Trabajo Práctico Final
Bases de Datos por Grafos

Nombre: Eduardo Echeverria
Email: edwardminus1db@gmail.com
Tema: Bases de Datos Relacionales

Objetivos

El objetivo del presente trabajo práctico es poder crear y configurar una base de datos en Postgres a partir de varias tablas en formato CSV, tomando un diagrama de interdependencias entre las tablas como referencias. La idea es dejar una base de datos operativa para finalmente poder realizar consultas a la misma.

Descripción de la Base de Datos

Dentro de los archivos disponibles en la base de datos de LEGO se encuentran 8 tablas escritas en formato CSV:

- Inventory sets
- Inventory parts
- Inventories
- Colors
- Sets
- Parts
- Themes
- Part categories

Adicionalmente se encuentra un diagrama indicativo de las relaciones que deberían existir entre estas tablas.

Montaje Base de Datos en Postgres

El montaje de la base de datos en Postgres consiste en agregar las 8 tablas mencionadas en Postgres. Estos son los pasos seguidos.

1. Creación de la Base de Datos.
   Dentro del cliente de DBeaver se realizó la creación de la base de datos a través del siguiente comando:

   ```
   create database lego_database;
   ```

2. Creación de tablas dentro de la Base de Datos.
   El siguiente paso es la creación de las tablas. En esta parte solo realizamos la creación de la tabla y la definición del tipo de los datos, vamos a definir las Primary Key y Foreign Key más adelante. Para la creación de las tablas utilizamos los siguientes comandos:

Tabla “inventory_parts”:

```
create table inventory_parts (
   inventory_id INTEGER,
   part_num VARCHAR(50),
   color_id INTEGER,
   is_spare VARCHAR(50),
   quantity INTEGER
);
```

Tabla “inventory_sets”:

```
create table inventory_sets (
   inventory_id INTEGER,
   set_num VARCHAR(50),
   quantity INTEGER
);
```

Tabla “parts”:
En esta tabla, en un principio se definió el data type de “name” como VARCHAR(50), sin embargo, al momento de importar la data de la tabla CSV se encontraron errores debido a que en varios casos el nombre de la pieza excede los 50 caracteres, por lo que se tuvo que redefinir la tabla como VARCHAR(128):

```
create table parts (
   part_num VARCHAR(50),
   name VARCHAR(128),
   part_cat_id INTEGER
);
```

Tabla “colors”:

```
create table colors (
   id INTEGER,
   name VARCHAR(50),
   rgb VARCHAR(50),
   is_trans VARCHAR(50)
);
```

Tabla “inventories”:

```
create table inventories (
   id INTEGER,
   version INTEGER,
   set_num VARCHAR(50)
);
```

Tabla “part_categories”:

```
create table part_categories (
   id INTEGER,
   name VARCHAR(50)
);
```

Tabla “sets”:
En este caso, se tuvo que redefinir la tabla porque, de la misma forma que en el caso de la tabla “parts”, el nombre de algunos sets excede los 50 caracteres. En este caso la variable “name” se redefinió como VARCHAR(512).

```
create table sets (
   set_num VARCHAR(50),
   name VARCHAR(512),
   year INTEGER,
   theme_id INTEGER,
   num_parts INTEGER
);
```

Tabla “themes”:

```
create table themes (
   id INTEGER,
   name VARCHAR(50),
   parent_id INTEGER
);
```

Una vez creadas las tablas, se procedió a importar la data de las tablas CSV en las tablas creadas en Postgres. Para realizar la importación se utilizó la interfaz gráfica de DBeaver.

3. Definición de las Primary Keys y Foreign keys

En este paso se definieron las Primary Keys y Foreign keys en cada una de las tablas, con lo que se establecieron las relaciones entre las tablas. Para definir estas keys se emplearon los siguientes comandos:

Tabla “inventory_sets”
Primary Key: En este caso la primary key se trata de una clave primaria compuesta por las claves “*inventory_id”* y “*set_num*”

```
ALTER TABLE public.inventory_sets
ADD CONSTRAINT inventory_sets_pkey PRIMARY KEY (inventory_id, set_num);
```

Foreign Keys: Se tienen como foreign keys a "*inventory_id*" que corresponde a la primary key "*id*" en la tabla "inventories".

```
ALTER TABLE public.inventory_sets
ADD CONSTRAINT inventory_sets_inventory_id_fkey FOREIGN KEY (inventory_id)
REFERENCES public.inventories (id)
ON DELETE CASCADE;
```

Otra foreign key es "*set_num*" que corresponde a la primary key "*set_num*" en la tabla "sets"

```
ALTER TABLE public.inventory_sets
ADD CONSTRAINT inventory_sets_set_num_fkey FOREIGN KEY (set_num)
REFERENCES public.sets (set_num)
ON DELETE CASCADE;
```

Tabla "inventory_parts"
Primary Keys: En este caso la Primary Key se trata de una composite primary key o clave primaria compuesta. Esto debido a que las claves "*inventory_id*", "*part_num*" y "*color_id*" son claves primarias en otras tablas, por tanto ninguna de estas podría ser la clave primaria por sí sola en la tabla de "*inventory_parts*".

Adicionalmente, se observó que dentro de la tabla de "*inventory_parts*" existen algunos registros en los que la información contenida en las claves "*inventory_id*", "*part_num*" y "*color_id*" no es única, por tanto se tuvo que agregar una clave más: "is_spare"

En conclusión, la clave primaria compuesta la forman las claves: "*inventory_id*", "*part_num*", "*color_id*" is "*is_spare*";

```
ALTER TABLE public.inventory_parts
ADD CONSTRAINT inventory_parts_pkey PRIMARY KEY (inventory_id, part_num,
color_id, is_spare);
```

Foreign Keys: Tenemos como foreign key a "*inventory_id*" que es la primary key "*id*" en la tabla "inventories"

```
ALTER TABLE public.inventory_parts
ADD CONSTRAINT inventory_parts_inventory_id_fkey FOREIGN KEY
(inventory_id)
REFERENCES public.inventories (id)
ON DELETE CASCADE;
```

Luego tenemos como foreign key a "*part_num*" que es la primary key "*part_num*" en la tabla "parts"

```
ALTER TABLE public.inventory_parts
ADD CONSTRAINT inventory_parts_part_num_fkey FOREIGN KEY (part_num)
REFERENCES public.parts (part_num)
ON DELETE CASCADE;
```

Finalmente tenemos la foreigh key “*color_id*” que es la primary key “*id*” en la tabla “colors”

```
ALTER TABLE public.inventory_parts
ADD CONSTRAINT inventory_parts_color_id_fkey FOREIGN KEY (color_id)
REFERENCES public.colors (id)
ON DELETE CASCADE;
```

Tabla “inventories”
Primary key: Se definió “*id*” como primary key.

```
ALTER TABLE public.inventories
ADD CONSTRAINT inventories_pkey PRIMARY KEY (id);
```

Foreign keys: Tenemos la foreign key “*set_num*” que corresponde con la primary key “*set_num*” de la tabla “sets”

```
ALTER TABLE public.inventories
ADD CONSTRAINT inventories_set_num_fkey FOREIGN KEY (set_num)
REFERENCES public.sets (set_num)
ON DELETE CASCADE;
```

Tabla “sets”
Primary key: Se definió a “*set_num*” como clave primaria.

```
ALTER TABLE public.sets
ADD CONSTRAINT sets_pkey PRIMARY KEY (set_num);
```

Foreign Keys: Se tiene la foreign key “*theme_id*” correspondiente a la primary key “*id*” de la tabla “themes”

```
ALTER TABLE public.sets
ADD CONSTRAINT sets_theme_id_fkey FOREIGN KEY (theme_id)
REFERENCES public.themes (id)
ON DELETE CASCADE;
```

Tabla “parts”
Primary key: Se definió a “*part_num*” como clave primaria.

```
ALTER TABLE public.parts
ADD CONSTRAINT parts_pkey PRIMARY KEY (part_num);
```

Foreign key: Tenemos la foreign key "*part_cat_id*" que corresponde a al priamry key "*id*" de la tabla "part_categories"

```
ALTER TABLE public.parts
ADD CONSTRAINT parts_part_cat_id_fkey FOREIGN KEY (part_cat_id)
REFERENCES public.part_categories (id)
ON DELETE CASCADE;
```

Tabla "themes"
Primary key: Se definió a "*id*" como clave primaria.

```
ALTER TABLE public.themes
ADD CONSTRAINT themes_pkey PRIMARY KEY (id);
```

Foreign keys: No se tienen foreign keys en esta tabla

Tabla "colors"
Primary key: Se definió "*id*" como primary key

```
ALTER TABLE public.colors
ADD CONSTRAINT colors_pkey PRIMARY KEY (id);
```

Foreign keys: No se tienen foreign keys en esta tabla

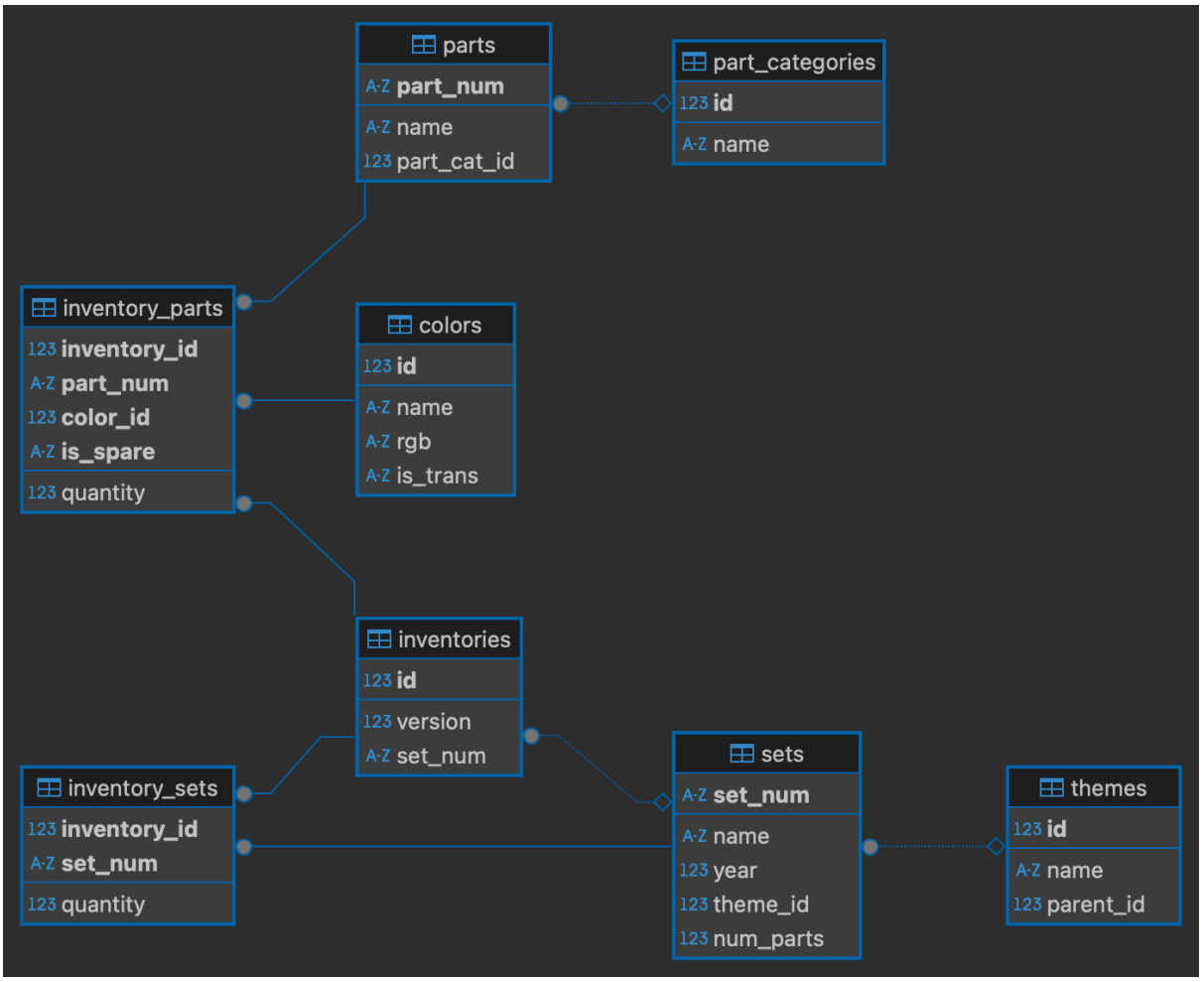
Tabla "part_categories"
Primary key: Se definió a "*id*" como clave primaria.

```
ALTER TABLE public.part_categories
ADD CONSTRAINT part_categories_pkey PRIMARY KEY (id);
```

Foreign keys: No se tienen foreign keys en esta tabla

Diagrama de la Base de Datos

Luego de definir las relaciones entre cada una de las tablas a traves de las primary keys y foreign keys, la estructura de la base de datos quedo como se aprecia en el siguiente diagrama:



Con la base de datos establecida, podemos pasar a realizar las consultas del ejercicio

Consultas a la Base de Datos

1. Colores más utilizados en los 90

Identifica cuáles son los 10 colores más frecuentemente usados en los sets de LEGO durante la década de los 90.

Para resolver este punto se utilizó la siguiente consulta:

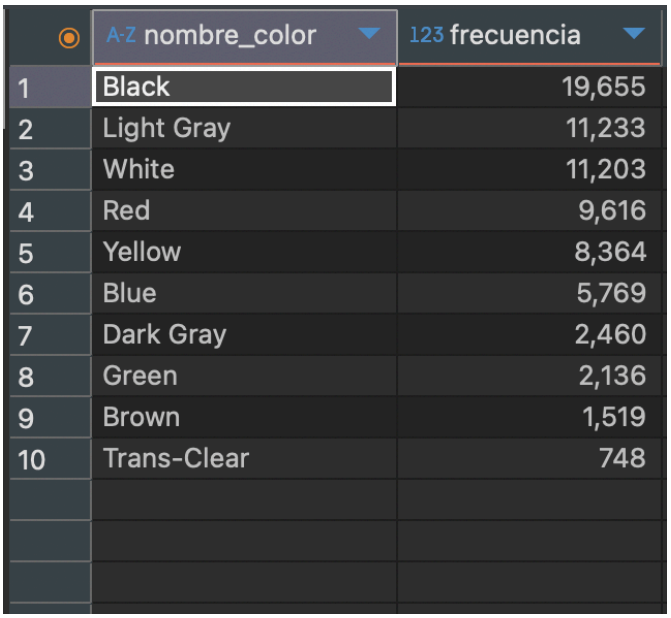
```
select c.name as nombre_color, count(*) as frecuencia
from colors c
left join inventory_parts ip on c.id = ip.color_id
left join inventories i on i.id = ip.inventory_id
left join sets s on s.set_num = i.set_num
where s.year between 1990 and 1999
group by c.name
order by frecuencia desc
limit 10;
```

Donde lo que se está haciendo es consultar por “name” de la tabla “colors” y redefiniendo esta variable como “nombre_color”, para luego contar todas las ocurrencias.

La tabla “colors” se une por izquierda con la tabla “inventory_parts”, luego estas tablas se unen por izquierda con la tabla “inventories” y luego estas se unen por izquierda con la tabla “sets”. Estos joins se realizan para poder vincular el nombre dentro de la tabla de “color” con el año de lanzamiento de la tabla “sets”.

Luego se filtran los sets lanzados entre 1990 y 1999. Finalmente se agrupan los resultados por “name” y se los ordena de forma descendente de acuerdo a la cantidad de la cuenta definida en la variable “frecuencia” limitando la salida a los primeros 10 resultados.

Resultado:



	nombre_color	frecuencia
1	Black	19,655
2	Light Gray	11,233
3	White	11,203
4	Red	9,616
5	Yellow	8,364
6	Blue	5,769
7	Dark Gray	2,460
8	Green	2,136
9	Brown	1,519
10	Trans-Clear	748

2. Colores únicos
Determina la cantidad de colores que son únicos en toda la base de datos.

Se obtuvo esta información con la siguiente consulta:

```
select c.name as nombre_color, s.name as nombre_set, s.set_num
from colors c
join (
   select ip.color_id, i.set_num
   from inventory_parts ip
   join inventories i on ip.inventory_id = i.id
   group by ip.color_id, i.set_num
) conjunto_color_set on c.id = conjunto_color_set.color_id
join sets s on s.set_num = conjunto_color_set.set_num
group by c.name, s.name, s.set_num
having COUNT(*) = 1
order by c.name;
```

Partimos de una consulta interna (“conjunto_color_set”) donde unimos las tablas de “inventory_parts” y “inventories” para poder vincular las piezas con sus respectivos sets. Luego seleccionamos “color_id” y “set_num” para cada combinacion de color y set.

Luego en la consulta externa vinculamos el resultado de la consulta interna con la tabla de ”colors” y luego unimos la tabla “sets” con el resultado de la consulta interna, de esta manera podemos identificar las combinaciones donde el color es único, es decir aparece una sola vez en cada set.

Resultado:

nombre_color	nombre_set	set_num
Aqua	Pretty in Pink Jewels-n-More	7533-1
Aqua	Advent Calendar 2004 Clikits (Day 18) Paper Clip / Hair C	7575-19
Aqua	Pretty in Pink Beauty Set	7527-1
Aqua	Bears on the Beach	5977-1
Aqua	Tropical Breeze Jewels 'n' More	7546-1
Aqua	Advent Calendar 2004 Clikits (Day 7) Gift Tag with Icons	7575-8
Aqua	Jewels-n-Clips	7509-1
Aqua	My Starry Notes	7526-1
Black	Darth Maul	6005188-1
Black	Pick-up Truck	6146-1
Black	Advent Calendar 1998 Classic Basic (Day 19) Boat	1298-20
Black	Maggie Simpson	71005-5
Black	Fight on the Flying Wing	7683-1
Black	Amusement Park Hot Dog Van	41129-1
Black	Metroliner	4558-1
Black	Zesk	8977-1
Black	Motorbike	3506-1
Black	Lake-town Guard	30216-1

3. Tendencia de piezas por sets a lo largo de los años
Analiza cómo ha evolucionado la cantidad de piezas incluidas en los sets de LEGO a través del tiempo.

Este punto se resolvió con la siguiente consulta a la base:

```
select s.year,
      count(distinct s.set_num) as numero_sets,
      sum(ip.quantity) as piezas_totales,
         round(sum(ip.quantity) * 1.0 / count(distinct s.set_num), 2) as
promedio_piezas_set
from sets s
join inventories i on s.set_num = i.set_num
join inventory_parts ip on i.id = ip.inventory_id
group by s.year
order by s.year;
```

En esta consulta en primer lugar se calcula la cantidad de sets por año dentro de la tabla “sets”, luego se obtiene la cantidad de piezas en total, con lo que se puede calcular el promedio de piezas por set.

Finalmente se ordena esta cantidad por año de lanzamiento y de las ordena de forma ascendente por año.

Resultado:

	year	numero_sets	piezas_totales	promedio_piezas_set
1	1,950	7	71	10.14
2	1,953	4	66	16.5
3	1,954	14	173	12.36
4	1,955	27	1,038	38.44
5	1,956	11	222	20.18
6	1,957	18	895	49.72
7	1,958	42	1,872	44.57
8	1,959	4	65	16.25
9	1,960	2	526	263
10	1,961	16	1,215	75.94
11	1,962	38	3,270	86.05
12	1,963	13	600	46.15
13	1,964	9	909	101
14	1,965	10	1,071	107.1
15	1,966	86	3,702	43.05

4. Temáticas más populares de los 2000
Identifica cuáles fueron las temáticas de sets más populares durante la década de los 2000.

Esta información se obtuvo con la siguiente consulta:

```
select t.name as tematica, count(s.set_num ) as cantidad_sets
from themes t
left join sets s on s.theme_id  = t.id
where s.year between 2000 and 2009
group by t.name
order by cantidad_sets desc;
```

Seleccionamos “name” de la tabla de “themes” y “set_num” de la tabla “sets”, la tabla “themes” se une por izquierda con la de “sets”. De esta forma vinculamos el nombre de la tematica con la cantidad de sets lanzados por año, filtrando entre los años 2000 y 2009.

Finalmente agrupamos por “name” y ordenamos de forma descendente por cantidad.

Resultado:

	A-Z tematica	123 cantidad_sets
1	City	142
2	Bulk Bricks	124
3	Basic Set	121
4	Creator	118
5	Clikits	105
6	Technic	104
7	Star Wars Episode 4/5/6	88
8	Soccer	80
9	Supplemental	73
10	Knights Kingdom II	69
11	Construction	66
12	Airport	57
13	Studios	53
14	Tiny Turbos	51
15	Advent Sub-Set	48