HW1_109550017_黃品云

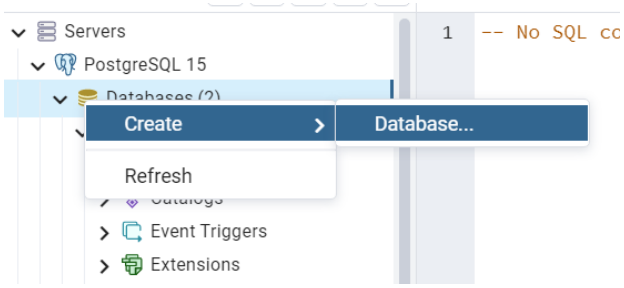
HW1_report：

Q1. The process of creating the “covid19” databases (can be screenshot and/or SQL/non-SQL statements with text explanation) (10pts)

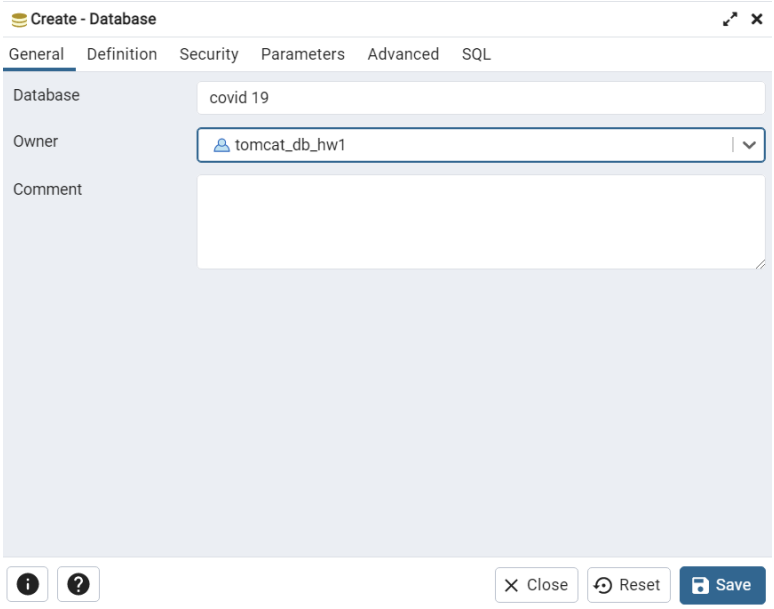
Ans 1:

I followed the following YouTube tutorial to set up my postgreSQL, and it was downloaded successfully. (https://www.youtube.com/watch?v=K5-Ed18rOb0)

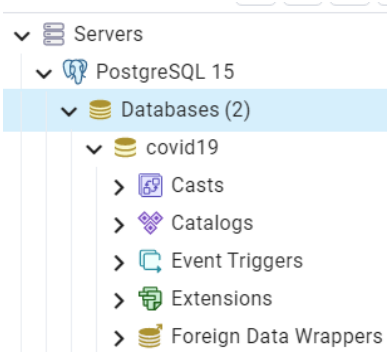
I clicked on Databases and created a bew database within it.



I typed in the database’s name, which is “covid 19”, and saved it.



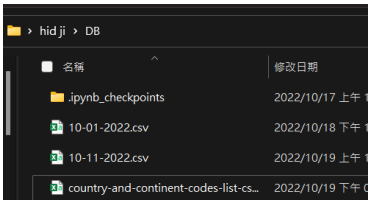
As you can see, the “covid 19” database has been created successfully.



Q2. The process of importing three required .csv files into covid19 database (can be screenshot and/or SQL/non-SQL statements with text explanation). Please included/described the data type and keys of the imported table in your screenshot, SQL statements, and explanations (30pts)

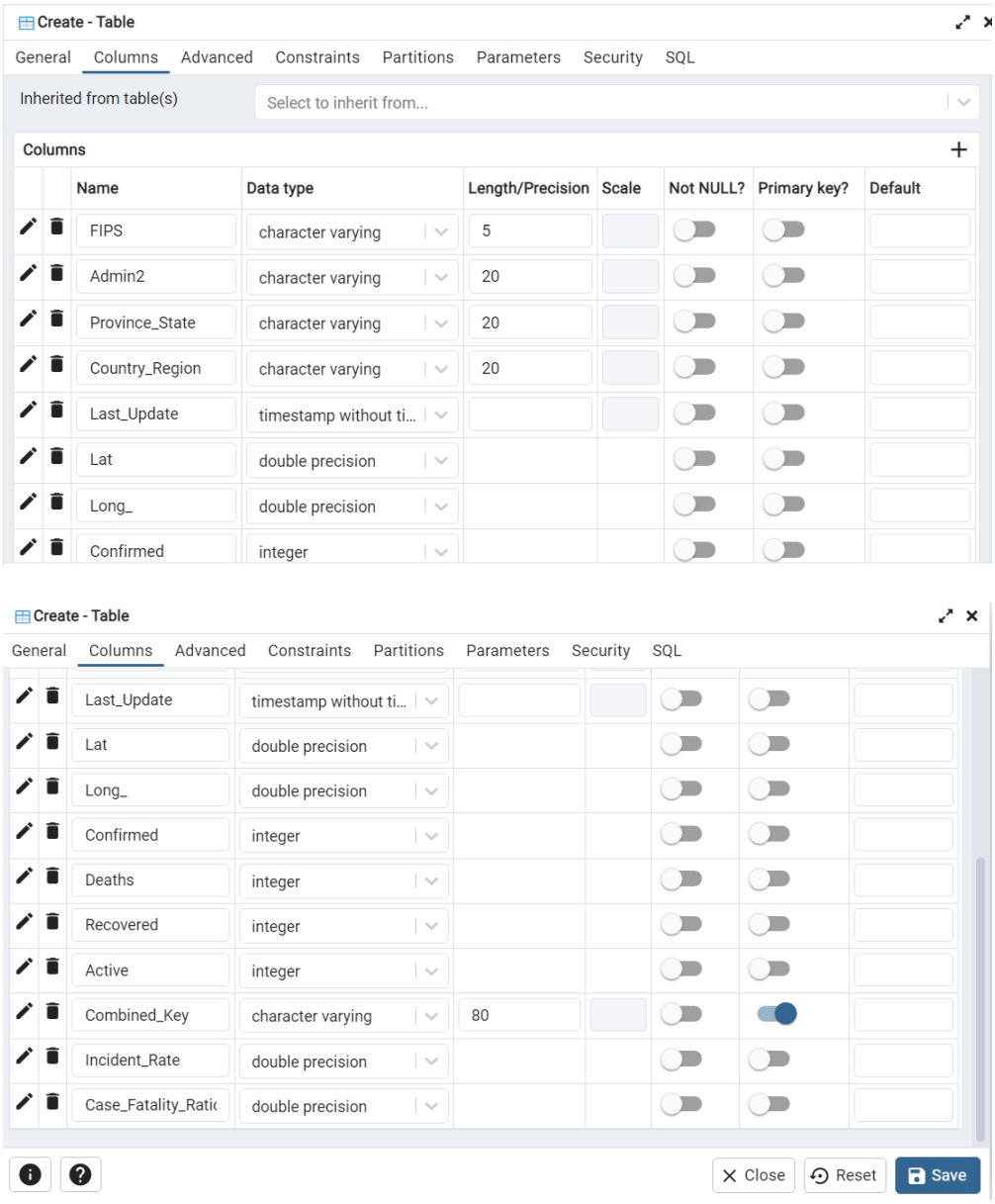
Ans 2:

First, I downloaded the required three csv files(using python in Jupyter Notebook) from github following the tutorial in the link, and it worked successfully.(https://www.youtube.com/watch?v=SMCoNBQupaE)



Second, I started to set the schema for “10012022” table according to “10-01-2022.csv” file, and specified the column names and data types.

“10012022” table(from “10-01-2022.csv”):



💡 However, I have changed the datatypes of “FIPS”, “Admin2”, “Province_State”, “Country_Region” because of the errors that I encountered later, the updated version of table schema is as following 👇

Dashboard | Properties | SQL | Statistics | Dependencies | Dependents | Processes | covid19/postgr...

```sql
-- Table: public.10012022

-- DROP TABLE IF EXISTS public."10012022";

CREATE TABLE IF NOT EXISTS public."10012022"
(
    "FIPS" character varying(5) COLLATE pg_catalog."default",
    "Admin2" character varying(100) COLLATE pg_catalog."default",
    "Province_State" character varying(100) COLLATE pg_catalog."default",
    "Country_Region" character varying(100) COLLATE pg_catalog."default",
    "Last_Update" time with time zone,
    "Lat" double precision,
    "Long_" double precision,
    "Confirmed" integer,
    "Deaths" integer,
    "Recovered" integer,
    "Active" integer,
    "Combined_Key" character varying(300) COLLATE pg_catalog."default" NOT NULL,
    "Incident_Rate" double precision,
    "Case_Fatality_Ratio" double precision,
    CONSTRAINT "10012022_pkey" PRIMARY KEY ("Combined_Key")
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS public."10012022"
    OWNER to tomcat_db_hw1;
```

As you can see, in the relation “10012022”, there are 14 attributes, and explanations are as follows:

- **FIPS**: Federal Information Processing Standards code that uniquely identifies counties within the USA. Since it is a code, I set it as varchar(5).
- **Admin2**: County name, which can be represented as varchar(100).
- **Province_State**: Province, state or dependency name, which can be represented as varchar(100).
- **Country_Region**: Country, region or sovereignty name, which can be represented as varchar(100).
- **Last Update**: MM/DD/YYYY HH:mm:ss (24 hour format, in UTC), which can be represented as timestamp without time zone
- **Lat** and **Long_**: Dot locations on the dashboard, it is a float type with many numbers, I think it would be better if it is set as double precision.
- **Confirmed**: Counts include confirmed and probable (where reported), it is a number, which can be represented as integer.
- **Deaths**: Counts include confirmed and probable (where reported), it is a number, which can be represented as integer.
- **Recovered**: Recovered cases, it is a number, which can be represented as integer.
- **Active:** Active cases = total cases - total recovered - total deaths, it is a number, which can be represented as integer.

- **Combined_Key:** A string that stores the combination of fips, admin2, province_state, and country_region, since it is a very long string, I set it as varchar(305)
- **Incident_Rate**: Incidence Rate = cases per 100,000 persons, float type with many numbers, which can be represented as double precision.
- **Case_Fatality_Ratio (%)**: Case-Fatality Ratio (%) = Number recorded deaths / Number cases, float type with many numbers, which can be represented as double precision.

I set **Combined_Key** as **primary key** because it is the only column that I think can distinguish different data.

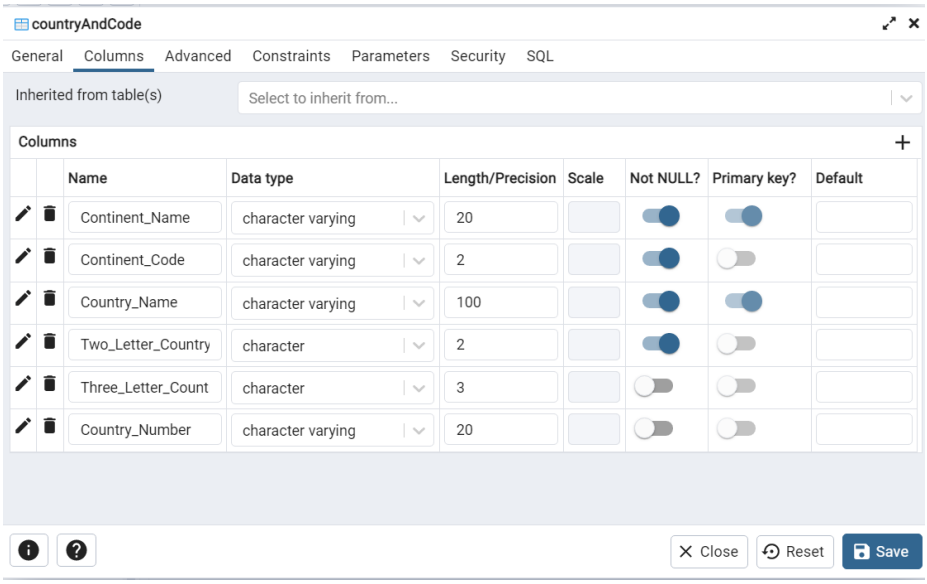
As for tables “10112022(from “10-11-2022.csv”)” and “countryAndCode”(from “country-and-continent-codes-list-csv.csv”), I repeated the above instructions to create them.

10112022 table(from “10-11-2022.csv”) :

Dashboard Properties SQL Statistics Dependencies Dependents Processes covid19/postgr…

```sql
-- Table: public.10112022

-- DROP TABLE IF EXISTS public."10112022";

CREATE TABLE IF NOT EXISTS public."10112022"
(
    "FIPS" character varying(5) COLLATE pg_catalog."default",
    "Admin2" character varying(100) COLLATE pg_catalog."default",
    "Province_State" character varying(100) COLLATE pg_catalog."default",
    "Country_Region" character varying(100) COLLATE pg_catalog."default",
    "Last_Update" time with time zone,
    "Lat" double precision,
    "Long_" double precision,
    "Confirmed" integer,
    "Deaths" integer,
    "Recovered" integer,
    "Active" integer,
    "Combined_Key" character varying(300) COLLATE pg_catalog."default" NOT NULL,
    "Incident_Rate" double precision,
    "Case_Fatality_Ratio" double precision,
    CONSTRAINT "10112022_pkey" PRIMARY KEY ("Combined_Key")
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS public."10112022"
    OWNER to tomcat_db_hw1;
```

country-and-continent-codes-list table(from “country-and-continent-codes-list-csv.csv”):




```
CREATE TABLE IF NOT EXISTS public."countryAndCode"
(
    "Continent_Name" character varying(20) COLLATE pg_catalog."default" NOT NULL,
    "Continent_Code" character varying(2) COLLATE pg_catalog."default" NOT NULL,
    "Country_Name" character varying(100) COLLATE pg_catalog."default" NOT NULL,
    "Two_Letter_Country_Code" character(2) COLLATE pg_catalog."default" NOT NULL,
    "Three_Letter_Country_Code" character(3) COLLATE pg_catalog."default",
    "Country_Number" character varying(20) COLLATE pg_catalog."default",
    CONSTRAINT "countryAndCode_pkey" PRIMARY KEY ("Continent_Name", "Country_Name")
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS public."countryAndCode"
    OWNER to tomcat_db_hw1;
```

As you can see, in the relation “countryAndCode”, there are 6 attributes, and explanations are as follows:

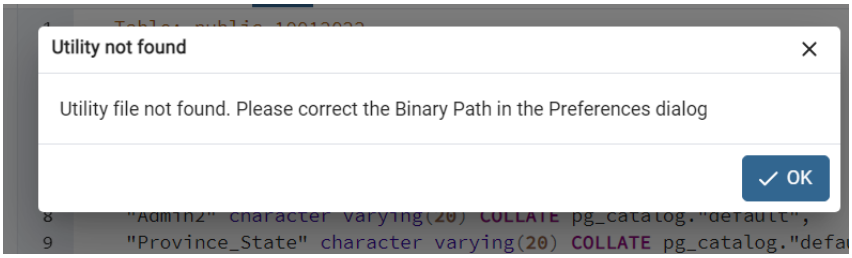
- Continent_Name: the name of continent, I set it as varchar(20)
- Continent_Code: the code version of representing continent, I set it as varchar(2)
- Country_Name: the name of country, I set it as varchar(100)
- Two_Letter_Country_Code: codes with two digits to represent countries, I set it as char(2)
- Three_Letter_Country_Code: codes with three digits to represent countries, I set it as char(3)
- Country_Number: the number of representing country, I set it as varchar(20)

I set **Continent_Name** and **Country_Name** as **primary key** because I think they are two useful elements to distinguish data.

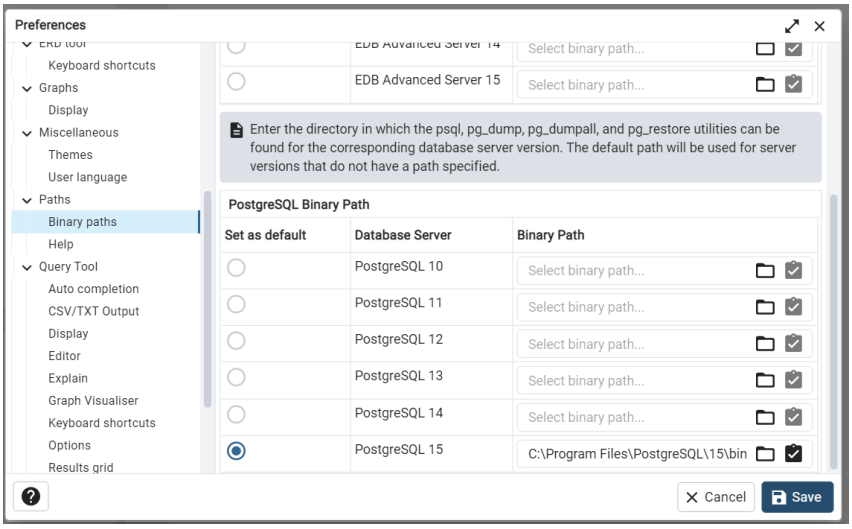
I found that the downloaded “country-and-continent-codes-list-csv.csv” file not only had data inconsistency, but also had missing elements, so I editted it according to the original version in github, and to ensure I did not do something wrong in the csv file, I set columns before “Three_Letter_Count” and “Country_Number” into NULL.

Import csv file

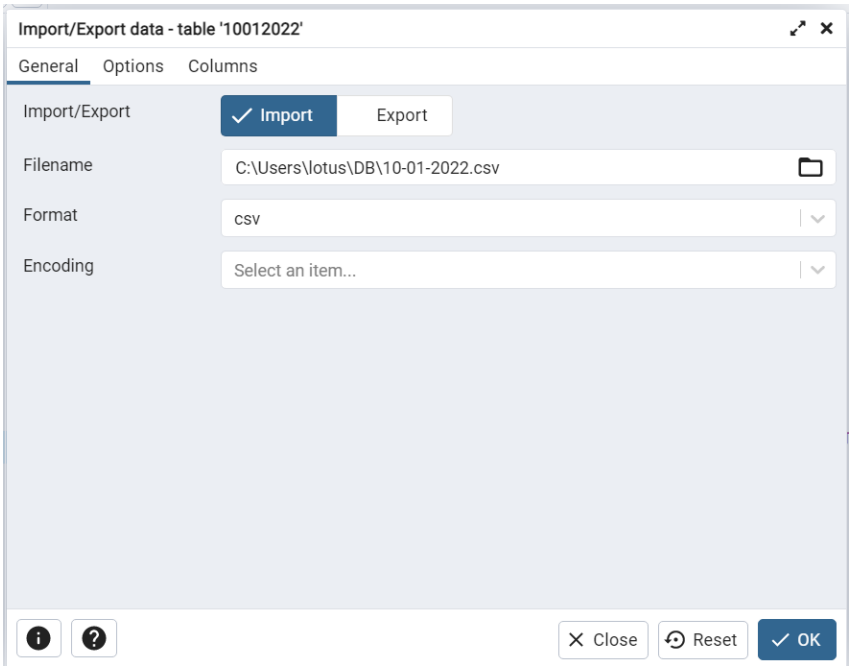
I clicked on table and imported csv file, but the following error happened. I got an “utility not found” error.



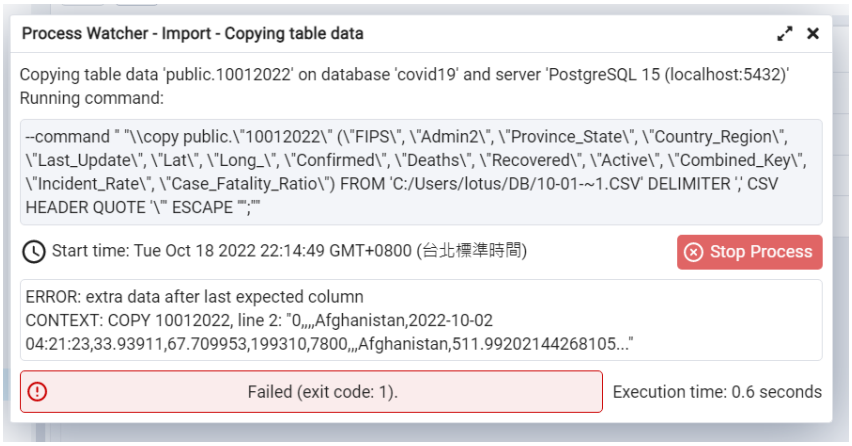
To solve this error, I went to Preferences to edit my Binary paths.



When I tried to import data again, the error disappeared.



When I was trying to importing the csv file again, this new error message kept showing.

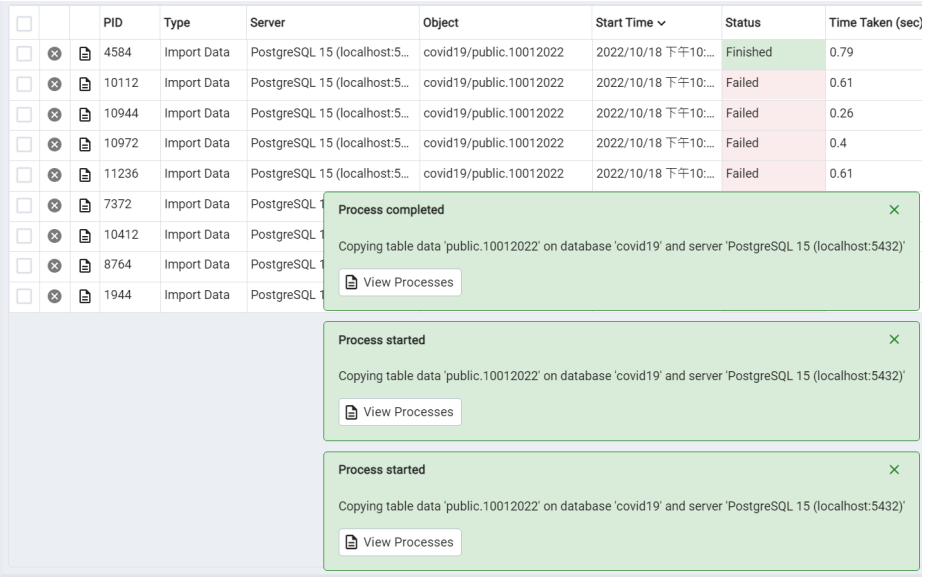


And I found that it was because when I downloaded the data from github, it simultaneously created a column before FIPS without my knowing. By deleting the column before FIPS, I solved this error.

	FIPS	Admin2	Province_	Country_F	Last_Upd	Lat	Long_	Confirmed	Deaths	Recovered	Active	Combined	Incident_F	Case_Fatality_
0				Afghanista	########	33.93911	67.70995	199310	7800			Afghanista	511.992	3.913502
1				Albania	########	41.1533	20.1683	332263	3589			Albania	11545.73	1.080168
2				Algeria	########	28.0339	1.6596	270676	6879			Algeria	617.2624	2.541415
3				Andorra	########	42.5063	1.5218	46227	155			Andorra	59829.16	0.335302
4				Angola	########	-11.2027	17.8739	103131	1917			Angola	313.7898	1.858801
5				Antarctica	########	-71.9499	23.347	11	0			Antarctica		0
6				Antigua ar	########	17.0608	-61.7964	9098	146			Antigua ar	9290.499	1.604748
7				Argentina	########	-38.4161	-63.6167	9708420	129897			Argentina	21480.81	1.337983
8				Armenia	########	40.0691	45.0382	442875	8683			Armenia	14945.66	1.960598
9			Australian	Australia	########	-35.4735	149.0124	205752	126			Australian	48061.67	0.061239
10			New Soutl	Australia	########	-33.8688	151.2093	3507880	5314			New Soutl	43211.14	0.151488
11			Northern T	Australia	########	-12.4634	130.8456	97483	73			Northern T	39691.78	0.074885
12			Queenslan	Australia	########	-27.4698	153.0251	1641389	2239			Queenslan	32086.58	0.136409
13			South Aus	Australia	########	-34.9285	138.6007	768961	946			South Aus	43778.02	0.123023
14			Tasmania	Australia	########	-42.8821	147.3272	249621	187			Tasmania,	46614.57	0.074914
15			Victoria	Australia	########	-37.8136	144.9631	2614813	5678			Victoria, A	39439.7	0.217147
16			Western A	Australia	########	-31.9505	115.8605	1154732	658			Western A	43896.15	0.056983
17				Austria	########	47.5162	14.5501	5144116	20754			Austria	57116.23	0.403451

HOWEVER…

After I solved the errors mentioned above, tons of errors came after it 😻, and it was because some of the province names, country names and admin2 are REEEEEEAAALLLLYY long😍, then I just set every column datatype into varchar(100)🥰🥰 And it finally worked. ^_^



I finally successfully imported all the data from csv file!!

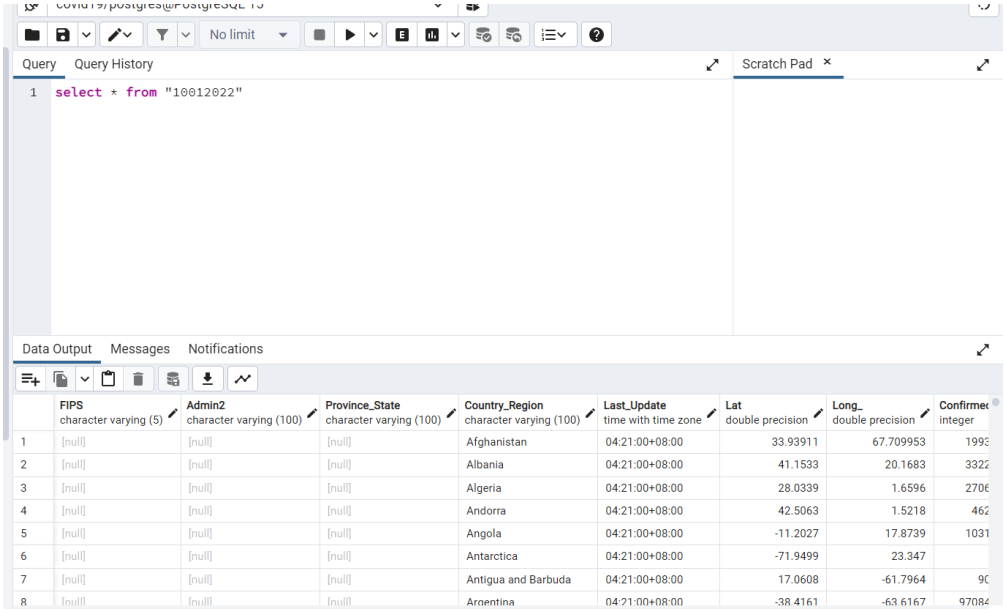
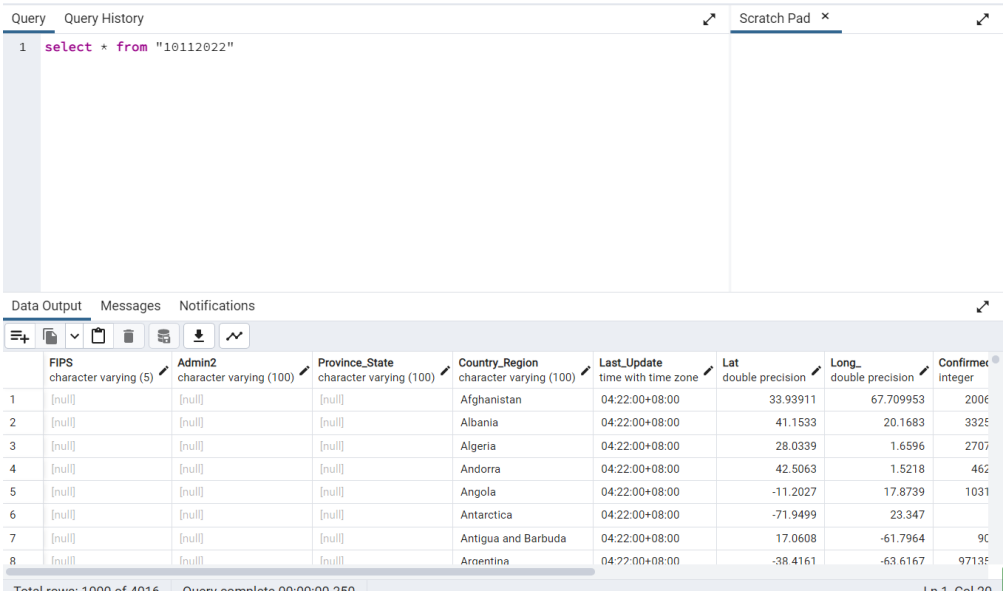


table “10112022” (from “10-01-2022.csv”)

“10112022” table was also imported the data successfully.🙆👍👍



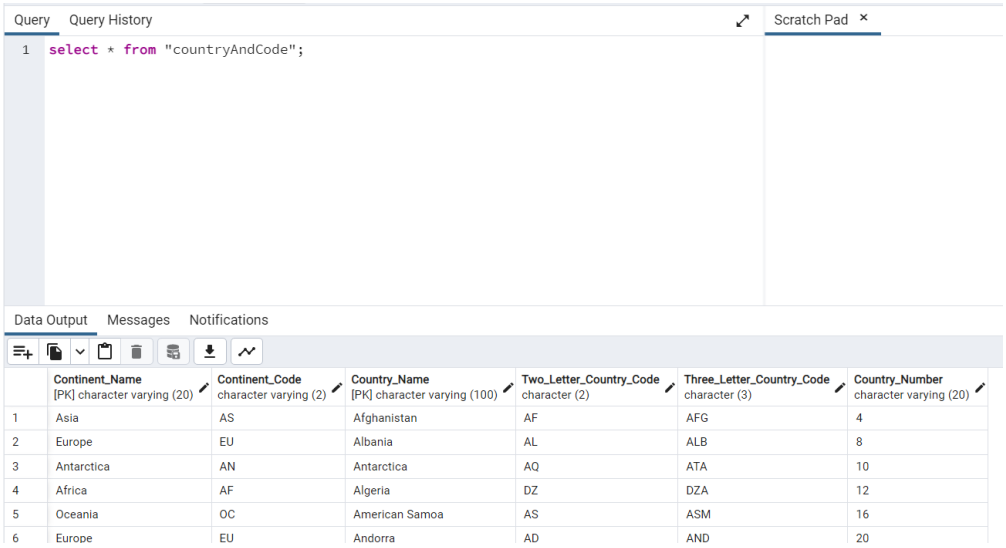
“10012022” table(from “10-01-2022.csv”)

⚠️ Since there were some data inconsistency(different names for the same country) between country-and-continent-codes-list.csv and (10-01-2022.csv & 10-11-2022.csv), I editted the country names in country-and-continent-codes-list.csv to make it consistent.

(e.g. `Afghanistan, the republic of` → `Afghanistan`)

It is diffucult to choose which side of country name representation should stay, but since it is a database homework to me, I just editted the names in country-and-continent-codes-list.csv due to convenience, hope no one would be disappointed to my choice, LOVE and PEACE.❤️🧡💛💚💙💜

the imported result also worked fine, we are good for the next section.



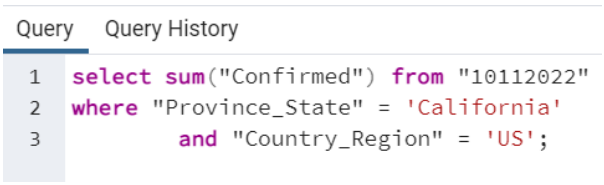
	Continent_Name [PK] character varying (20)	Continent_Code character varying (2)	Country_Name [PK] character varying (100)	Two_Letter_Country_Code character (2)	Three_Letter_Country_Code character (3)	Country_Number character varying (20)
1	Asia	AS	Afghanistan	AF	AFG	4
2	Europe	EU	Albania	AL	ALB	8
3	Antarctica	AN	Antarctica	AQ	ATA	10
4	Africa	AF	Algeria	DZ	DZA	12
5	Oceania	OC	American Samoa	AS	ASM	16
6	Europe	EU	Andorra	AD	AND	20

“countryAndCode” table(from “country-and-continent-codes-list-csv.csv”) :

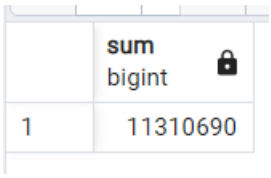
Q3. The SQL statements and output results of 4a (10pt). If the SQL statements or output results are not provided, you will not get the points.

4a: extract the total case number (Confirmed) in California, US on 2022-10-11 (data update date) (10 pts)

Ans 3:



query codes



	sum bigint
1	11310690

result

use `select` clause to get the table from table “10112022” with `where` clause specifying the condition, which is “Province_State” = ‘California’ and “Country_Region” = ‘US’.

Q4. The SQL statements and output results of 4b (10pt)

4b: extract the total case number (Confirmed) in California, US on 2022-10-01 (data update date) (10 pts)

Ans 4:

Query | Query History

```
1  select sum("Confirmed") from "10012022"
2  where "Province_State" = 'California'
3          and "Country_Region" = 'US';
```

query codes

	sum bigint
1	11268292

result

use `select` clause to get the table from table “10012022” with `where` clause specifying the condition, which is “Province_State” = ‘California’ and “Country_Region” = ‘US’.

Q5. The SQL statements and output results of 4c (10pt)

4c: extract the newly diagnosed case number (Confirmed) in California, US on 2022-10-11, compared with 2022-10-01 (data update date), in one SQL statement (please don’t just do 4a-4b) (10 pts)

Ans 5:

Query | Query History

```
1  select
2      sum("Confirmed") - (select sum("Confirmed") from "10012022"
3                          where "Province_State" = 'California' and
4                                "Country_Region" = 'US')
5      as newly_diagnosed_case_number
6  from "10112022"
7  where "Province_State" = 'California' and
8        "Country_Region" = 'US'
```

query codes

	newly_diagnosed_case_number bigint
1	42398

result

I used `Scalar Subquery` to get the total confirmed cases number from “10012022” table, and then subtracted it from the total confirmed cases number from “10112022” table to get the newly diagnosed cases number in California, US.

6. The SQL statements and output results of 4d (10pt)

4d. extract the country names (return Country_Region column) and total confirmed COVID cases (return Confirmed column) with more than 20,000,000 total COVID-19 cases on 2022-10-11 (data update date) (10 pts)

Ans 6:

Query | Query History

```
1  select
2      "Country_Region",
3      sum("Confirmed") as confirmed_cases
4  from "10112022"
5  group by "Country_Region"
6  having sum("Confirmed") > 20000000;
```

query codes

	Country_Region character varying (100)	confirmed_cases bigint
1	France	36187658
2	Korea, South	25025749
3	Italy	22896742
4	US	96783524
5	United Kingdom	23957457
6	Germany	34257916
7	India	44616235
8	Japan	21593704
9	Russia	20929929
10	Brazil	34731539

result

Since one Country_Region might have multiple different Province_States, I grouped the data with Country_Region using `group by` and then used the `having` clause with aggregate function(sum > 20000000).

7. The SQL statements and output results of 4e (10pt)

4e. extract the country names (return Country_Region column) and total confirmed COVID cases (return Confirmed column) with more than 20,000,000 total COVID-19 cases on 2022-10-11 (data update date). Try to join the Country code and continents mapping table, and return only the data from countries in Asia. (10 pts)

Ans 7:

Query | Query History

```
1  select
2      "Country_Region",
3      sum("Confirmed") as confirmed_cases
4  from "10112022"
5  join "countryAndCode"
6  on "countryAndCode"."Country_Name" = "10112022"."Country_Region"
7  where "countryAndCode"."Continent_Name" = 'Asia'
8  group by "Country_Region"
9  having sum("Confirmed") > 20000000;
```

query

	Country_Region character varying (100)	confirmed_cases bigint
1	Korea, South	25025749
2	India	44616235
3	Japan	21593704
4	Russia	20929929

result

Since I need to select countries that are in Asia, combining “10112022” table with “countryAndCode” is necessary.

First, I joined “countryAndCode” with “10112022” on the data that have the same contry name `"countryAndCode"."Country_Name" = "10112022"."Country_Region”`

and then filter the data whose location is in Asia

`"countryAndCode"."Continent_Name" = 'Asia’`

the aggregate function part is similar to Ans.6, and we successfully get the result.

8. The SQL statements and output results of 4f (10pt)

4f. extract the country names (return Country_Region column) and newly diagnosed case number of countries with a newly diagnosed case number (calculate the number by yourself) > 100,000 on 2022-10-11 (compare with 2022-10-01). In descending order of newly diagnosed case numbers. (10 pts)

Ans 8:

Query | Query History

```sql
with
old_data(country_region, old_confirmed) as(
    select "Country_Region", sum("Confirmed")
    from "10012022"
    group by "Country_Region"
),
new_data (country_region, new_confirmed) as(
    select "Country_Region", sum("Confirmed")
    from "10112022"
    group by "Country_Region")
select
    "old_data"."country_region",
    ("new_data"."new_confirmed" - "old_data"."old_confirmed") as newly_diagnosed_case
from "old_data"
join "new_data" on "new_data"."country_region" = "old_data"."country_region"
where "new_data"."new_confirmed" - "old_data"."old_confirmed" > 100000
order by newly_diagnosed_case desc
```

query codes

	country_region character varying (100)	newly_diagnosed_case bigint
1	Germany	871687
2	France	579373
3	Taiwan*	440596
4	Italy	396396
5	US	386493
6	Japan	264185
7	Russia	212106
8	Korea, South	206138
9	Austria	129544
10	Greece	106302

result

Since the newly diagnosed number comes from two different tables(”10012022” and ”10012022”), I first created two temporary table using the `with` clause. (old_data and new_data)

After getting two new temporary tables with **Country_Region** and **Confirmed** on the specific day respectively, I did subtraction of newly confirmed cases(from new_data table) and the confirmed cases(from old_data table) to get **newly diagnosed case number**.

I then filtered the table with the **newly_diagnosed_case>10000** condition and ordered the number with **descending order** to get the final output table.