Database - HW1

Link to this notion (which is better to read)

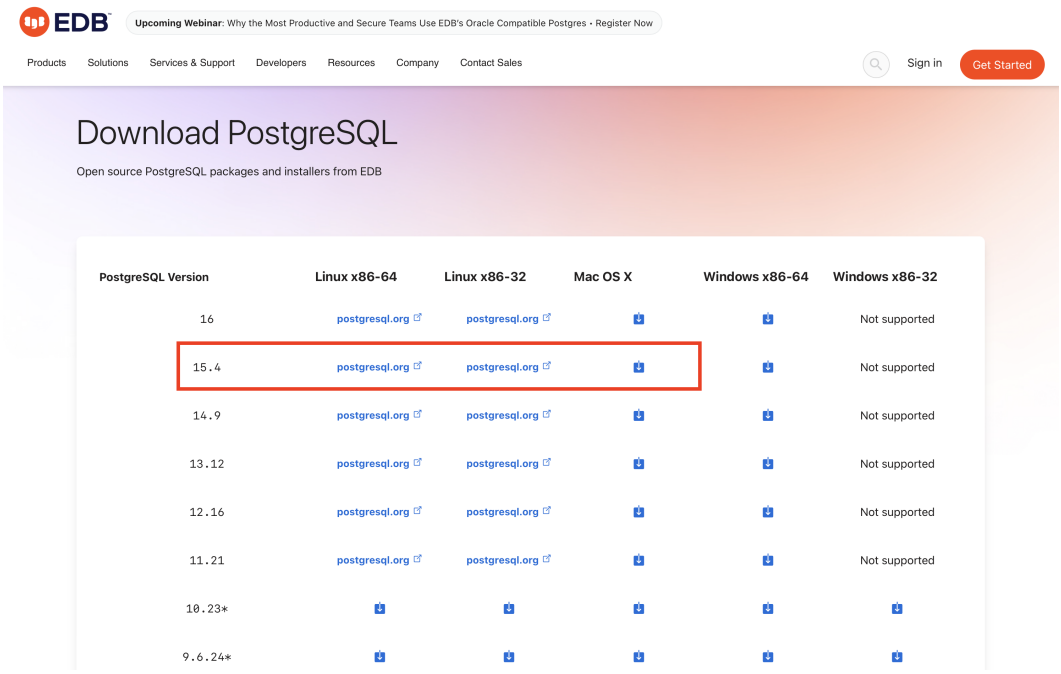
Database - HW1

Link to each question:

Q1. The process of creating the “lego” databases

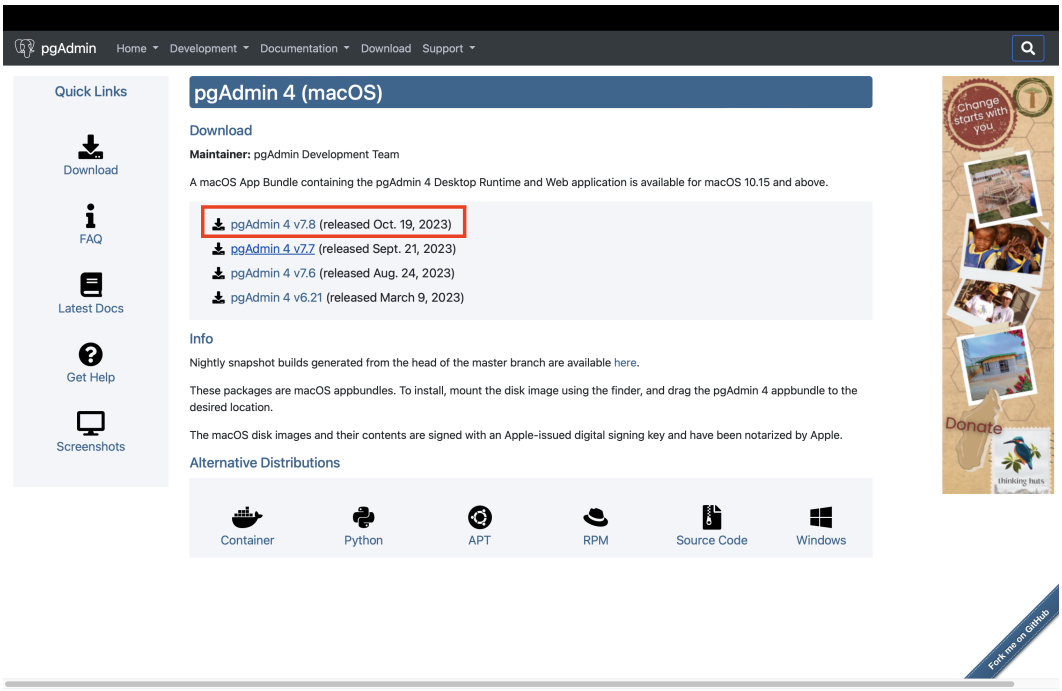
- **STEP 1.**

  First of all, before creating my own database, I need to set up all the environment that I can run SQL query and create a database. So I went to the website of PostgreSQL and download the Mac OS version of the interactive installer by EDB, which is shown below. Here I chose 15.4 version because I was worried that the latest version may have some bugs or unknown problems.



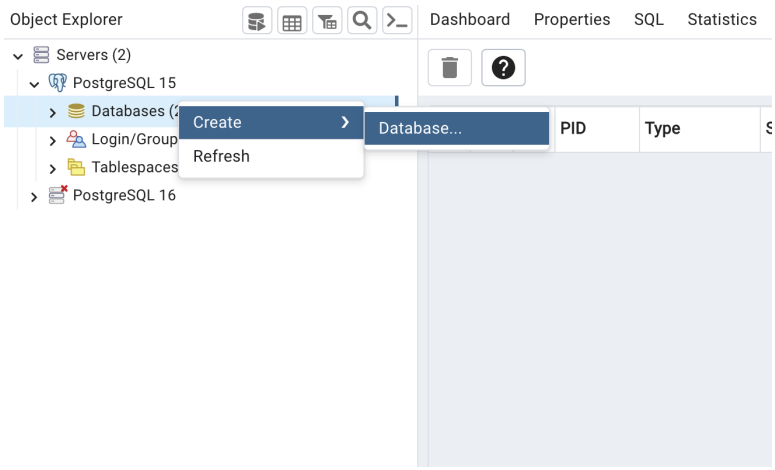
- **STEP 2.**

  After downloading the installer, I used it to downloading all the PostgreSQL 15.4 tools except for ***pgAdmin***, including ***SQL Shell(psql)*** and ***Application Stack Builder.*** The reason why I didn't download ***pgAdmin*** is that would cause a problem which I could not open it. So I came to the official website of pgAdmin 4 and download it.



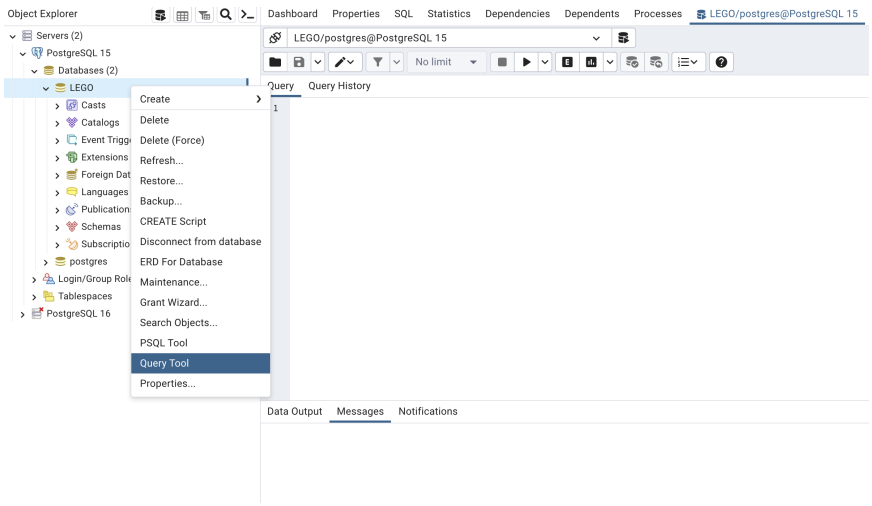
- **STEP 3.**

  After downloading the pgAdmin 4 and PostgreSQL 15.4, I could finally create my first database using UI provided by pgAdmin 4. I entered the app and we could see the *“Server”* icon one left hand side. Clicking it and which shown was PostgreSQL 15 icon and *“Database”* button down below. Right clicked the Database button and click *“Create→Database”* and then a window came up. Here I could type the name of this database and some other attributes. After setting all down, I clicked the save button and the “LEGO” database was created.



Q2. The process of importing eight required .csv files into lego database.

To import all the *.csv* files, I used the query tool in the pgAdmin UI, which is shown below. And I could then type in the schema of database datas using *DDL.*



Before importing datas from 8 csv files, we need to create the tables for each file first. I read the schema from the LEGO website and wrote the creating table query for each .csv file:

Create “colors” table:

- choose *id* as primary key because it’s unique

```
CREATE TABLE public.colors
(
    id VARCHAR(15),
    name VARCHAR(50),
    rgb CHAR(6),
    is_trans BOOLEAN,
    primary key (id)
);
```

Create “themes” table:

- choose *id* as primary key because it’s unique

```
CREATE TABLE themes
(
    id VARCHAR(15),
    name VARCHAR(100),
    parent_id VARCHAR(15),
    primary key (id)
);
```

Create “sets” table:

- choose *set_num* as primary key because it’s unique
- choose *theme_id* as foreign key reference from *themes*

```
CREATE TABLE sets
(
```

Create “pars” table:

- choose *part_num* as primary key because it’s unique
- choose *part_cat_id* as foreign key reference from *part_categories*

```
CREATE TABLE public.parts
(
```

```
    set_num VARCHAR(20),
    name VARCHAR(100),
    year INT,
    theme_id VARCHAR(15),
    num_parts INT,
    primary key (set_num),
    foreign key (theme_id)
        references themes(id)
);
```

```
    part_num VARCHAR(20),
    name VARCHAR(300),
    part_cat_id VARCHAR(15),
    primary key (part_num),
    foreign key (part_cat_id)
    references part_categories(id)
);
```

Create “parts_categories” table:

- choose *id* as primary key because it’s unique

```
CREATE TABLE public.part_categories
(
    id VARCHAR(15),
    name VARCHAR(100),
    primary key (id)
);
```

Create “inventories” table:

- choose *id* as primary key because it’s unique
- choose *set_num* as foreign key reference from sets

```
CREATE TABLE public.inventories
(
    id VARCHAR(15),
    version INT,
    set_num VARCHAR(20),
    primary key (id),
    foreign key (set_num)
        references sets(set_num)
);
```

Create “inventory_sets” table:

- choose *inventory_id* and *set_num* as primary key because it’s unique
- choose *inventory_id* as foreign key reference from *inventories*
- choose *set_num* as foreign key reference from *sets*

```
CREATE TABLE public.inventory_sets
(
    inventory_id VARCHAR(15),
    set_num VARCHAR(20),
    quantity INT,
    primary key (inventory_id, set_num),
    foreign key (inventory_id) references inventories(id),
```

```
    foreign key (set_num) references sets(set_num)
);
```

Create “inventory_parts” table:

- no primary key because there is no unique attribute in ***“inventory_parts.csv”***
- choose *inventory_id* as foreign key reference from *inventories*
- choose color_*id* as foreign key reference from *colors*

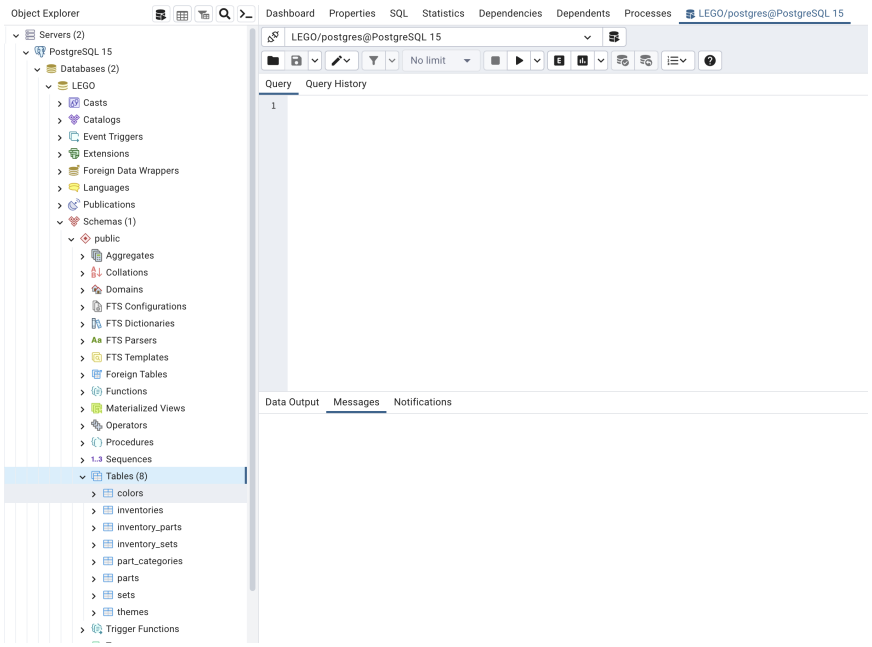
```
CREATE TABLE public.inventory_parts
(
    inventory_id VARCHAR(15),
    part_num VARCHAR(20),
    color_id VARCHAR(15),
    quantity INT,
    is_spare BOOLEAN,
    foreign key (inventory_id) references inventories(id),
    foreign key (color_id) references colors(id)
);
```

After creating each tables, we could import csv file by SQL query, too. Here is my code:

🔗*(Link to all the query)*

```
COPY public.colors(id, name, rgb, is_trans)
FROM '/Library/PostgreSQL/15/bin/HW1_Datas/colors.csv'
DELIMITER ','
CSV HEADER;
```

I’ve put all the files under the path ***'/Library/PostgreSQL/15/bin/HW1_Datas*** because “postgre” can directly access the datas from here without permission. And then the database could import datas from here. To import different file, all I need to do is to change the path of file and its table name and attributes, which was very easy to do. The result is shown below:



Q3. The SQL statements and output results of 4a.

- ***SQL statements:***

```
SELECT
    sets.name as Sets_name,
    themes.name as Themes_name
FROM
    sets,
    themes
WHERE
    themes.id = sets.theme_id AND
    sets.year = 2017
```

- ***Output results:*** (A part of the table)

total 296 datas

Data Output | Messages | Notifications

	sets_name character varying (100)	themes_name character varying (100)
1	Assembly Square	Modular Buildings
2	Carousel	Creator
3	Creative Builder Box	Classic
4	Creative Box	Classic
5	Blue Creative Box	Classic
6	Red Creative Box	Classic
7	Green Creative Box	Classic
8	Orange Creative Box	Classic
9	Demolition Site	Juniors
10	Police Truck Chase	Juniors
11	Anna & Elsa's Frozen Playground	Juniors
12	Batman vs. Mr. Freeze	Juniors
13	Fire Patrol Suitcase	Juniors
14	Mia's Farm Suitcase	Juniors
15	Andrea and Stephanie's Beach Holiday	Juniors
16	Miles' Space Adventures	Duplo
17	My First Number Train	Duplo
18	My First Plane	Duplo
19	My First Bird	Duplo
20	Sydney	Skylines
21	Chicago	Skylines

🔗: *Link to Github to show all the output result*

Q4. The SQL statements and output results of 4b.

- ***SQL statements:***

```
SELECT
    COUNT(set_num) AS Num_of_Set,
    year
FROM
    sets
WHERE
    year <= 2017 AND
    year >= 1950
GROUP BY
    year
ORDER BY
    Num_of_Set DESC
```

- ***Output results:*** (A part of the table)

total 66 datas

Data Output | Messages | Notifications

	num_of_set bigint	year integer
1	713	2014
2	665	2015
3	615	2012
4	596	2016
5	593	2013
6	503	2011
7	447	2002
8	444	2010
9	415	2003
10	402	2009
11	371	2004
12	349	2008
13	339	2001
14	330	2005
15	327	2000
16	325	1998
17	321	2007
18	300	1999

🔗: *Link to Github to show all the output result*

Q5. The SQL statements and output results of 4c.

- ***SQL statements:***

- ***Output results:***

```
WITH
    themes_cnt(name, total_set) AS(
        SELECT themes.name, COUNT(sets.name)
        FROM sets, themes
        WHERE themes.id = sets.theme_id
        GROUP BY themes.name)
SELECT
    name, total_set as max_set
FROM
    themes_cnt
WHERE
    total_set = (
        SELECT MAX(total_set)
        FROM themes_cnt
    );
```

Data Output | Messages | Notifications

	name character varying (100)	max_set bigint
1	Supplemental	496

Q6. The SQL statements and output results of 4d.

- ***SQL statements:***

```
WITH
    themes_avg(name, part) AS(
        SELECT themes.name, AVG(sets.num_parts)
        FROM sets, themes
        WHERE themes.id = sets.theme_id
        GROUP BY themes.name)
SELECT
    name, part as avg_num_parts
FROM
    themes_avg
ORDER BY
    avg_num_parts
```

- ***Output results:***

total 386 datas

Data Output | Messages | Notifications

	name character varying (100)	avg_num_parts numeric
1	Wooden Box Set	-1.0000000000000000000
2	Samsonite	0.0000000000000000000
3	Key Chain	0.18181818181818181818
4	Imperial Guards	1.0000000000000000000
5	Power Functions	1.8823529411764706
6	Control Lab	2.0000000000000000
7	Planet Series 1	3.0000000000000000
8	Western	3.0000000000000000
9	Value Packs	3.1666666666666667
10	Minifig Pack	3.5000000000000000
11	Classic Town	3.8181818181818182
12	Indiana Jones	4.0000000000000000
13	Series 14 Minifigures	5.3684210526315789
14	The Hobbit	5.5000000000000000
15	DFB Minifigures	5.6470588235294118
16	Series 15 Minifigures	6.3684210526315789
17	Series 16 Minifigures	7.1764705882352941
18	Tohunga	8.0000000000000000
19	The Simpsons	8.7894736842105263
20	Series 10 Minifigures	9.3500000000000000

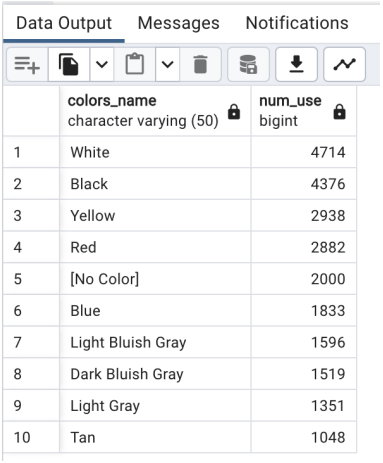
🔗: *Link to Github to show all the output result*

Q7. The SQL statements and output results of 4e.

- ***SQL statements:***

```
WITH
    color_use(name, num_use) AS(
        SELECT
            colors.name,
            COUNT(DISTINCT inventory_parts.part_num)
        FROM
            inventory_parts,
            colors
        WHERE
            colors.id = inventory_parts.color_id
        GROUP BY
            colors.id)
SELECT
    colors.name AS colors_name,
    color_use.num_use
FROM
    color_use,
    colors
WHERE
  colors.name = color_use.name
ORDER BY
    num_use DESC
LIMIT 10;
```

- ***Output results:***



Data Output | Messages | Notifications

	colors_name character varying (50)	num_use bigint
1	White	4714
2	Black	4376
3	Yellow	2938
4	Red	2882
5	[No Color]	2000
6	Blue	1833
7	Light Bluish Gray	1596
8	Dark Bluish Gray	1519
9	Light Gray	1351
10	Tan	1048

Q8. The SQL statements and output results of 4f.

- ***SQL statements:***

```
WITH
    -- to count the number of parts in each color and inventory
    quantity(color_name, inventory_id, quantity_sum, part_num) AS(

        SELECT
            colors.name,
            inventory_id,
            SUM(inventory_parts.quantity),
            inventory_parts.part_num
        FROM
            inventory_parts JOIN colors ON colors.id = inventory_parts.color_id
        GROUP BY
            colors.id, inventory_id, inventory_parts.part_num
```

```
    ),
    total_quantity(themes_name, color_name, total_quantity) AS(
        SELECT
            themes.name,
            quantity.color_name,
            SUM(quantity.quantity_sum)
        FROM
            ((themes JOIN sets ON themes.id = sets.theme_id)
            JOIN inventories ON sets.set_num = inventories.set_num)
            JOIN quantity ON inventories.id = quantity.inventory_id
        GROUP BY
            themes.name, quantity.color_name
    )
SELECT
    Origin.themes_name AS Theme_name,
    Origin.color_name AS Most_used_color
FROM
    total_quantity AS Origin
    LEFT OUTER JOIN total_quantity AS Bigger
        ON Origin.themes_name = Bigger.themes_name AND
           Origin.total_quantity < Bigger.total_quantity
WHERE
    Bigger.themes_name IS NULL
ORDER BY
    Origin.themes_name
```

- ***Output results:*** (A part of the table)

  total 384 datas

Data Output | Messages | Notifications

	theme_name character varying (100)	most_used_color character varying (50)
1	12V	Black
2	4 Juniors	White
3	4.5V	Black
4	9V	Black
5	Advent	Red
6	Advent Sub-Set	Red
7	Adventurers	Black
8	Agents	Black
9	Agori	Black
10	Airjitzu	Black
11	Airport	White
12	Alien Conquest	Light Bluish Gray
13	Alpha Team	Black
14	An Unexpected Journey	Reddish Brown
15	Angry Birds	Light Bluish Gray
16	Animals	White
17	Aquanauts	Yellow
18	Aquaraiders I	Black
19	Aquaraiders II	Black
20	Aquasharks	Black
21	Aquazone	Chrome Silver

💡 It's worth noticing that the theme *"Control Lab"* has two most_used_color because it uses the both color only 1 time, so these two colors are both list on the output result.

69	Control Lab	Dark Gray
70	Control Lab	Black

🔗: *Link to Github to show all the output result*