Pesticide Residue in the United States Food Supply

Introduction

The narratives of food and nutrition in the United States have undergone dramatic changes in the past thirty years. From the rise of farmers markets, GMOs, and unusual diets like gluten-free and the keto diet to name a few, citizens are increasingly aware and concerned about their food consumption habits. Food journalists, chefs, and doctors are igniting numerous conversations about American eating habits, chief among those is the encouragement to eat more daily portions of fruits and vegetables. While by and large, this is an incredibly positive step, there may be some unforeseen negative consequences. For obvious reasons, eating a pear instead of a bag of Doritos can only improve an individual’s health, but what people may not realize is that is not entirely a distinction of natural versus synthetic. The food production system in the United States has been radically transformed in the past few decades and a major part of this alteration is the increased usage of pesticides. While this practice has allowed for the increase in overall volume of production, allowing more people access to such commodities, it has at the same time decreased the overall nutritional value by introducing toxins into our food supply. Additionally, producers and sellers are not legally required to label their products with the pesticides used, entirely masking the truth from consumers.

Fortunately, there are many organizations dedicated to enlightening consumers, chief among those are the United States Department of Agriculture (USDA) and the Environmental Protection Agency (EPA). For the past twenty-five years, the USDA has been collecting samples for testing from across the US, in order to create a more complete picture in regards to pesticide residues in the food supply. This Pesticide Data Program (PDP) is quite extensive, but
unfortunately, upon first examination is unintelligible to the average consumer. A large portion of the data is scientifically motivated, which delivers important details to experts in the industry, while conversely oversaturating everyone else. The proposed project is to build a more practical database, which allows consumers to explore the dataset more thoroughly. Though the PDP maintains their own online interface, it is cumbersome and does not easily lend itself to investigations of trends.\(^1\) These explorations were guided by Roger Fry, a member of the database administration team for the PDP. He helped define what questions the data could and could not answer, because of various anonymity assurances and scientific constraints.\(^2\)

The new database and its interface will focus on producing a much-improved user experience. It will include more customizable queries, more informative result summaries, and more appealing aesthetics, all of which are important aspects that effect an application’s usability. This project consists of the following stages:

**Step I:** Data Assessment,

**Step II:** Data Cleaning,

**Step III:** Database Design,

**Step IV:** Database Creation,

**Step V:** Web Application Construction

**Step VI:** Query Development

The project utilizes a set of open-source tools, all of which have extensive documentation. The stages previously listed made use of the following: Jupyter Notebooks, Pandas a data science

---

\(^1\) Screenshots from the PDP’s current interface are included in Appendix B

\(^2\) Specific locations where samples were taken are withheld in order to protect the growers and producers
library in Python, SQLAlchemy a database ORM\textsuperscript{3} library for Python, Django an MVC\textsuperscript{4} framework for building web applications, PostgreSQL a structured query language, and finally Bootstrap a framework for building standardized HTML webpages.

Step I: **Data Assessment**

Over the course of 25 years, the USDA and their Pesticide Data Program has recorded the results of over 33 million tests performed on various samples of the US food supply. This complete dataset is available for download in the form of Excel and CSV files from their website. These files contained approximately 500 separate table, which break down to about 20 per year. Within those sets of 20 tables, 18 are references which hold codes and descriptions for values held in the other 2 tables. While most of the tables have no more than 800 records, the table containing information about established EPA pesticide tolerances has over 3,000 records, the table which contains information about the samples collected has over 11,000 records, and the table which contains information about the tests ran on those samples has over 1.3 million records. Once this data was downloaded, there were a few immediate steps to take: clean the data, create a normalized schema, load the data into a relational database.

Step II: **Data Cleaning**

Cleaning the data means to detect and correct any corrupted or inaccurate records by either deleting them or modifying them appropriately. Using Jupyter Notebook and the Python libraries NumPy and Pandas, a script loaded all the data into data frames, which allowed for

\textsuperscript{3} Object-relational-mapping or ORM is a technique for translating data between incompatible systems using object-oriented programming languages

\textsuperscript{4} Model-View-Controller or MVC is a design pattern that defines the model as an object, a view as a visualization of that model, and a controller, which governs the action between them
easier manipulation. Determining exactly which records were inaccurate and if they could simply be modified or instead needed to be deleted, Roger Fry, a member of the database administration team at the USDA, provided expertise about the dataset and information that guided this process. A few specific instances in which Fry was instrumental were when he indicated which attributes varied by year or conversely which should be consistent across time, if the meaning of some values were kept confidential from the public, and how a record without atomic values should be handled. Most communication with Fry occurred when initially trying to understand the data, as the documentation provided by the USDA was not entirely complete. Now with a procedure for modifying the data, the next step of the data cleaning process was to merge all comparable tables together. To be more specific there were 25 instances of the following 20 tables provided by the USDA: Annotations, Claims, Commodities, Commodity Types, Concentration Units, Confirmation Methods, Countries, Determinative Methods, Distribution Types, Extraction Methods, Labs, Mean Results, Origins, Pesticides, Quantitation Methods, Results, Samples, States, Test Classes, Tolerances. Some of these tables were directly merged, while others had a column “Year” added if the specific values were year dependent. For example, all claim codes and meanings are consistent across time, so the Claims tables were directly merged, while tolerances vary by year, so the year attribute was appended to each record within the Tolerance tables before they were merged. With the collated tables, the next step of the data cleaning process was to minimally normalize to 1NF. This entailed eliminating redundant entries, ensuring that columns all have atomic values, and that the types and formats of those values were consistent within those columns. Some decisions that needed to be made included handling columns that had both string and floating-point variables, eliminating nulls by replacing them with an “unknown” or “not available” variable, selecting which non-atomic
values to keep (sometimes only one, or other times all that appeared). After this, the data adhered to all the principles of First Normal Form.

Step III: **Database Design**

Further normalization required defining a database schema. Those 20 merged tables, now in 1NF very quickly could become 4NF by creating one additional table, eliminating a few redundant columns from one table, and eliminating another table entirely. To start with, the Annotations table was dropped, because no foreign key dependencies existed in reference to it.

At the time, the Pesticides table had the functional dependencies,

\[
\text{pesticide\_code}, \text{description} \rightarrow \text{test\_class}^5
\]

\[
\text{pesticide\_code} \rightarrow \text{test\_class}
\]

\[
\text{description} \rightarrow \text{test\_class}
\]

In order for a set of relations to be in 2NF, all partial dependencies, AC \( \rightarrow \) B, where AC is a candidate key and either A \( \rightarrow \) B or A \( \rightarrow \) C, have to be eliminated. To remove this dependency, the schema was transformed by adding a table, Pesticide Class, with the dependency,

\[
\text{pesticide\_code} \rightarrow \text{test\_class}
\]

and dropping the column, test\_class, from the Pesticides table to produce the dependency,

\[
\text{pesticide\_code} \rightarrow \text{description}.
\]

There are a couple remaining partial dependencies to eliminate from the Results table, in order for the entire schema to be in 2NF. This was accomplished by eliminating commodity\_code, commodity\_type, and test\_class. The previous partial dependency was,

---

^5 **Bolded** attributes indicate primary keys, **italicized** attributes indicate foreign keys, and attribute that are **both**, are primary keys and foreign keys!
\texttt{result\_id, year, pesticide\_code} \rightarrow \texttt{commodity\_code, commodity\_type, test\_class}

\texttt{result\_id, year} \rightarrow \texttt{commodity\_code, commodity\_type}

\texttt{pesticide\_code} \rightarrow \texttt{test\_class}

Previously, the partial dependency was eliminated by decomposing the Pesticides table into two separate relations, but in this instance, \texttt{commodity\_code, commodity\_type, test\_class}, simply need to be removed from the Results table. This transformation is lose-less, because the dependency,

\texttt{pesticide\_code} \rightarrow \texttt{test\_class}

is defined in the Pesticide Class table and \{\texttt{result\_id, year}\} is a foreign composite key referencing the primary key of the Samples table, which itself has the functional dependency,

\texttt{sample\_id, year} \rightarrow \texttt{commodity\_code, commodity\_type}.

Therefore, the partial dependencies can be resolved by eliminating these three columns from Results instead of decomposing the table into multiple relations, which would otherwise need to be done to avoid information lose.

Once these simple alterations were made, the entire database schema was in 4NF. There are no transitive dependencies, \(X \rightarrow Z\), where \(X \rightarrow Y\), \(Y \rightarrow Z\), and not \(Y \rightarrow X\), so it was immediately in 3NF. For each functional dependency, \(X \rightarrow Y\), \(X\) is a superkey, so the schema was not only 3NF, but also in BCNF. Additionally, the schema does not have multi-value dependencies. Most of the tables have no more than two attributes, so multi-value dependencies are impossible by definition. With the Results, Samples, and Tolerances tables, each entry within the relation is an instance of a real-world object, defined by exactly one value for each attribute. The Tolerances table holds definitions of tolerances set by the EPA for some commodity-pesticide pairing in some particular year and in no year did they set multiple tolerances for any particular pairing. The Samples table descriptions of specific samples of some commodity taken
by the USDA from some specific location on some specific date. The Results table holds the outcomes from specific tests run on specific samples, with no test recorded more than once on any sample. In these three tables, no attribute can have multiple values, meaning there are no multi-value dependencies.

However, the normalization stop here. The database schema is not in 5NF, because not every natural table join results in usable tables. For example, the natural join of Pesticides and Claims would be possible, because both relations have candidate keys \{code, description\} but the resulting table would be semantically meaningless, just an inter-mixing of pesticide and claim entities.

The final database schema has the following functional dependencies\(^6\):

- Claims: \textbf{code} \rightarrow description
- Commodities: \textbf{code} \rightarrow description
- Commodity Types: \textbf{code} \rightarrow description
- Concentration Units: \textbf{code} \rightarrow description
- Confirmation Methods: \textbf{code} \rightarrow description
- Countries: \textbf{code} \rightarrow description
- Determinative Methods: \textbf{code} \rightarrow description
- Distribution Types: \textbf{code} \rightarrow description
- Extraction Methods: \textbf{code} \rightarrow description
- Labs: \textbf{code} \rightarrow description
- Mean Results: \textbf{code} \rightarrow description
- Origins: \textbf{code} \rightarrow description
- Pesticides: \textbf{code} \rightarrow description
- Pesticide Classes: \textbf{code} \rightarrow \textit{test_class}
- Quantitation Methods: \textbf{code} \rightarrow description
- Results: \{\textbf{result_id, year, pesticide_code}\} \rightarrow \{\textit{commodity_code, commodity_type, lab_code, test_class, concentration, lod, concentration_unit_code, confirmation_method, confirmation_method2, quantitation_method, mean, extract_code, determin_code}\}

\(^6\) As before, \textbf{bolded} attributes indicate primary keys, \textit{italicized} attributes indicate foreign keys, and attributes with \textit{both} are primary and foreign keys!
- Samples: \{\text{sample_id, year}\} → \{month, day, state, site, commodity_code, source, variety, origin, country, dist_type, commodity_type, claim, grow_state, pack_state, dist_state\}
- States: \text{code} → \text{description}
- Test Classes: \text{code} → \text{description}
- Tolerances: \{\text{commodity_code, pesticide_code, year}\} → \{epa_tolerance, concentration_unit_code, note, comment\}

The Entity-Relationship diagram is as follows:

Step IV: **Database Creation**

The final task involved in the setup of this database is table creation and data insertion. A script which used a Python library called SQLAlchemy, defined the tables and their relationships, created the tables in the given PostgreSQL database, then took the cleaned data and loaded it into the tables. This process ensured that any data in the tables adhered to the
relationships defined above and did not violate any primary or foreign key constraints. If a record was encountered that did violate these rules, it was not inserted, but added to an error list. Anything that accumulated in this list was evaluated and either deleted or modified, based on the principles defined in Step I. Again, Roger Fry helped to make these decisions, so the data integrity was not corrupted. The final total of records in each table were counted, using the command `SELECT count(*) FROM table_name;` and are as follows:

- Claims = 5
- Commodities = 125
- Commodity Types = 26
- Concentration Units = 4
- Confirmation Methods = 25
- Countries = 265
- Determinative Methods = 42
- Distribution Types = 18
- Extraction Methods = 46
- Labs = 17
- Mean Results = 9
- Origin = 3
- Pesticides = 713
- Pesticide Classes = 713
- Quantitation Methods = 16
- Results = 33,836,648
- Samples = 278,676
- States = 56
- Test Classes = 22
- Tolerances = 78,658

Step V: **Web Application Creation**

The focus of this project, providing more information to consumers, makes the interface with which they interact incredibly integral to the project’s success and deserves much consideration. Especially important with large datasets, focusing the scope of a query can greatly enhance a user’s experience. Defining scope can be done either through static query design or
dynamic query options provided to the user. Both come with pros and cons and in different
situations the merits of one outweigh that of the other. For this project, users can both customize
simple searches and also run more complex queries with few or no variable options.

Providing users with dynamic query options first requires defining which attributes have
the most contextual relationship to a user’s needs. This includes not only defining the available
search inputs, but also the corresponding search outputs. In theory it is possible to search by any
value of any attribute within the database, but this would quickly become too complex and would
overwhelm the user. This set of available options can be trimmed by considering which would be
the most relevant and meaningful to the user. For the simple queries which have the largest
opportunity for customization, the available options are as follows: commodity, claim, origin,
state grown in, state packed in, state distributed from, state sold in, month collected, year
collected, lab, and pesticide. Some of the attributes that were eliminated are level of detection
limit (lod), confirmation method, extraction method, and site code. These were all removed,
because their values largely have no bearing on whether or not a consumer considers a
commodity healthy. Although test methods and their results of course define what is and what is
not in code, merely the outcome matters to the consumer, not the methods. The same principles
are used to determine which attributes should be displayed as the result of a query. Only varying
slightly, the result attributes are as follows: commodity, claim, origin, state grown in, state
packaged in, state distributed in, state sold in, year, pesticide, concentration, tolerance, and
concentration units. Of course, these are not the only viable options, but in terms of limitations
on a computer’s display, it does not make sense to try and pack many more into the view port.
The values would be cut off or navigation through the table would be complicated, negating the
usefulness of adding more attributes.
For the more complex queries, which are aimed at answering much more specific questions, there is not as much opportunity for customization and not as much of a need to display many attributes. For example, when asking the question, what states have grown healthy peaches, it is only necessary to display the state in which the peaches were grown, they would not necessarily need to know where those peaches were packaged and what pesticides were tested for. On the search end of this query, users select which commodity, they wish to search for in place of peaches, making the interaction more dynamic, though markedly less so when compared to the level of customization with simple queries.

Both the search and result interfaces were designed using Django, an MVC framework for creating web applications in Python. This easily abstracted away the complexities of connecting a web application to a database, as well as provided prepackaged tools to for managing the user experience. A Model-View-Controller or MVC framework defines the model as an object, a view as a visualization of that model, and a controller, as the module that governs the action between models and views. For this specific project, the tables of the database are represented as models, the views are the webpages that interact with the database, and the remainder of Django’s framework, which include URL routers and query functions among others, is the controller.

The first steps of creating a Django application are much like any other project, package installation and environment setup. Once these two steps are completed, a Django project can be created by running the command, `django-admin startproject project_name`. Though this next step may seem a bit redundant, within the project folder, there must also be an initialized application, which can be created with the command, `python3 manage.py startapp app_name`. These simple scripts create the base framework in which models and views can be created. This is the core design feature of Django: abstraction of complexities. The
peripheral work required to create a secure connection to and a graphic interface for a database are completed with just these two simple commands.

As mentioned previously, the models created are classes that represent the individual tables within the database, while the views are HTML web pages that present the models in a specific format. When defining the models, because the database already exists Django provides a simple way to take the existing schema and represent it as python classes. The command, `python3 manage.py inspectdb -database db_name > models.py`, pipes what amounts to a translation from the PostgreSQL schema to models defined as python classes. While almost perfect, the auto-generated models need some manual touch ups, but much less work that defining them from scratch. The final step needed to create the models is to migrate the changes to the Django project. Tracking changes to the structure of the application is necessary in order for the project to maintain an accurate picture of its internal state. Again, Django aims to streamline this process and provides a set of two simple methods: `python3 manage.py makemigrations`, which creates the script that lists all the changes and their dependencies, and then `python3 manage.py migrate`, which actually runs those migrations.

Step VI: **Query Development**

Django provides a module for model abstraction, which comes with pros and cons. The major pro being it allows an application to be database agnostic, or in other words, the code which executes queries does not need to be altered if the underlying database is changed from PostgreSQL to MySQL. The major drawback of the abstraction is that it obscures the actual structure of the SQL command it executes at the database level. As a consequence, fine tuning a more complex query is very difficult. Django also provides a way of bypassing the model level to
execute raw SQL queries, but it is worth noting that by circumventing the model abstraction, the application is no longer database agnostic, sacrificing the application’s portability.

The simple queries ask questions like, which blueberries grown in 2013 and labeled as organic, tested positive for pesticides? This query is simply built by adding filters to the models in the following manner:

```python
Sample.objects.values('samp_id', 'year', 'month', 'state_code', 'commod_code', '
    'origin_code', 'claim_code', 'grow_st', 'pack_st', 'dist_st')
    .filter(commod_code='BB').filter(year='2013').filter(claim='PO')
```

Django does not execute the query until an attempt is made to access the records produced by it. This so-called lazy loading, adds to the overall efficiency of the application, by not initializing any objects unnecessarily. Running as few time-consuming operations at the database level as possible will improve performance as well, so once a query is accessed, its contents are hashed and any further manipulation done to them is all in memory. This approach differs for the complex queries, all of which are executed entirely at the database level and whose results return already formatted as a python dictionary mapping. These complex queries are executed by first establishing a new connection to the database, executing the statement, and the returning the results in the desired format as follows:

```python
cursor = connection['pdp_db'].cursor()
cursor.execute(sql_statement)
results = fetchall(cursor)
```

Some examples of questions the complex queries answer and their corresponding SQL statements are:
Which states have only ever grown healthy peaches?

```
select healthy.grow_st
from
  (select distinct grow_st
   from
    (select res_id, res.year, pest_code, concen, grow_st, commod_code
     from (select res_id, year, pest_code, concen from result_table) as res
      inner join (select samp_id, year, grow_st, commod_code
        from sample_table where commod_code='PC') as samp
      on res.res_id=samp.samp_id and res.year=samp.year) as t
    inner join (select commod_code, pest_code, year, epa_tol from tolerance_table) as tol
    on t.pest_code=tol.pest_code and t.commod_code=tol.commod_code and t.year=tol.year
    where concen <= cast_to_float(epa_tol, 0) and grow_st<>'NA') as healthy
left join (select distinct grow_st
   from
    (select res_id, res.year, pest_code, concen, grow_st, commod_code
     from (select res_id, year, pest_code, concen from result_table) as res
      inner join (select samp_id, year, commod_code
        from sample_table where commod_code='PC') as samp
      on res.res_id=samp.samp_id and res.year=samp.year) as t
    inner join (select commod_code, pest_code, year, epa_tol from tolerance_table) as tol
    on t.pest_code=tol.pest_code and t.commod_code=tol.commod_code and t.year=tol.year
    where concen > cast_to_float(epa_tol, 0) and grow_st<>'NA') as unhealthy
  on healthy.grow_st=unhealthy.grow_st
where unhealthy.grow_st is null;
```

Are there certain commodities with more violations than others?

```
select
tol.commod_code as commodity,
  10000*count(case when concen > cast_to_float(epa_tol, 0) then 1 else null end)/count(*) as rate_per_ten_thousand
from
  (select samp_id, samp.year as year, pest_code, concen, commod_code
   from
    (select res_id, year, pest_code, concen from result_table) as res
    inner join (select samp_id, year, commod_code from sample_table) as samp
    on res_id=samp.samp_id and res.year=samp.year) as res_samp
  inner join (select commod_code, pest_code, year, epa_tol from tolerance_table) a as tol
    on res_samp.year=tol.year and res_samp.pest_code=tol.pest_code and res_samp.commod_code=tol.commod_code
  group by commodity
order by rate_per_ten_thousand desc;
```

Further Work

This project just begins to scratch the surface of potential avenues of exploration with this dataset. A large opportunity exists for simpler additions, which add similar functionalities to the application as it already exists. An analysis of the geographic traits of a commodity, which could trace its path from grower to packager to distributor and finally to the seller would be a very
interesting exercise and perhaps reveal something about the supply chain and potentially locate regions of concern or compliance. Another addition could explore the severity of EPA regulation violations in terms of degree above acceptable residue levels and maybe even with respect to the toxicity of a given pesticide. Current queries could be improved by allowing for multi-selects or exclusion of options based on previous values selected.

These three suggestions would add great value to the application, but there are also much more extensive additions that can increase the usefulness to an even larger degree. As it stands, the application would benefit most from more localized data and a robust statistical analysis. This is not the only dataset that exists providing information about pesticides in the US food supply, so a merger could widen the domain of answerable questions, perhaps expanding the consumer’s understanding of food supply patterns past the state level and down to the county or even lower. A statistical analysis would expand the set of answerable questions even further, but would require an individual well versed in the subject.

**Conclusion**

Overall, this project successfully expanded upon consumers’ knowledge of commodity production tendencies. Though there is still significant work to be done in this regard, the steps made here lay the groundwork for future strides. It also stands as a small example of how barriers between scientific or academic research and popular consumption should and can be broken. Experts should do their best to not only educate individuals like themselves, but also to impart that information to a much wider population.

The database and interface built here accomplishes just that. It brings forth new information to the marketplace, shedding light on things otherwise unseen. The most exciting part about a project like this is that every step forward opens another door, making even more
information ready for discovery. The US is increasingly ready for a food revolution and every added piece of information brings consumers that much closer to healthier lifestyles and food production practices.
## Appendix A – Data Dictionaries

**Table 1: Results Table (33,836,648 Records)**

<table>
<thead>
<tr>
<th>Column</th>
<th>Type (Length)</th>
<th>Key</th>
<th>Description</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>res_id</strong></td>
<td>Integer</td>
<td>Primary, Foreign</td>
<td>ID of sample</td>
<td>Unique to year</td>
</tr>
<tr>
<td><strong>year</strong></td>
<td>Integer</td>
<td>Primary</td>
<td>Year of test</td>
<td></td>
</tr>
<tr>
<td><strong>pest_code</strong></td>
<td>Character Varying (3)</td>
<td>Primary, Foreign</td>
<td>ID of pesticide tested for</td>
<td></td>
</tr>
<tr>
<td><strong>concen</strong></td>
<td>Double Precision</td>
<td></td>
<td>Concentration detected</td>
<td></td>
</tr>
<tr>
<td><strong>Lod</strong></td>
<td>Double Precision</td>
<td></td>
<td>Limit of detection</td>
<td>Varies across years, labs, and pesticides</td>
</tr>
<tr>
<td><strong>con_unit_code</strong></td>
<td>Character Varying (3)</td>
<td>Foreign</td>
<td>Units for concentration</td>
<td></td>
</tr>
<tr>
<td><strong>conf_method_code</strong></td>
<td>Character Varying (3)</td>
<td>Foreign</td>
<td>Primary confirmation method</td>
<td></td>
</tr>
<tr>
<td><strong>conf_method2_code</strong></td>
<td>Character Varying (3)</td>
<td>Foreign</td>
<td>Secondary confirmation method</td>
<td></td>
</tr>
<tr>
<td><strong>quantitate</strong></td>
<td>Character Varying (3)</td>
<td>Foreign</td>
<td>Quantitative method</td>
<td></td>
</tr>
<tr>
<td><strong>mean_code</strong></td>
<td>Character Varying (3)</td>
<td>Foreign</td>
<td>Code for mean result finding</td>
<td></td>
</tr>
<tr>
<td><strong>extract_code</strong></td>
<td>Character Varying (3)</td>
<td>Foreign</td>
<td>Extraction method</td>
<td></td>
</tr>
<tr>
<td><strong>determin_code</strong></td>
<td>Character Varying (3)</td>
<td>Foreign</td>
<td>Determinative method</td>
<td></td>
</tr>
<tr>
<td>Column</td>
<td>Type (Length)</td>
<td>Key</td>
<td>Description</td>
<td>Note</td>
</tr>
<tr>
<td>---------------</td>
<td>---------------</td>
<td>------</td>
<td>--------------------------------------------------</td>
<td>-------------------------------</td>
</tr>
<tr>
<td>Samp_id</td>
<td>Integer</td>
<td>Primary</td>
<td>ID of sample taken</td>
<td>Unique to year</td>
</tr>
<tr>
<td>Year</td>
<td>Integer</td>
<td>Primary</td>
<td>Year sample was taken</td>
<td></td>
</tr>
<tr>
<td>Month</td>
<td>Integer</td>
<td></td>
<td>Month sample was taken</td>
<td></td>
</tr>
<tr>
<td>Day</td>
<td>Integer</td>
<td></td>
<td>Day of month sample was taken</td>
<td></td>
</tr>
<tr>
<td>State_code</td>
<td>Character Varying (3)</td>
<td>Foreign</td>
<td>State commodity was sold in</td>
<td></td>
</tr>
<tr>
<td>Commod_code</td>
<td>Character Varying (3)</td>
<td>Foreign</td>
<td>Commodity sampled</td>
<td></td>
</tr>
<tr>
<td>Source_id</td>
<td>Character Varying (3)</td>
<td></td>
<td>Code (A-Z) to make samp_id unique without year</td>
<td>Not currently used</td>
</tr>
<tr>
<td>Variety</td>
<td>Character Varying (25)</td>
<td></td>
<td>Variety/Class of commodity</td>
<td>“Free form”</td>
</tr>
<tr>
<td>Origin_code</td>
<td>Character Varying (3)</td>
<td>Foreign</td>
<td>Code indicating domestic or import</td>
<td>Can also be unknown</td>
</tr>
<tr>
<td>Country_code</td>
<td>Character Varying (3)</td>
<td>Foreign</td>
<td>Origin countries or set of origin countries</td>
<td>Only for imports</td>
</tr>
<tr>
<td>Dist_type_code</td>
<td>Character Varying (3)</td>
<td>Foreign</td>
<td>Type of collection/distribution facility</td>
<td></td>
</tr>
<tr>
<td>Commod_type_code</td>
<td>Character Varying (3)</td>
<td>Foreign</td>
<td>Type of commodity</td>
<td>i.e. – Fresh, canned, etc.</td>
</tr>
<tr>
<td>Claim_code</td>
<td>Character Varying (3)</td>
<td>Foreign</td>
<td>Claim made on product</td>
<td>e.g.- organic</td>
</tr>
<tr>
<td>Grow_st</td>
<td>Character Varying (3)</td>
<td>Foreign</td>
<td>State code for location of sample grower</td>
<td></td>
</tr>
<tr>
<td>Pack_st</td>
<td>Character Varying (3)</td>
<td>Foreign</td>
<td>State code for location of sample packager</td>
<td></td>
</tr>
<tr>
<td>Dist_st</td>
<td>Character Varying (3)</td>
<td>Foreign</td>
<td>State code for location of sample distributor</td>
<td></td>
</tr>
<tr>
<td>Column</td>
<td>Type (Length)</td>
<td>Key</td>
<td>Description</td>
<td>Note</td>
</tr>
<tr>
<td>-----------</td>
<td>---------------</td>
<td>----------------</td>
<td>--------------------------------------------------</td>
<td>-------------------------------------------</td>
</tr>
<tr>
<td>Commodity</td>
<td>Character Varying (3)</td>
<td>Primary, Foreign</td>
<td>Code for commodity</td>
<td></td>
</tr>
<tr>
<td>Pest_code</td>
<td>Character Varying (3)</td>
<td>Primary, Foreign</td>
<td>Code for pesticide</td>
<td></td>
</tr>
<tr>
<td>Year</td>
<td>Character Varying (3)</td>
<td>Primary</td>
<td>Code for year tolerance was in effect</td>
<td></td>
</tr>
<tr>
<td>EPA_tol</td>
<td>Character Varying (5)</td>
<td></td>
<td>Established EPA Tolerance for pesticide-commodity pairing</td>
<td>Can be EX (Exempt) or NT (No tolerance established)</td>
</tr>
<tr>
<td>Con_unit_code</td>
<td>Character Varying (3)</td>
<td>Foreign</td>
<td>Code for units of tolerance</td>
<td></td>
</tr>
<tr>
<td>Note</td>
<td>Character Varying (50)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Comment</td>
<td>Character Varying (50)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Appendix B – USDA’s PDP Interface

Figure 1: Initial query interface

<table>
<thead>
<tr>
<th>PDP Database Search</th>
<th>Output Preference</th>
<th>Results Preference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Check All Commodities</td>
<td>Check All Pesticides</td>
<td>Check All years</td>
</tr>
<tr>
<td>Almonds</td>
<td>1,2,4-Triazole</td>
<td>2017</td>
</tr>
<tr>
<td>Apple Juice</td>
<td>1-Naphtol</td>
<td>2016</td>
</tr>
<tr>
<td>Apple Sauce</td>
<td>2,4,5-T</td>
<td>2015</td>
</tr>
<tr>
<td>Apples</td>
<td>2,4-TP</td>
<td>2014</td>
</tr>
<tr>
<td>Apples-Single Servings</td>
<td>2,4-D</td>
<td>2013</td>
</tr>
<tr>
<td>Asparagus</td>
<td></td>
<td>2012</td>
</tr>
<tr>
<td>Asparagus, Canned</td>
<td>2,4-dimethyl aniline (2,4 DMA)</td>
<td>2011</td>
</tr>
<tr>
<td>Avocado</td>
<td>2,4-dimethylphenyl formamidine (2,4-DMPF)</td>
<td>2010</td>
</tr>
<tr>
<td>Baby Food - Applesauce</td>
<td>2,6-dichlorobenzamide</td>
<td>2009</td>
</tr>
<tr>
<td>Baby Food - Carrots</td>
<td>2,6-DIPN</td>
<td>2008</td>
</tr>
<tr>
<td>Baby Food - Green Beans</td>
<td>3,5-Dichloroaniline</td>
<td>2007</td>
</tr>
<tr>
<td>Baby Food - Peaches</td>
<td>3-Hydroxycarbofuran</td>
<td>2006</td>
</tr>
<tr>
<td>Baby Food - Pears</td>
<td>3-ketocarbofuran</td>
<td>2005</td>
</tr>
<tr>
<td>Baby Food - Peas</td>
<td>4,4-dibromobenzophenone</td>
<td>2004</td>
</tr>
<tr>
<td>Baby Food - Sweet Potatoes</td>
<td>4-Hydroxychlorothalonil</td>
<td>2003</td>
</tr>
<tr>
<td>Benenes</td>
<td>4-Hydroxyxylphenylemine</td>
<td>2002</td>
</tr>
<tr>
<td>Barley</td>
<td>5-Hydroxylbendazolone</td>
<td>2001</td>
</tr>
<tr>
<td>Beef Adipose</td>
<td>Abamectin</td>
<td>2000</td>
</tr>
<tr>
<td>Beef Liver</td>
<td>Azoxystrobin</td>
<td>1999</td>
</tr>
</tbody>
</table>

Figure 2: Example query
### Analytical Results

<table>
<thead>
<tr>
<th>Sample ID#</th>
<th>Com</th>
<th>Pest Code</th>
<th>Pest Name</th>
<th>Tst</th>
<th>Concen</th>
<th>LOD</th>
<th>pp_</th>
<th>Co1</th>
<th>Co2</th>
<th>Ann</th>
<th>Qua</th>
<th>Mea</th>
<th>Ext</th>
<th>Det</th>
<th>Tol (ppm)</th>
</tr>
</thead>
<tbody>
<tr>
<td>WA1799050023ACWA1</td>
<td>AC</td>
<td>B65</td>
<td>Spirodiclofen</td>
<td>J</td>
<td>0</td>
<td>0.01</td>
<td>M</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>NP</td>
</tr>
<tr>
<td>WA1799050023ACWA1</td>
<td>AC</td>
<td>B80</td>
<td>Acetamiprid</td>
<td>A</td>
<td>0.004</td>
<td>0.0015</td>
<td>M</td>
<td>LU</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>O</td>
</tr>
<tr>
<td>WA1799050023ACWA1</td>
<td>AC</td>
<td>B79</td>
<td>Trifloxystrobin</td>
<td>F</td>
<td>0</td>
<td>0.0015</td>
<td>M</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WA1799050023ACWA1</td>
<td>AC</td>
<td>B78</td>
<td>Fluquinconazole</td>
<td>L</td>
<td>0</td>
<td>0.01</td>
<td>M</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>ND</td>
</tr>
<tr>
<td>WA1799050023ACWA1</td>
<td>AC</td>
<td>B77</td>
<td>Dimethomorph</td>
<td>W</td>
<td>0</td>
<td>0.003</td>
<td>M</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>ND</td>
</tr>
<tr>
<td>WA1799050023ACWA1</td>
<td>AC</td>
<td>B75</td>
<td>Boscalid</td>
<td>A</td>
<td>0</td>
<td>0.003</td>
<td>M</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>ND</td>
</tr>
<tr>
<td>WA1799050023ACWA1</td>
<td>AC</td>
<td>B73</td>
<td>Fenazaquin</td>
<td>J</td>
<td>0</td>
<td>0.005</td>
<td>M</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>ND</td>
</tr>
<tr>
<td>WA1799050023ACWA1</td>
<td>AC</td>
<td>B72</td>
<td>Tetraconazole</td>
<td>L</td>
<td>0</td>
<td>0.01</td>
<td>M</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>ND</td>
</tr>
<tr>
<td>WA1799050023ACWA1</td>
<td>AC</td>
<td>B68</td>
<td>Thiadiazep</td>
<td>A</td>
<td>0</td>
<td>0.001</td>
<td>M</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>ND</td>
</tr>
<tr>
<td>WA1799050023ACWA1</td>
<td>AC</td>
<td>B64</td>
<td>Fenomidone</td>
<td>A</td>
<td>0</td>
<td>0.005</td>
<td>M</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>ND</td>
</tr>
<tr>
<td>WA1799050023ACWA1</td>
<td>AC</td>
<td>B63</td>
<td>Fludolanil</td>
<td>A</td>
<td>0</td>
<td>0.0015</td>
<td>M</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>ND</td>
</tr>
</tbody>
</table>

**Figure 3:** Results from example query
Appendix C – Improved Database Interface

Figure 4: Exploration options (More coming soon)
Figure 5: Simple query search form

Figure 6: Results from simple query
Figure 7: Result from simple query with search filter