Final Project: PostgreSQL

Larchelle Tuifao, Elora Tonaki, Alexis-Rachelle, Richard Mackey, Christopher Knowles

Chaminade University

CS-200 SQL and Relational Databases

Dr. Rylan Chong

December 1, 2023

I. DESCRIPTION

PostgreSQL is a "powerful, open source object-relational database system that uses and extends the SQL language combined with many features that safely store and scale the most complicated data workloads. The origins of PostgreSQL date back to 1986 as part of the POSTGRES project at the University of California at Berkeley and has more than 35 years of active development on the core platform" (PostgreSQL, *About* 1996-2023).

PostgreSQL has become a popular choice for open source relational databases for many people and organizations. This strong reputation has been earned through its proven architecture, reliability, data integrity, robust feature set, extensibility, and its dedication to making sure PostgreSQL runs well and consistently innovative.

II. KEY FEATURES IN COMPARISON

Key Features	MySQL	PostgreSQL
Database Type	Relational	Object-Relational
Syntax	Limited	Easy to read
ACID Compliance	ACID compliance, but MyISAM doesn't support ACID	Fully supported
Backup and Recovery	Good	Efficient
Cross-Platform	Yes	Optimal on UNIX-based systems
Save & Code Retrieval	Supported	Supported to an extent
Foreign Keys	Supported, not in MyISAM	Fully supported
Indexing Techniques	Various Techniques	Advanced types like GIN
Stored Procedures	Supported	Advanced with PL/pgSQL language
Web Applications	Moderate	Easy

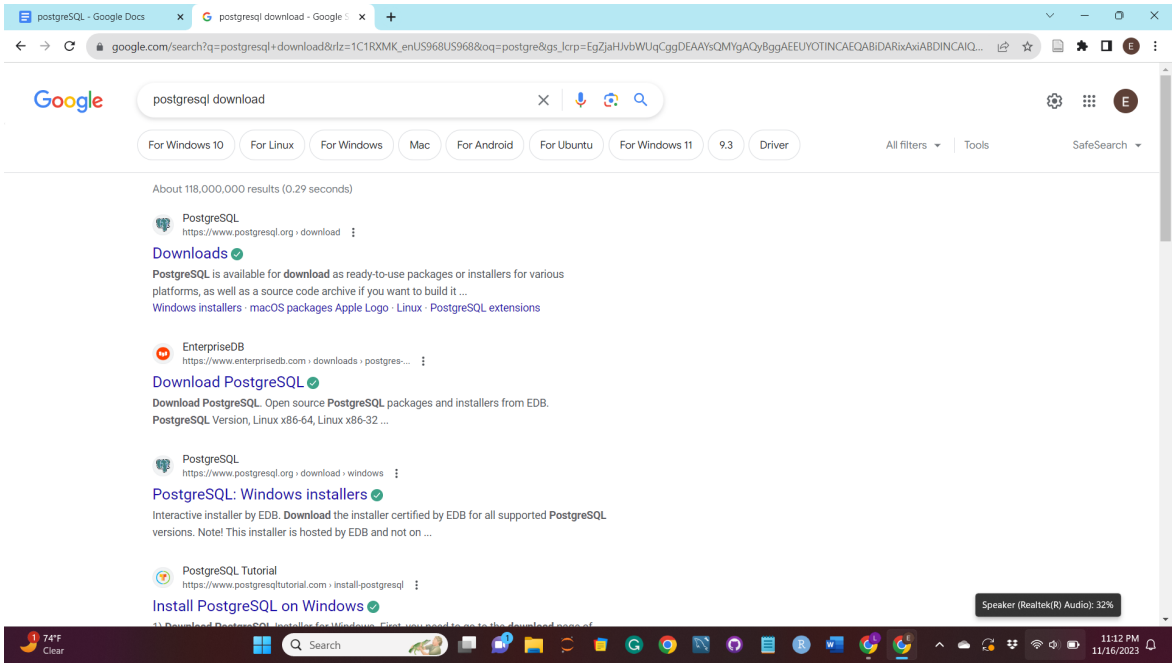
- **Database Type:** For MySQL, it is generally faster and easier to run with large datasets. For PostgreSQL, it is better for enterprise-level applications and can handle more complex datasets. However , PostgreSQL doesn't run its database as fast and efficiently as MySQL.

- **Syntax:** For MySQL the syntax isn't always clear about errors in your code. The syntax will let you know that you have an error in your code but will not tell you where or what is wrong with it. In contrast, PostgreSQL's syntax is much more clear about where in your code there is an error.
- **ACID properties:** "Both are ACID-compliant systems. However, some MySQL storage engines—like MyISAM—don't support ACID. If you need ACID compliance with MySQL, try InnoDB" (Smallcombe, 2023).
- **Backup and Recovery:** "Both provide backup and recovery features. The backup and recovery tool that comes with PostgreSQL is known for its high level of efficiency" (Smallcombe, 2023).
- **Cross-platform:** "Both MySQL and PostgreSQL are cross-platform solutions. However, PostgreSQL is famous for its performance optimization features on UNIX-based systems" (Smallcombe, 2023).
- **Save & Code Retrieval:** Both MySQL and PostgreSQL have easy ways of saving your codes to your database. Although, the code retrievals after saving them are different. When you save a code in MySQL, you will have it saved as a file and saved in your MySQL application. As opposed to saving your code in PostgreSQL, it will save as an SQL file. This makes it to where we weren't able to take our saved code and go back to coding in PostgreSQL. Saving it as an SQL file resulted in it taking us to MySQL.
- **Foreign Keys:** "Both allow foreign key constraints. However, the MyISAM storage engine in MySQL doesn't" (Smallcombe, 2023).
- **Indexes:** Both PostgreSQL and MySQL offer distinct indexing options. PostgreSQL index types include the following:
    - Partial indexes that only arrange information from a section of the table
    - B-tree indexes and hash indexes
    - Expression indexes that generate an index resulting from express functions instead of column values (Ravoof, 2023)

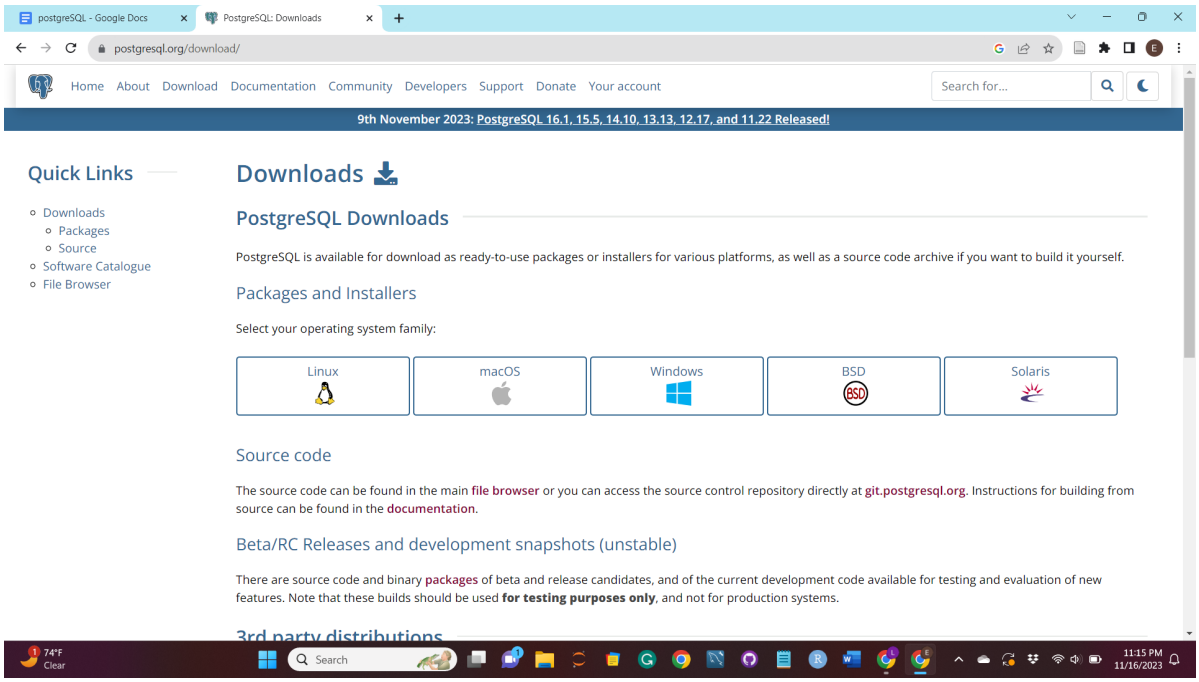
  MySQL, on the other hand, offers the following index options:
    - Indexes stored on R-trees, such as indexes found on spatial data types
    - Indexes stored on B-trees, such as PRIMARY KEY, INDEX, FULLTEXT, and UNIQUE

    - Inverted lists and hash indexes when utilizing FULLTEXT indexes (Ravoof, 2023)
- **Stored procedures:** “Both support stored procedures. However, compared to the routine syntax of MySQL, PostgreSQL provides more complete support through its PL/pgSQL language. The PL/pgSQL language allows PostgreSQL users to be more creative in developing stored procedures to fit their unique use cases” (Smallcombe, 2023).
- **Web applications:** When it comes to installing PostgreSQL, the steps and instructions are relatively easy to follow along. Finding which type of download best fits your pc or mac was easy to differentiate and the process of the installation took less than ten minutes. For MySQL, the installation process was moderately easy to follow along, as the steps were longer to get through compared to PostgreSQL. When it came to installing especially on Mac computers, it provided a difficult task as certain versions were needed and were harder to locate on the web application.

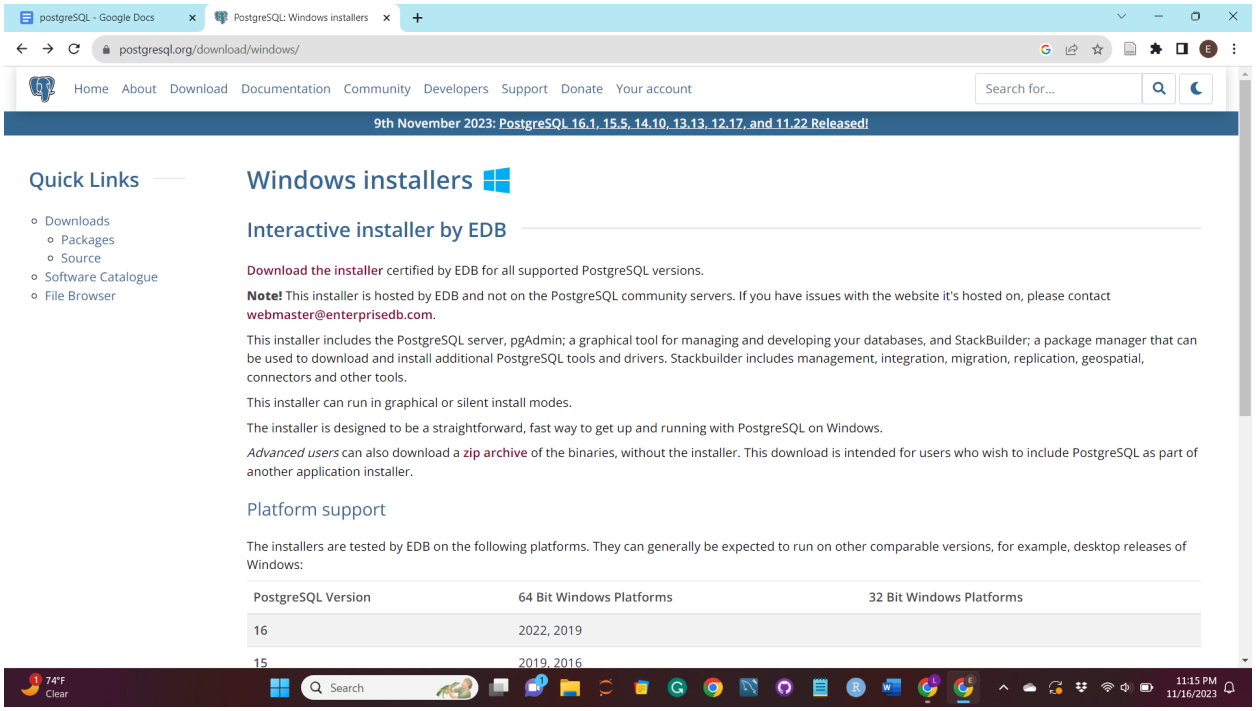
III. INSTALLATION ON PC



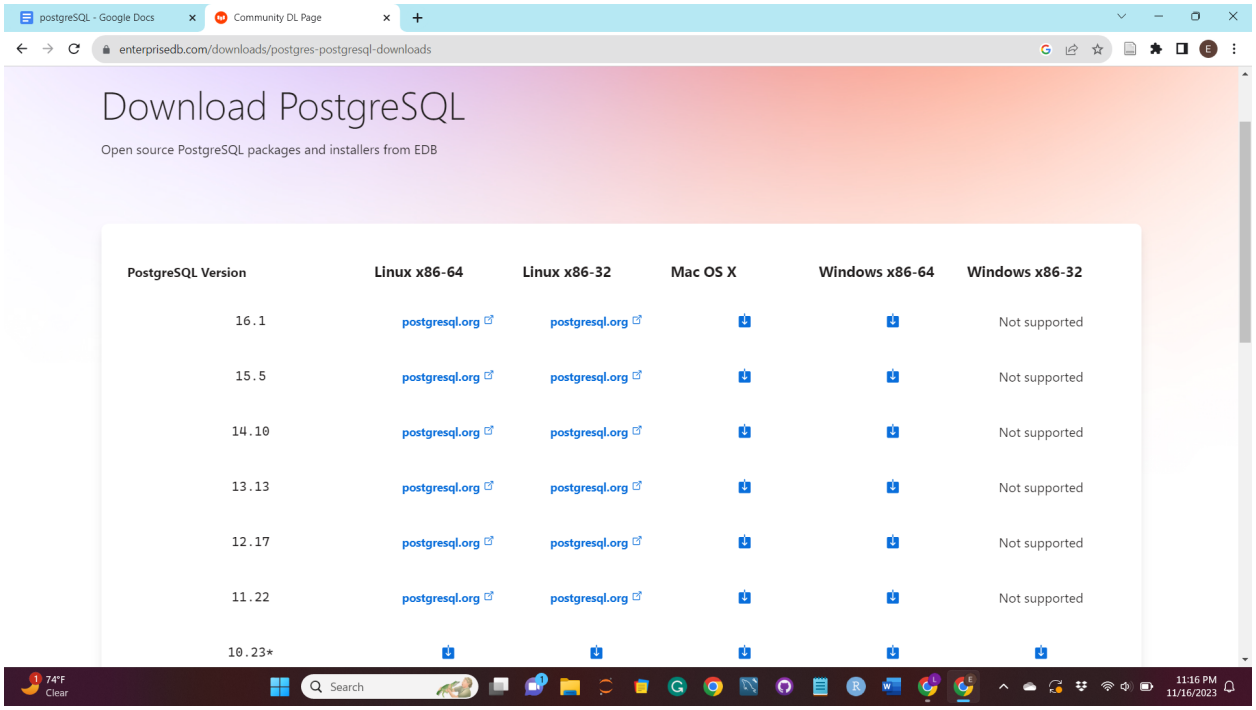
Open your web browser and search for “postgreSQL”; make sure to click the link https://www.postgresql.org › download



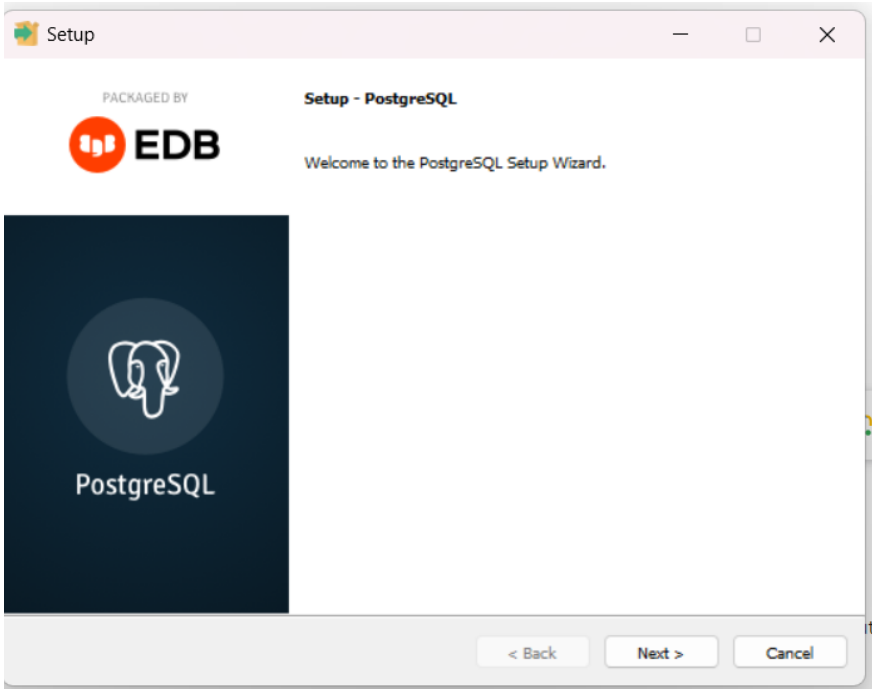
Click on “Windows” to download windows version



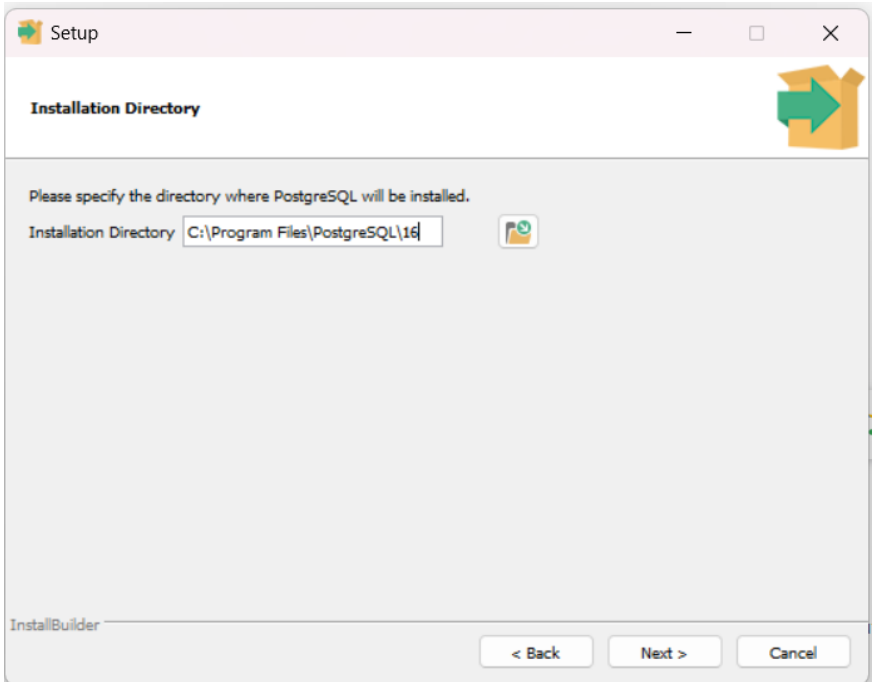
next, click on “download the installer”



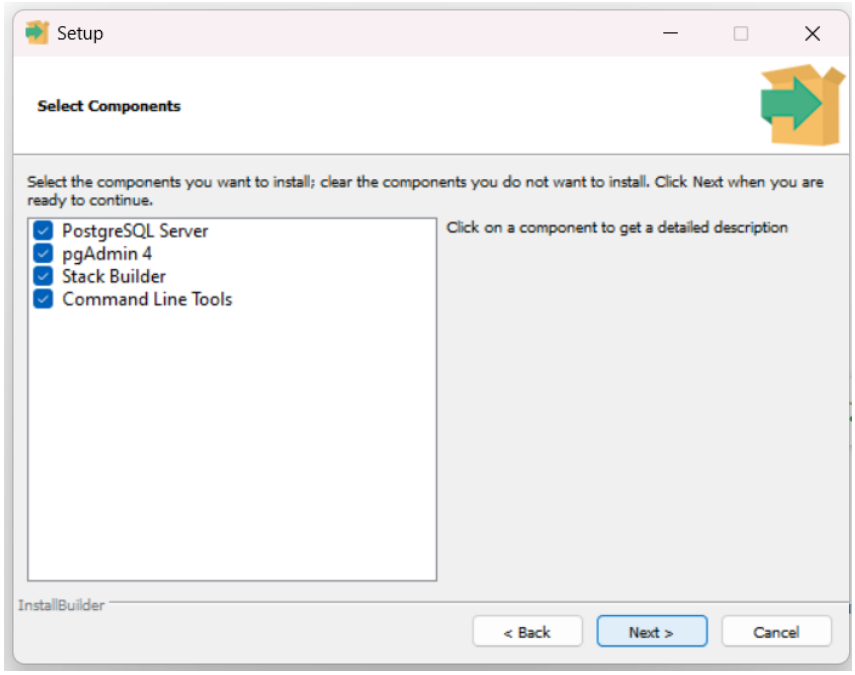
Select the most up to date version which is 16.1, and click on the icon below “windows x86-64”



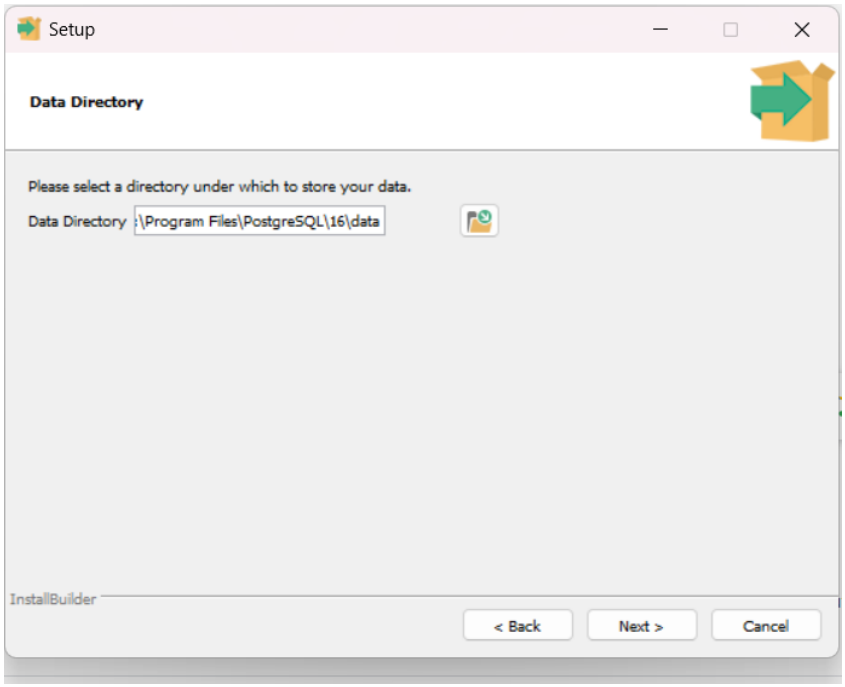
Open the downloaded version and begin the setup process



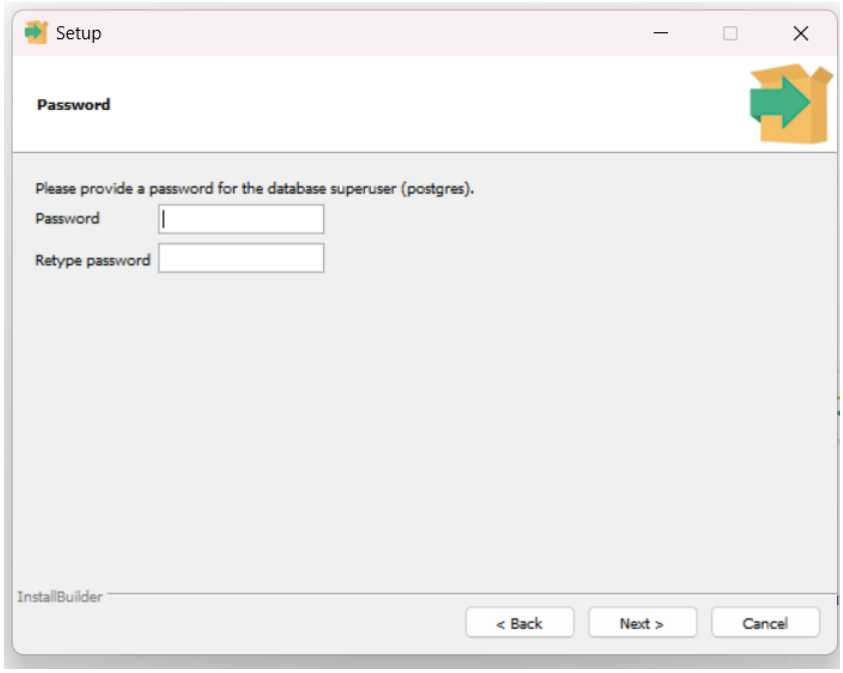
Specify the directory where PostgreSQL will be installed; so which file it will be located



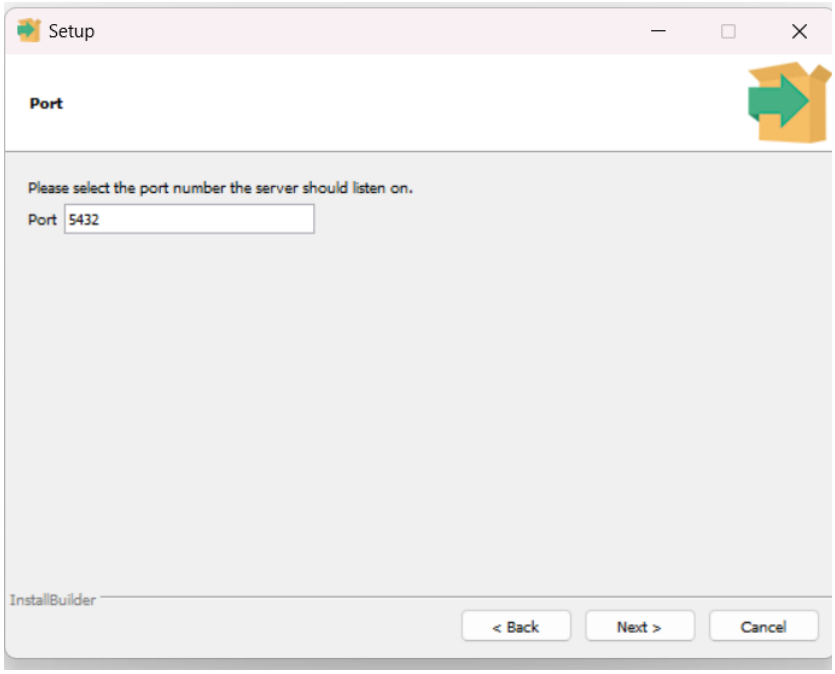
Select all components to be installed



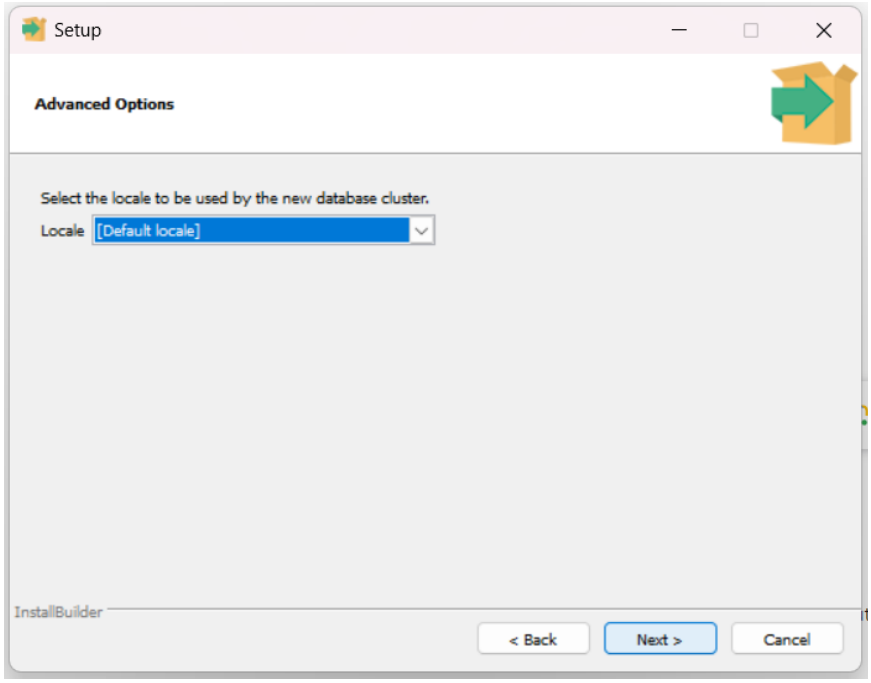
Specify the directory where the data will be saved



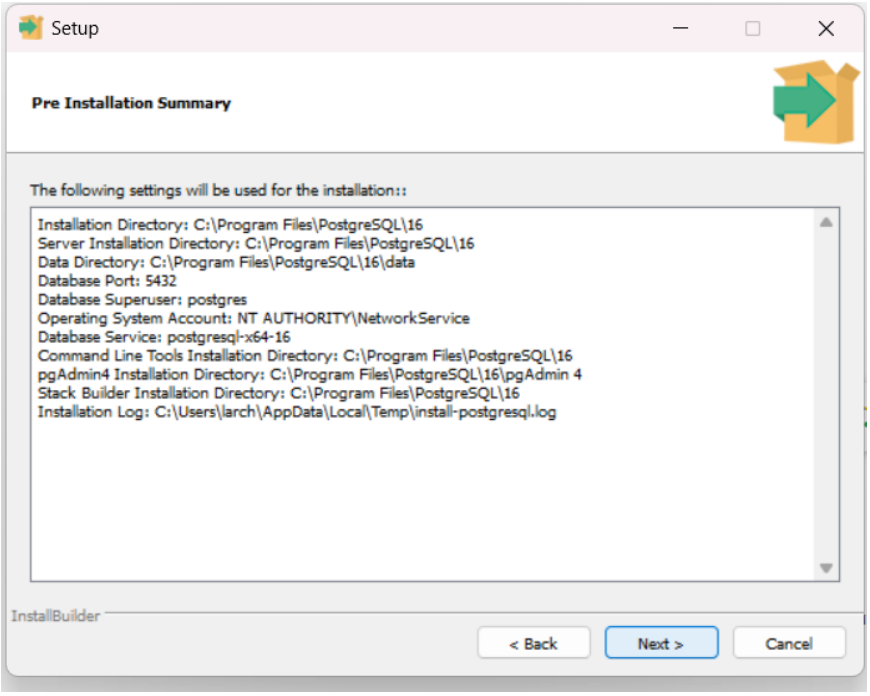
Now create a password to access your database; keep in mind that you will need your password every time you open PostgreSQL



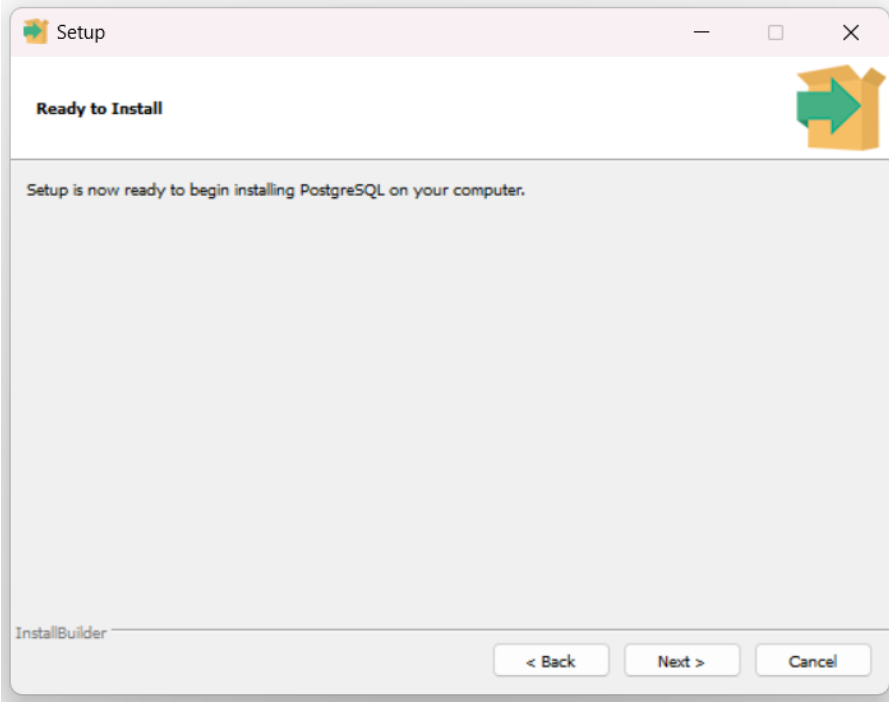
The port number is also important because it is the default port number for PostgreSQL



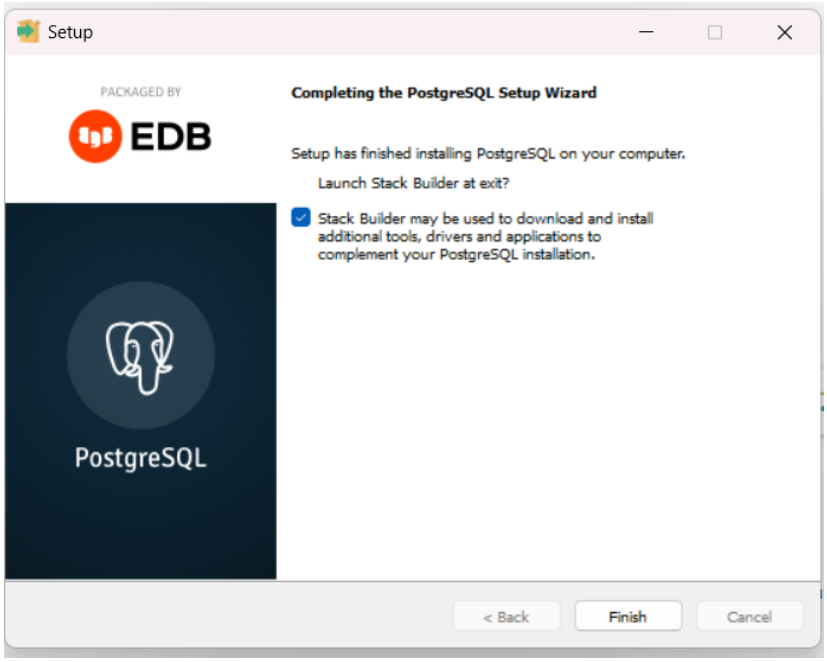
Select “Default locale” which will be used by the new database



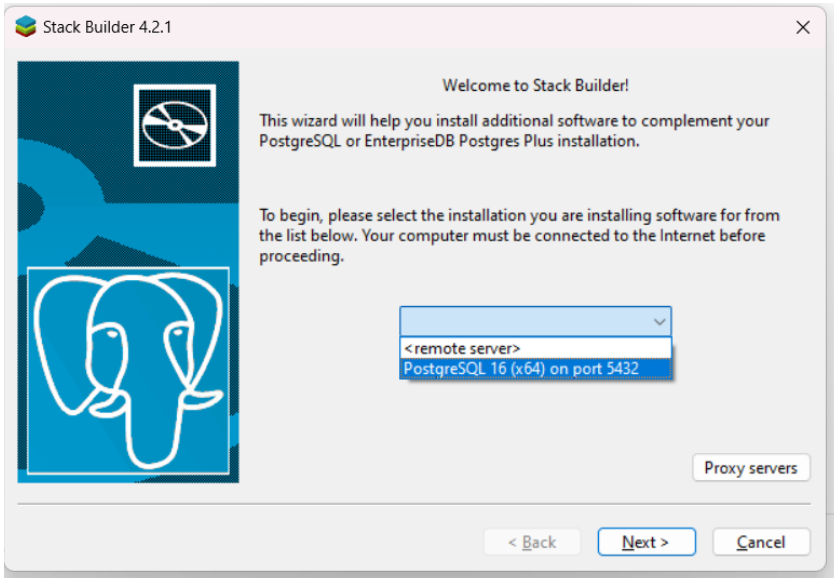
This is the installation summary and will show the details of both where the installation will be located and the overall details of the download



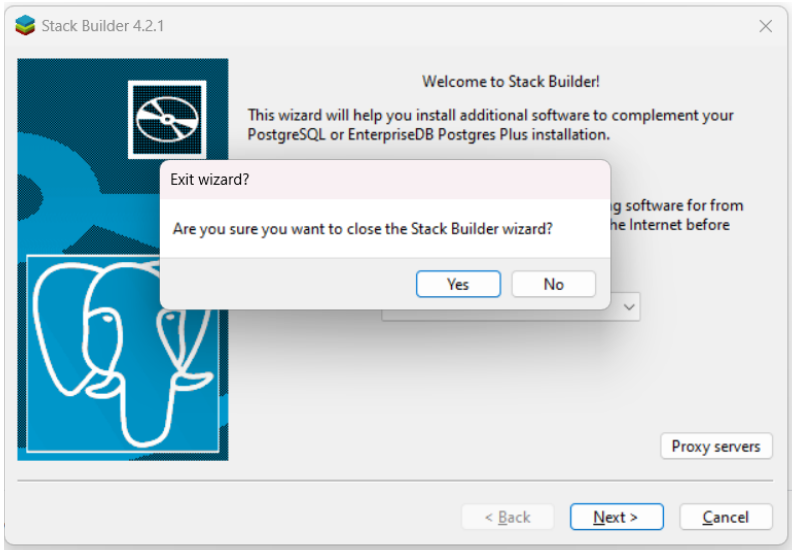
Now PostgreSQL is ready to be installed



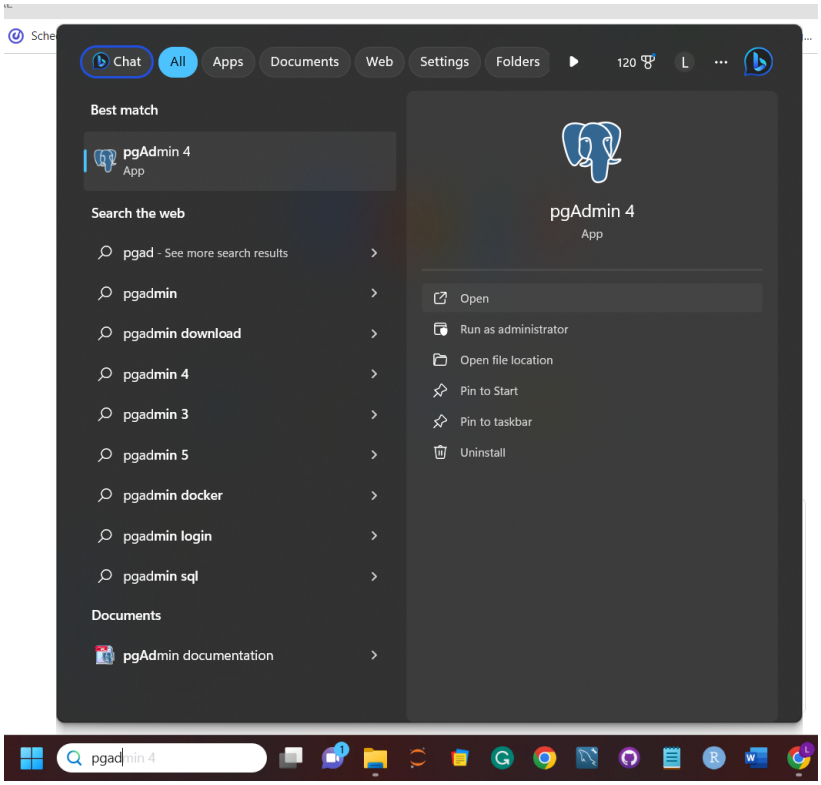
Once the installation process has been completed, it will ask to launch “stack builder” at the end you may select if you want to but it is not needed to install



As previously mentioned, Stack Builder isn’t necessary to install PostgreSQL, so for this part you may press “cancel” to exit



Click “Yes” to exit out of the Stack Builder completely



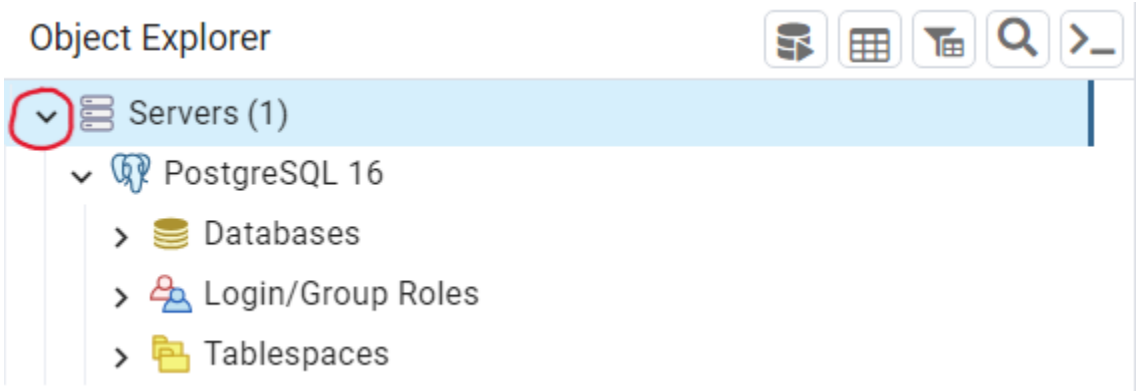
Now that you have PostgreSQL installed, search for the application and open

IV. TWO CODE CASES

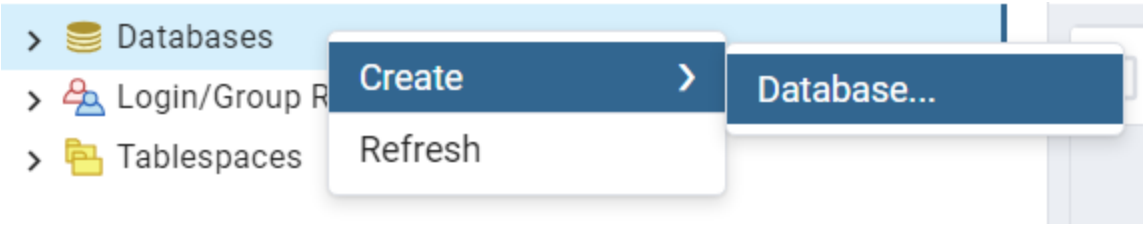
CODE CASE 1: VIDEO GAME CHARACTERS

I. Create Database

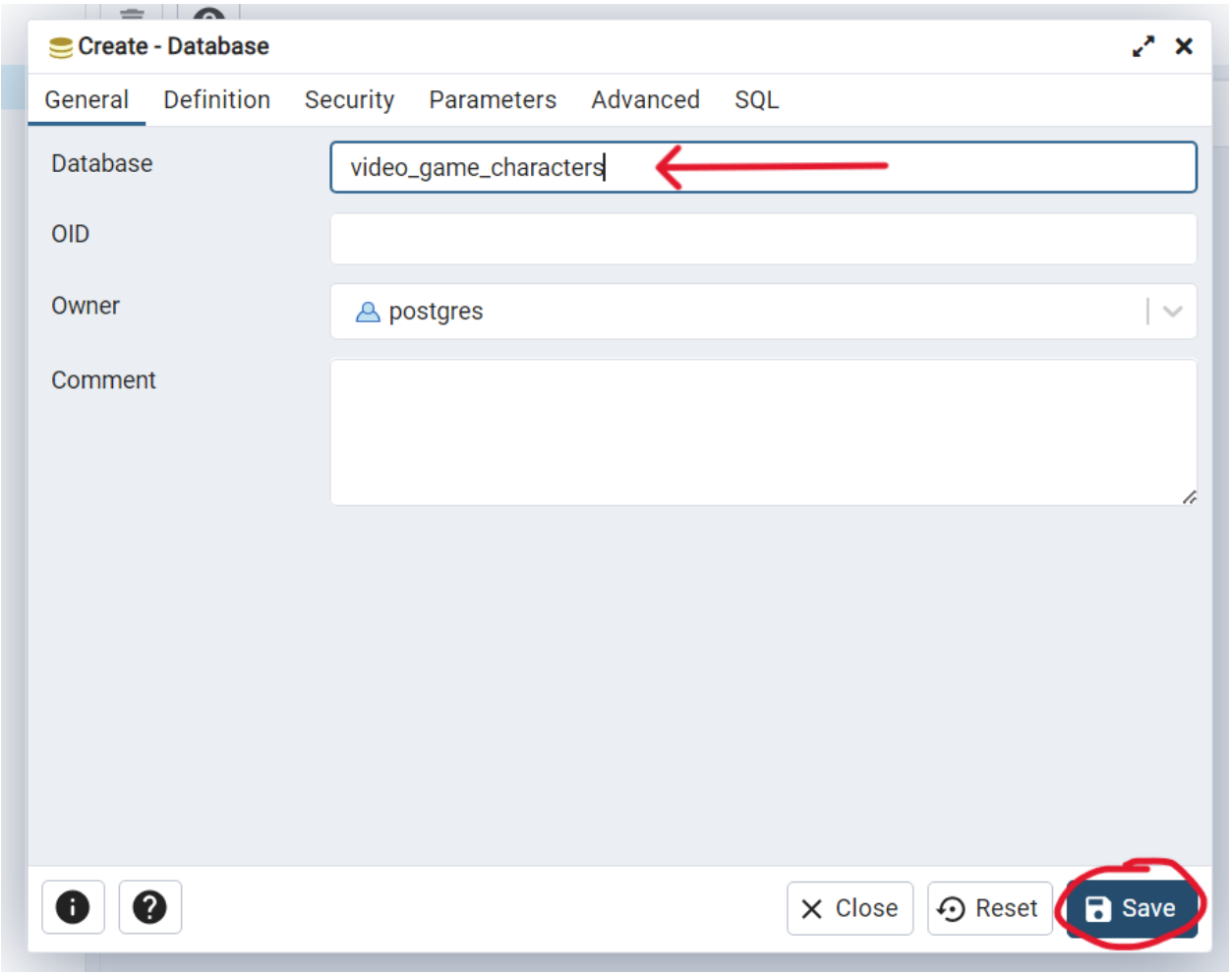
1. Open PostgreSQL on your device



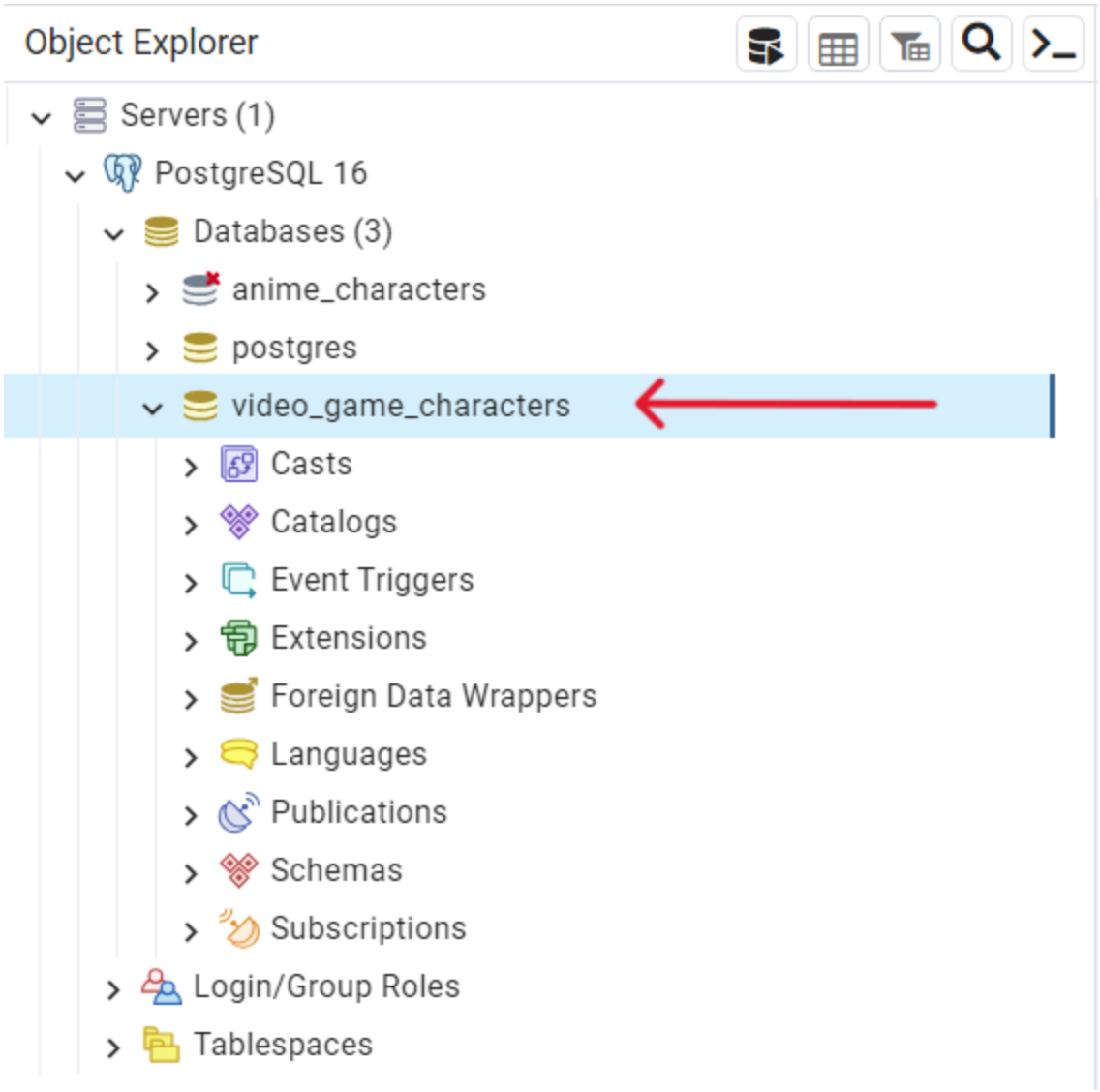
2. On the left hand side of screen click on the arrow next to “Servers”



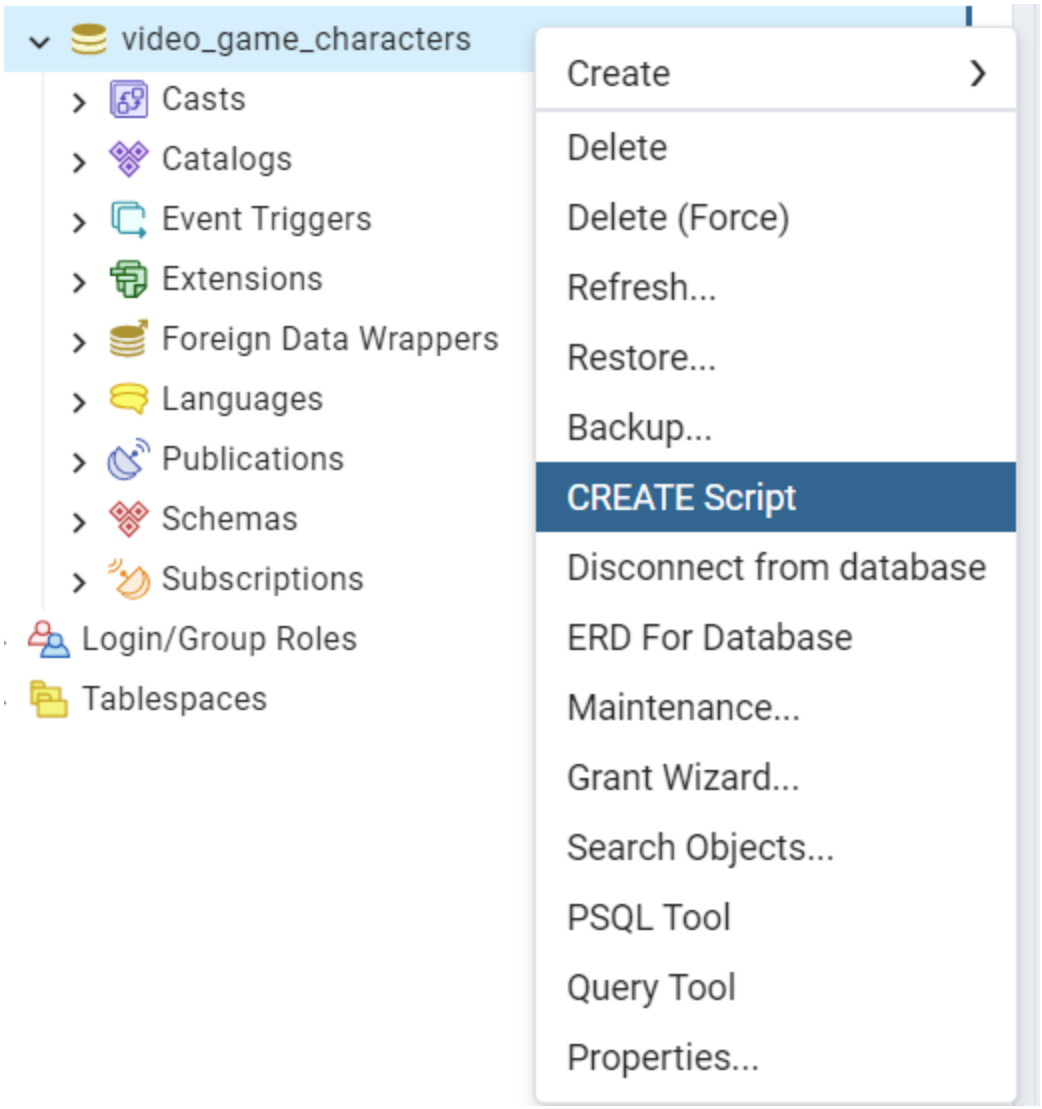
3. Right Click “Databases”, hover over “Create”, Click “Database…”



4. A new prompt will appear showing an empty line next to where “Database” is, there you will name your database “video_game_characters”. Click “Save” on the bottom right of prompt once you are done naming your database



5. Once you've created your database, it should pop up on left side of your screen under "Databases" as "video_game_characters"



6. Right click on your new database and a list will pop up, click “CREATE Script”

II. Create 3x Tables

Example code:

```
CREATE TABLE genshin_impact
```

The code CREATE means what it implies, you want to create something in your database. And TABLE means that you would like the format of a table. So when you put it together as CREATE TABLE, this means that within your database you would like to create a table to insert data.

We have chosen “genshin_impact” as our table name. Since the topic is about video game characters and we are interested in Genshin Impact, we’ve decided to create our first table about

Genshin Impact. Next we have the parentheses. The parentheses are there to "group filter criteria statements and provide the structure necessary to perform complex queries" (*Blackbaud*).

Example code:

```
CREATE TABLE genshin_impact (
    genName
    genBirthday
    genElement
    genWeaponType
    genRegion
    genRarity
```

When you create a table, you also must create an acronym that is easy to remember as well as matches your table name. This is important especially when you are creating multiple tables in the same database that would contain similar information. For example, in our table "genshin_impact" we've chosen "gen" (as shown highlighted above) to distinguish between the other tables. Then once we've created our acronym, you will now put what type of data you want to collect (genName, genBirthday, etc.). When creating the type of data you would like to collect, you must create it specifically for your desired table. Since our first table is about Genshin Impact characters, we would like to create a table that will specifically be related to that game. In our table we have the characters name, birthday, element that they use, weapon, region they're from, and rarity. These characteristics are also to help be able to differentiate between each character that will be inserted into the table.

Example code:

```
CREATE TABLE genshin_imp
    genName VARCHAR(20)
```

VARCHAR(20) "a datatype in SQL that holds characters of variable length" (*United States*, IBM). VARCHAR is mostly used for data values that require special characters such as a hyphen, underscore, exclamation point, or even having letters and numbers in that one value. For example, we have genName VARCHAR(20). We've chosen to set it as various characters because there are characters who may have a hyphen in their name. We've limited it to 20 characters because no character's name is longer than 20 characters. If there were characters with a longer name, then you would want to make the amount of characters allowed larger such as 30.

Example Code:

```
CREATE TABLE genshin_imp
    genName VARCHAR(20)
    genBirthday INTEGER
    genElement CHAR(10)
```

CHAR (as highlighted above) “stores character data in a fixed-length field” (*United States,* IBM).. When it comes to data that only requires the alphabet A-Z, CHAR is a good choice to have as your variable. And again, the amount of characters that you put as your limit depends on the values that you would like to insert into your table. For the variable genElement, there are only 7 elements used in that game which are Anemo, Geo, Electro, Dendro, Hydro, Pyro, and Cryo. So based on this information, having the variable be limited to CHAR(10) is sufficient.

Example code:

```
CREATE TABLE genshin_imp
    genName VARCHAR(20)
    genBirthday INTEGER
```

What INTEGER (as highlighted above) represents is “stores whole numbers that range from -2,147,483,647 to 2,147,483,647 for 9 or 10 digits of precision” (*United States,* IBM). Looking at the variable “genBirthday” set at INTEGER, means that you are only allowed to enter the value as a number. An example that we would use in this case is “0430” meaning that that character’s birthday is on April 30th.

Example code:

```
CREATE TABLE genshin_impact (
    genName VARCHAR(20) NOT NULL
    genBirthday INTEGER NOT NULL,
    genElement CHAR(10) NOT NULL,
```

NOT NULL “used to ensure that a given column of a table is never assigned the null value” (*United States,* IBM).. In other words, this value is not allowed to be left blank or have nothing.

```
CREATE TABLE genshin_impact (
    genName VARCHAR(20) NOT NULL
    genBirthday INTEGER NOT NULL,
    genElement CHAR(10) NOT NULL,
    genWeaponType CHAR(10) NOT NU
    genRegion CHAR(10) NULL,
    genRarity VARCHAR(10) NOT NUL
```

In contrast, NULL (as highlighted above) is a special "marker used in SQL to indicate that a data value does not exist in the database" (*United States,* IBM). The reason "genRegion" is left as NULL is because there are characters who do not have a place that they are from. And for that reason, we would need to have that variable as NULL for cases such as that.

Example code:

```
CREATE TABLE genshin_impact (
    genName VARCHAR(20) NOT NULL PRIMARY KEY,
```

A PRIMARY KEY (as highlighted above) is a column in a relational database table that's distinctive for each record. One thing about a PRIMARY KEY is that if you have a variable set as a PRIMARY KEY, you must also set it to NOT NULL at all times.

Example code:

```
CREATE TABLE genshin_impact (
    genName VARCHAR(20) NOT NULL PRIMARY KEY,
    genBirthday INTEGER NOT NULL,
    genElement CHAR(10) NOT NULL,
    genWeaponType CHAR(10) NOT NULL,
    genRegion CHAR(10) NULL,
    genRarity VARCHAR(10) NOT NULL
);
```

"Parentheses must always be used to enclose subqueries" (MariaDB KnowledgeBase, *Parentheses*). They are important because they mark an opening and closing statement of a group of code. As you can see above, the code within the parentheses is a group of code that represents the columns that will go into your table.

Example code:

```
CREATE TABLE genshin_impact (
    genName VARCHAR(20) NOT NULL PRIMARY KEY,
    genBirthday INTEGER NOT NULL,
    genElement CHAR(10) NOT NULL,
    genWeaponType CHAR(10) NOT NULL,
    genRegion CHAR(10) NULL,
    genRarity VARCHAR(10) NOT NULL
);
```

Lastly, the reason you need a semicolon (as highlighted above) at the end of each code is to distinguish where the code ends. A "semicolon is the standard way to separate each SQL statement in database systems that allow more than one SQL statement to be executed in the same call to the server" (*United States,* IBM). Just as important as the semicolon, the comma after each line of code represents the separation of each column in your table. If you were to not put any commas, everything within the parentheses would read as one column of the Genshin Impact table, which we would not want.

```
CREATE TABLE genshin_impact (
    genName VARCHAR(20) NOT NULL PRIMARY KEY,
    genBirthday INTEGER NOT NULL,
    genElement CHAR(10) NOT NULL,
    genWeaponType CHAR(10) NOT NULL,
    genRegion CHAR(10) NULL,
    genRarity VARCHAR(10) NOT NULL
);
```

If you observe the way the code is written above, you'll notice that the functions are in all caps, the parentheses are separated from the variables, after each variable it's entered, and the variables are indented. The reason the code is written this way is for visual purposes. Having the functions in all caps is optional, if you desire the lower case look rather than the all caps, then that is sufficient as well. You could also have the code all on one line instead of indented and how each variable is on a new line if you desire. For us, it is much easier to see that we are creating a table named genshin_impact, that we are recording the characters name, birthday, element, weapon type, region, and rarity. This way of coding has been acquired due to what makes more sense to us. You are able to alter the way you code to ensure that the codes make complete sense to you.

```
CREATE TABLE honkai_star_rail (
    honName VARCHAR (30) NOT NULL PRIMARY KEY,
    honRarity VARCHAR(10) NOT NULL,
```

```
    honPath CHAR(15) NOT NULL,
    honCombatType CHAR(10) NOT NULL,
    honOrganization VARCHAR(30) NOT NULL
);

CREATE TABLE apex_legends (
    apxLegendName VARCHAR(20) NOT NULL PRIMARY KEY,
    apxHomeworld VARCHAR(20) NULL,
    apxClass CHAR(10) NOT NULL,
    apxTacticalAbility VARCHAR(20) NOT NULL,
    apxUltimateAbility VARCHAR(20) NOT NULL
);
```

III. Insert Data Into Tables

Example code:

```
INSERT INTO genshin_impact
```

"An SQL INSERT statement writes new rows of data into a table. If the INSERT activity is successful, it returns the number of rows inserted into the table. If the row already exists, it returns an error. Multiple rows can be inserted into a table" (*The SQL insert statement*).

Example Code:

```
INSERT INTO genshin_impact
```

The code INTO "creates a new table in the default filegroup and inserts the resulting rows from the query into it" (VanMSFT, *INTO clause (transact-SQL) - SQL server*). In simpler terms, the code INTO is just like saying "I want the following information to go in the following table".

Example Code:

```
INSERT INTO genshin_impact
    VALUES ('Diluc', 0430, 'Pyro', 'Claymore', 'Mondstadt', '5 Star'),
           ('Zhongli', 1231, 'Geo', 'Polearm', 'Liyue', '5 Star'),
           ('Raiden Shogun', 0626, 'Electro', 'Polearm', 'Inazuma', '5 Star'),
           ('Nahida', 1027, 'Dendro', 'Catalyst', 'Sumeru', '5 Star'),
           ('Neuvillete', 1218, 'Hydro', 'Catalyst', 'Fontaine', '5 Star'),
           ('Childe', 0720, 'Hydro', 'Bow', 'Snezhnaya', '5 Star');
```

VALUES is a statement that returns a set of one or more rows as a table (*MySQL 8.0 Reference Manual :: 13.2.19 values statement*). ('Diluc', 'Pyro', 'Claymore', 'Mondstadt', '5 Star') are the values that represent the information that you'll be putting into each column of your table to be read.

```
INSERT INTO genshin_impact
	VALUES ('Diluc', 0430, 'Pyro', 'Claymore', 'Mondstadt', '5 Star'),
			('Zhongli', 1231, 'Geo', 'Polearm', 'Liyue', '5 Star'),
			('Raiden Shogun', 0626, 'Electro', 'Polearm', 'Inazuma', '5 Star'),
			('Nahida', 1027, 'Dendro', 'Catalyst', 'Sumeru', '5 Star'),
			('Neuvillete', 1218, 'Hydro', 'Catalyst', 'Fontaine', '5 Star'),
			('Childe', 0720, 'Hydro', 'Bow', 'Snezhnaya', '5 Star');

INSERT INTO honkai_star_rail
	VALUES ('Himeko', '5 Star', 'Erudition', 'Fire', 'Astral Express'),
			('Herta', '4 Star', 'Erudition', 'Ice', 'Herta Space Station'),
			('Seele', '5 Star', 'The Hunt', 'Quantum', 'Jarilo-IV'),
			('Dan Heng Imbibitor Lunae', '5 Star', 'Destruction', 'Imaginary', 'The
Xianzhou Luofu'),
			('Blade', '5 Star', 'Destruction', 'Wind', 'Stellaron Hunters'),
			('Kafka', '5 Star', 'Nihility', 'Lightning', 'Stellaron Hunters');

INSERT INTO apex_legends
	VALUES ('Mad Maggie', 'Salvo', 'Assault', 'Riot Drill', 'Wrecking Ball'),
			('Bloodhound', 'Talos', 'Recon', 'Eye of the Allfather', 'Beast of the Hunt'),
			('Wraith', 'Typhon', 'Skirmisher', 'Into the Void', 'Dimensional Rift'),
			('Revenant', null, 'Skirmisher', 'Shadow Punch', 'Forged Shadows'),
			('Conduit', 'Nexus', 'Support', 'Radiant Transfer', 'Energy Barricade'),
			('Crypto', 'Gaea', 'Recon', 'Surveillance Drone', 'Drone EMP');
```

<u>CODE CASE 2: ANIME CHARACTERS</u>

Note: As you code CASE 2, the instructions are exactly the same as CASE 1. Here we are just using different names.

I. Create Database or Starting Application

i. Open PostgreSQL, on left hand side of screen click on “Servers”

ii. Right Click “Databases”, hover over “Create”, Click “Database…”

iii. New prompt will appear showing an empty line where “Database” is, there you will name your database “anime_characters”, Click “Save” on bottom right of prompt

iiii. Should pop up on left side of your screen under “Databases” as “anime_characters”, Right click on your new database and list will pop up, click “CREATE Script”

II. Create 3x Tables

```
CREATE TABLE one_piece (
	opcName VARCHAR(30) NOT NULL PRIMARY KEY,
	opcBounty VARCHAR(30) NOT NULL,
	opcOccupation CHAR(30) NOT NULL,
	opcDevilFruit CHAR(30) NULL,
	opcBirthday INTEGER NOT NULL
);

CREATE TABLE jujutsu_kaisen (
	jjkName VARCHAR(30) NOT NULL PRIMARY KEY,
	jjkCursedTechnique VARCHAR(50) NULL,
	jjkDomainExpansion VARCHAR(50) NULL,
	jjkGrade VARCHAR(30) NULL,
	jjkBirthday INTEGER NULL
);

CREATE TABLE fairy_tail (
	fytName VARCHAR(30) NOT NULL PRIMARY KEY,
	fytMagic VARCHAR(50) NULL,
	fytWeapon VARCHAR(20) NOT NULL,
	fytAlias VARCHAR(30) NULL,
	fytBirthday VARCHAR(10) NOT NULL
);
```

III. Insert Data Into Tables

```
INSERT INTO one_piece
VALUES ('Monkey D Luffy', '3000000000', 'Straw Hats Captain', 'Gum-Gum Fruit', 0505),
		('Rorona Zoro', '1111000000', 'Swordsman', null, 1111),
		('Nami', '366000000', 'Navigator', null, 0703),
		('Usopp', '50000000', 'Sniper', null, 0401),
		('Sanji', '1032000000', 'Cook', null, 0302),
		('Tony Tony Chopper', '1000', 'Doctor', 'Human-Human Fruit', 1224);

INSERT INTO jujutsu_kaisen
VALUES ('Gojo Satoru', 'Six Eyes', 'Unlimited Void', 'Special Grade', 1207),
		('Megumi Fushiguro', 'Ten Shadows', 'Chimera Shadow Garden', 'Grade 2', 1222),
		('Nobara Kugisaki', 'Straw Doll', null, 'Grade 3', 0807),
```

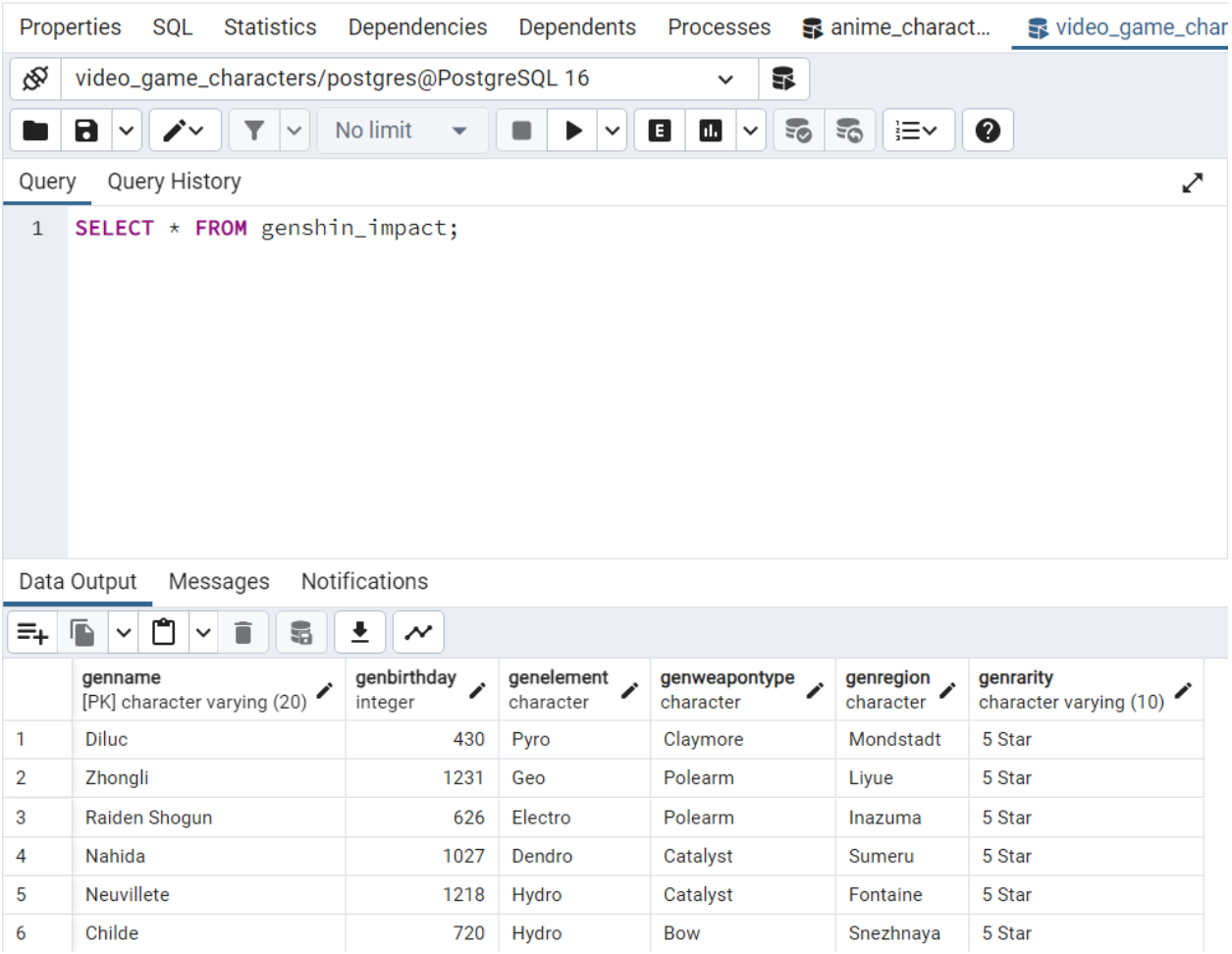
```
('Sukuna', 'Unknown: Fire-Based', 'Malevolent Shrine', 'Special Grade', null),
('Choso', 'Blood Manipulation', null, 'Special Grade', null),
('Toji Fushiguro', null, null, null, 1231);

INSERT INTO fairy_tail
VALUES ('Natsu Dragneel', 'Fire Dragon Slayer', 'Sealed Flame Blade', 'Salamander', 'Year777'),
('Lucy Heartfilia', 'Spatial Magic', 'Zodiac Keys', null, 'Year X767'),
('Gray Fullbuster', 'Ice Devil Slayer', 'Ice Make', null, 'Year X766'),
('Erza Scarlet', 'Requip: The Knight', '200+ Weapons', 'Titania', 'Year X765'),
('Gajeel Redfox', 'Iron Dragon Slayer', 'Solid Script', 'Black Steel', 'Year777'),
('Mirajane Strauss', 'Take Over', 'Transformation', 'The Demon', 'Year X765');
```

QUERY DATA EXAMPLES

When you are querying your database, you are basically asking to retrieve data from your database. This is useful for when you need to manipulate, access, delete, or retrieve data from your relational database. You are giving your database instructions on what to do based on the type of data that you would like to retrieve.

I. Querying Example #1

```
SELECT * FROM genshin_impact;
```



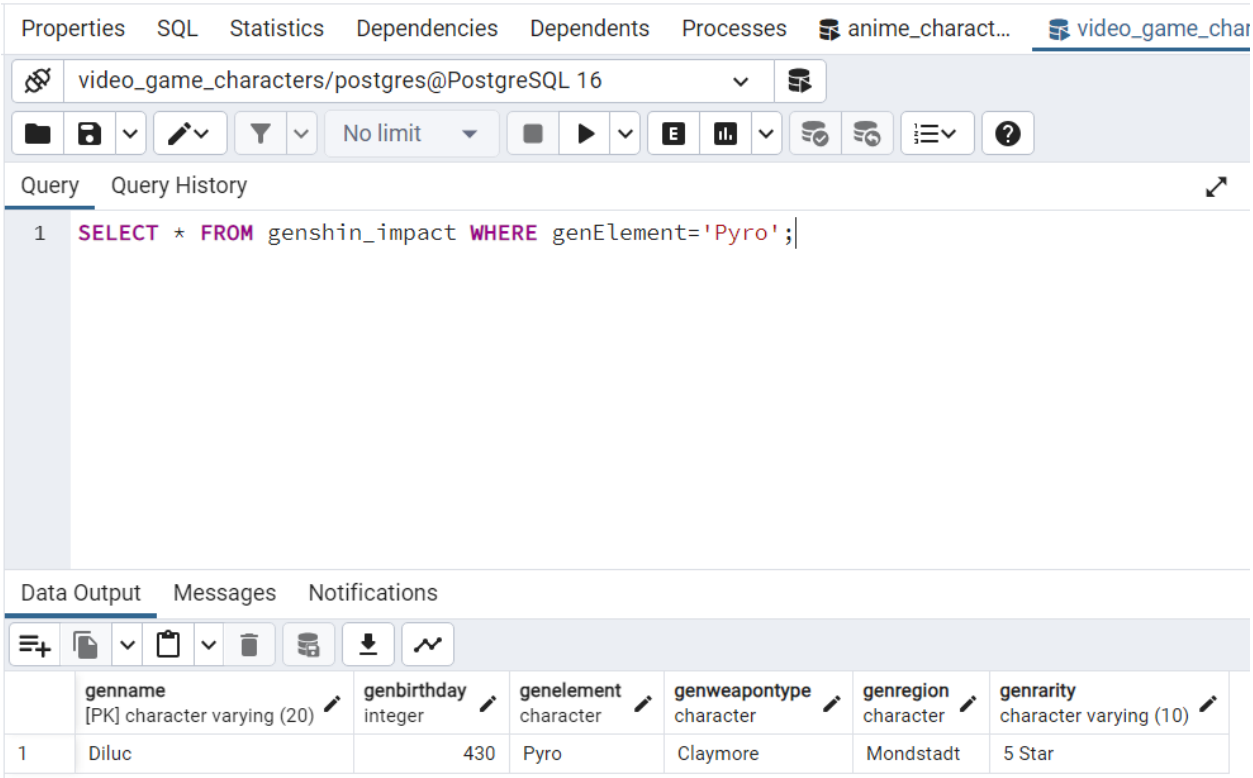
	genname [PK] character varying (20)	genbirthday integer	genelement character	genweapontype character	genregion character	genrarity character varying (10)
1	Diluc	430	Pyro	Claymore	Mondstadt	5 Star
2	Zhongli	1231	Geo	Polearm	Liyue	5 Star
3	Raiden Shogun	626	Electro	Polearm	Inazuma	5 Star
4	Nahida	1027	Dendro	Catalyst	Sumeru	5 Star
5	Neuvillete	1218	Hydro	Catalyst	Fontaine	5 Star
6	Childe	720	Hydro	Bow	Snezhnaya	5 Star

When using the code “SELECT” this means that you are asking PostgreSQL to select or take something from your database. The symbol “*” represents “everything”. So when you input the code “SELECT *...” you are really asking your database to “SELECT EVERYTHING”. Selecting “everything” means basically that you are asking your database to show all the data from the table that you created. Now, having the code “FROM” means that you want PostgreSQL to only pull data from a specific table(s). So when you put all three of those codes together, “SELECT * FROM genshin_impact”, the instructions you as the user are giving to your database is that you want to select all the data from the table called genshin_impact.

Again, using the code above you should be able to see everything from your table such as your columns and values that have been inserted. This query would be useful if you want to see all of your data from a specific table you’ve created. So for this example, we wanted to see all of the data that we have just inserted into our table called “genshin_impact”.

I. Querying Example #2

SELECT * FROM genshin_impact WHERE genElement='Pyro';

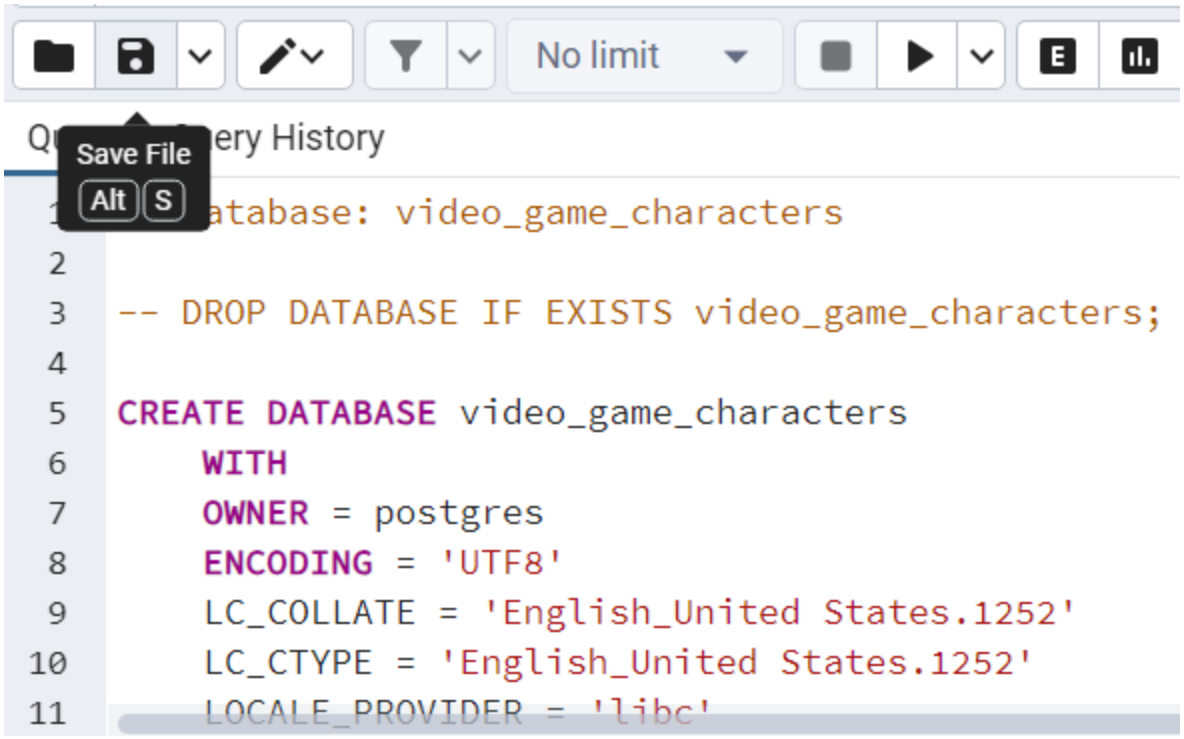


As we've discussed in the previous query, SELECT * FROM genshin_impact means you are asking PostgreSQL to show you all of the data from the table known as "genshin_impact". Now in this code, we introduce WHERE and "=".

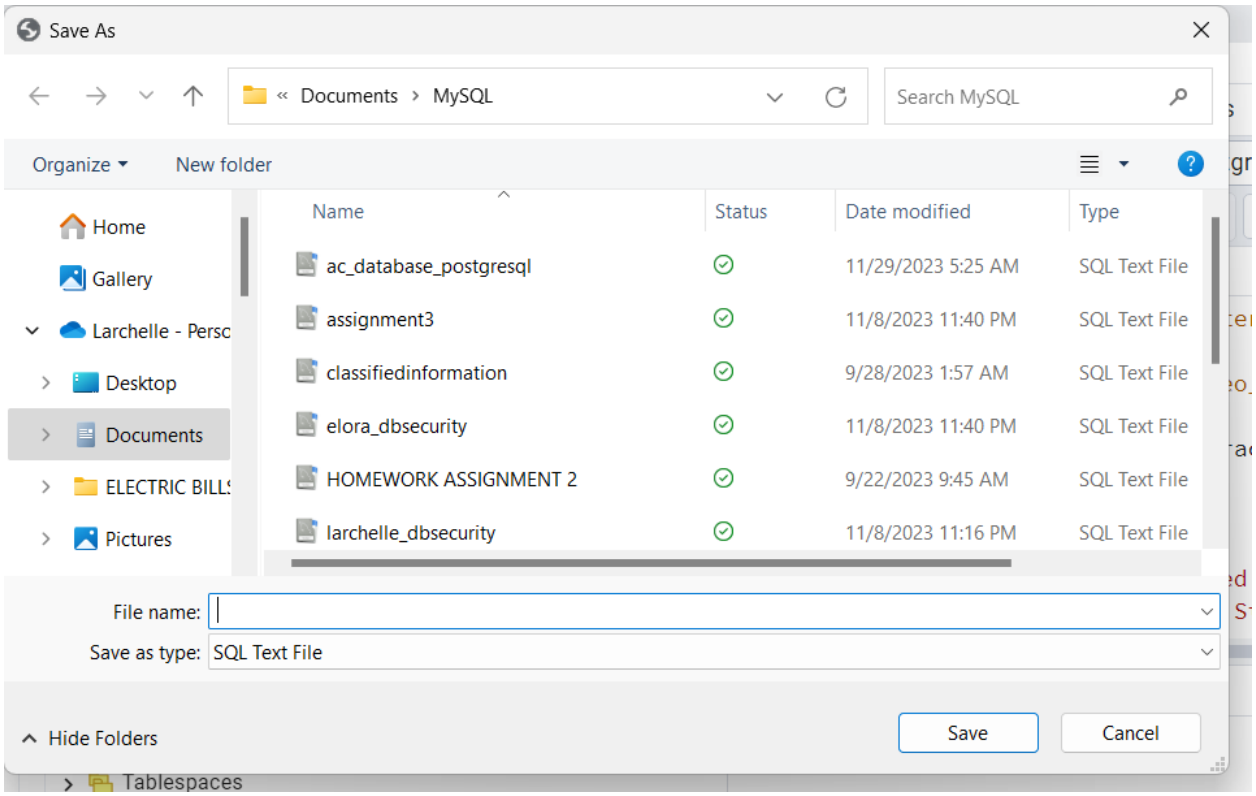
The code WHERE helps to specify exactly what you want. For example, in the figure above we have SELECT * FROM genshin_impact WHERE genElement='Pyro'. This means that you are asking your database to select everything from your table known as "genshin_impact" specifically where a character possesses Pyro as their element. Now, after the WHERE command, we have 'genElement='Pyro'. What this string of code with the "=" means is that you only want the information from characters who use Pyro as their element. Another way to think of "=" could be we are asking the database to pull all of the information from our table where the character using Pyro as an element is true.

The code shown above would be useful for when you only want data that contains one specific value you are looking for. So for this example, we only wanted to take all the information on characters who have Pyro as their element.

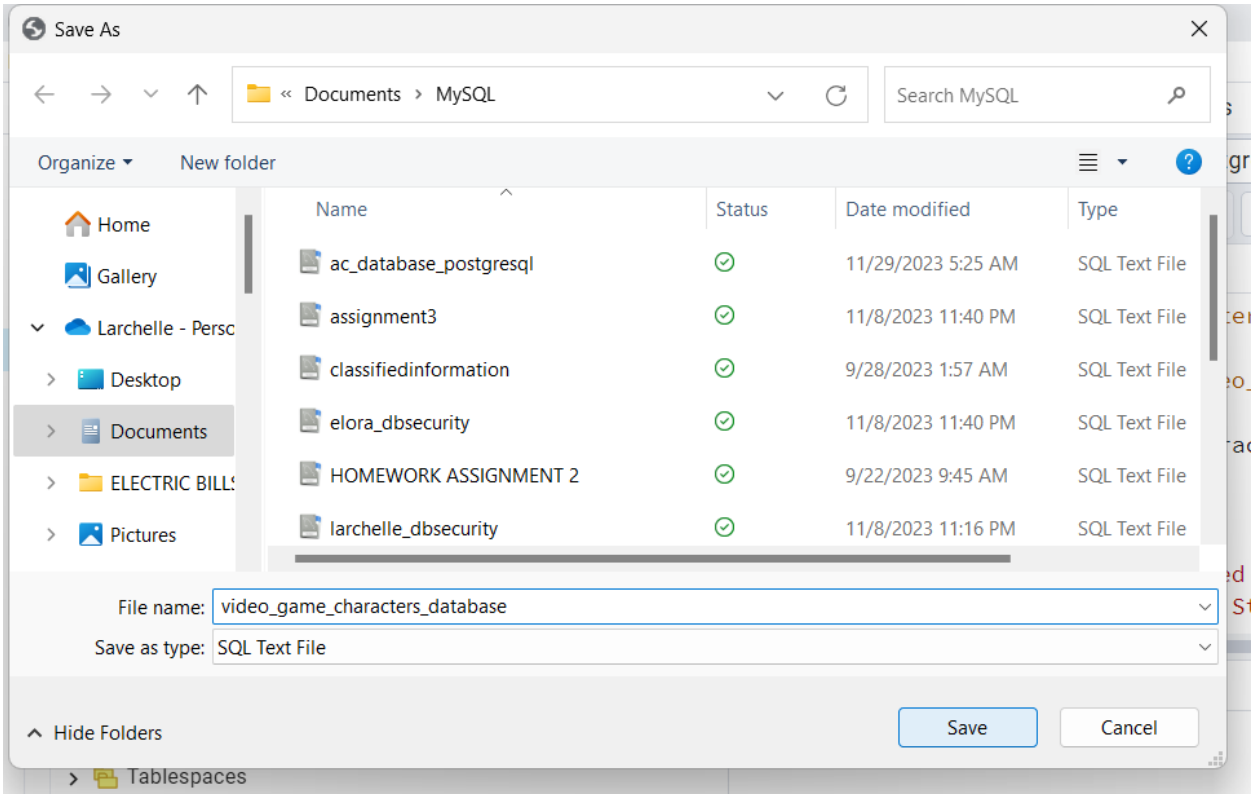
HOW TO SAVE YOUR CODES



1. Click on the save button as shown in the figure above



2. After clicking on the save button, a new prompt will appear showing “Save As”



3. Once the prompt appears, you wanna go ahead and save your code as a name you desire, so for us we have it as “video_game_characters_database”, then click “Save” at the bottom right of the prompt

V. USEFUL LINKS

A. **PostgreSQL Admin Tutorial:** this is the Youtube video we found useful to start coding in PostgreSQL.

VI. SHORT SUMMARY

Overall, we feel that PostgreSQL is very similar to MySQL Workbench. The installation was easy and smooth, we didn’t encounter any obstacles whilst downloading the PostgreSQL application. MySQL was also easy to download.
We thought that PostgreSQL was not beginner friendly because we had to look up youtube videos on how to start coding in PostgreSQL. But, after figuring out how to start coding in PostgreSQL, it was smooth sailing. From there it was about learning the language which is not much different from MySQL. Though, we found that there were features of PostgreSQL that were different from MySQL.

The first feature that we noticed was different is how to create a database. We found it inefficient in PostgreSQL compared to creating a database in MySQL. We rather code to CREATE DATABASE db_name than having to right click to create the database.
The second feature we noticed was different was how to run the codes. In MySQL, you would have to manually Ctrl+Enter each section of code. In PostgreSQL, it was much easier to run large amounts of code simply by highlighting all of the codes you want to run and clicking on what looks like a play/pause button. In this case PostgreSQL was more efficient.
The third feature was that PostgreSQL has a scratch pad on the right side of the screen. We felt that this was good for commenting or taking notes of any kind.
As we described in the key features, we found that PostgreSQL has a much more efficient syntax. When learning about how to code in MySQL, it was difficult to know where exactly the error was in our codes just by looking at the syntax. While in PostgreSQL it was very easy to see where the error occurred in our codes.
Lastly, we found that MySQL had a more visually appealing data output of the tables. In PostgreSQL, as shown below, the columns are shown as lowercase. Although, we liked the fact that in the data output table in PostgreSQL, it shows data types on the data output table (such as PK, character varying, integer, etc.).

	genname [PK] character varying (20)	genbirthday integer	genelement character	genweapontype character	genregion character	genrarity character varying (10)
1	Diluc	430	Pyro	Claymore	Mondstadt	5 Star
2	Zhongli	1231	Geo	Polearm	Liyue	5 Star
3	Raiden Shogun	626	Electro	Polearm	Inazuma	5 Star
4	Nahida	1027	Dendro	Catalyst	Sumeru	5 Star
5	Neuvillete	1218	Hydro	Catalyst	Fontaine	5 Star
6	Childe	720	Hydro	Bow	Snezhnaya	5 Star

But as you can see here, in MySQL the columns are much easier to read as shown below but do not provide the different data types.

	genName	genBirthday	genElement	genWeaponType	genRegion	genRarity
▶	Childe	720	Hydro	Bow	Snezhnaya	5 Star
	Diluc	430	Pyro	Claymore	Mondstadt	5 Star
	Nahida	1027	Dendro	Catalyst	Sumeru	5 Star
	Neuvillete	1218	Hydro	Catalyst	Fontaine	5 Star
	Raiden Shogun	626	Electro	Polearm	Inazuma	5 Star
	Zhongli	1231	Geo	Polearm	Liyue	5 Star
*	NULL	NULL	NULL	NULL	NULL	NULL

VII. REFERENCE LIST

About. PostgreSQL. (n.d.). https://www.postgresql.org/about/

Blackbaud. (n.d.). https://kb.blackbaud.com/knowledgebase/articles/Article/41398

Education, I. C. (2021, November 29). *PostgreSQL vs. mysql: What's the difference?* IBM Blog. https://www.ibm.com/blog/postgresql-vs-mysql-whats-the-difference/

MySQL 8.0 Reference Manual :: 13.2.19 values statement. MySQL. (n.d.). https://dev.mysql.com/doc/refman/8.0/en/values.html#:~:text=VALUES%20is%20a%20DML%20statement,as%20a%20standalone%20SQL%20statement.

Parentheses. MariaDB KnowledgeBase. (n.d.). https://mariadb.com/kb/en/parentheses/#:~:text=Parentheses%20must%20always%20be%20used,defined%20functions%20and%20stored%20routines.

Ravoof, S. (2023, October 18). *PostgreSQL vs MySQL: Explore their 12 critical differences*. Kinsta®. https://kinsta.com/blog/postgresql-vs-mysql/#postgresql-vs-mysql-headtohead-comparison

Smallcombe, M. (2019, June 14). *PostgreSQL vs MySQL: The critical differences*. Integrate.io. https://www.integrate.io/blog/postgresql-vs-mysql-which-one-is-better-for-your-use-case/

The SQL insert statement. TIBCO Product Documentation. (n.d.). https://docs.tibco.com/pub/as/4.2.0/doc/html/GUID-B822D80C-D79D-4A11-A7C5-DC2C5A54EE22.html#:~:text=An%20SQL%20INSERT%20statement%20writes,be%20inserted%20into%20a%20table.

United States. IBM. (n.d.). https://www.ibm.com/us-en

VanMSFT. (n.d.). *INTO clause (transact-SQL) - SQL server*. INTO Clause (Transact-SQL) - SQL Server | Microsoft Learn. https://learn.microsoft.com/en-us/sql/t-sql/queries/select-into-clause-transact-sql?view=sql-server-ver16

VIII. UPLOADED TO GITHUB

Each of us have to show that we did it in Github