Intro to Databases Project Part 1

The relatively informal, one-paragraph description:

The database will seek to solve the problem of searching up methods (and their documentation) for programming languages, specifically focusing on implementing a good way of searching up Python packages, modules, and methods while also leaving room for implementation of other programming languages. The entities involved would be *Packages*, *Modules*, *Methods*, *Constants*, *Users*, *Authors*, *User Code*, *Source Code*, *Keywords* and *History*. The *Package* entity contains a package name and filepath, and are the folders for *Modules*. *Modules* have a module name, list of *Constants*, and list of *Methods*, and a list of *Dependencies*. *Constants* have a constant name, *Module it belongs to*, constant type, a list of *Keywords*, and a constant description and are contained within *Modules*. *Methods* have a method name, *Module* it belongs to, parameters, a return value, return value type, a list of keywords, a method description, and a list of similar methods and are also contained within *Modules*. *Modules* also contain the *Source Code* and/or *User Code*, which have text and a file size. The *User Code* (ID'ed by *Filepath*) contains the text and file size in bytes and must be written by a *User* who contributes as an *Author*. *Source Code* and *User Code* must be written by an *Author* with an ID, an author name, and the list of files worked on. The *Users* have a unique ID, a *Username*, list of *Favorite Packages*, list of *Favorite Modules*, list of *Favorite Methods*, and a *History* associated with them. *Keywords* have a list of related *Keywords*, and *History* has the date searched, time searched, and *Filepath* of the thing searched. We will be pursuing the Web Front-End Option.

https://lucid.app/lucidchart/invitations/accept/34793a6c-6874-4205-93d6-31b365957c61

A brief "data plan":

To start off with, we will be using a database of methods that Sedona has been creating over the past few years of Computer Science classes. Eventually, we can move onto using the Python documentation as well.

https://1drv.ms/u/s!AjqoqwpaY7NMlsBJY2vs_y2335kkbQ

https://docs.python.org/3/py-modindex.html

(Web Front-End) A description of your user interaction plans:

The *User* can search the database for *Keywords*, *Method Names*, *Package Names*, and *Module Names* within a specified *Language*. For each search, they will receive a list of possible *Methods* that would match and otherwise would get the *Method* itself. The *User* would then select the method and the program would return a page containing the *Parameters* and *Returns* as well as a general *Method Description*. The page would also

display *Methods* within the same *Module* that may be of use and suggestions for related *Methods*. For a *Constant*, it would display a general description of what the constant represents, the *Constant Type*, and the *Constant Value*. *Users* are encouraged to add in their own *Keywords* on the page of each *Method*/*Constant* that would be added to the database and then associated with the *Method*/*Constant* when searching. The *User* can also look through the *History* of *Methods*/*Packages*/*Modules* that they have visited, as well as ones they have starred as *Favorites*.

Contingency plan:

Should either one of us drop the class, then we will remove the entire *User* and *Author* functionality, along with the *History* and *Favorites* relations, and then the remaining team member will be able to work with just the *Code*, *Packages*, *Modules*, and *Constants*/*Methods*.

SQL Schema:

```
CREATE TABLE package
(
        name varchar(50);
        primary key(name);
)

CREATE TABLE module
(
        name varchar(50);
        primary key(name);
)

CREATE TABLE constant
(
        name varchar(50);
        type varchar(50);
        value varchar(500);
        description varchar(50);
        primary key(name);
)

CREATE TABLE method
(
        name varchar(50);
        returns varchar(50);
        return_type varchar(50);
        description varchar(500);
        primary key(name);
)

CREATE TABLE parameter
(
        method_name varchar(50);
        parameter varchar(50);
        type varchar(50);
        primary key(method_name, parameter);
)

CREATE TABLE code
(
        filepath varchar(50);
        filename varchar(50);
```

```
        text varchar(10000);
        file_size int;
        primary key(filepath);
)

CREATE TABLE source_code
(
        filepath varchar(50);
        primary key(filepath);
        author-id int;
        foreign key(filepath) references code;
        foreign key(author-id) references author
)

CREATE TABLE user_code
(
        filepath varchar(50);
        primary key(filepath);
        user-id int;
        foreign key(filepath) references code;
        foreign key(user-id) references user
)

CREATE TABLE author
(
        author-id int;
        name varchar(50);
        primary key(author-id);
)

CREATE TABLE user
(
        user-id int;
        name varchar(50);
        primary key(user-id);
)

CREATE TABLE keyword
(
        keyword varchar(50);
        primary key(keyword);
)

CREATE TABLE history
```

```
(
	date_time datetime;
	type varchar(50);
	filepath varchar(50);
	search varchar(50);
	primary key(date_time);
	foreign key(filepath) references code;
)

CREATE TABLE package_containment
(
	filepath varchar(50);
	package_name varchar(50);
	primary key(filepath, package_name);
)

CREATE TABLE module_containment
(
	filepath varchar(50);
	module_name varchar(50);
	primary key(filepath, module_name);
)

CREATE TABLE package_module_containment
(
	package_name varchar(50);
	module_name varchar(50);
	primary key(package_name, module_name);
)

CREATE TABLE module_constant_containment
(
	module_name varchar(50);
	constant_name varchar(50);
	primary key(package_name, module_name);
)

CREATE TABLE module_method_containment
(
	module_name varchar(50);
	method_name varchar(50);
	primary key(module_name, method_name);
)
```

```
CREATE TABLE module_dependencies
(
        module_name varchar(50);
        imported_module_name varchar(50);
        primary key(package_name, module_name);
)

CREATE TABLE contributor
(
        author-id int;
        user-id int;
        primary key(author-id, user-id);
        foreign key(author-id) references author;
        foreign key(user-id) references user;
)

CREATE TABLE searched
(
        date_time datetime;
        user-id int;
        primary key(user-id, date_time);
        foreign key(date_time) references history;
        foreign key(user-id) references user;
)

CREATE TABLE module_favorite
(
        user-id int;
        module_name varchar(50);
        primary key(user-id, module_name);
        foreign key(user-id) references user;
        foreign key(module_name) references module;
)

CREATE TABLE method_favorite
(
        user-id int;
        method_name varchar(50);
        primary key(user-id, method_name);
        foreign key(user-id) references user;
        foreign key(method_name) references method;
)

CREATE TABLE constant_favorite
```

(

user-id int;
constant_name varchar(50);
primary key(user-id, constant_name);
foreign key(user-id) references user;
foreign key(constant_name) references constant;

)

Additional Constraints:

- module_dependencies refer to all modules imported by a given module