# BUSINESS INTELLIGENCE COURSEWORK REPORT

**Table of contents:**

# 1. INTRODUCTION

## 1.1 The client

The scenario follows the narrative of a veterinary clinic, offering specialized veterinary services, comprised of a number of experienced experts in the field. This clinic wishes to gain a more straight-forward and in-depth access to available data that will help them in their decision making process. This will be achieved with the development of a fully integrated business intelligence (BI) system.

Since information provided for the clinic was limited a combination of imagination, research and existing knowledge was used to make some assumptions regarding the client that would help the development and the deployment processes.

More specifically it was assumed that the clinic would aim to offer specialized veterinary services to companion animals on a daily basis with minimal breaks, while also they strive to disseminate their scientific knowledge and experience to all clinic associates who wish to participate in its activities.

From a technological view, it was assumed that the members of the clinic can be classified as “basic” users of digital software, possessing a fundamental knowledge of computers with no real expertise or preference. It can also be safely assumed that the clients will, most likely, not be willing to allocate any funds towards further expanding their physical IT infrastructure.

On the topic of infrastructure there were also some assumptions made. Expanding upon the previous statements it can be said that the members of the clinic have access to at least a single computer, with average characteristics, in their working environment along with a reliable connection to the internet. Centralized or decentralized large volume systems were assumed to not exist in either a physical or a digital form. These include systems like servers and databases. Last but not least, it was assumed that the specialized experts do not currently use any other form of external data source, such as an API, and largely rely on their, and their colleagues, expertise.

## 1.2 Project requirements, planning and problem analysis

The above assumptions directly translated into the scope of the work of this project, the end goals of which are numbered below:

1. A data extraction architecture.
2. A local and minimal cost fully operational database.
3. An efficient ETL pipeline to perform the necessary data manipulation.
4. An efficient DW architecture.
5. An interactive and intuitive front-end interface that users can interact with.

Last but not least it is important to note that any business analytics functionality, other than informational reports, is not implemented since it was assumed that there was no significant advantage to be gained compared to traditional methods, when it comes to a veterinary clinic business environment.

Once the requirements where laid out and understood the next step was to decide upon an organization plan to help with distributing the workload during the course of the designated timeline. For this, we made use of an agile workflow pattern with 2-week iteration intervals. Each iteration included profiling, download of data, ETL pipeline execution and finally a DW deployment with each part of the project being refined with each iteration.

To help us keep on track we had a few tools in our disposal which focused on improving the efficiency of the process. We specifically made use of the following tools for the corresponding purposes throughout the entirety of the process:

- GitHub repositories for code sharing and versioning.
- A Gantt chart for an overview of the workflow and the focus distribution throughout a particular week.
- Trello for task management and
- Microsoft teams for communicating.

It is worth mentioning here that although an iteration development pattern was used, for the scope of this report we are going present each step separately in its entirety.

## 2. DATA REQUIREMENTS, SYSTEM DESIGN AND DEVELOPMENT

### 2.1 IT systems

The system to be integrated heavily relies on two open data sources the Food and Drug Administration's (FDA) veterinary API which includes data separated in quarters, dating back all the way to 1987 until today. The FDA's API is also continuously expanding year by year. There was a second API that was a candidate for implementing it into the system, namely theDogAPI. However, although a staging environment was created for it and a download script was developed for it, it was ultimately decided to not utilize this source since it was incompatible with the main database used for this project. The second source is theDogAPI which, as its name suggests, specializes in information for dog pets.

The system will be built upon open source software, namely the python and SQL development languages will be used.

#### 2.1.1 FDA API

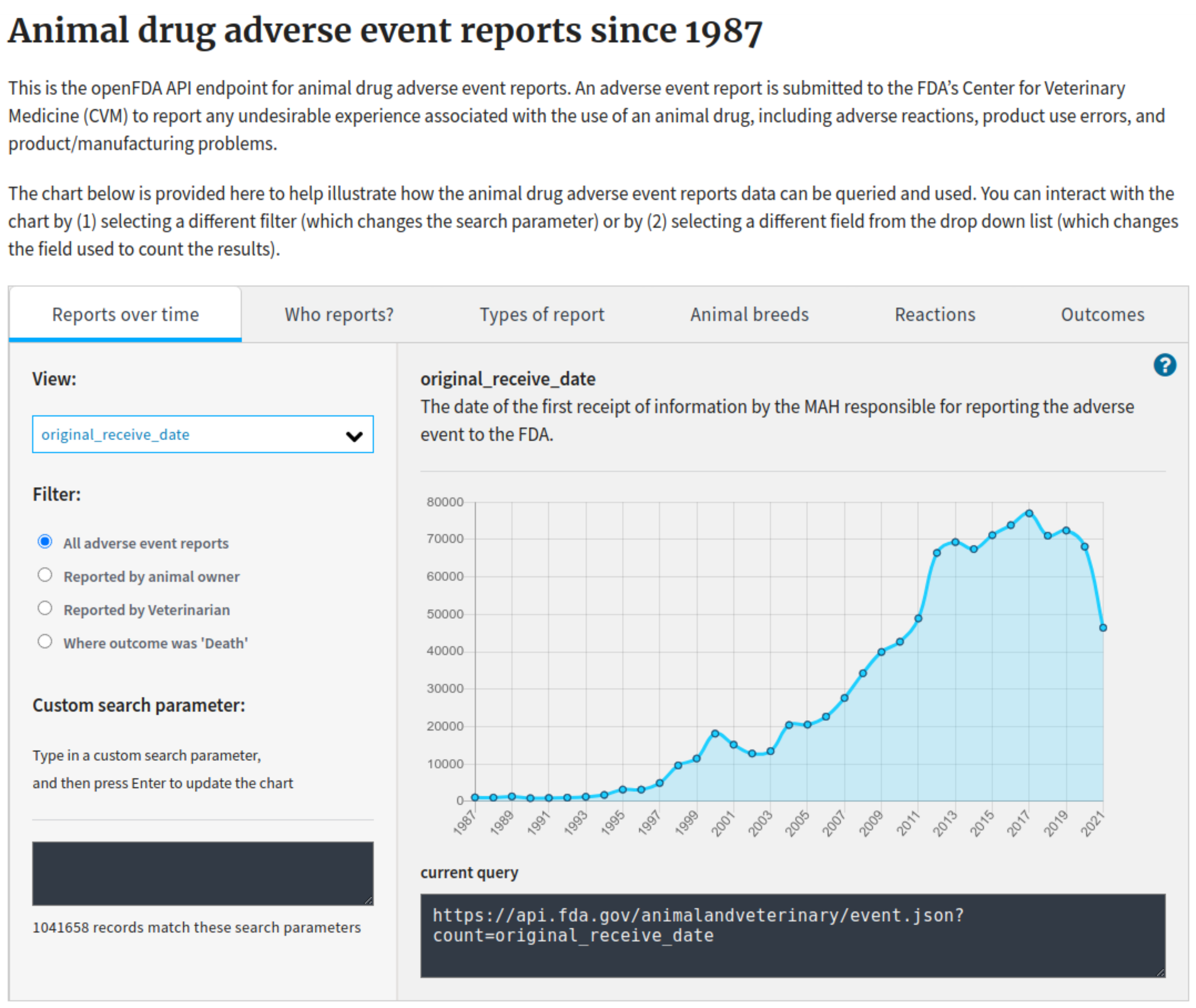
##### 2.1.1.1 Introduction

So the data source for this project is a database build from the FDA where veterinary experts and animal owners are encouraged to report drug-related adverse events where animals were involved. Reporters are allowed to enter information about the event ranging from the drug that was used to general medical information regarding the animal such as any recent procedures performed on it, bloodwork and even x-ray findings. The database, as of the time of this report, has a collection of more than 1,07 million reported incidents and since more than 1 animal can be part of a single incident the total number of individual animal cases exceeds 95 million.

It was outlined earlier that the FDA encourages individuals to report cases of their own through the FDA's platform. Although, this helped the database reach the numbers mentioned above it also introduced a considerable amount of data with no analytical value. Thus, a process had to take place to filter the raw data and remove any unwanted entries, namely profiling.

#### 2.1.1.2 Profiling

Upon designing a satisfying project plan, what followed was an initial profiling process. For this, the first point of reference was the FDA API's documentation page [1] which provided detailed information about the data the system would handle. This included information from titles, descriptions and data types of the fields, to the location of the fields inside the nested JSON files. What is more, on the website there also existed a profiling tool which allowed one to access the entire dataset and construct custom queries. The tool visualized the result into suitable graphs that matched the one's query. A screenshot of the interactive interface can be seen below (see image 1)



Animal drug adverse event reports since 1987

This is the openFDA API endpoint for animal drug adverse event reports. An adverse event report is submitted to the FDA's Center for Veterinary Medicine (CVM) to report any undesirable experience associated with the use of an animal drug, including adverse reactions, product use errors, and product/manufacturing problems.

The chart below is provided here to help illustrate how the animal drug adverse event reports data can be queried and used. You can interact with the chart by (1) selecting a different filter (which changes the search parameter) or by (2) selecting a different field from the drop down list (which changes the field used to count the results).

Reports over time | Who reports? | Types of report | Animal breeds | Reactions | Outcomes

View: original_receive_date

Filter:
- [✓ All adverse event reports]
- Reported by animal owner
- Reported by Veterinarian
- Where outcome was 'Death'

Custom search parameter:

Type in a custom search parameter, and then press Enter to update the chart

1041658 records match these search parameters

original_receive_date
The date of the first receipt of information by the MAH responsible for reporting the adverse event to the FDA.

current query

```
https://api.fda.gov/animalandveterinary/event.json?
count=original_receive_date
```

*Figure 1: FDA's profiling tool*

After experimenting with the tool and querying for every field available the first set of adjustments could be derived for the data. More specifically, it was observed that the following fields would be discarded due to a considerable percentage of them being missing. (At this stage, the threshold corresponds to more than 50% of the data missing from the field:

- animal.age_max
- animal.weight_max
- drug.lot_expiration
- drug.lot_number
- duration.unit
- duration.value
- secondary_reporter
- type_of_information

What was also observed was that there existed fields dominated by a single value:

- animal.weight_unit
- receiver.city
- receiver.country
- receiver.organization
- receiver.postal_code
- receiver.state
- receiver.street_address
- health_assessment_prior_to_exposure.assessed_by

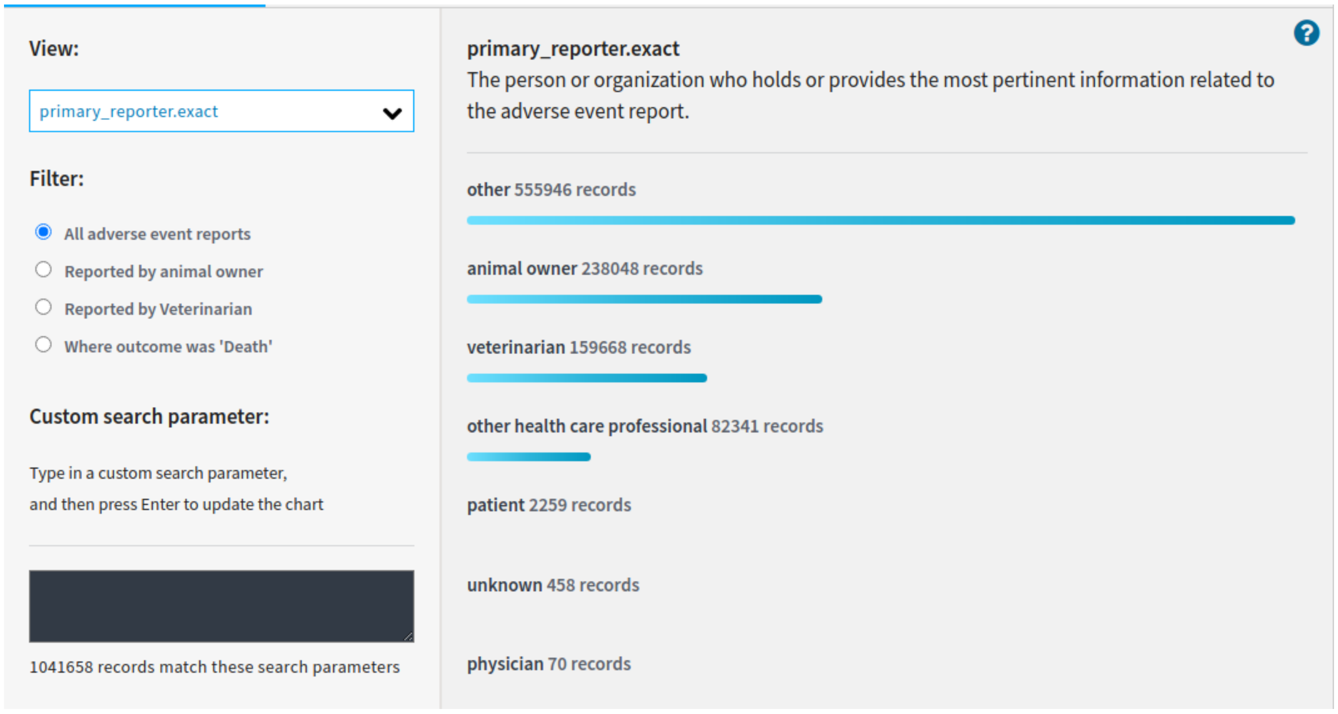
In these cases it was decided to drop these fields and implement their value in the corresponding numeric field associated with them. For fields that this did not apply to (e.g. receiver object data fields and health_assessment) it was decided to drop the fields and exclude this information completely since they would not add any statistical value to the system.

It was also during this stage that it was decided which fields would be dropped due to ethical and/or censoring reasons. Such fields are the ones including data which could potentially have an effect on an external person and/or organization who does not directly associate with the veterinary clinic. The same applies for data that could cause a decision bias to the veterinary experts using the system for reasons not related to scientific facts. An example of such data would be the name of a particular drug or manufacturer. What follows is a list of the fields that satisfy the aforementioned requirements:

- drug.lot_expiration
- drug.lot_number
- drug.manufacturer.name
- drug.manufacturer.registration_number
- drug.number_of_defective_items
- drug.number_of_items_returned
- product_ndc

It was also during this stage that outlier checks were performed on both numeric and text fields. Both categories presented outliers all of which were crosschecked before deciding if it was worth maintaining into the dataset. There are two cases of outliers that are worth discussing, one coming from the number of animals affected field and the other one from the primary reporter field.

To start from the latter, a search query about the individual who reported the incident revealed the following:



View:

primary_reporter.exact

Filter:

- [✓ All adverse event reports]
- Reported by animal owner
- Reported by Veterinarian
- Where outcome was 'Death'

Custom search parameter:

Type in a custom search parameter, and then press Enter to update the chart

1041658 records match these search parameters

primary_reporter.exact

The person or organization who holds or provides the most pertinent information related to the adverse event report.

other 555946 records

animal owner 238048 records

veterinarian 159668 records

other health care professional 82341 records

patient 2259 records

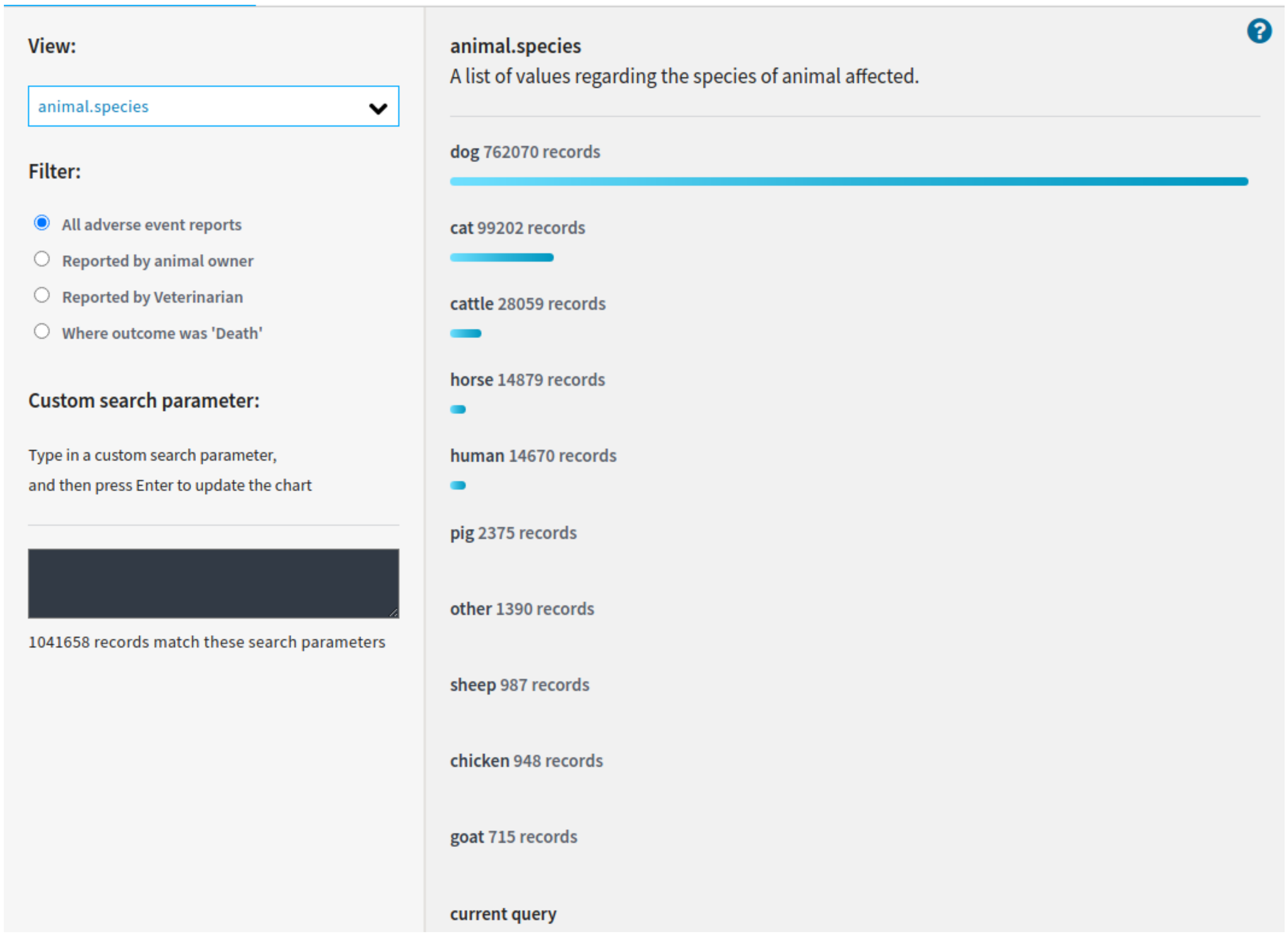
unknown 458 records

physician 70 records

*Figure 2: Querying FDA's interactive graph tool about the primary reporter field. Note that in the tool this field is referred to as primary_reporter.exact and not primary_reporter. The results indicate that there are human related incidents cataloged inside the dataset.*

A total of 2259 cases, totaling to 0.2% of the total incidents, were reported by the patient! This directly translates to the fact that there exist incidents within the database where the adverse event is related to a human and not an animal, a fact further supported by querying for information related to the animal's species.

There, the results are even more surprising with a total of 14879 cases reported for human as the patient. Similarly striking is that 'human' is the 5th most reported species throughout the whole dataset. However, that does not affect its integrity since 'human' for animal species account for a total of 1.43% of the entire dataset.


View:

animal.species

Filter:

- [✓ All adverse event reports]
- Reported by animal owner
- Reported by Veterinarian
- Where outcome was 'Death'

Custom search parameter:

Type in a custom search parameter, and then press Enter to update the chart

1041658 records match these search parameters

animal.species
A list of values regarding the species of animal affected.

dog 762070 records

cat 99202 records

cattle 28059 records

horse 14879 records

human 14670 records

pig 2375 records

other 1390 records

sheep 987 records

chicken 948 records

goat 715 records

current query

*Figure 3: Querying FDA's interactive graph tool about the animal species field. The query reveals more human-related cases for a total of 14670.*

It would be intuitive to discard all of these cases since they are not related to animals and they represent a statistically insignificant amount of the dataset, however a closer look at the individual incidents revealed that information contained within these cases has statistical value for a veterincary clinic.

| Veddra_term_name | Active_ingredient_name |
|---|---|
| Unclassifiable adverse event | Selamectin |
| Accidental exposure | Selamectin |
| Cough | Selamectin |
| Rash | Selamectin |
| Pruritus | Selamectin |

*Table 1: An example incident where 'human' was reported as the animal species.*

Table 1 provides an in-depth look into one of such an incident. It can be seen that 5 individual incidents were nested inside the single reported case all of which describe adverse events related to human-animal interactions, a phenomenon which is not uncommon for veterinary clinics to deal with. Thus, it can be derived that human-related entries can provide a useful insight to the veterinary experts dealing with similar cases.

Moving on to the numeric outliers found inside the number_of_animals_affected field. So, this data fields indicates some of the largest outliers from the whole dataset, with values ranging from 1 to 1 million. Upon inspecting cases with extraordinary large values one can observe that the majority of them refers to cases where animals originating from production lines were involved, such as chickens, pigs and cattles. It is not uncommon for such species to grow and live inside large-scale facilities with high animal-per-square-meter densities. Battery cages for example are infamous for containing animal populations in the thousands. So, it can be assumed that reported cases with extreme numbers are representations of real-world events and thus hold statistical value for veterinary experts. Thus, it was decided that all outliers would not be discarded.

This completed the first stage of the development process and led to the next step which would be the design, development and deployment of the staging environment.

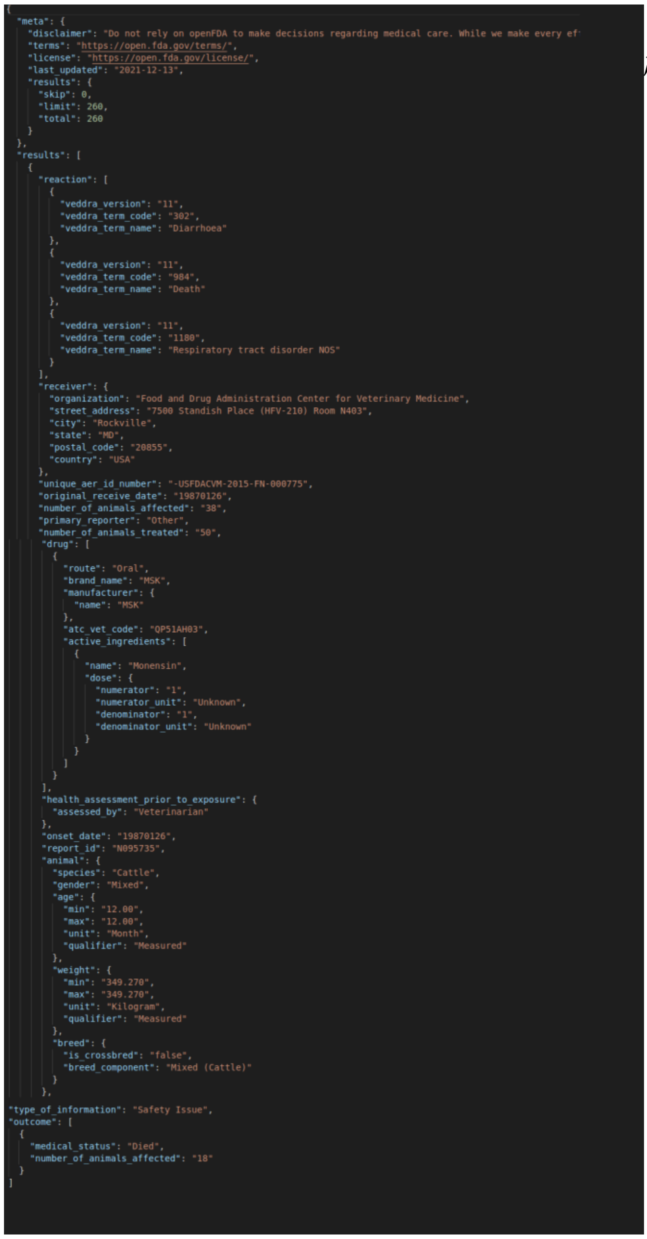
## 2.2 Staging

For the creation and deployment of the staging environment there were two options that were developed and tested during the course of the project. Both approaches had to follow the convention that all raw data had to be stored without omitting any fields and/or data entries. All of the rules that were a consequence of the profiling process were to be applied during the Extract Load Transform (ETL) process of the system. This was made in an attempt to have a one-to-one representation of the online database downloaded locally.

### 2.2.1 Approach 1

The first approach followed a minimal workflow and consisted of a single python script. The script automated the download process for all available (140 at the time of this report) data files available from the FDA’s website and stored them in an orderly fashion in JSON format. The proof of concept was promising. The system could download the entire dataset in short amount of time, store it in an orderly manner, allow access to the entire raw dataset and it required minimal human intervention allowing it to be scheduled in advance.

This idea was developed and the system operated properly but was discarded since it did not serve as a reliable foundation for the next phases of the system. This happened due to the structure of the JSON files that had to be handled in order to achieve the desired Data Warehouse (DW) schema.



```
{
  "meta": {
    "disclaimer": "Do not rely on openFDA to make decisions regarding medical care. While we make every ef
    "terms": "https://open.fda.gov/terms/",
    "license": "https://open.fda.gov/license/",
    "last_updated": "2021-12-13",
    "results": {
      "skip": 0,
      "limit": 260,
      "total": 260
    }
  },
  "results": [
    {
      "reaction": [
        {
          "veddra_version": "11",
          "veddra_term_code": "302",
          "veddra_term_name": "Diarrhoea"
        },
        {
          "veddra_version": "11",
          "veddra_term_code": "984",
          "veddra_term_name": "Death"
        },
        {
          "veddra_version": "11",
          "veddra_term_code": "1180",
          "veddra_term_name": "Respiratory tract disorder NOS"
        }
      ],
      "receiver": {
        "organization": "Food and Drug Administration Center for Veterinary Medicine",
        "street_address": "7500 Standish Place (HFV-210) Room N403",
        "city": "Rockville",
        "state": "MD",
        "postal_code": "20855",
        "country": "USA"
      },
      "unique_aer_id_number": "-USFDACVM-2015-FN-000775",
      "original_receive_date": "19870126",
      "number_of_animals_affected": "38",
      "primary_reporter": "Other",
      "number_of_animals_treated": "50",
      "drug": [
        {
          "route": "Oral",
          "brand_name": "MSK",
          "manufacturer": {
            "name": "MSK"
          },
          "atc_vet_code": "QP51AH03",
          "active_ingredients": [
            {
              "name": "Monensin",
              "dose": {
                "numerator": "1",
                "numerator_unit": "Unknown",
                "denominator": "1",
                "denominator_unit": "Unknown"
              }
            }
          ]
        }
      ],
      "health_assessment_prior_to_exposure": {
        "assessed_by": "Veterinarian"
      },
      "onset_date": "19870126",
      "report_id": "N095735",
      "animal": {
        "species": "Cattle",
        "gender": "Mixed",
        "age": {
          "min": "12.00",
          "max": "12.00",
          "unit": "Month",
          "qualifier": "Measured"
        },
        "weight": {
          "min": "349.270",
          "max": "349.270",
          "unit": "Kilogram",
          "qualifier": "Measured"
        },
        "breed": {
          "is_crossbred": "false",
          "breed_component": "Mixed (Cattle)"
        }
      },
"type_of_information": "Safety Issue",
"outcome": [
  {
    "medical_status": "Died",
    "number_of_animals_affected": "18"
  }
]
```

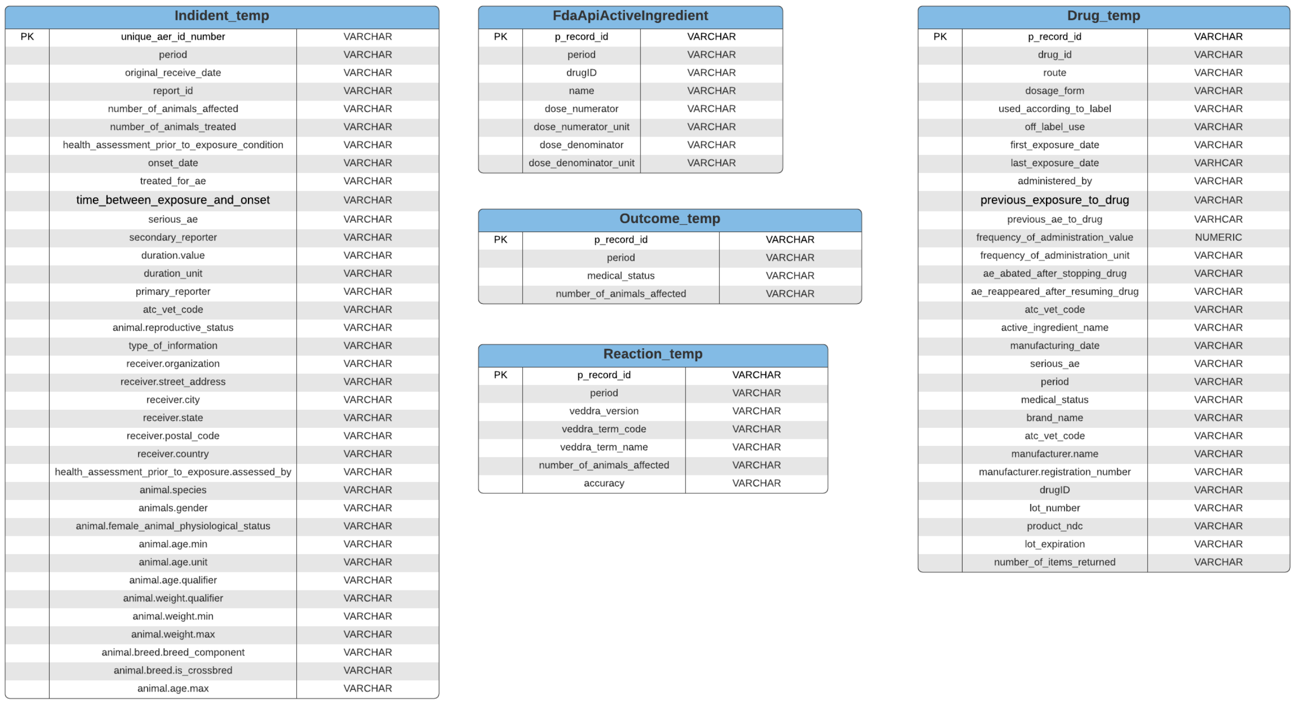
*Figure 4: JSON file structure as it is downloaded from the FDA's database.*

The JSON file was, primarily, comprised of information that was nested which made python queries to extract the data increasingly more difficult as the nesting increased. What is more, mentioned on the API documentation is that the dataset is constantly changing with new fields being added, changes to the structure of the file and the removal of redundant fields. It is easy to understand how this fact can easily have considerable consequences to the pipeline of our system since constant monitoring for each change is required.

For these reasons it was decided that the staging pipeline would follow approach 2 which involved a more complicated and, consequently, longer to execute system.

### 2.2.2 Approach 2

Approach 2 involved fetching the results from a request to the API and, through the use of pandas [2] dataframes , normalizing the data and storing in into tables with pre-defined structure. The structure of the tables can be seen below:



| Indident_temp | | |
|---|---|---|
| PK | unique_aer_id_number | VARCHAR |
| | period | VARCHAR |
| | original_receive_date | VARCHAR |
| | report_id | VARCHAR |
| | number_of_animals_affected | VARCHAR |
| | number_of_animals_treated | VARCHAR |
| | health_assessment_prior_to_exposure_condition | VARCHAR |
| | onset_date | VARCHAR |
| | treated_for_ae | VARCHAR |
| | time_between_exposure_and_onset | VARCHAR |
| | serious_ae | VARCHAR |
| | secondary_reporter | VARCHAR |
| | duration.value | VARCHAR |
| | duration_unit | VARCHAR |
| | primary_reporter | VARCHAR |
| | atc_vet_code | VARCHAR |
| | animal.reproductive_status | VARCHAR |
| | type_of_information | VARCHAR |
| | receiver.organization | VARCHAR |
| | receiver.street_address | VARCHAR |
| | receiver.city | VARCHAR |
| | receiver.state | VARCHAR |
| | receiver.postal_code | VARCHAR |
| | receiver.country | VARCHAR |
| | health_assessment_prior_to_exposure.assessed_by | VARCHAR |
| | animal.species | VARCHAR |
| | animals.gender | VARCHAR |
| | animal.female_animal_physiological_status | VARCHAR |
| | animal.age.min | VARCHAR |
| | animal.age.unit | VARCHAR |
| | animal.age.qualifier | VARCHAR |
| | animal.weight.qualifier | VARCHAR |
| | animal.weight.min | VARCHAR |
| | animal.weight.max | VARCHAR |
| | animal.breed.breed_component | VARCHAR |
| | animal.breed.is_crossbred | VARCHAR |
| | animal.age.max | VARCHAR |

| FdaApiActiveIngredient | | |
|---|---|---|
| PK | p_record_id | VARCHAR |
| | period | VARCHAR |
| | drugID | VARCHAR |
| | name | VARCHAR |
| | dose_numerator | VARCHAR |
| | dose_numerator_unit | VARCHAR |
| | dose_denominator | VARCHAR |
| | dose_denominator_unit | VARCHAR |

| Outcome_temp | | |
|---|---|---|
| PK | p_record_id | VARCHAR |
| | period | VARCHAR |
| | medical_status | VARCHAR |
| | number_of_animals_affected | VARCHAR |

| Reaction_temp | | |
|---|---|---|
| PK | p_record_id | VARCHAR |
| | period | VARCHAR |
| | veddra_version | VARCHAR |
| | veddra_term_code | VARCHAR |
| | veddra_term_name | VARCHAR |
| | number_of_animals_affected | VARCHAR |
| | accuracy | VARCHAR |

| Drug_temp | | |
|---|---|---|
| PK | p_record_id | VARCHAR |
| | drug_id | VARCHAR |
| | route | VARCHAR |
| | dosage_form | VARCHAR |
| | used_according_to_label | VARCHAR |
| | off_label_use | VARCHAR |
| | first_exposure_date | VARCHAR |
| | last_exposure_date | VARHCAR |
| | administered_by | VARCHAR |
| | previous_exposure_to_drug | VARCHAR |
| | previous_ae_to_drug | VARHCAR |
| | frequency_of_administration_value | NUMERIC |
| | frequency_of_administration_unit | VARCHAR |
| | ae_abated_after_stopping_drug | VARCHAR |
| | ae_reappeared_after_resuming_drug | VARCHAR |
| | atc_vet_code | VARCHAR |
| | active_ingredient_name | VARCHAR |
| | manufacturing_date | VARCHAR |
| | serious_ae | VARCHAR |
| | period | VARCHAR |
| | medical_status | VARCHAR |
| | brand_name | VARCHAR |
| | atc_vet_code | VARCHAR |
| | manufacturer.name | VARCHAR |
| | manufacturer.registration_number | VARCHAR |
| | drugID | VARCHAR |
| | lot_number | VARCHAR |
| | product_ndc | VARCHAR |
| | lot_expiration | VARCHAR |
| | number_of_items_returned | VARCHAR |

*Figure 5: Entity relational diagram for staging environment. Data is split into tables according to the nesting of the JSON file. It should be noted however that the animal-related dimensions are nested inside the 'FdaApiIncident' table.*

As it can be seen, all data fields have been kept from the JSON file and the data is not split up into 5 tables. The categorization of the data follows the nesting pattern of the JSON file. Each table also has a common column which stores the natural key ("unique_aer_id_number") of the incident, as assigned to it from the FDA, in order to maintain a connection between the tables.

Furthermore, we had to add an additional key to maintain the relationship between the drug and its active ingredients. Thus in each iteration of the process, a key drug_id was generated to link drugs and active ingredients. The unique aer key was also present to ensure the data extraction validity.

It is also during this stage when it was revealed that an incident can have nested more than one instance of a reaction, an outcome or a drug. For example, it could be the case that a single incident reports adverse events for 5 animals each of which had a different reaction to a specific substance and subsequently a different outcome. For such cases, the algorithm was calibrated to extract each case separately so as to differentiate the cases. It follows that the rest of the system treated the new extracted cases as separated incidents while maintaining however the original incident's natural key. To add to this connection, a key was also assigned to the corresponding incidents to help identify individual reactions, outcomes and / or drugs from within their 'parent' incidents.

Last but not least, it is worth mentioning that all data field types where assigned to be variable characters at this stage due to inconsistencies to the format of the entries. These inconsistencies would be later regulated during the ETL stage

With the staging phase complete, what follows is the ETL process.

## 2.3 Extract Transform Load (ETL)

The ETL phase is a pipeline, each step of which specializes on a particular task with the end goal of loading the data, to the DW environment of the database, in a digestible form for the front-end part of the system to query on.

Below is a diagram of the ETL pipeline:

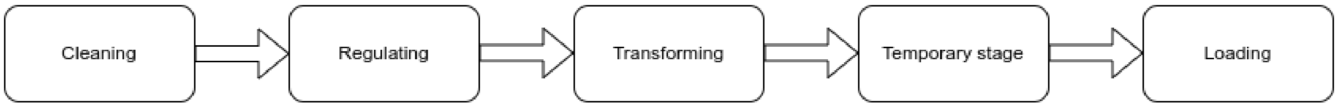


*Figure 6: Extract, Transform and Load (ETL) pipeline steps visualized.*

Each step is designed to perform a specialized task on the data stored.  This is where all of the rules derived from the first profiling process will be applied.

Taking a closer look at each step.

### 2.3.1 Cleaning

As the name suggests, the first step of the pipeline cleans the data. This step can be broken down into the following key stages:

- Dropping unwanted columns according to the rules specified during the first profiling phase.
- Unifying how unknown data is presented.
- Regulating date type values.
- Imputing data when necessary.

The columns dropped matched the requirements specified during the initial profiling process.

### 2.3.2 Regulating

Unification of data involved operations about all different data types present in the database. Namely:

- Missing values of numeric fields were converted to 0, so that they would have no impact on algebraic operations during the front-end phase.
- Missing values of numeric fields that represented non-operational values (e.g., codes) were replaced by the null value. It must be mentioned here that missing values in these columns were represented using strings values (‘unknown’, ‘not applicable’).

- Missing entries from text and varying character fields were replaced by the value ‘NaN’. This choice was based on the fact that ‘NaN’ was the predominant value used by the FDA to handle such missing entries while other representations, (‘not applicable’, ‘unknown’, ’NOT APPLICABLE’, ‘Unknown’), applied for less than 5% of the total missing values
- Date fields missing entries were replaced with the null value.

Also taking place during this stage of the ETL pipeline was the addition of surrogate keys to all tables that did not have one in the staging environment in the form of ‘drug_id’, ‘animal_id’, ‘active_ingredient_id’, ‘reaction_id’ and ‘outcome_id’. These would later be used to join data from temporary tables and form the fact and dimension tables in the relational data warehouse environment.

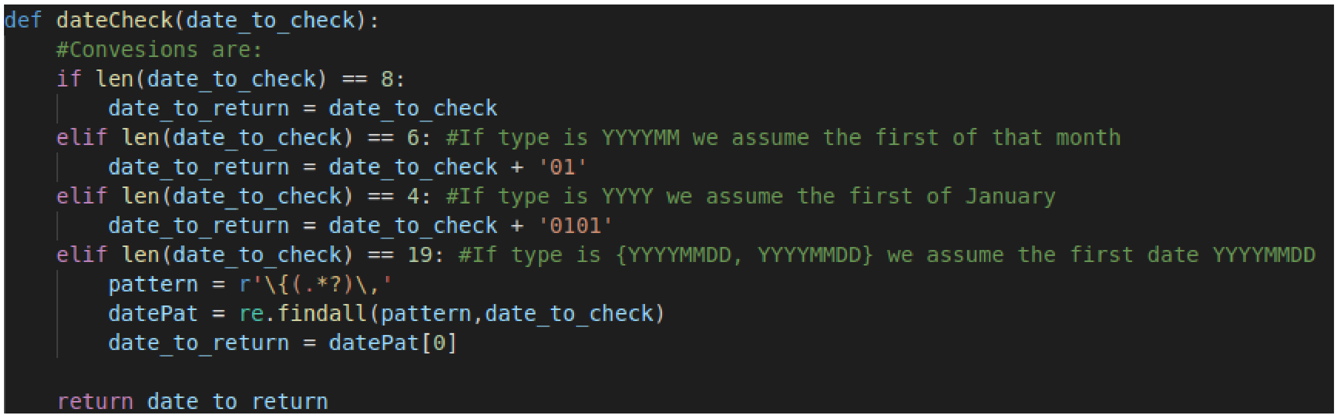
**2.3.3 Transforming**

This is also where a second profiling phase took place. Mistakes made during the previous steps of the process could be more easily spotted. Having a more simplified and targeted, towards the end goal, dataset revealed imperfections of the set that needed to be resolved before reaching the DW environment. It was observed that a a number of entries reporting incidents were not correctly formatted to match their destination data types.

More specifically, discrepancies in the entries mainly appeared in fields containing non string type data but also the string character field ‘breed_component’ was observed to have inconsistencies.

All date type fields did not have a standardized means of representing the data with many incidents having been reported with missing information (in the form of missing month and day information YYYY or just day information YYYYMM) while others were completely missing or had a set of dates into them (in the form {YYYYMMDD, YYYYMMDD}). For consistency it was decided to use the YYYYMMDD format universally, since it was the most commonly used format from the source database. This translated into imputing incomplete data to match the new format. For this, two assumptions applied: All entries with a YYYY format were directly assigned the 1st of January as the default date to complete the entry. While for YYYYMM format the 1st of that month was assigned as the default value. One could potentially argue that this can introduce a bias to our data, however these values were specifically chosen to place the entry to the biggest rounding possible whether that was the month or the year during which the incident took place. It is also worth mentioning here that all missing values where assigned a null value.

A snippet of the code used for these transformations can be seen in the following figure.



```python
def dateCheck(date_to_check):
    #Convesions are:
    if len(date_to_check) == 8:
        date_to_return = date_to_check
    elif len(date_to_check) == 6: #If type is YYYYMM we assume the first of that month
        date_to_return = date_to_check + '01'
    elif len(date_to_check) == 4: #If type is YYYY we assume the first of January
        date_to_return = date_to_check + '0101'
    elif len(date_to_check) == 19: #If type is {YYYYMMDD, YYYYMMDD} we assume the first date YYYYMMDD
        pattern = r'\{(.*?)\,'
        datePat = re.findall(pattern,date_to_check)
        date_to_return = datePat[0]

    return date_to_return
```

*Figure 7: Code snippet of transformation mechanism for date fields. The input value is the corresponding date value one wants to check. The series of if statements was adjusted according to the variation of the field (found by querying the dataset after the staging environment was loaded). A transformation takes place accordingly and the value returned is the regulated value.*

As far as numeric fields are concerned, all data that passed through the filter for missing and / or non numeric values were converted to floats to ensure that they would fit into the numeric data type of the end table. Data which was filtered to have either missing or non-compatible values was imputed in two ways based on what they represented. Numeric fields which served as dimensions in the DW environment were emptied (meaning that their value was replaced by 'Null'). These fields were:

- Veddra_version
- Veddra_term_code

Fact numeric fields, on the other hand, were imputed with 0 in an attempt to reduce the effect they would have on numeric operations during the query stage of the system. These fields were:

- Age_min
- Weight_kg
- Total_number_of_animals_affected
- Total_number_of_animals_treated
- Reaction_number_of_animals_affected
- Outcome_number_of_animals_affected

The breed component field posed the biggest challenge during this process. Profiling revealed large inconsistencies in the way in which the data was recorded, especially apparent on cases with more than 1 animals involved. Table 2 displays three of such instances.

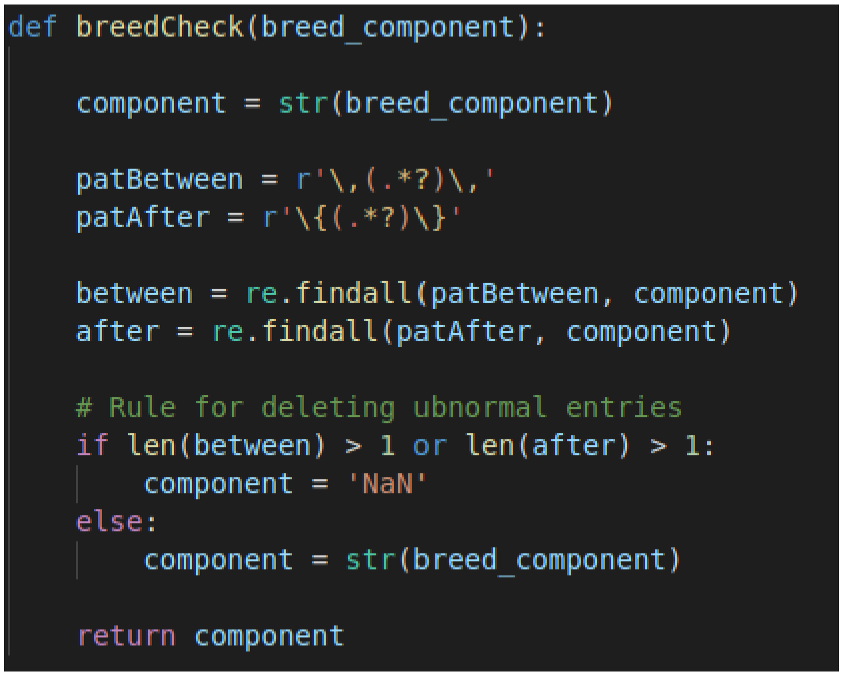
| Breed Component | Animals Affected |
|---|---|
| {"Retriever - Golden","Retriever (unspecified)","Retriever - Golden","Poodle (unspecified)","Retriever - Golden","Poodle | 12 |

| Breed Component | Animals Affected |
|---|---|
| (unspecified)","Retriever - Golden","Poodle (unspecified)","Retriever - Golden","Poodle (unspecified)","Retriever - Golden","Poodle (unspecified)","Retriever - Golden","Poodle (unspecified)","Retriever - Golden","Poodle (unspecified)","Retriever - Golden","Poodle (unspecified)","Retriever - Golden","Poodle (unspecified)","Retriever - Golden","Poodle (unspecified)","Retriever - Golden","Poodle (unspecified)"} | |
| {Greyhound,"Deutsche Dogge, Great Dane","Deutsche Dogge, Great Dane"} | 3 |
| {"Poodle - Miniature","Poodle - Standard",Rottweiler} | 4 |

*Table 2: Instances of bad practices within the breed component field of the dataset.*

It was a common occurrence for breed_components to appear in odd values and/or with inconsistent separation between the multiple entries of the field as seen in the examples above.

The typical representation for breed components inside the field usually come in the form of the first case of the table, with quotes defining a single component and commas separating different components from each other. However, as it can be seen from the second and third cases, that is not standardized across the dataset. There exist a number of entries like these two examples, with a mixture of commas and quotes separating different components. Such entries made developing proper algorithm to distinguish between breed components, an increasingly difficult task. The effort required to do so greatly outweighs the benefits gained from it. Thus, it was decided to filter out such cases, using regular expressions in python, and replace their value with 'NaN'. The low percentage of such cases, accounting for about 0.5% of the total cases, allowed for such a transformation to occur since there was no distinguishable value lost by doing so. Figure 8 is the code snippet of the function performing this filtering.



```python
def breedCheck(breed_component):

    component = str(breed_component)

    patBetween = r'\,(.*?)\,'
    patAfter = r'\{(.*?)\}'

    between = re.findall(patBetween, component)
    after = re.findall(patAfter, component)

    # Rule for deleting ubnormal entries
    if len(between) > 1 or len(after) > 1:
        component = 'NaN'
    else:
        component = str(breed_component)

    return component
```

*Figure 8: Code snippet of breedCheck function, the filtering mechanism for the breed_component field. Two regular expressions identify the variation from the input case and filter out unwanted values. Variation in this case is indicated from the number of in-between values that exist nested inside the breed component field.*

Regarding the rest of the cases, there were no transformations applied to them despite of the number of breed components inside the field. One may argue that it is bad practice to have fields with more than one values inside them, however that is not the case here. Entries inside the breed component field can include more than one values, where here the convention is that a single value is defined as a series of string characters inside double quotes (“ Labrador ”) separated by commas from other values inside the field. What is important to understand however is the fact that animal(s) can be crossbred, as indicated from the corresponding data field. Thus, there can be cases that a crossbred animal(s) have two values inside the corresponding breed component field that are effectively acting as one.  An example of this scenario would be an entry (“Labrador”, “Labrador-Golden retriever”)

This made understanding the data increasingly harder and it was decided that it could also lead to an incorrect representation of the data. Thus, it was decided to again apply a filter which filtered out such abnormalities in the entries and replaced offending entries with ‘Null’.
Also during this stage the data fields age_unit and age_min were regulated. More specifically, all age unit entries, instead of the unknown ones, where assigned the value ‘Year’ which directly translated into the age_min values. The age_unit field contained a total of 9 different values:

- C17998
- Day
- Hour
- Minute
- Month
- NaN

- Second
- Week
- Year

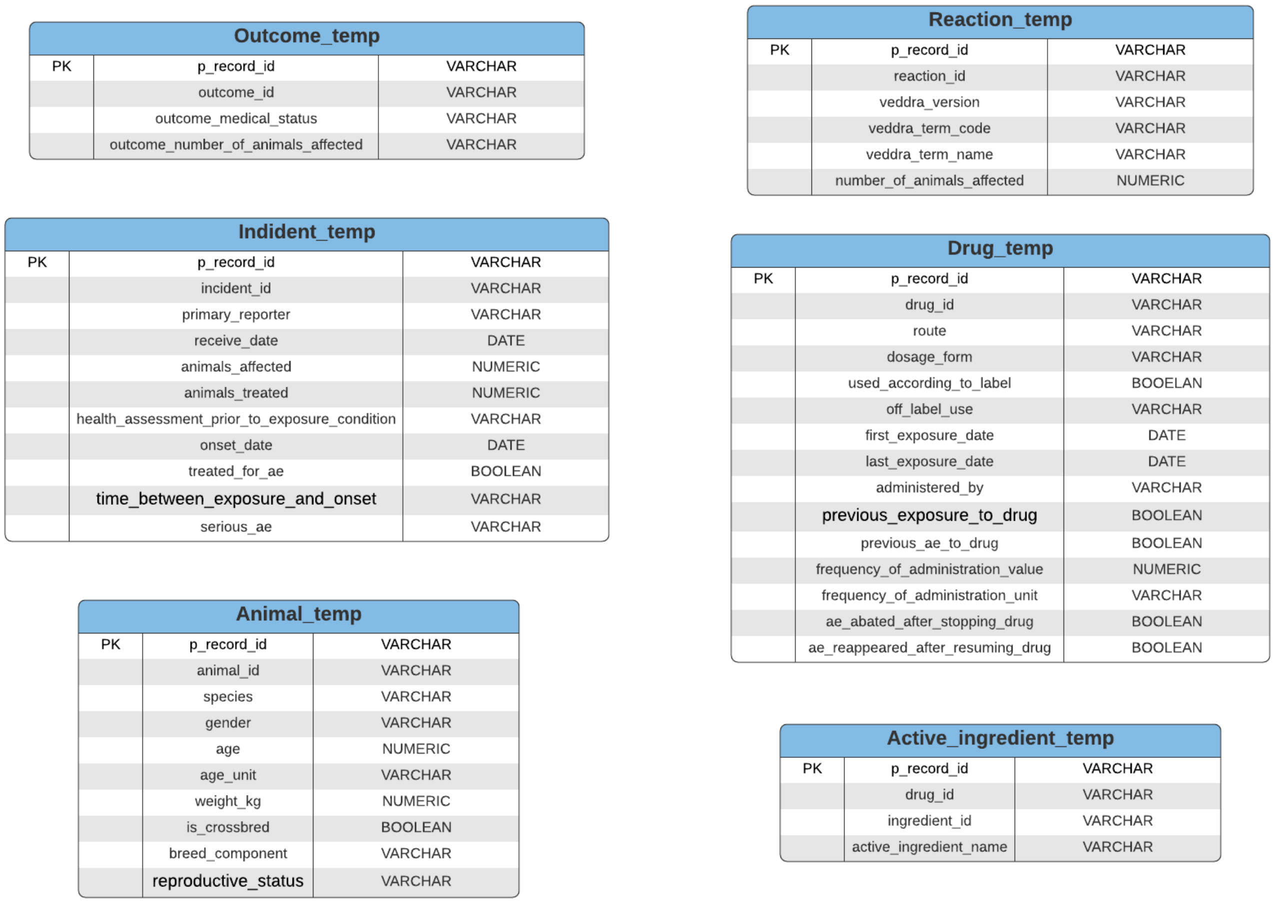
The value that appeared the most was ‘Year’ occupying more than 50% of the entries. Thus, it was decided to convert all other entries, from their corresponding unit, to year. Age_min entries with an age_unit value of ‘Day’ where transformed by dividing them by 365 and correspondingly entries with ‘Month’ and ‘Week’ were divided by 12 and 52. Entries with age_unit ‘C17998’ were also assumed to be years, as query testing revealed that such entries had the value ‘human’ in their corresponding animal.species field.

The remaining age_min entries with corresponding age_unit values of ‘Hour’, ‘Minute’, ‘Second’ were discarded. Regulating these would require a division with 8760, 525600 and 31536000 accordingly. It is easy to understand that both displaying such information and, more importantly, drawing conclusions from it would be very demanding tasks. Thus, it was decided to replace these age_min values with 0.

### 2.3.4 Temporary

Following the addition of the surrogate keys the data moves to a temporary stage positioned after the staging environment and before the final data warehouse environment. This stage acted as a preparation step for loading the data. It served the purpose of arranging the data into tables that would make it easier for the final load, while it also allowed for some preliminary query executions.

The data is now arranged in a manner as indicated by the following entity relational diagram (ERD) where p_record_id is the natural key of the dataset (aer):



**Outcome_temp**

| | | |
|---|---|---|
| PK | p_record_id | VARCHAR |
| | outcome_id | VARCHAR |
| | outcome_medical_status | VARCHAR |
| | outcome_number_of_animals_affected | VARCHAR |

**Reaction_temp**

| | | |
|---|---|---|
| PK | p_record_id | VARCHAR |
| | reaction_id | VARCHAR |
| | veddra_version | VARCHAR |
| | veddra_term_code | VARCHAR |
| | veddra_term_name | VARCHAR |
| | number_of_animals_affected | NUMERIC |

**Indident_temp**

| | | |
|---|---|---|
| PK | p_record_id | VARCHAR |
| | incident_id | VARCHAR |
| | primary_reporter | VARCHAR |
| | receive_date | DATE |
| | animals_affected | NUMERIC |
| | animals_treated | NUMERIC |
| | health_assessment_prior_to_exposure_condition | VARCHAR |
| | onset_date | DATE |
| | treated_for_ae | BOOLEAN |
| | time_between_exposure_and_onset | VARCHAR |
| | serious_ae | VARCHAR |

**Drug_temp**

| | | |
|---|---|---|
| PK | p_record_id | VARCHAR |
| | drug_id | VARCHAR |
| | route | VARCHAR |
| | dosage_form | VARCHAR |
| | used_according_to_label | BOOELAN |
| | off_label_use | VARCHAR |
| | first_exposure_date | DATE |
| | last_exposure_date | DATE |
| | administered_by | VARCHAR |
| | previous_exposure_to_drug | BOOLEAN |
| | previous_ae_to_drug | BOOLEAN |
| | frequency_of_administration_value | NUMERIC |
| | frequency_of_administration_unit | VARCHAR |
| | ae_abated_after_stopping_drug | BOOLEAN |
| | ae_reappeared_after_resuming_drug | BOOLEAN |

**Animal_temp**

| | | |
|---|---|---|
| PK | p_record_id | VARCHAR |
| | animal_id | VARCHAR |
| | species | VARCHAR |
| | gender | VARCHAR |
| | age | NUMERIC |
| | age_unit | VARCHAR |
| | weight_kg | NUMERIC |
| | is_crossbred | BOOLEAN |
| | breed_component | VARCHAR |
| | reproductive_status | VARCHAR |

**Active_ingredient_temp**

| | | |
|---|---|---|
| PK | p_record_id | VARCHAR |
| | drug_id | VARCHAR |
| | ingredient_id | VARCHAR |
| | active_ingredient_name | VARCHAR |

*Figure 9: Entity relationship diagram for the temporary stage. Relations between table entries are maintained with the use of the p_record_id dimension which corresponds to the aer natural key of the dataset.*

A distinct feature of this ERD, is the complete absence of relations between tables. This is only partially true because of the existence of the p_record_id dimension inside the tables which associates entries between tables. Additionally the drug_id field, associates drugs and active ingredients. These associations will later prove crucial during the next step of the ETL pipeline.

### 2.3.5 Normalization, grouping, joining DW (loading)

The third step of the ETL pipeline involved transforming the structure of the dimension tables and the implementation of the associations between them. It is also where the fact tables are created by joining data entries from all dimension tables through the means of the universally common p_record_id dimension. Subsequently this dimension acted as the natural key of the dataset.

The ERD of the DW environment is displayed in figure 10.

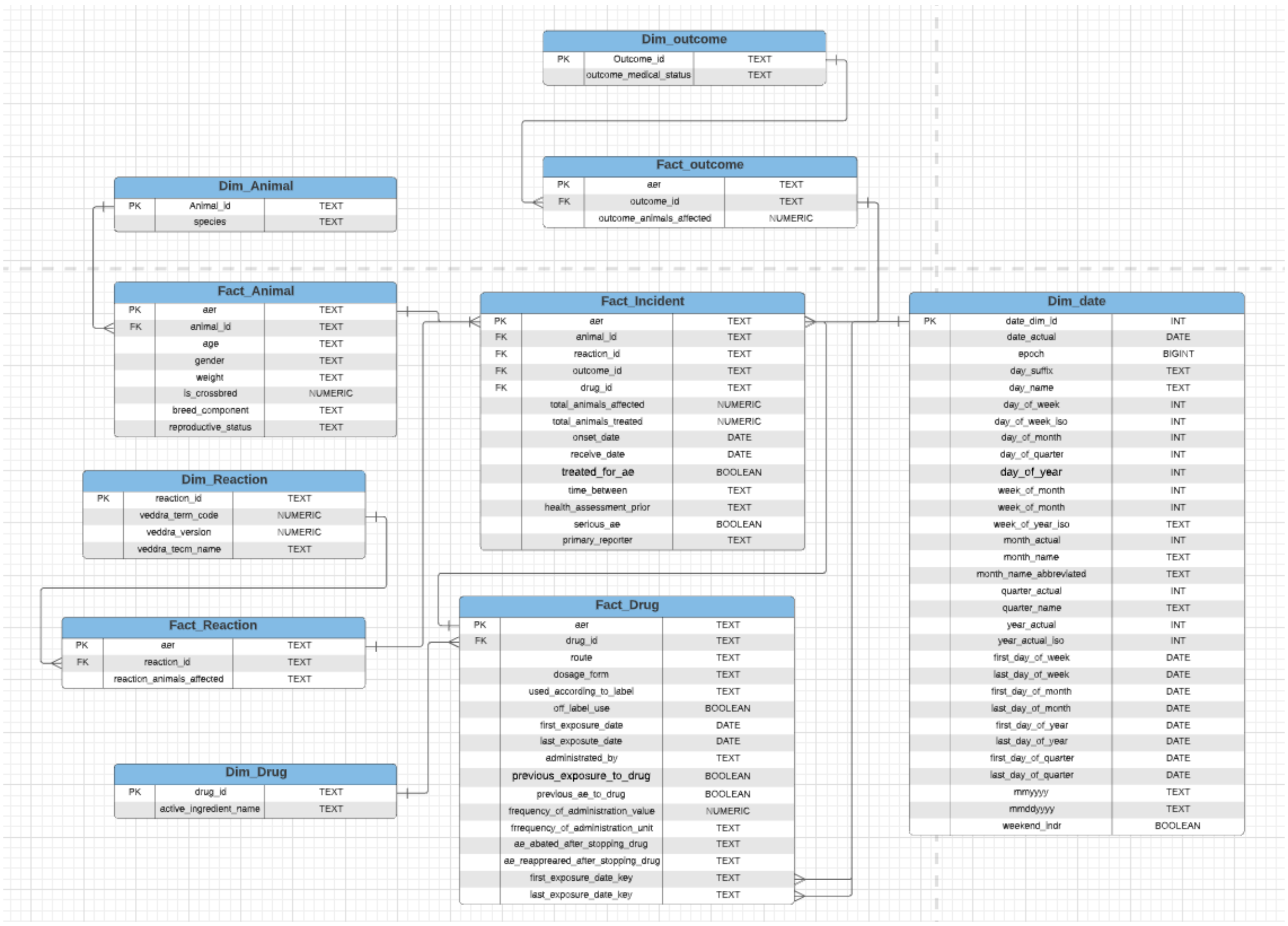


*Figure 10: Entity relationship diagram for the data warehouse environment of the database. A Constellation schema centered on the 'Fact_incident' fact table.*

So, for the DW the constellation schema architecture was followed. The schema consists of 5 fact tables accompanied by 5 dimension tables. Fact tables Animal and Drug were introduced to reduce the complexity that a large individual fact table would have for this scenario. For example, an early attempt for an ERD design for the DW environment included a total of 20 facts along with 6 foreign key columns. The current design is centered on the fact_incident table which serves as the main reference point for the DW, while the other two fact tables are connected to this table by sharing the natural key of the dataset, to allow for complex queries. The process of loading the data was executed using SQL queries, run through a python script.

Each dimension table was loaded from its corresponding table from the temporary stage. However, at this step all duplicate values remaining were discarded using the 'GROUP' SQL function, maintaining only a single instance for every unique value of the dimension fields. This step reduced the amount of data stored within the tables considerably when compared to the temporary stage. Dimension tables are loaded first when converting from the temporary to the DW environments because of their contribution to the loading of the fact tables later on. Dimension tables are now normalized and are comprised of rows ranging in the thousands, compared to the millions of rows in the temporary stage.

That is not the case for the fact tables. Fact tables are both tall and wide with an average of 10 columns each and more than 2 million rows. Fact tables are loaded by joining data from the tables of the temporary environment utilizing some key dimensions connecting them and the dimension tables created.

More specifically:

- The ‘dw.fact_animal’ table is connected with the ‘dw.dim_animal’ table on the animal.species field by joining the ‘temp.animal’ and ‘dw.dim_animal’ tables
- The ‘dw.fact_drug’ table is connected with the ‘dw.dim_drug’ table on the active_ingredient_name field by joining the ‘temp.active_ingredient’ and ‘dw.dim_drug’ tables
- The dw.fact_reaction is connected with the dw.dim_reaction table with the reaction_id field
- The dw.fact_outcome is connected with the dw.dim_outcome table with the outcome_id field
- The dw.fact_Incident table is connected with the Fact_drug table on the drug_id field and the aer field and with all other fact tables with the aer field

**Resources:**
**[1]: https://open.fda.gov/apis/animalandveterinary/event/**
**[2]: https://pandas.pydata.org/**