A Database for Indexable Carbide Inserts

A Senior Project

presented to

the Faculty of California Polytechnic State University

San Luis Obispo

In Partial Fulfillment

of the Requirements for the Degree

Bachelor of Science in Computer Engineering

by

Andrew Neal Yoder

June, 2019

Table of Contents

1. INTRODUCTION

The indexable inserts project is a collaborative effort to aggregate into a single database as many indexable carbide inserts from as many manufacturers as possible. Inserts are generally labeled with a part number following a specific standard determined by shapes and measurements, however specifications for certain aspects of carbide inserts—such as which materials they can cut—can vary by manufacturer. There currently is not a way to search a comprehensive database containing tools from multiple manufacturers for a handful of inserts that would satisfy some necessary parameters, making finding the correct tool in a shop a much more time-consuming process than it needs to be. By compiling a single, searchable database of indexable inserts from multiple manufacturers, as well as incorporating a shop-specific inventory system and easy-to-read labeling system, we will make tool organization and selection a much simpler task.

For the back end of this project, we will be making a database of carbide insert specifications from the most prominent manufacturers; time limiting, we plan to have two to three manufacturers in the database by the end of the project and potentially more in the future. Adam, my partner for this project, developed a web scraper to collect this data from manufacturers' websites. For the front end, we will construct a website with the primary focus being displaying this data in a way that is easy to search and read. We also plan to implement an inventory support for clients, allowing them to search their own shop's inventory for whatever tool is needed and catalog their stock of each tool. Another aspect to our project is generating custom labels for each insert to help organize tools in a client's shop, though this is a secondary objective.

2. BACKGROUND

Carbide Inserts

Cutting metal with a CNC machine requires indexable carbide inserts attached to a holder; the inserts are the piece that make contact with the raw material. Depending on the type of material used and the desired cut, an operator will need to find specific inserts to use. These inserts vary by shape, size, materials they are rated to cut, and type of cutting (e.g. turning or milling), and there are thousands of possible variations. Most inserts follow a standard naming convention that is determined by several characteristics, such as the insert’s shape, size, nose radius, and thickness, among others; however, manufacturers often have unique conventions for determining which materials an insert can cut and the appropriate speeds and feeds for those inserts—an identifier called the grade.

For the scope of this project, we decided to only focus on the ANSI cataloging convention.[1] The standard is outlined below [1], followed by a brief example.

> According to ANSI B212.4-2002 standard, identification of the indexable insert includes 10 positions denoted by a capital letter. Each position (from 1 to 10) defines a characteristic of the insert in the following order:
>
> – Shape;
> – Clearance [or Relief Angle];
> – Tolerance class;
> – Hole / Chip Breaker;
> – Size;
> – Thickness;
> – Cutting-point configuration;
> – Edge preparation;
> – Hand;
> – Facet size.

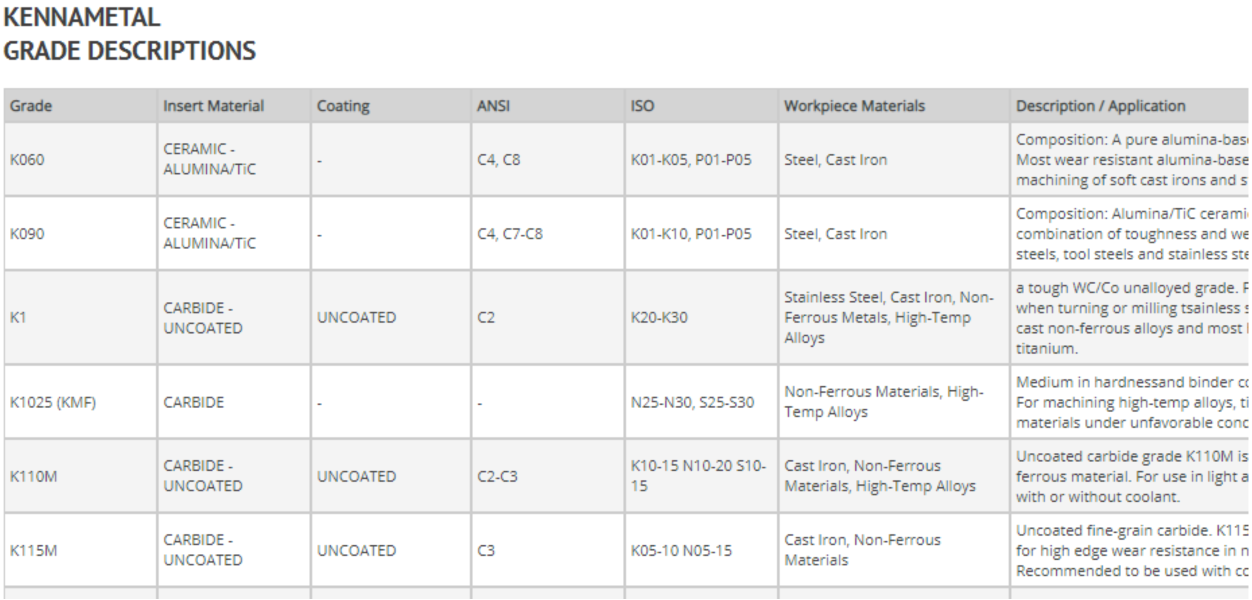
[1] We hope to include the ISO convention at a later date.

For example,

`CPMT32.51FL`

`C:`	diamond 80°
`P:`	11° relief angle
`M:`	M-class, 0.005” tolerance
`T:`	40-60° double countersink hole, single-sided chip breaker
`3:`	3/8” cutting edge length
`2.5:`	5/32” thickness
`1:`	1/64” radius
`F:`	sharp cutting edge
`L:`	left-hand
`-:`	-

Another important identifier is the grade. Since these vary by manufacturer, operators must rely on different look-up tables depending on the inserts they have at hand—sometimes they may need to compare inserts from several manufacturers. The first few rows from Kennametal’s look-up table[2] can be seen in Figure A.



KENNAMETAL
GRADE DESCRIPTIONS

Grade	Insert Material	Coating	ANSI	ISO	Workpiece Materials	Description / Application
K060	CERAMIC - ALUMINA/TiC	-	C4, C8	K01-K05, P01-P05	Steel, Cast Iron	Composition: A pure alumina-bas Most wear resistant alumina-base machining of soft cast irons and s
K090	CERAMIC - ALUMINA/TiC	-	C4, C7-C8	K01-K10, P01-P05	Steel, Cast Iron	Composition: Alumina/TiC cerami combination of toughness and we steels, tool steels and stainless ste
K1	CARBIDE - UNCOATED	UNCOATED	C2	K20-K30	Stainless Steel, Cast Iron, Non-Ferrous Metals, High-Temp Alloys	a tough WC/Co unalloyed grade. F when turning or milling tsainless s cast non-ferrous alloys and most l titanium.
K1025 (KMF)	CARBIDE	-	-	N25-N30, S25-S30	Non-Ferrous Materials, High-Temp Alloys	Medium in hardnessand binder co For machining high-temp alloys, ti materials under unfavorable cond
K110M	CARBIDE - UNCOATED	UNCOATED	C2-C3	K10-15 N10-20 S10-15	Cast Iron, Non-Ferrous Materials, High-Temp Alloys	Uncoated carbide grade K110M is ferrous material. For use in light a with or without coolant.
K115M	CARBIDE - UNCOATED	UNCOATED	C3	K05-10 N05-15	Cast Iron, Non-Ferrous Materials	Uncoated fine-grain carbide. K115 for high edge wear resistance in n Recommended to be used with co

Figure A. Kennametal’s Grade Look-Up Table, given as an example of a grade.

[2] Full table available: https://cets.com/resources/kennametal-grades

While a lot of useful sizing information about a given insert is available by examining the ANSI or ISO identification number, an insert's grade is especially important for determining which materials a part can be used with. From that we can determine the appropriate speed and feed rates, or how fast the machine and workpiece material should be moving in relation to each other to result in the smoothest cuts. As has been stated, inserts' grades vary by manufacturer, and this is the big limitation that has prevented a database being compiled that allows operators to search for inserts across multiple manufacturers. By going to a manufacturer's website, operators are able to search for appropriate inserts in a similar manner to this project, however this project is aimed at the majority machining shops which have parts from multiple manufacturers. By eliminating the need to search for parts across multiple manufacturer databases, as well as creating a consistent label, we will reduce the overhead required for shops to complete projects.

The Database

A database is a memory tool that can be used to manage and search a lot of data. A database table is essentially a big spreadsheet that we can interact with by using specific commands and arguments. A relational database is one which has tables that are organized by rows of objects that have different attributes all relating to the same object; these attributes are organized as columns [2]. A relational database is preferred over a non-relational (or a "NoSQL") database when data is structured and consistent [3]. For this project, we opted to use a relational database because we are categorizing a large amount of data that will have similar, if not identical, attributes. Relational databases are also advantageous since they can be searched and indexed quite efficiently [3]. Structure Query Language, or SQL, is the language used to interact with relational databases [4]. Commands allow a person to create, edit, search, or delete any aspects of a table. Arguments specify which table(s) we care about, which columns or specific values we care about, and so on. An example SQL query follows:

```
SELECT name, age FROM people WHERE name = ‘Jerry’;
```

In this example, we are stating that we wish for the function to return a list of everyone named Jerry, as well as their age, from our table (called ‘people’ in this example). Queries can become much more complex since we have the ability to sort the data that is returned to us, grab data from multiple tables, specify multiple return parameters (e.g. `AND age > 21`), return only the `MAX` value of some parameter, as well as many other controls.

There are many different databases that we can use to maintain our data. The most popular are Oracle, MySQL, Microsoft SQL Server, PostgreSQL, and MongoDB [5]. They are all relatively similar and all rely on the same language—SQL—but there are minor differences. Oracle’s database has a price tag above what we were willing to spend, MySQL is incredibly popular with a lot of support around the Internet, and PostgreSQL is a slightly newer competitor to MySQL [6]. Having no experience with databases prior to this project, and with the only significant difference in most databases being the actual programs used to interact with them, I decided to go with PostgreSQL.

PostgreSQL can be used offline on local machines, but since this project involves clients accessing our website and searching our online database, hosting the PostgreSQL database online is a requirement. There are a variety of hosting services to choose from, but using one that provides Database as a Service can eliminate the backend server hosting requirements. I will elaborate more in the **Description** section below.

The Website

Finding a place to host a website is relatively easy. My only web experience prior to this project required me to learn a little bit about WordPress, so I started my research there. What I learned is that WordPress is a very flexible open source platform for creating websites, approximately 33% of websites on the Internet use WordPress, and it works really well with PHP and SQL [7]. I also learned that WordPress is just a platform, and I needed to find a hosting service that provided it. After exploring options such as WordPress (they have a hosting service as well), GoDaddy, and SiteGround, I determined SiteGround provides all of the tools I would need at their base cost. Since this website only needs one or two pages, I did not find it necessary to spend several hundred dollars for extraneous features. It did seem important to pay for a stable hosting service and our own domain name: indexable-inserts.com.

Putting the website together took more time than expected; it took some trial and error to find satisfactory plugins. The important characteristics of the website include an area to collect user input, or a form, and some php code to get that input, create an SQL query, and display the results from the database. WordPress makes it easy to create static text pages, but more dynamic pages require the use of plugins, which are addons created by others. This will be further explained in the **Description** section.

Putting It All Together

This project requires the use of a database and a website. It is not too complicated to put the two together, but it does take a few extra steps. Interacting with a database requires the use of SQL, and formatting a website is done using HTML. Another language, PHP, works as a good intermediary between HTML and SQL. We can paste PHP code snippets directly into our HTML code, and this basically tells HTML that we will be dynamically populating that section of the page. For example, if we wish to print the current date or use a user's name somewhere—things that cannot be hardcoded into HTML—PHP is necessary. In a PHP code snippet, we can connect to our database using the credentials given by Heroku, get user input to create an appropriate SQL query using PHP's `GET` method to find specific variable values, and then we can dynamically populate the HTML page using the results.

Perhaps the most important feature of PHP—at least in the context of this project—is the ability to gather and use client input. We can collect user input by creating a form in HTML and linking the Submit button to a php file, essentially telling the HTML file to `POST` the results to the PHP file, which can then `GET` them. An alternative method to `POST` and `GET` client input is a format called Ajax. Using Ajax is ideal for search functions because the search parameters are included in the URL of the results page, making it possible to save or share specific results. The basic syntax for Ajax follows:

http://www.somewebsite.com/welcome?name=Jerry

This URL would send us to the Welcome page of somewebsite.com, passing in the value "Jerry" for the variable "name". If we shared this URL, we would not need to collect user input again as we already have our input values.

To summarize, we are hosting our PostgreSQL database with Heroku, hosting our WordPress website with SiteGround, and using PHP to allow clients to interact with the database and show the results on our web page. The chosen method for getting user input is Ajax.

3. DESCRIPTION

The Database

There are several options for online database hosting; I explored Amazon's Relational Database Service ("RDS") and Heroku. Amazon's RDS allows customers to create "instances" of a server of our choosing, such as Linux. We can create a "local" database through that instance and connect to it with the proper credentials, and AWS only charges for the processing used—byproducts of uptime, total connections, and how often one interacts with the database. The issue with Amazon's RDS, however, is that it is just extremely difficult to work with. I spent days trying to troubleshoot, but always came to the same issue of not being able to reconnect to a database once I had disconnected from it.

After a lot of frustration trying to work with Amazon's RDS, I decided to try a different hosting service. Heroku is one that I had heard of in the past but did not know much about. Looking into it, I found that they have a Database as a Service feature that supports PostgreSQL [8]. There is a paid option for bigger projects and a free option for smaller projects (with a maximum of 10,000 rows), such as the scope of this one. Setting up the database with Heroku was much more straightforward compared to Amazon's RDS, and Heroku provides quite a few setup guides to help with the process and troubleshoot common issues [8]. Once set up, Heroku provides a URL to access the database, the name of the database, and login credentials, which can be used to access the database through a program like pgAdmin or through the CLI. The program pgAdmin is an administrative tool with a GUI to help maintain a PostgreSQL database; it is advantageous for simply viewing the attributes of a database, such as which tables exist, which columns exist in each table, and how many rows of data we have stored in a table. An alternative is simply interacting with a database through the command line. The command line is better for actually modifying the database using SQL, at least from my experience.

Creating a table in our database is not very complicated. After downloading Heroku's CLI package, we can simply open our command prompt, connect to the database using the credentials given by Heroku, and start typing SQL queries. We can connect to our database through the command line by typing `heroku psql --app app-name`, where my app name is indexable-inserts. From there, we can create a table by typing the SQL query

```
CREATE TABLE public.inserts (
     ANSI_catalog_num varchar(30),
     ISO_catalog_num  varchar(30),
     .
     .
     .
     nose_radius      varchar(30)
);
```

This tells the database to create a public table called "inserts" and to create a column for each of the given variables and data types. Adam scraped data for a few thousand indexable inserts and placed the results in a CSV file. We can relatively easily copy the data from a CSV file into an SQL table [9] by typing

```
COPY table(field1, field2, …) FROM ‘path’ DELIMETER ‘,’ CSV
HEADER;
```

The table and fields would be replaced with our table name and each column in our table, the path denotes the local path to the CSV file on our machine, and the CSV HEADER option essentially just skips the first row in the CSV file since it contains column names and not any actual data. One caveat I experienced is that the `COPY` command requires superuser privilege—while this is not a big issue, I had some trouble gaining superuser privileges with Heroku and I found it was easier to simply replace `COPY` with `/copy`, a command available with Heroku's command line interface.

The Website

Hosting the website required a lot less background knowledge than learning about creating, hosting, and interacting with a database. After paying for SiteGround's hosting service, I gained access to the WordPress Administrative Dashboard for the website. From the dashboard, it is relatively straightforward to create different pages on the website and add navigation. We decided that we really only need two pages on the website, at least at this stage: one for a form to collect client search inputs and another to display the resulting table.

WordPress provides the fundamentals for creating a website, but additional plugins help to expand functionality. For this website, I needed a plugin to create PHP code snippets that can be copied into an HTML block on a given page. I also needed a way to create forms that can post input values in the Ajax format.

I explored several options for using PHP code snippets, but eventually landed with the Woody Snippets plugin because the free version had all of the functionality I needed. I learned about Woody Snippets by examining the backend of Central Pacific Ski Club's website so I could see how they went about solving similar problems. (I was formerly CPSC's webmaster and simply asked for temporary administrative access). To use a PHP code snippet, we just need to create a php snippet file through Woody Snippets, which will then provide a reference link (e.g. `[wbcr_php_snippet id="19"]`) that can be pasted into an HTML page. When the page loads, the HTML will know to run the PHP code to populate that section of the page.

For collecting client search queries, it is necessary to use a form, a utility that allows users to enter various types of data and "submit" that data to be used by the webpage. Forms can be created through HTML code and entries can be collected by using PHP requests. Alternatively, many plugins also exist for creating forms with a GUI, which is helpful for more complex forms with multiple input options. For this project, forms were constructed through the Ninja Forms plugin. I created two forms: the first for user-entered text corresponding to ANSI part numbers and grades, and the second containing dropdown menus populated with each possibility for an insert's shape and sizing characteristics. Form submissions result in a page redirect to the /result page with Ajax parameter data. For example, if I entered the workpiece material as "N – Non-ferrous" and the relief angle as "P – 11°", the redirect URL would be

/result?material=N&shape=&relief=P&size=&thickness=&nose=

with parameters specified after the question mark, separated by ampersands, and either containing a value or NULL. On the result page, we can paste a reference to one of our PHP code snippets that gets the form submission values, constructs an SQL query, and displays the final table.

Searching the Database and Displaying the Results

Once the input form has been submitted, we can construct SQL queries as a PHP code snippet. The process is to first define our connection variables, such as the host URL, the name of the database, the username and password to access this database, and the port number [10]. Once we have these values, we can pass them into the `pg_connect()` function, which returns a PostgreSQL connection; we can store this connection as a variable, i.e. `$connection = pg_connect().` The next step is to set up our SQL query, which will be of the form `SELECT *`

FROM $table. This query, specifically, will simply return every entry from our table; for our case, $table is a variable with the value ‘inserts’. The * symbol acts as a wildcard, saying "everything." In PHP, the $ symbol denotes a variable [11]. After setting up our query string ($query), we can pass the connection and query variables to the pg_query() function, like so:

```
$result = pg_query($connection, $query);
```

From here, we just need to iterate through $result by calling pg_fetch_array(), which returns one row at a time as an array. We can use the array for each row to populate the results table that we wish to display, selecting whichever columns we care about and iterating through $result until pg_fetch_array() returns NULL [6].

Since inserts' grades are manufacturer-specific, we must search multiple tables to construct our query if a specific grade or material is given as a search parameter. The queries for each table will differ by manufacturer. For Kennametal, the table that we scraped (Figure A) conveniently has a column for the ISO grade. Interestingly, most inserts are not characterized by an ANSI or an ISO grade, despite the naming conventions existing. Each workpiece material has a corresponding letter associated with it, which is identical to the letters given for ISO grades. To construct an array of grades fitting our search parameters, we can use the SQL query

```
SELECT grade FROM grades_table WHERE iso LIKE ‘%$material%’;
```

Once we have a table containing each matching grade, we can iterate through the table and add the manufacturer grade to an array, which will be used for the next database query. For other manufacturers, the grades table will often contain a column specifying workpiece materials an insert can be used with which can be searched to find a list of matching grades.

Once we have constructed variables pertaining to a part number, grade, or array of grades, we can insert them into another SQL query to search the actual inserts database. The final query will look something like this:

```
SELECT ansi_part_number, grade, manufacturer FROM inserts_table
WHERE ansi_part_number LIKE ‘%$partNumber%’ AND grade IN
$gradesArray;
```

This will return all of the inserts available in our database that fit the search parameters, and from here we can iterate through the result to construct a readable table on the result page.

4. EVALUATION

My portion of this project involved two major components: (1) creating and populating the inserts database, and (2) setting up a website for clients to interact with that database. For the database, I needed to populate the actual inserts database with all of the data that Adam scraped from manufacturer websites. Additional look-up tables are required to find appropriate grades depending on a part's manufacturer, as well as to determine the proper speeds and feeds for a particular insert and material. We also had a plan to implement an inventory management system on the website, but since almost every aspect of this project was new to both Adam and I, everything took considerably longer than anticipated. Our best approach for this, at the moment, is to implement a table for each client that would contain the quantity of all inserts available in their shop. Ideally, we would have the ability for clients to login and access their shop's inventory table, but I left that out of the evaluation metric because I was not sure if we would get to that point by the end of the project.

The evaluation metric I specified for the project follows:

Database

End Goal	*Requirement Met?*	*Steps Required to Complete*
Inserts table populated	Yes	N/A
Grades tables	Partially	Add more manufacturers
Inventory management	No	Unsure

The inserts table is populated with inserts from Kennametal and Mitsubishi; we plan to add more manufacturers in the future. The database currently only has a grade table for Kennametal, however more are needed to extend the usefulness of our search utility. The inventory management aspect would also improve the usefulness of our search utility for shop owners, though the current state of the project simply returns relevant inserts that a shop may or may not have.

Website

End Goal	*Requirement Met?*	*Steps Required to Complete*
Accessible	Yes	N/A
Displays Table Cleanly	Partially	Interviews and research to determine which columns people find most helpful
Searchable	Yes	N/A

As for the website requirements, it exists and is accessible to anyone with the URL. It accepts user search parameters and displays the correct results from the database in an easy-to-read format, however inserts tables contain a lot of data about each insert which may also be useful if displayed as a part of the results table. More work is needed to determine a way to display a lot more data about each insert while maintaining the easy-to-read aspect.

5. CONCLUSION

The indexable inserts project allowed me to learn about databases and SQL and gain a little experience with web development. Overall, I would consider this project mostly a success. The main framework and functionality are implemented, but a lot of aspects of the project—such as the number of manufacturers we have data for—would ideally be more extensive; due to time constraints, however, it is understandable that we were limited in the amount of data we could collect. I also wish I had speed and feed tables in the database and knew how to display them in an easy-to-read format alongside the resulting parts table. Additionally, it has been difficult to determine what data is important for each insert and what data is unnecessary to include because neither Adam nor I have any experience actually using carbide inserts.

Creating the website gave me exposure to PHP and HTML and using forms to collect user input. If I had more time, the website could be much more aesthetically pleasing, but I learned the fundamentals for what was necessary for the project and it seems to function just fine. In addition to simply improving the look and feel of the website, future goals for this project include implementing the inventory management system and creating a way to generate physical labels for shop owners to use in their shops to find parts and corresponding data much more easily. Data protection and system security is also a future goal as I did not have much time or knowledge to focus on security principles in the scope of this project.

This project may not yet be in its final stages, but the overall business idea is still a possibility in our minds since there is currently no other comprehensive inserts database that is marketed to machine shops. Once more manufacturers are added to the database and we have an inventory management system in place, we can begin marketing the service to clients.

BIBLIOGRAPHY

[1] E. Isakov, "Understanding the Identification System for Indexable Inserts," 1 October 2014. [Online]. Available: https://www.ctemag.com/news/articles/understanding-identification-system-indexable-inserts. [Accessed May 2019].

[2] Amazon Web Services, "What is a Relational Database?," [Online]. Available: https://aws.amazon.com/relational-database/. [Accessed March 2019].

[3] K. D. Foote, "A Review of Different Database Types: Relational versus Non-Relational," 21 December 2016. [Online]. Available: https://www.dataversity.net/review-pros-cons-different-databases-relational-versus-non-relational/. [Accessed February 2019].

[4] w3resource, "PostgreSQL SELECT," 11 April 2019. [Online]. Available: https://www.w3resource.com/PostgreSQL/select.php. [Accessed May 2019].

[5] DB-Engines, "DB-Engines Ranking," [Online]. Available: https://db-engines.com/en/ranking. [Accessed February 2019].

[6] PostgreSQL, "PostgreSQL: The World's Most Advanced Open Source Relational Database," [Online]. Available: https://www.postgresql.org/. [Accessed February 2019].

[7] WordPress, "About Us," [Online]. Available: https://wordpress.org/about/. [Accessed May 2019].

[8] Heroku, "Heroku Postgres," [Online]. Available: https://www.heroku.com/postgres. [Accessed March 2019].

[9] PostgreSQL Tutorial, "Import CSV File Into PostgreSQL Table," [Online]. Available: http://www.postgresqltutorial.com/import-csv-file-into-posgresql-table/. [Accessed March 2019].

[10] T. Sisson, "Connecting to a Database using PHP," 30 October 2018. [Online]. Available: https://www.inmotionhosting.com/support/website/database-connections/connecting-to-a-database-using-php. [Accessed April 2019].

[11] PHP, "PostgreSQL," [Online]. Available: https://www.php.net/manual/en/book.pgsql.php. [Accessed April 2019].

[12] C. Kerstiens, "Postgres Guide," [Online]. Available: http://postgresguide.com/. [Accessed February 2019].

[13] T. Boronczyk, "Hosted PostgreSQL with Heroku," 20 December 2011. [Online]. Available: https://www.sitepoint.com/hosted-postgresql-with-heroku/. [Accessed March 2019].

[14] NuSphere Corp, "PostgreSQL Functions," 19 September 2006. [Online]. Available: http://www.nusphere.com/kb/phpmanual/ref.pgsql.htm. [Accessed May 2019].

[15] K. Yank, "Build Your Own Database Driven Web Site Using PHP & MySQL," 7 July 2009. [Online]. Available: https://www.sitepoint.com/php-amp-mysql-1-installation/. [Accessed May 2019].

APPENDIX: PHP Code Snippet for Querying the Database

Note: Some values, such as login credentials and table names, have been changed for security purposes.

```
# database connection code modeled from
# https://www.inmotionhosting.com/support/website/database-
# connections/connecting-to-a-database-using-php

# Setup connection variables, such as database username and password
$hostname="host";
$username="username";
$password="password";
$dbname="database";
$port="port";
$usertable="table_name";
$gradestable="grades_table";

# Connect to the database with pg_connect()
# https://www.php.net/manual/en/function.pg-connect.php
$connection = pg_connect("host='$hostname' port='$port' user='$username'
password='$password' dbname='$dbname'");

# check connection
if (!$connection) {
      echo "an error occurred.<br>";
      exit;
}

# these will be set if we came from the "search" page
$pn = $_GET['pn'];
$gr = $_GET['gr'];
if ( strlen($pn) > 0 ) {
      $partNum = "%" . strtoupper($pn);
}
if ( strlen($gr) > 0 ) {
      $grade = strtoupper($_GET['gr']);
}

# otherwise get data from the "find" page
if ( (strlen($partNum) == 0) && (strlen($grade) == 0) ) {
      $ma = $_GET['material'];
      $sh = $_GET['shape'];
      $ra = $_GET['relief'];
      $si = $_GET['size'];
      $th = $_GET['thickness'];
      $no = $_GET['nose'];

      # figure out the ANSI part number from given parameters
      $partNum = "";

      # append value or placeholder to partNum
      $partNum .= ((strlen($sh) > 0) ? $sh : "_");
      $partNum .= ((strlen($ra) > 0) ? $ra : "_");
      $partNum .= "__";
      $partNum .= ((strlen($si) > 0) ? $si : "%");
      $partNum .= ((strlen($th) > 0) ? $th : "%");
      $partNum .= ((strlen($no) > 0) ? $no : "%");

      # if material is specified, we only care about certain grades
      if (strlen($ma) > 0) {
            $gradequery = "SELECT Grade FROM $gradestable WHERE ISO LIKE
                  '%$ma%'";
            $grades = pg_query($connection, $gradequery);
            $gradesarray = array();

            # append ‘em all to the array
            while ( $row=pg_fetch_array($grades, NULL, PGSQL_BOTH) ) {
                  array_push($gradesarray, $row['grade']);
            }
      }
}

# get the last part of the query together, WHERE $sel
$sel = "";

# if the partNum variable has anything...
if (strlen($partNum) > 0) {
            $sel .= "ANSI_catalog_number LIKE '$partNum%'";
}
# if both are set, we need the AND
if ((strlen($grade) > 0) && (strlen($partNum) > 0)) {
      $sel .= " AND grade LIKE '$grade%'";
}
# don’t want that AND if there’s nothing before it
else if (strlen($grade) > 0) {
      $sel .= "grade LIKE '$grade%'";
}
# in case we’re dealing with the array rather than a specific grade
else if (strlen($ma) > 0) {
      $sel .= " AND grade IN ('".implode("','", $gradesarray)."')";
}

# Setup our query
$query = "SELECT ANSI_catalog_number, grade FROM $usertable WHERE $sel";

# Run the Query
$result = pg_query($connection, $query);
```

```
?>

# display the results
<html>
<style>
{    border: 1px solid black;    }
</style>
<table id="database-results" style="border: 1px solid black">
    <tr>
        <th>ANSI</th>
        <th>Grade</th>
        <th>Manufacturer</th>
    </tr>
<?php   while ( $row=pg_fetch_array($result, NULL, PGSQL_BOTH) )  { ?>
        <tr>
            <td><?=$row['ansi_catalog_number'];?></td>
            <td><?=$row['grade'];?></td>
            <td><?=$row['manufacturer'];?></td>
        </tr>
<?php } ?>

</table>
</html>
```