detdb.com

PROJECT OVERVIEW

Jamiel Rahi

December 22, 2018

DetonationDB

Goal

The original Detonation Database is an online catalog of experimental data on gaseous detonations. It was created by Joe Shepherd at the GALCIT Explosion Dynamics Laboratory at Caltech in 1997, and consists of tables (with associated citations) and scatter plots that combine data from different tables. The database itself was last updated in 1999, and the website is no longer maintained. The goal of `detdb.com` is to revitalize Dr. Shepherd's original database into a modern and expandable format. The primary features will include: all the features of the original database, a fast Google-style keyword search, and a data upload system strictly reserved for administrators.

1 Joe Shepherd's Database

The old database can be found here.

- 387 tables, 101 figures, and 130 citations. Some authors appear in several different citations. The smallest complete unit of data is a table. It is essentially the results of an experiment. A figure is a scatter plot combining multiple tables with common properties. Figure 1 shows a table and Figure 4 shows a scatter plot.
- Each table is indexed with an alphanumeric ID, for example "at195c" or "mk184d". The logic behind the naming is still unclear.
- On the website, each table can be downloaded as a `.txt` file with the name being its ID mentioned above. Figure 3 shows an example.
- Every table currently has the same eight properties (category, fuel, etc) seen in Figure 1. Some are sometimes blank, and some are sometimes ranges or multiple values. Sometimes they are mixed with the dataset itself. Figure 2 shows another example. Section 3 explains how this variability can handled easily with 'properties' and 'details'.
- There is a print version of the entire database written in LaTeX. The figures are also clearly generated in LaTeX. See Section 2 for concerns about figures.
- There used to be a search feature, but it no longer works. The old back end was written in Perl (similar to PHP).

Table 131: at198b [37, Desbordes (1993)]

Initial Pressure (atm)	Initial Pressure (kPa)	Cell Width (mm)
0.2517	25.49721	6.9507
0.3023	30.62299	5.9668
0.3004	30.43052	4.9908
0.4012	40.64156	3.9759
0.5025	50.90325	2.9779
0.6416	64.99408	2.1944
0.6457	65.40941	1.9972
0.8036	81.40468	1.6923
1	101.3	1.3526
1	101.3	1.1956

Category:	cell size	Fuel:	C2H2
Sub-Category:	width	Oxidizer:	O2
Initial Pressure:	0.13-1 kPa	Diluent:	Ar
Initial Temperature:	293 K	Equivalence Ratio:	1

Figure 1: Example table.

Table 100: at21c [130, Zitoun (1995)]

Pressure (bar)	Pressure (kPa)	Cell Width
0.4918	49.18	1.4819
0.6953	69.53	0.9901
0.9827	98.27	0.6889

Category:	cell size	Fuel:	H2
Sub-Category:	width	Oxidizer:	O2
Initial Pressure:	50.7-101.3 kPa	Diluent:	
Initial Temperature:	123 K	Equivalence Ratio:	1

Figure 2: Another example table.

```
#Equivalence Ratio, Critical Energy
0.7733, 101.0650
1.0000, 53.7913
1.1080, 48.2483
1.2034, 54.7121
1.3351, 91.1051
```

Figure 3: Example table text file.

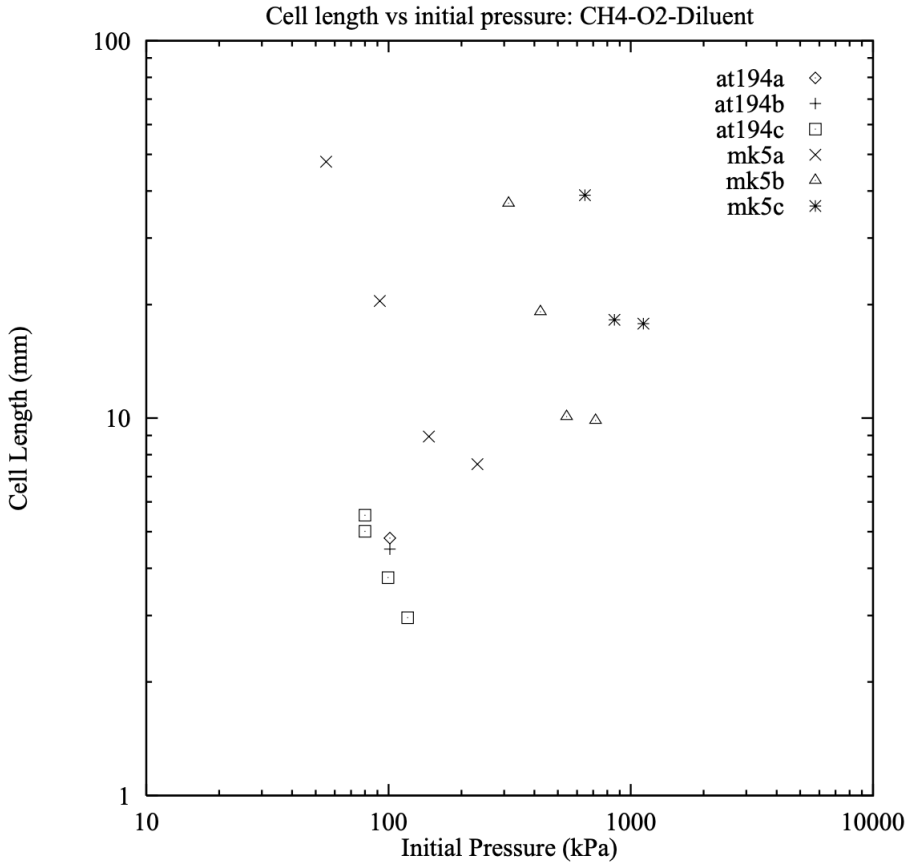


Figure 31: Cell length vs initial pressure; CH4-O2-Diluent

Figure 4: Example figure.

2 Development

There is a lot involved in completely porting the existing functionalities into a modern site. Note that it is very important that the site works on old browsers – you never know what kind of computers are being used in a combustion lab. **The first baby-step is to finish the alpha version before the Winter 2019 semester begins**.

- **Alpha**. Pleasant front-end, default Django admin interface, functioning search, and manual index. Page refreshing is fine. No figures.
- **Beta**. Refined admin system based on professor input. Moderate amount of dynamic UI elements (i.e. more React and AJAX). Figures.
- **Release**. Smooth interface with minimal page refreshing on modern browsers. Fall-back system (perhaps using modernizr) in case of old browsers. Approval from Joe Shepherd (Caltech) and Charles Kiyanda (Concordia). Many download options.

2.1 A Note About Figures

Figures fundamentally depend on a knowledge of the material. Although the existing figure could technically be copied, developing a system for creating new figures requires third-party input for what exactly is useful. Even copying the old figures into the new system is a considerable task. Should they still be generated from LaTeX? Should we implement one of the many JS or Python data plot libraries? Should there be a dynamic front-end plotting tool?

Given these considerations, figures can be implemented in a later version. Adding more structure to the database when this feature is included should not be a problem, since figures depend on tables, but tables do not depend on figures in Joe Shepherd's database.

2.2 A Note About LaTeX

The new database should also have a print copy that updates whenever new data is added. This would require automated LaTeX generating tools similar to those used in the previous Perl back end. Like figures, this can be implemented later.

2.3 A Note About Citations

A dialogue with Joe Shepherd will be important to completely clarify how to credit him (and his team) for their original work. How should we be formally credited? Input from other professors may be useful.

3 Database Design

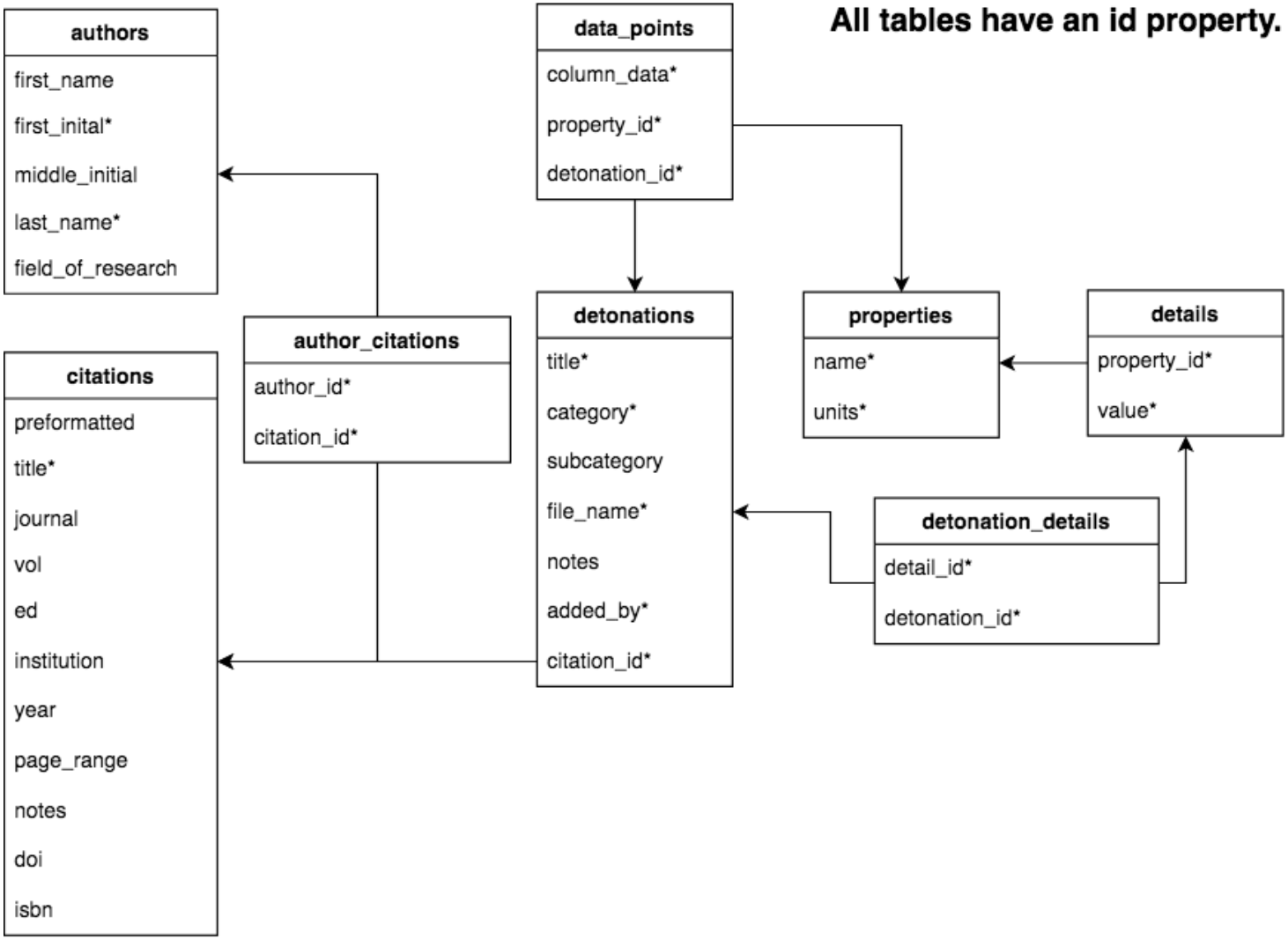
The following preliminary SQL database structure is meant to **minimize code duplication** and remain **flexible for any new entries**.

SQL Table	Description	Examples
authors	Authors often appear in multiple citations. Instead of repeatedly entering the author, we save that author here and reference them by ID.	Self explanatory.
citations	Self explanatory.	Self explanatory.
author_citations	Many authors can be associated with a single citation, and many citations can be associated with a single author. Therefore, it is a many-to-many relationship and needs an extra table to create author-citation pairs.	Self explanatory.
detonations	This contains general information about a single detonation experiment (what I previously called a 'table' in Section 1). It acts as a single entity that other tables need to point to. It doesn't directly hold any actual experimental data, but instead that experimental data will point to it. The 'title' property can correspond to the alphanumeric code mentioned in Section 1.	A 'category' might be 'Cell Length'. 'added_by' is some administrator (possibly default to 'Joe Shepherd').
properties	These are essentially labels with units. The same properties appear in both metadata and in experimental data, so instead of rewriting the property name with all sorts of different possible values every time it comes up, simply reference it if it already exists in this table. Otherwise, create a new entry in this table. This is particularly useful for ranges, since max and min values can be indexed individually (i.e. '1.6-2.1' is searchable as '1.6' or '2.1').	'Initial Pressure' and 'kPa'. 'Fuel' and 'compound'. 'Equivalence Ratio' and 'dimensionless'. 'Min Temperature' and 'K'.
details	A detail is a single value associated with a single property. A lot of properties have the same value for different experiments, so they can always point to the same ID in this table.	'293' associated with 'Initial Temp', 'K'.
detonation_details	Similar to author_citations, this creates detonation-detail pairs (many-to-many relationship).	Self-explanatory.
data_point	A single data point holds a column (saved as a JSON array) of experimental data (Figures 1-3), associated with a property. It is then linked to a detonation.	`'[0.492,0.695,0.983]'` with `'Pressure','bar'`.

Table 1: Overview of tables.

*** indicates required field.**

All tables have an id property.



∞

4 Technologies

The highly-structured nature of the data lends itself well to a relational (SQL) database. Along with this, a Python back end is chosen for quick, high-level integration with otherwise complicated tools. Table 1 summarizes all potential technologies being used in the project.

Tech	Open Source	Justification	Confirmed
Django	Yes	Fast development, built-in admin site, well-established, extensive documentation, and personal interest.	Yes
MySQL	Yes	Industry-standard database, more comprehensive than SQLite but less than PostgreSQL, built-in integration with Django.	Yes
Elastic Search	Yes	Well-known search engine based on Apache Lucene. Slightly simpler to use over Sphinx, fast, and easy to integrate with Django.	No
Haystack	Yes	API for simplified search engine integration into Django. Supports common search engines including Elastic Search.	No
React	Yes	Popular front-end framework for creating dynamic user interfaces. Chosen over Angular due to its more natural 'component' approach. Also, personal interest.	No
Backbone.js	Yes	Lightweight framework for structuring front-end code into the standard Model-View-Controller pattern. In other words, it helps reduce spaghetti code and figuratively act as a 'backbone'. Compatible with React.	No
Bootstrap 4	Yes	CSS essentials. Chosen over Semantic UI for its nav bar.	Yes
BeautifulSoup	Yes	Python web-scraping utility to gather old database information.	No

Table 2: Technologies overview.

5 Interface

There is still plenty of front-end design work to do, and only the pages that are essentially already designed are to be discussed here. Designs for search results and individual table pages are still undecided, and implementation of figures will be a significant design task in its own right.

5.1 Home Page

A clean and simple Google-style home page should suffice, albeit with a bit more complexity than Google. Perhaps some information about the database and website could be provided in a new section under the main section with the search bar.

5.2 Manual Index

This should be similar in structure to the original database, organized by category. As a starting point, it can be functionally the same as the old one, but look nicer. There should be a nav bar including (but not limited to) "About", "Downloads", and "Cite".

5.3 Logo

The logo should be relatively simple. The following figures show some possible logo options.

Figure 5: Logo A.

Figure 6: Logo B.