname	drugname
prod_ai	prod_ai
val_vbm	val_vbm

primaryid	caseid	drug_seq	role_cod	drugname	prod_ai	val_vbm	route	dose_vbm	cum_dos
1000661...	10006618	1	PS	LIPITOR	ATORVAS...	1		UNK	
1000661...	10006618	2	SS	BENTYL	DICYCLO...	1		UNK	
1000661...	10006618	3	SS	DICYCLO...	DICYCLO...	1		UNK	

Indication Entity



Configure Import Settings

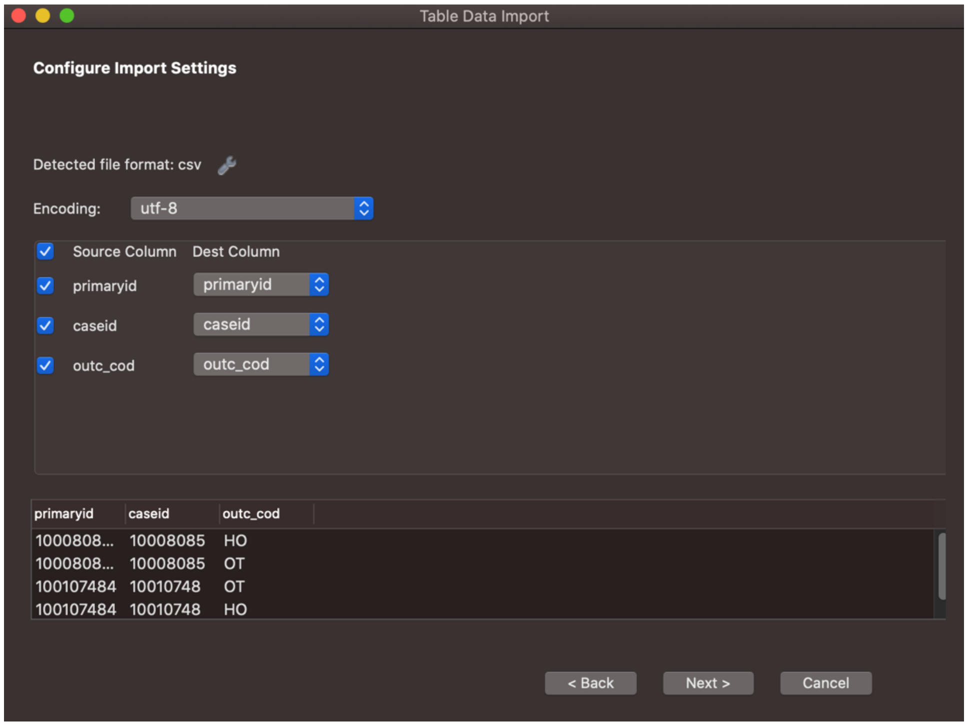
Detected file format: csv

Encoding: utf-8

Source Column	Dest Column
primaryid	primaryid
caseid	caseid
indi_drug_seq	indi_drug_seq
indi_pt	indi_pt

primaryid	caseid	indi_drug_seq	indi_pt
1000808...	10008085	1	Neuroend...
1000808...	10008085	2	Metastas...
1000808...	10008085	8	Neuroend...
1000808...	10008085	9	Metastas...

Outcome Entity



Configure Import Settings

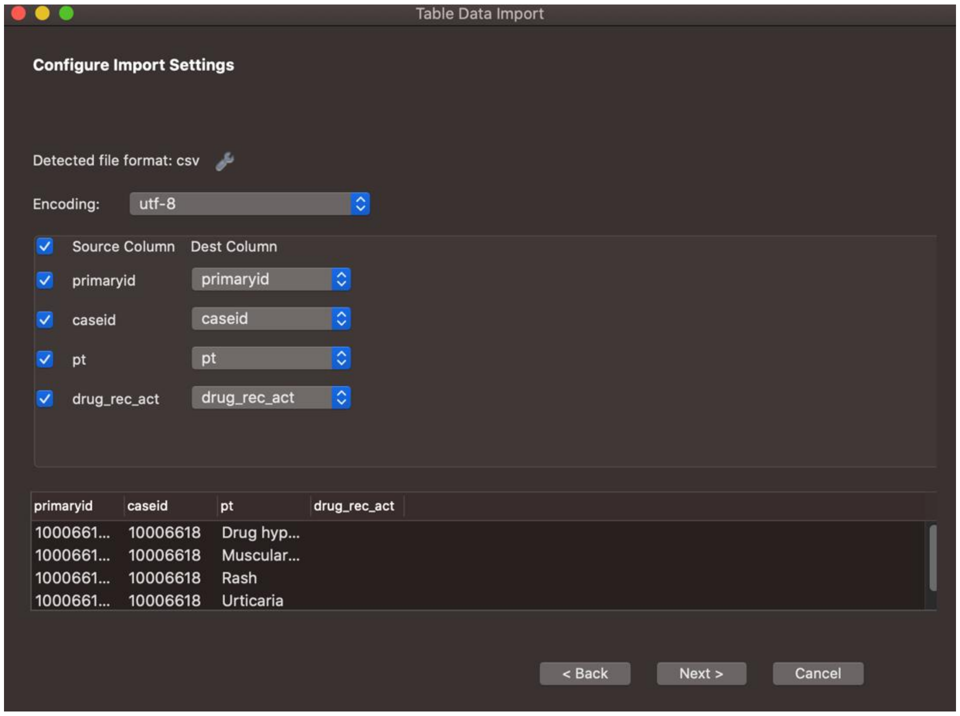
Detected file format: csv

Encoding: utf-8

✓	Source Column	Dest Column
✓	primaryid	primaryid
✓	caseid	caseid
✓	outc_cod	outc_cod

primaryid	caseid	outc_cod
1000808...	10008085	HO
1000808...	10008085	OT
100107484	10010748	OT
100107484	10010748	HO

Reaction Entity



Configure Import Settings

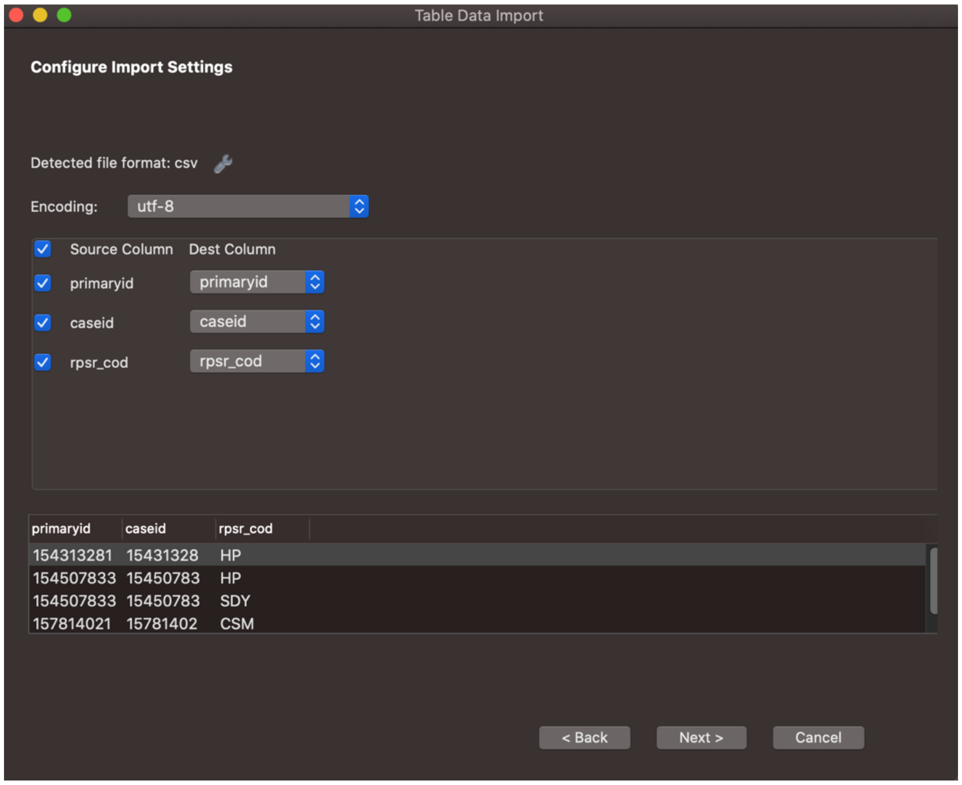
Detected file format: csv

Encoding: utf-8

✓	Source Column	Dest Column
✓	primaryid	primaryid
✓	caseid	caseid
✓	pt	pt
✓	drug_rec_act	drug_rec_act

primaryid	caseid	pt	drug_rec_act
1000661...	10006618	Drug hyp...	
1000661...	10006618	Muscular...	
1000661...	10006618	Rash	
1000661...	10006618	Urticaria	

Report Source Table



Configure Import Settings

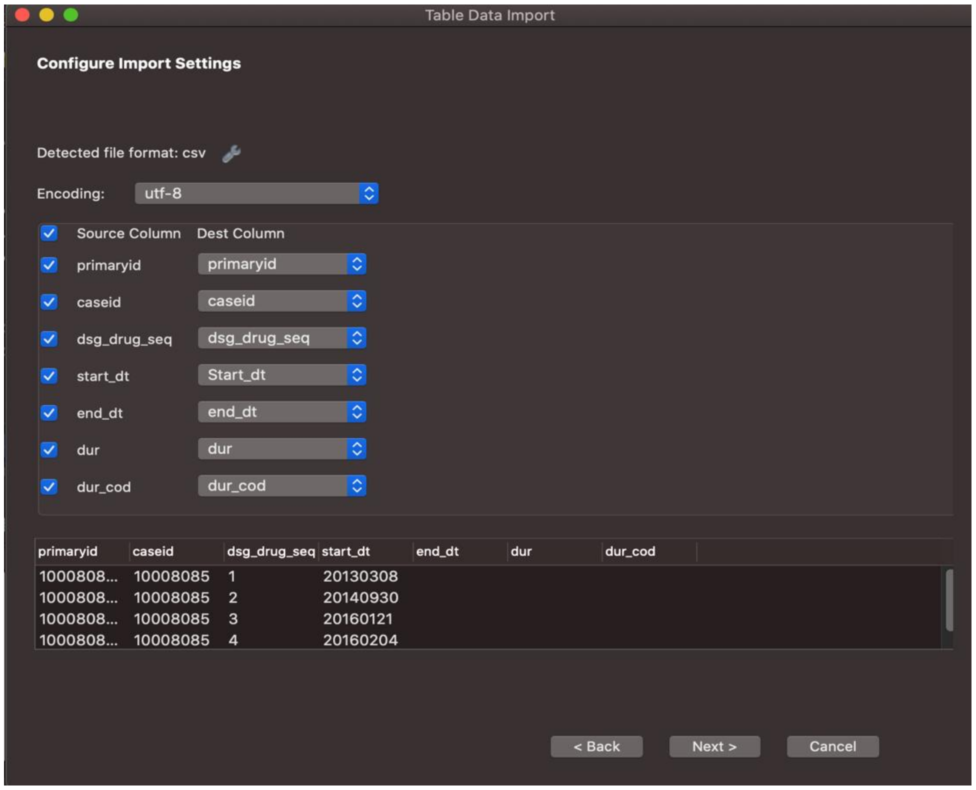
Detected file format: csv

Encoding: utf-8

✓	Source Column	Dest Column
✓	primaryid	primaryid
✓	caseid	caseid
✓	rpsr_cod	rpsr_cod

primaryid	caseid	rpsr_cod
154313281	15431328	HP
154507833	15450783	HP
154507833	15450783	SDY
157814021	15781402	CSM

Therapy Table



Configure Import Settings

Detected file format: csv

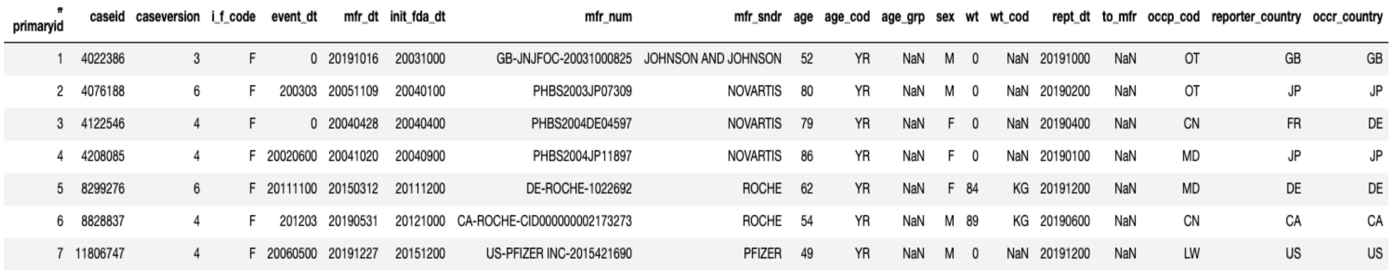
Encoding: utf-8

Source Column	Dest Column
✓ primaryid	primaryid
✓ caseid	caseid
✓ dsg_drug_seq	dsg_drug_seq
✓ start_dt	Start_dt
✓ end_dt	end_dt
✓ dur	dur
✓ dur_cod	dur_cod

primaryid	caseid	dsg_drug_seq	start_dt	end_dt	dur	dur_cod
1000808...	10008085	1	20130308			
1000808...	10008085	2	20140930			
1000808...	10008085	3	20160121			
1000808...	10008085	4	20160204			

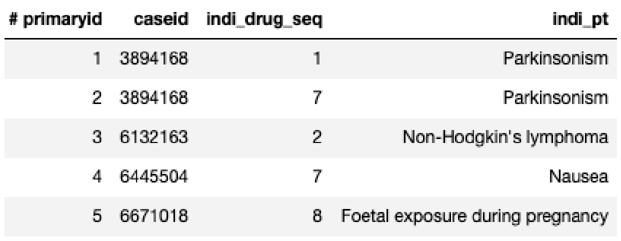
3. Appendix C

- Demographic Table



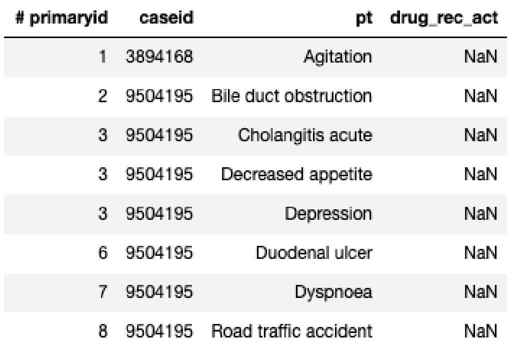
# primaryid	caseid	caseversion	i_f_code	event_dt	mfr_dt	init_fda_dt	mfr_num	mfr_sndr	age	age_cod	age_grp	sex	wt	wt_cod	rept_dt	to_mfr	occp_cod	reporter_country	occr_country
1	4022386	3	F	0	20191016	20031000	GB-JNJFOC-20031000825	JOHNSON AND JOHNSON	52	YR	NaN	M	0	NaN	20191000	NaN	OT	GB	GB
2	4076188	6	F	200303	20051109	20040100	PHBS2003JP07309	NOVARTIS	80	YR	NaN	M	0	NaN	20190200	NaN	OT	JP	JP
3	4122546	4	F	0	20040428	20040400	PHBS2004DE04597	NOVARTIS	79	YR	NaN	F	0	NaN	20190400	NaN	CN	FR	DE
4	4208085	4	F	20020600	20041020	20040900	PHBS2004JP11897	NOVARTIS	86	YR	NaN	F	0	NaN	20190100	NaN	MD	JP	JP
5	8299276	6	F	20111100	20150312	20111200	DE-ROCHE-1022692	ROCHE	62	YR	NaN	F	84	KG	20191200	NaN	MD	DE	DE
6	8828837	4	F	201203	20190531	20121000	CA-ROCHE-CID000000002173273	ROCHE	54	YR	NaN	M	89	KG	20190600	NaN	CN	CA	CA
7	11806747	4	F	20060500	20191227	20151200	US-PFIZER INC-2015421690	PFIZER	49	YR	NaN	M	0	NaN	20191200	NaN	LW	US	US

- Indication Table



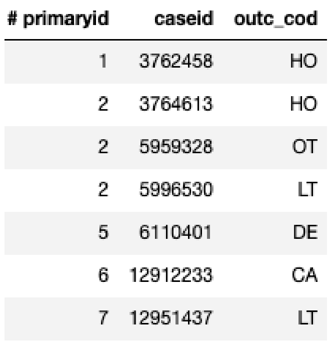
# primaryid	caseid	indi_drug_seq	indi_pt
1	3894168	1	Parkinsonism
2	3894168	7	Parkinsonism
3	6132163	2	Non-Hodgkin's lymphoma
4	6445504	7	Nausea
5	6671018	8	Foetal exposure during pregnancy

- Reaction Table



# primaryid	caseid	pt	drug_rec_act
1	3894168	Agitation	NaN
2	9504195	Bile duct obstruction	NaN
3	9504195	Cholangitis acute	NaN
3	9504195	Decreased appetite	NaN
3	9504195	Depression	NaN
6	9504195	Duodenal ulcer	NaN
7	9504195	Dyspnoea	NaN
8	9504195	Road traffic accident	NaN

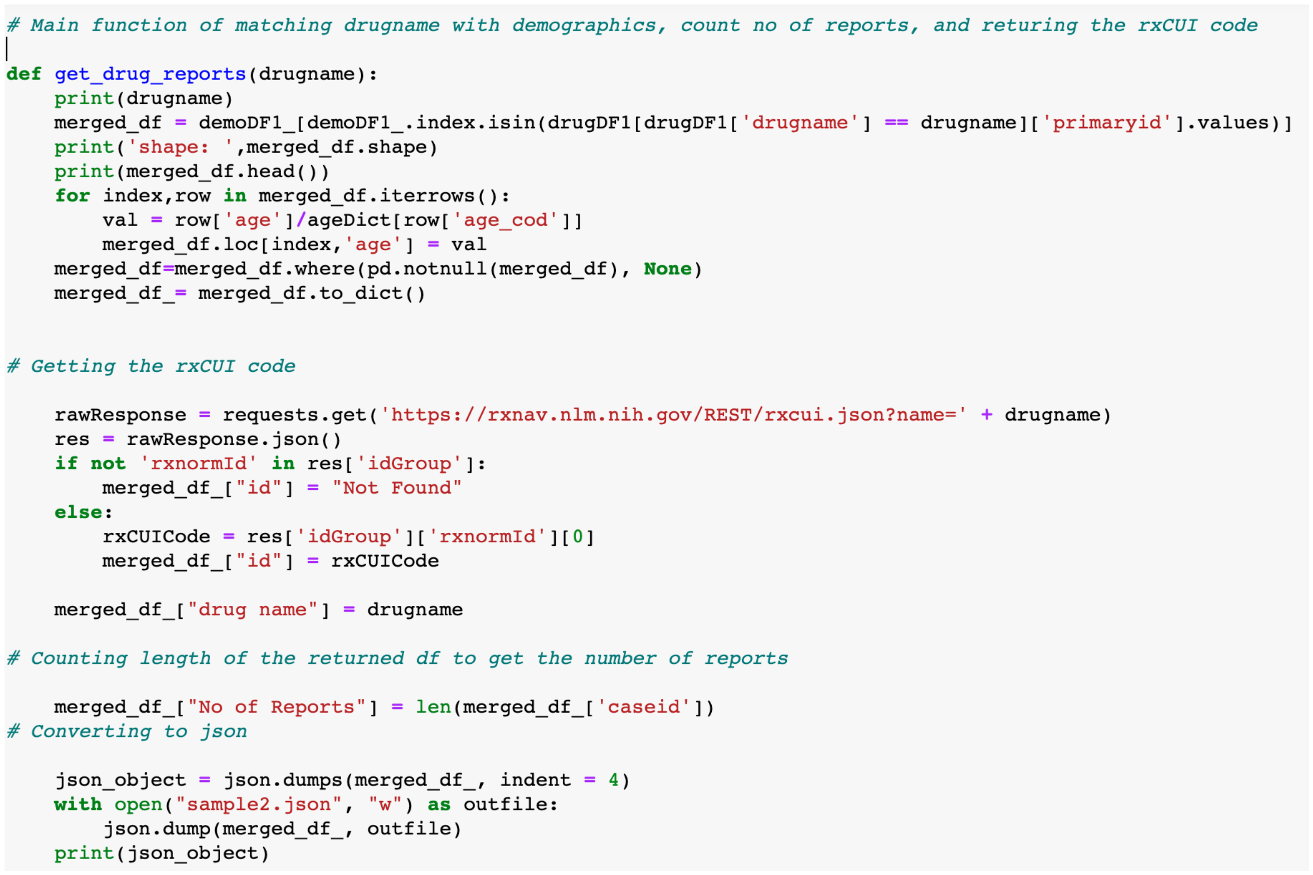
- Outcome Table



# primaryid	caseid	outc_cod
1	3762458	HO
2	3764613	HO
2	5959328	OT
2	5996530	LT
5	6110401	DE
6	12912233	CA
7	12951437	LT

4. Appendix D

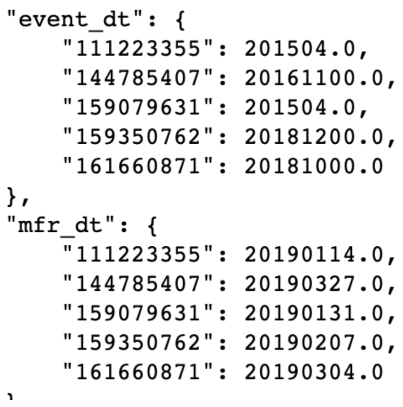
1. Knowledge Extraction Script



```python
# Main function of matching drugname with demographics, count no of reports, and returing the rxCUI code

def get_drug_reports(drugname):
    print(drugname)
    merged_df = demoDF1_[demoDF1_.index.isin(drugDF1[drugDF1['drugname'] == drugname]['primaryid'].values)]
    print('shape: ',merged_df.shape)
    print(merged_df.head())
    for index,row in merged_df.iterrows():
        val = row['age']/ageDict[row['age_cod']]
        merged_df.loc[index,'age'] = val
    merged_df=merged_df.where(pd.notnull(merged_df), None)
    merged_df_= merged_df.to_dict()


# Getting the rxCUI code

    rawResponse = requests.get('https://rxnav.nlm.nih.gov/REST/rxcui.json?name=' + drugname)
    res = rawResponse.json()
    if not 'rxnormId' in res['idGroup']:
        merged_df_["id"] = "Not Found"
    else:
        rxCUICode = res['idGroup']['rxnormId'][0]
        merged_df_["id"] = rxCUICode

    merged_df_["drug name"] = drugname

# Counting length of the returned df to get the number of reports

    merged_df_["No of Reports"] = len(merged_df_['caseid'])
# Converting to json

    json_object = json.dumps(merged_df_, indent = 4)
    with open("sample2.json", "w") as outfile:
        json.dump(merged_df_, outfile)
    print(json_object)
```

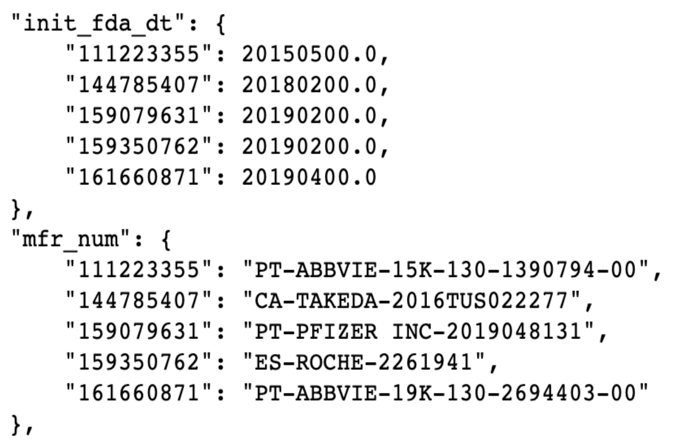
2. Analysis Results

I. Adverse event date and manufacture date.



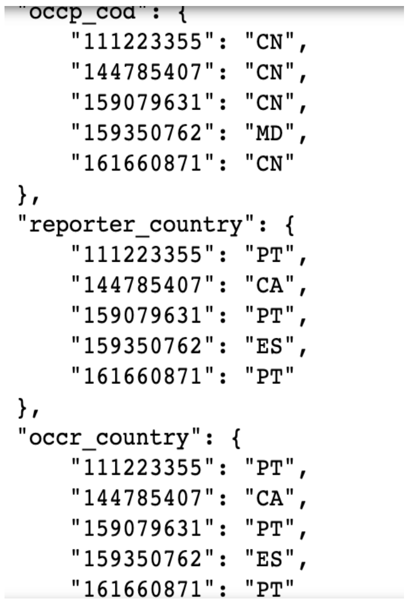
```
"event_dt": {
    "111223355": 201504.0,
    "144785407": 20161100.0,
    "159079631": 201504.0,
    "159350762": 20181200.0,
    "161660871": 20181000.0
},
"mfr_dt": {
    "111223355": 20190114.0,
    "144785407": 20190327.0,
    "159079631": 20190131.0,
    "159350762": 20190207.0,
    "161660871": 20190304.0
```

II. Manufacturer number and date when FDA received the adverse report



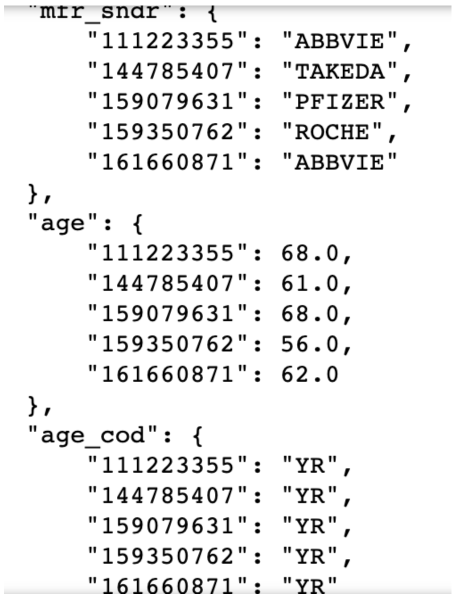
```
"init_fda_dt": {
    "111223355": 20150500.0,
    "144785407": 20180200.0,
    "159079631": 20190200.0,
    "159350762": 20190200.0,
    "161660871": 20190400.0
},
"mfr_num": {
    "111223355": "PT-ABBVIE-15K-130-1390794-00",
    "144785407": "CA-TAKEDA-2016TUS022277",
    "159079631": "PT-PFIZER INC-2019048131",
    "159350762": "ES-ROCHE-2261941",
    "161660871": "PT-ABBVIE-19K-130-2694403-00"
},
```

III. Reporter occupation code, country and occurrence country



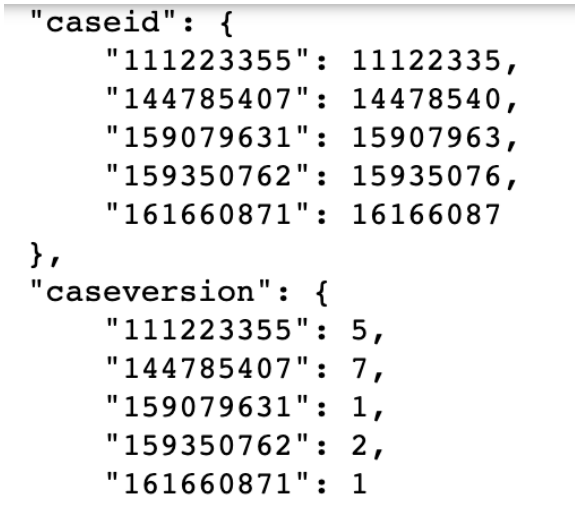
```
"occp_cod": {
    "111223355": "CN",
    "144785407": "CN",
    "159079631": "CN",
    "159350762": "MD",
    "161660871": "CN"
},
"reporter_country": {
    "111223355": "PT",
    "144785407": "CA",
    "159079631": "PT",
    "159350762": "ES",
    "161660871": "PT"
},
"occr_country": {
    "111223355": "PT",
    "144785407": "CA",
    "159079631": "PT",
    "159350762": "ES",
    "161660871": "PT"
```

IV. Manufacturer sender, age and age code



```
"mfr_sndr": {
    "111223355": "ABBVIE",
    "144785407": "TAKEDA",
    "159079631": "PFIZER",
    "159350762": "ROCHE",
    "161660871": "ABBVIE"
},
"age": {
    "111223355": 68.0,
    "144785407": 61.0,
    "159079631": 68.0,
    "159350762": 56.0,
    "161660871": 62.0
},
"age_cod": {
    "111223355": "YR",
    "144785407": "YR",
    "159079631": "YR",
    "159350762": "YR",
    "161660871": "YR"
```

V. Case id and case version



```
"caseid": {
    "111223355": 11122335,
    "144785407": 14478540,
    "159079631": 15907963,
    "159350762": 15935076,
    "161660871": 16166087
},
"caseversion": {
    "111223355": 5,
    "144785407": 7,
    "159079631": 1,
    "159350762": 2,
    "161660871": 1
```