Source Code Documentation

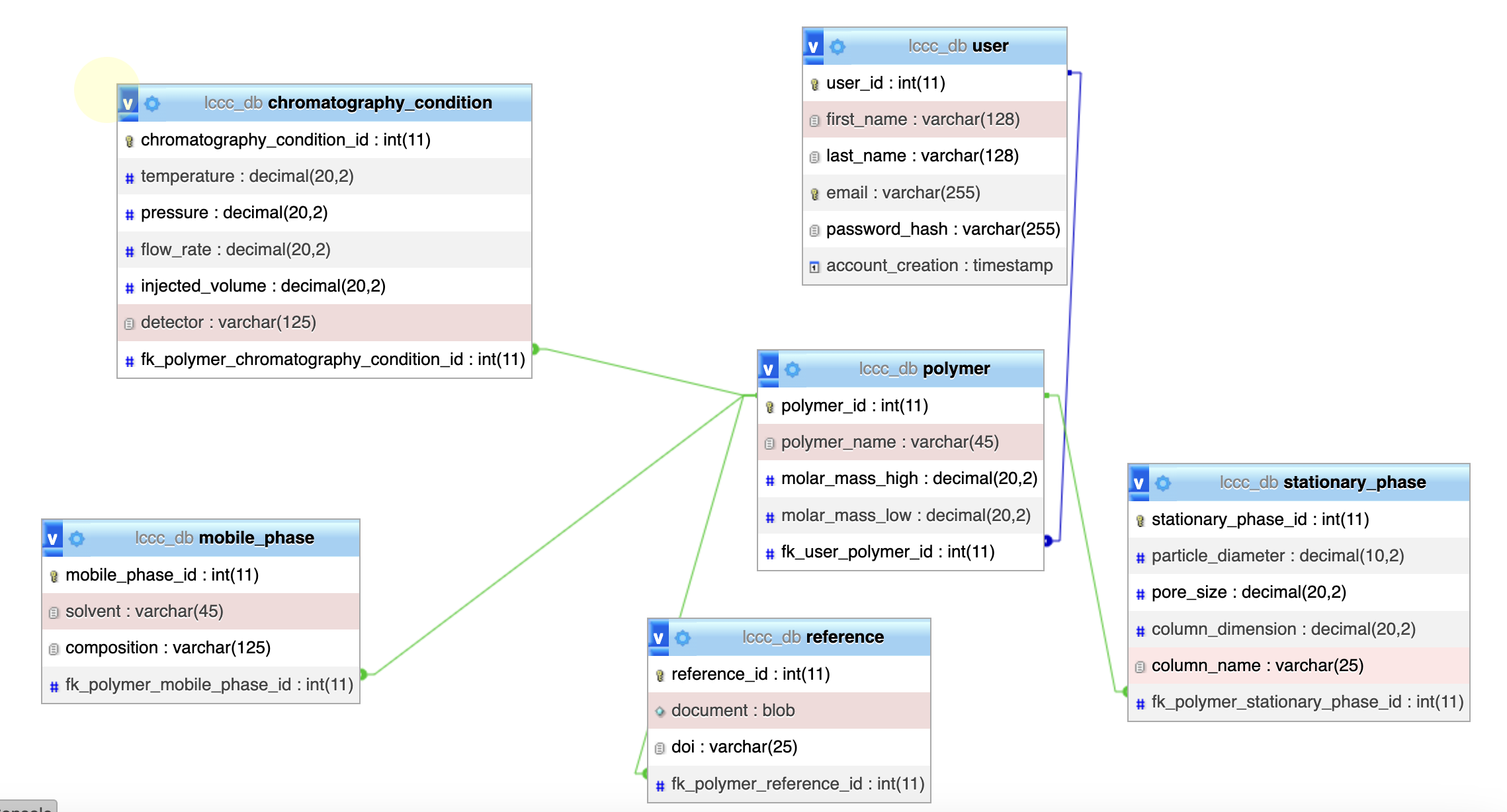
**Database**:

The name of the database is lccc\_db which is short for liquid chromatography at the critical condition database.

**Tables**

* polymer – This table holds the attrubutes for the polymer name, molar high, and molar low
* chromatography\_conditions - This table holds the attributes for temperature, pressure, flow rate, injected volume, and detector.
* mobile\_phase – This table holds the attributes for the solvent, and composition
* stationary\_phase – This table is for the attribute's particle diameter, pore size, column dimension, and column name
* references – This table holds the attributes for the documentation and DOI number. Currently, the documentation attribute is not used on the website but may be used in the future.
* user – This table stores the users first name, last name, email, password, and the date in which the user was registered. The password in the database is stored as a hash

Foreign key to primary keys – The database associates each of the polymers with the user's id by making the foreign key in the polymer table the same value as the user's unique primary key. Each other table in the database is connected to the polymer table by sharing the same primary key as the polymer's primary key. This means that the primary keys in each of the other tables also acts as the foreign key which will map to an individual polymer. The following image shows the attribute names and data types in their corresponding tables.



**Header file**:

In order to keep all the code throughout this project a little bit more organized and neater we have created a file called header.php that eliminates the redundancy of code by including the database connection, sourced in libraries, and navigation bar. This file is included at the top of every file for the website. The file also contains some conditionals for the navigation bar that will allow only the signed-in users to see certain navigation items. The session starts at the top of the header allows one to be able to use session variables in any of the files. Most of the scripts inside the header section are for importing in some functionality of bootstrap, ajax, and jQuery library.

**Polymer Entry file**:

In order to have the title of each of the files correct each of the files contains a PHP variable for making the title. The page checks if the user is signed in because a user is not allowed to enter any polymer data into the database if they have not registered their account yet. The page also includes some session variables for status checking. These variables are created on other pages and are meant to be used if the user is directed back to the polymer entry page. The file is a normal HTML form with an action that sends the user to the polymer preview page. There is a bunch of helpful tips that are commented out throughout the file and were meant to be in use, but the client never specified what exactly they wanted in the help tips so, they remain commented out to make the page look a bit cleaner. There is also a section for uploading documentation that is commented out. The reasoning for commenting it is that the client originally wanted a way to be able to upload a document but never really gave any clarification into how it is any different from just utilizing the DOI URL to go to the official documentation that is published. The file also includes a script at the bottom which utilizes the form display js file. The file's purpose is to display the composition based on what the user inputs into the solvent. Most of the content for the form was organized by using bootstraps grid functionality.

**Polymer Preview**:

The preview file is meant to give the user a way to preview the entry page data before submitting. The file first checks to ensure the user clicked on the continue button on the entry page. If the name was not set the user will be redirected back to the entry page. The file also contains validation for ensuring that all required inputs have been filled in and that the user has not entered invalid data. The file also uses PHP to get the values past from the form on the entry page and then assign them to session variables. Any entry that the user did not enter will be displayed as N/A by default. The file also checks to see if the user entered a DOI number or DOI URL. The file has a script at the bottom which allows the user to decide to enter multiple entries. When the user clicks submit it triggers the JavaScript function to execute prompting the use to enter in the text yes or no. If you type in yes, the entry will be submitted to the database and redirect the user back to the entry page with all their previous data still in the form. If the user enters no, then they will be redirected to the polymer search page where they can begin to search up any entry. The file also contains another script linked to the form display comp on prev js file. This file is just meant to have the composition information displayed. The edit button allows users to return to the previous page with all their data intact. In case the user entered in any spaces each of the strings are trimmed to ensure proper validation.

**Polymer Insert**:

This file is the one responsible for storing all the information that was entered into the form on the polymer entry page. Each session variable created in the preview file is now assigned to separate PHP variables being inserted into the database. The first thing inserted into the database needs to be the polymer information. The reason for doing this first is that polymer id is needed to properly create the foreign key associated with the other tables. The query success variable is just a Boolean variable that checks if the query into the database was successful or not. The only query that is checked is the polymer entry mostly because of how important it is. The file contains a bunch of echo statements that are there mostly for debugging. If any of the echo statements show, then something went wrong with one of the database inserts. For adding null values into the database, you must use the NULLIF function in the query statement. If you do not use that operation, the database will not know what value to put into it. After creating a query statement, you must use the mysqli\_query function to grab the connection to the database and then run the insert. If something goes wrong with this part the page would originally run into a 500 error or just a blank page. Inserting the data into the database can be difficult because when something goes wrong with communication with the server, there is no good error messaging. I believe there may some sort of extension you get for the visual studio code editor or some alternative way to get better error messages. The last part of this file is just checking whether the user decided to enter multiple entries or not. It is important to note that the header function allows you to redirect users with PHP. If you are not familiar with file navigation the “../” means to back out of a file and then each “/” allows you to navigate into a file. This is especially essential for understanding how the different files in the overall code navigate to each other. The bottom functions are responsible for also setting the session status variable which is used to display a message on whatever page the user is directed to.

**Connection**:

This file is responsible for establishing the connection to the server. It first requires the host, user, password, and database name. After that, you create the global connection variable by using the mysqli\_connect and include all that information as the arguments to be used for the function.

**Polymer search**:

This file uses something called pagination in order to nicely display the data on the page. The Way to control the number of entries that are displayed on each page is to change the value of the num\_per\_page and the value in the start\_from variable which is responsible for the page start. The file immediately runs to queries. The first one is the actual query that is being generated on the page and the second one is just used to get the total amount of entries that are stored in the database. The file also contains some PHP for checking the session status variables. The file contains code for the keyword search bar which is just a small form with a post method that will direct the user to the search results. The file utilizes aspects of JQuery to incorporate a multi-search option. Multi-search is designed to help the user narrow down whatever they are searching for. The multi-search duplicates the same input box which is an input group that contains a select box and an input box. When the user clicks on the add button it executes the JavaScript code put in more input boxes. The input boxes are capped at 13. To track all the input data the name value submitted in this form must be arrays. This will allow there to be multiple entries to be sent to the next page. The data's content is just a while loop that is grabbing all the rows of data from the database. It displays the information by using PHP to echo the queried results. The last section of code on the page is responsible for gathering the data that will display on the following pages by running a pre-query to get the content for the next pages. The code just includes some methods for navigating to the last page, first page, next page, and previous page/.

**Polymer search results**:

This file is responsible for taking the information from the user on the polymer search page and displaying the results accordingly. The main first set of conditionals that run is checking to see which form the user submitted via either the search or multi-search. If the user entered just the keyword search the text that the user input is assigned to a PHP variable and then compare to all the data in the database. This search produced a lot of duplicate entries. To keep the data from being duplicated the query runs the comparison over very specific queries and then unions the results. This needs to probably later be optimized to use a join function or something to make the code take fewer lines. If the query is successful, then the code utilizes the same method from the search page and runs a while loop on the results and displays all the results on the page. It is important to note that the pagination for this file was not working properly so, so one of the biggest issues with this film is that it could potentially in the future display more data than the web page can handle. Multi-search is a little bit trickier because it needs to compare two different input fields that have data stored as an array. In order to ensure that the data is being compared to the selected option I ran the query inside an O(n2) nested for loop. Essentially the nested loop makes sure that the compared values are of the same index. If they are in the same index location in their corresponding arrays, then just run the like operation. After the query is completed, the results are run through a while loop to display all the appropriate data. This file just includes a back option to get to the search page.

**Sign\_up file**:

This file is used to allow the user to create an account. There were many links used to create this file. Google reCAPTCHA for reCAPTCHA version 2, water.css for registration form format. Font-awesome which is used to add symbols to the page, and just-validate which is a JavaScript validation tool that is used to make it easier to provide custom validation to a form. Below this is a block of code that handles the reCHAPTCHA V2. This uses the syntax that is on Google Developers page that is needed to setup the reCAPTCHA. This is where the keys are generated for the domain that is being used for this to work. The keys must be tied to a domain and the domain in this case is localhost. This block of code just allows the page and google reCAPTCHA to connect and show the box for the user to use. Below this code is where the form page is set up with the required fields. All of this is structured with HTML and CSS. There are textboxes for the necessary registration data, reCAPTCHA for more authentication, and then the CSS below this code. Also, imported in the “form\_validation.js” file which has all of the validation for the fields for this registration form. This file also has a header.php file at the top which shows the navbar at the top of the page and there is a footer.php file which shows the contact information.

**Sign-up-insert file**:

This file is used to connect the names of the fields in the Sign\_up file to the database. Here the names are stored into PHP variables. The data input through the registration page is then inserted into the attributes in the database. The table that this data is being inserted into is the “user” table. Once the data has been sent to the database then the user is then redirected to the Sign in page. This Sign-up-insert file is used to connect the data from the sign\_up file to the database.

**Sign\_in.php file**:

This file is used to allow the user to sign into their account once that account has been created. At the top of the code there is an if statement to check if user is signed in and if the user is signed in then it will show profile button on the navbar. The code block below is used to retrieve the data from the connection.php which is used to connect to the database. All the attributes from the user table are selected and then the user is checked to see if the login credentials being used are legitimate. A session is started for the unique user if they successfully sign in and then they are redirected to the polymer\_search page where they can search for different polymers. This checks to see if the data the user input is correct or not. Below this code is where there is a session created for a success alert. This will show a green success flash box when a user successfully registers their account. Below this is where the login form is created with the necessary textboxes, “Email” and “Password”. After the form HTML code is CSS. This styles the form page and the textboxes. The login form validation that is being used is the built-in validation from HTML.

**Process signup file**:

This file processes the signup data that is inputted from the sign\_up file. The code after the connection has been imported is used to take the password from the database and hash it. The password\_hash function was used to hash the password to a default value.

**User\_logged\_in** **file**:

This file is used to allow the user to sign in. The if statement at the top is used to check if there is a unique session value that has been set. This unique value is the primary key for the id in the user table in the database. If this has been set, then retrieve that unique id from the database and then connect it to the session. The if statement at the bottom of the code is used to display the first name of the user. This data is retrieved from the user table in the database.

**User\_logged\_out file**:

This file is used to take the session that was started when a user was signed into their account and then destroy it when they click on the sign out button on the navbar. This basically allows the user to sign out. When the user signs out then they get redirected back to the sign in page. The SQL statement below inserts the data from the sign-up-insert page into the attributes in the user table in the database. There is an if statement below to check to see if an error has occurred or not. The parameters are then binded to the variables that were created in the process-sign-up file. This allows the information to be sent to the database. An if statement is then used below this to check if the sign up was successful and the data was submitted to the database. If this was successful, then the user will then be redirected back to the sign in page. This will flash the green success flash on that page. Below this is an if statement that check to see if the email that was submitted has already been taken or not.

**Email\_validator file**:

This file checks to see if the email is available or not. There is a SQL query that is used to retrieve the emails in the database and check if they are in use. There is section of code in the “form\_validation.js” file that is used to validate the email. If it is in use, then show a message that says It is taken.

**Form\_validation file**:

This file is used to provide most of the validation for the registration page for the textboxes. JustValidate was being used here to provide a faster way to add custom validation. The validation was added for all of the textboxes on the sign\_up page. The fields are added for the unique fields and then the rules are set for the textboxes. Custom validation is then used to specify many constraints like the length, characters, and matching.

**User\_data file:**

The `user\_data.php` file is a web page that enables signed-in users to view and delete their own polymer entered data. The page displays one or more tables containing information about the user's polymer data, such as the polymer name and other relevant details. To delete an entry, the user simply clicks the delete button next to it. This triggers the `POST` action, which removes the data from the database table based on the poly\_id associated with that table.