Google drive database home made

César Segura, Valentina Herrera, Lucia Ardila

Mayo 2022

Índice general

0.1. Introducción

El almacenamiento de los archivos se ha convertido en una parte importante de la vida humana, debido a que esto nos permite tener un orden de los archivos, poderlos compartir, tener un espacio propio y privado para ellos, poderlos ordenar, entre otras. Similarmente, es este proyecto de una base de datos que almacena los archivos, *Google Drive database home made*, es propuesto, diseñado y implementado. En este reporte se mostrara el proceso de creación y implementación serán explicados. Después de la explicación del proyecto, se mostrara los requerimientos y el diseño conceptual.

0.2. Descripción del proyecto

La motivación de hacer este proyecto fue pensando en que se podría hacer un almacenamiento de archivos mas privado, siendo este compartido con amigos o familia, teniendo un disco duro con espacio de 1 Terabyte. Esta idea surgió a partir de un problema que le encontramos a un almacenamiento en la nube como lo es Drive el cual es que podría ser inseguro al tener el correo y algunos datos, estando además susceptible al borrado por parte de terceros al ser estas redes publicas. También el querer tener un almacenamiento de datos mas organizado al ser propio dejando paso a una mejora en la forma de encontrar los archivos. Sin embargo, algunas dificultades que analizamos es que podríamos tener datos duplicados, que el espacio de almacenamiento se sobrepase o se agote y el tener que trabajar con medidas de almacenamiento diferentes.

Este proyecto diseñara una base de datos de almacenamiento de archivos que sera usado por usuarios que necesiten guardar archivos de todo tipo, ya sea una imagen, video, archivo de excel, word, power point, pdf, cpp, exe, entre otros. Para esto el usuario tendrá acceso a los archivos, además de que estos usuarios tendrán 4 Roles; siendo Administrador: el cual tiene acceso a todo lo que ocurre dentro del archivo, Editor: que se le permite editar el archivo, Lector: que tiene únicamente el rol de leerlo mas no modificarlo y Comentarista: el cual puede hacer comentarios a los lados del archivo, aportando información a este.

El usuario tiene acceso a muchos archivos sin embargo hay archivos que tienen una clave para mantener la seguridad y privacidad de estos archivos. El usuario esta sujeto además de tener permisos para cada archivo, se tiene un rol pero además un permiso especial en cada caso, teniendo en primer lugar todos los permisos, en otro el poder insertar, otro de actualizar y el de leer.

0.2.1. Necesidad de la base de datos

Las razones por las cuales se necesita la base de datos en este proyecto es porque brinda la posibilidad de contar con grandes cantidades de datos, es decir, con nuestro proyecto tenemos la posibilidad de hacer consultas para llegar

a información mas delimitada que necesite el usuario y la segunda es que es seguro y privado, como cualquier programa los usuarios necesitan de algo que les de seguridad y confianza a la hora de meter archivos que pueden contener información de ellos, por lo cual se tiene este disco duro privado para ellos.

0.2.2. Utilización de la base de datos

Como se había mencionado anteriormente esta base de datos puede ser utilizada por cualquier persona que tenga que meter algún archivo, desde que cuente con un computador y espacio de almacenamiento.

0.3. Requerimientos funcionales

En esta sección se discuten los requerimientos funcionales en nuestro proyecto, además de mostrar las primeras entidades que se pensaron con sus atributos y como se fue cambiando esto. También se plantean las relaciones que tiene cada atributo.

0.3.1. Primer planteamiento de las entidades y atributos

Inicialmente en nuestro proyecto hicimos un bosquejo y pensamos en 4 tablas siendo:

- Propietario el cual contiene Id, nombre, administrador, espacio de almacenamiento, medida almacenamiento, numero archivos guardados.
- Archivo donde definimos su Id, nombre, clave archivo, tipo, peso, medida almacenamiento, fecha.
- Clave que solo contenía su clave.
- Admin el cual tiene permisos y roles

Entidad	Atributos
Propietario	-Id -Nombre -Administrador -Espacio almacenamiento -Medida almacenamiento -Número archivos guardados
Archivo	-Id -Nombre -Clave Archivo -Tipo -Peso -Medida almacenamiento -Fecha
Clave	-Clave
Admin	-Permisos -Roles

Sin embargo, las tablas que teníamos estaban mal planteadas, nos dimos cuenta que no teníamos una entidad de carpeta, que nos ayudaría a ordenar los archivos, no dándole solución a nuestro problema, además que caíamos en la redundancia, al tener atributos repetidos, y la entidad de clave era una tabla innecesaria ya que esta se podía colocar como un atributo y quitamos la tabla de admin, separando los atributos volviéndolas en entidades para definir bien las relaciones.

0.3.2. Planteamiento final de las entidades y atributos

Luego de corregir los errores que teníamos, planteamos de nuevo nuestras tablas, creando otros atributos como lo es Usuario, Archivo, Carpeta, Rol y Permisos. En estas ya las relaciones eran mas claras y tenia sentido las entidades que planteábamos mostrándolas a continuación:

- Los datos de los usuarios son su Id, nombre, correo, clave de usuario y numero de archivos guardados.
- Un archivo puede tener código, nombre, y datos del archivo como: su clave, tipo, peso, medida de almacenamiento y la fecha.
- Una carpeta puede tener un código, nombre y datos de la carpeta como: fecha, peso, medida de almacenamiento y numero de archivos en ella.
- Un Rol contiene el número que identifica al rol, y los roles que serian Administrador, Editor, Lector y Comentarista.

- Un Permiso contiene el numero que identifica al permiso, y los permisos que serian Todos, Insertar, Leer y Actualizar.

0.3.3. Planteamiento de las relaciones

- Una carpeta contiene muchos archivos
- Un archivo no puede estar en muchas carpetas debido a que este cambiaría su código
- Un usuario tiene muchos archivos
- Un archivo puede ser de muchos usuarios
- Un usuario posee muchos roles y permisos

Entidad	Atributos	Relación
Usuario	-Id -Nombre -Correo -Clave Usuario -Número archivos	Rol(1) -> Posee Permisos(1) -> Posee
Rol	-Numero del Rol -Rol	Usuario(m) -> Tiene
Permisos	-Número de Permisos -Permisos	Usuario(m) -> Tiene
Archivo	-Código -Nombre -Datos archivo: *Clave Archivo *Tipo *Peso *Medida almacenamiento *Fecha	Carpeta(1) -> Esta Usuario(m) -> Pertenece
Carpeta	-Código -Nombre -Datos Carpeta *Fecha *Peso *Medida almacenamiento *Número de archivos	Archivo(m) -> Contiene Usuario(m) -> Pertenece

Al tener listas las entidades con los atributos y todas las relaciones nos dispusimos a hacer el Diagrama Entidad Relación

0.4. Creaciones

0.4.1. Tablas Desnormalizadas

La creación de tablas en un principio, cuando estábamos planeando el proyecto se componía por la tablas propietario, archivo y carpeta, sin embargo, nuestro principal problema estaba en que no estaba normalizada esto era un problema frente a las consultas, debido a que aunque pudiéramos hacer consultas simples, cuando hiciéramos carga masiva de datos, empezaríamos a presentar muchos problemas, el esquema relacional estaba mal y además nos dimos cuenta que habían tablas que no eran necesarias ya que por ejemplo, podíamos crear un rol para un administrador, darle los permisos necesarios y además crear otros roles como el de escritor, editor, comentarista y lector.

Cuando iniciamos a implementar lo anterior en PgAdmin a partir del esquema creado anteriormente obtuvimos lo siguiente:

```sql
DROP TABLE IF EXISTS Propietario;
CREATE TABLE Propietario (
  id_propietario INTEGER PRIMARY KEY,
  correo VARCHAR (100) NOT NULL,
  Espacio_almacenamiento REAL,
  medida_almacenamiento VARCHAR(4),
  Usuario VARCHAR (100) NOT NULL,
  Archivos_guardados INTEGER,
  Permisos VARCHAR (50),
  Rol VARCHAR (50),
  contraseña_usuario INTEGER
);
DROP TABLE IF EXISTS Carpeta;
CREATE TABLE Carpeta (
  id_carpeta INTEGER PRIMARY KEY,
  nombre_archivo VARCHAR (100),
  Fecha_carpeta DATE,
  num_archivos INTEGER,
  id_pro INTEGER,
  FOREIGN KEY (id_pro) REFERENCES propietario (id_propietario)
);


DROP TABLE IF EXISTS Archivo;
CREATE TABLE Archivo(
  id_Archivo INTEGER PRIMARY KEY,
  Tipo_archivo VARCHAR (5) NOT NULL,
  Nombre VARCHAR (100),
  Fecha_archivo DATE,
  Peso_archivo REAL,
  contraseña_archivo INTEGER,
  id_prop INTEGER,
  id_carp INTEGER,
  FOREIGN KEY (id_prop) REFERENCES propietario(id_propietario),
  FOREIGN KEY (id_carp) REFERENCES carpeta(ID_carpeta)
);
```

Para este código, decidimos quitar las tablas de admin y clave ya que por un lado, nos dimos cuenta que la base de datos, podía ser mejor si se implementar el rol de admin que pueda tener todos los permisos y adicionalmente tener otros papeles como lo es de el editar, comentar y de lector, por otro lado, el de clave, nos pareció mejor asignarlos otra tabla para así poder tener más fácilmente si la clave es del propietario o si es para un archivo. También, decidimos agregar los parámetros medida de almacenamiento, para saber si un archivo o carpeta se mide en GB, MB, TB o KB y el rol que un usuario cumple.

0.4.2. Tablas Normalizadas

Como bien se sabe la normalización es una herramienta que nos facilita la eficiencia de consultas, dando paso a que de un modelo entidad-relación se convierta en un modelo relacional.
En nuestro proyecto se normalizo hasta la Tercera Forma Normal, a continuación se tiene el proceso para lograr la normalización:

0.4.3. 1NF

La primera forma normal (1FN) se refiere a que todos los atributos llave deben estar definidos, es decir, en los renglones o columnas se tiene un solo valor, no un conjunto de ellos. En nuestro proyecto observamos que teníamos varios datos, los cuales podían salir repetidos, siendo este un problema, debido a que un usuario puede tener varios archivos, o que una carpeta puede contener varios archivos, lo cual era necesario cambiar las tablas y dejar estas simples con cada llave.

Drive

Codigo	Usuario	Correo	Contraseña	Archivo_nombre	TipoArchivo	TamañoArchivo	Carpeta_nombre	Fechaarchivo	Papeles/Permisos	PesoCarpeta	Rol	archivosGuardados	numeroArchivos_carpeta	Fecha_carpeta
1	Valentina Herrera	valentina.herrera@gmail.com	NULL	Proyecto1	word	15 KB	ProyectoFinal	10/2/2021	Editar	165 KB	editor	30	3	10/2/2021
1	Valentina Herrera	valentina.herrera@gmail.com	NULL	Proyecto2	word	41 KB	ProyectoFinal	10/10/2021	Editar	165 KB	editor	30	3	10/10/2021
1	Valentina Herrera	valentina.herrera@gmail.com	NULL	Proyecto3	word	109 KB	ProyectoFinal	10/22/2021	Editar	165 KB	editor	30	3	10/22/2021
2	Lucia Ardila	lucia.ardila@gmail.com	1234	Entrega1	pdf	800 KB	Fundamentos	5/15/2021	Todo	800 KB	admin	60	10	5/15/2021
2	Lucia Ardila	lucia.ardila@gmail.com	4567	Entrega2	pdf	345 KB	Escritura	4/21/2022	Todo	345 KB	admin	100	20	4/21/2022
3	Cesar Segura	cesar.segura@gmail.com	9876	TrabajoCooperativo	word	50 KB	Logica	8/23/2020	Todo	50 KB	admin	1000	50	8/23/2020
4	Juan Contreras	juan.contreras@gmail.com	NULL	Defensa	pdf	145 KB	Universidad	1/11/2018	Ver	145 KB	observador	2	1	1/11/2018
5	Juan Rodriguez	juan.rodriguez@gmail.com	NULL	Constitucion	jpg	3.034 KB	Electivas	10/31/2019	comentar	5.083 KB	observador	3	2	10/31/2019
5	Juan Rodriguez	juan.rodriguez@gmail.com	NULL	Analisis	png	2.049 KB	Electivas	6/2/2018	Comentar	5.083 KB	comentarista	15	3	6/2/2018

En la tabla anterior observamos como hay varios usuarios que tienen el mismo código varias veces, separando esto para conseguir cumplir la primera regla formal.

codigo	usuario	correo	archivos_Guardados	contraseña
1	Valentina Herrera	valentina.herrera@gmail.com	30	NULL
2	Lucia Ardila	lucia.ardila@gmail.com	160	456
3	Cesar Segura	cesar.segura@gmail.com	1000	369
4	Juan Contreras	juan.contreras@gmail.com	2	NULL
5	Juan Rodriguez	juan.rodriguez@gmail.com	18	NULL

codigo	archivo_nombre	tipoArchivo	tamaño_archivo	carpeta_nombre	Fecha_archivo	PesoCarpeta	nummeroArchivos	Fecha_carpeta	Contraseña_archivo
1	Proyecto1	word	15 KB	ProyectoFinal	10/2/2021	165 KB	3	15/02/2021	NULL
1	Proyecto2	word	41 KB	ProyectoFinal	10/10/2021	165 KB	3	20/10/2021	NULL
1	Proyecto3	word	109 KB	ProyectoFinal	10/22/2021	165 KB	3	30/12/2021	NULL
2	Entrega1	pdf	800 KB	Fundamentos	5/15/2021	800 KB	10	6/9/2021	1234
2	Entrega2	pdf	345 KB	Escritura	4/21/2022	345 KB	20	21/12/2022	4567
3	TrabajoCooperativo	word	50 KB	Logica	8/23/2020	50 KB	50	31/10/2020	9876
4	Defensa	pdf	145 KB	Universidad	1/11/2018	145 KB	1	3/11/2018	NULL
5	Constitucion	jpg	3.034 KB	Electivas	10/31/2019	5.083 KB	2	13/06/2019	NULL
5	Analisis	png	2.049 KB	Electivas	6/2/2018	5.083 KB	3	9/2/2018	NULL

Como se mostró el usuario queda en una tabla solo con el código, su nombre, correo y otros datos. Y en otra tabla queda el código que tiene cada usuario para un archivo, dejándolo en 1FN

0.4.4. 2NF

La segunda forma normal (2FN) se refiere a que en primer lugar tiene que cumplir con 1FN y en segundo lugar nos dice que es requisito que los campos, los

cuales dependen de una llave o son no-llave dependan de toda la clave primaria. Al hacer la normalización en nuestro proyecto aplicando la segunda regla habían unos campos o atributos los cuales dependen solo de las tablas simples, como la fecha de un archivo o la fecha de la carpeta, el peso, etc. Volviendo estas tablas dependientes a las claves primaria que tenemos.

id_archivo	nombreArchivo
123	Proyecto1
345	Proyecto2
567	Proyecto3
693	Entrega1
119	Entrega2
456	TrabajoCooperativo
789	Defensa
987	Constitucion
365	Analisis

nombreCarpeta	id_carpeta
ProyectoFinal	111
Fundamentos	222
Logica	698
Universidad	731
Electivas	942

Como se observa se deja la clave primaria siendo estas Id del usuario, código de la carpeta y del archivo, numero del rol y del Permiso con sus respectivos nombres, teniendo nuestras tablas simples con solo un atributo, realizando la 2FN.

0.4.5. 3NF

La tercera forma normal (3FN) se refiere a la eliminación de dependencias transitivas en las relaciones, logrando así crear una nueva relación. En nuestro proyecto contábamos con una dependencia transitiva debido a que un usuario se relaciona con los archivos y estos archivos se relacionan con la carpeta, dándonos por transitividad que la carpeta se relaciona con los usuarios, para esto creamos una tabla intermedia para solucionar este problema.

Usuario_carpeta	
Id_usuario	Codigo_carpeta
3	100
4	400
5	700
6	1000
7	300
8	200
9	500
10	900
1	300
2	400

La tabla que creamos fue la de Usuariocarpeta la cual hace que se relacionen y logremos la 3FN.

Haciendo el paso a paso de como seria el proceso de normalización, vimos como se iba convirtiendo las tablas, dándonos una visualización mas profunda de como quedarían las tablas normalizadas las cuales se muestran a continuación:

0.4.6. Creación Tablas

- Tablas Simples

  Se muestran las tablas simples, las cuales cuentan con su clave primaria y el nombre de esta.

```
--Tablas Simples-------------------------

CREATE TABLE Carpeta (
    codigo INTEGER PRIMARY KEY,
    nombre VARCHAR (100) NOT NULL
);

CREATE TABLE Archivo (
    codigo INTEGER PRIMARY KEY,
    nombre VARCHAR (100) NOT NULL
);

CREATE TABLE Rol (
    no_rol INTEGER PRIMARY KEY,
    Rol VARCHAR(100) NOT NULL
);

CREATE TABLE Permisos (
    no_permisos INTEGER PRIMARY KEY,
    Permisos VARCHAR(100) NOT NULL
);
```

- Tablas Dependientes

  Las tablas dependientes nos cuentan sobre los datos que tiene cada archivo y carpeta, además de tener la tabla Usuario la cual es la principal que va relacionada con roles, permisos y las tablas intermedias.

```sql
--Tablas Dependientes----------------------------------------
DROP TABLE IF EXISTS Usuario;
CREATE TABLE Usuario (
    Id INTEGER PRIMARY KEY,
    nombre VARCHAR (100) NOT NULL,
    correo VARCHAR (100) NOT NULL,
    clave_usuario INTEGER,
    archivo_guardados INTEGER NOT NULL,
    no_rol INTEGER REFERENCES Rol (no_rol),
    no_permisos INTEGER REFERENCES Permisos (no_permisos)
);

CREATE TABLE Datos_Archivo (
    clave_archivo INTEGER,
    tipo_archivo VARCHAR(100) NOT NULL,
    peso_archivo INTEGER NOT NULL,
    medida_almacenamiento VARCHAR(100) NOT NULL,
    fecha DATE,
    codigo_archivo INTEGER REFERENCES Archivo (codigo),
    codigo_carpeta INTEGER REFERENCES Carpeta (codigo)
);

CREATE TABLE Datos_carpeta (
    fecha Date,
    peso_carpeta INTEGER NOT NULL,
    medida_almacenamiento VARCHAR(100) NOT NULL,
    num_archivos INTEGER,
    codigo_carpeta INTEGER REFERENCES Carpeta (codigo)
);
```

- Tablas Intermedias

  Las tablas intermedias son las tablas que sacamos de una relación muchos a muchos, además de tener nuestra tabla de Usuariocarpeta que nos permite cumplir la 3FN quitando una dependencia transitiva.

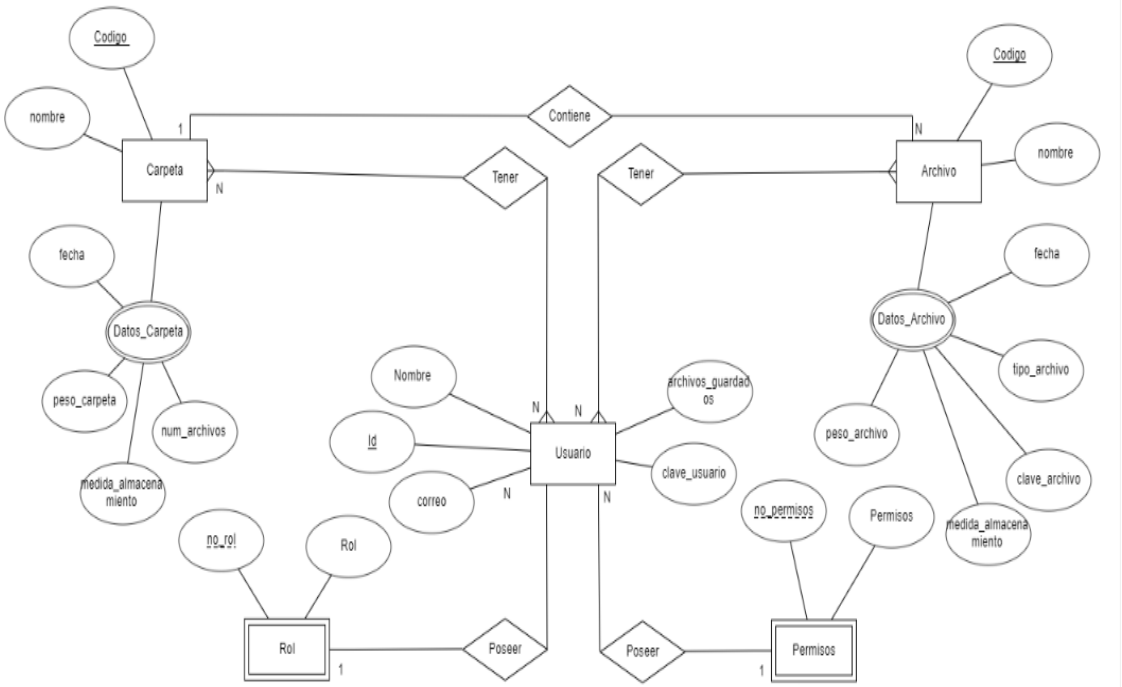
```
--Tablas Intermedias---------------------------------------

CREATE TABLE Usuario_archivo (
    Id_usuario INTEGER REFERENCES Usuario(Id),
    codigo_archivo INTEGER REFERENCES Archivo (codigo)
);

CREATE TABLE Usuario_carpeta (
    Id_usuario INTEGER REFERENCES Usuario(Id),
    codigo_carpeta INTEGER REFERENCES Carpeta (codigo)
);
```

0.5. Diseño conceptual de la base de datos

Luego de tener nuestras tablas normalizadas, seguimos con la construcción del diseño conceptual de nuestro proyecto, esto nos ayuda a tener un diagrama que nos permita conocer bien las relaciones y de conocer donde esta cada atributo y las relaciones así podemos hacer las consultas.

0.5.1. Diagrama Entidad Relacional

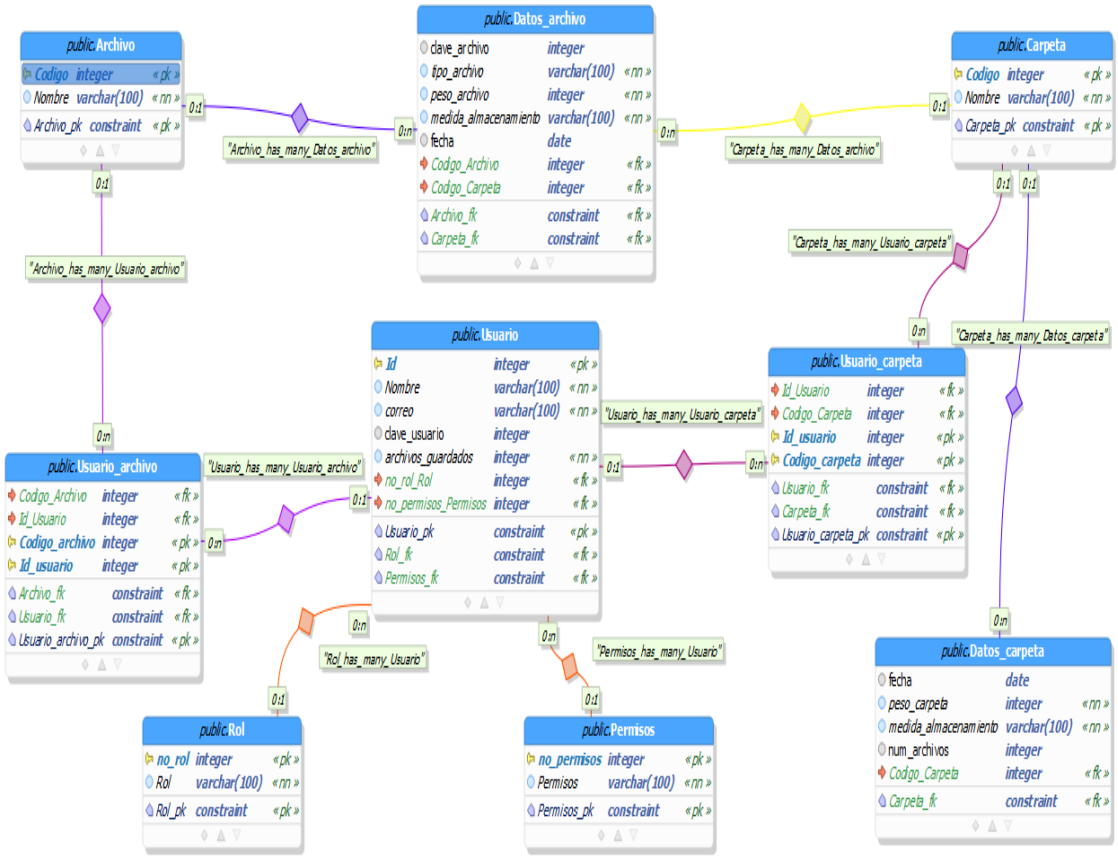
Un diagrama E/R (entidad-relación) es aquel que ilustra las entidades, atributos y relaciones. El diagrama que representa el nuestro, es el siguiente:



El diagrama de entidad relación, nos muestra que rol y permisos, son una entidad débil, que datos archivo y datos carpeta son atributos multivalor. Por otro lado, también, se pueden ver las relaciones existentes entre cada entidad y la relación que estos tienen como por ejemplo la entidad usuarios tiene una relación con permisos la cual es poseer y a su vez, la relación entre ellas de es de muchos a uno. Por último, el diagrama, también nos muestra las llaves únicas las cuales están subrayadas.

0.5.2. Diagrama Relacional

Un diagrama relacional es aquel modelo relacional que nos sirve para ver las tablas de datos y ver las relaciones que presenten. El diagrama que representa el nuestro, es el siguiente:



Con este diagrama, se nos facilita poder hacer las consultas para determinar las especificaciones al tener las relaciones y visualmente los atributos de cada entidad, logrando así poder unir las tablas y tener los resultados esperados.

0.6. Inserciones

En la parte de inserción de datos, metimos algunos manualmente, sobretodo los que tienen poquitas variables como la de Rol o Permisos y con los demás si hicimos carga masiva, ayudándonos de las tablas de excel a través de archivos CSV importamos los datos a nuestras tablas así se hacia automáticamente cargando mas archivos como se muestra a continuación:

```
COPY Archivo
FROM 'C:\Users\57320\Downloads\Archivo.csv'
DELIMITER ';'
CSV HEADER;

COPY Carpeta
FROM 'C:\Users\57320\Downloads\Carpeta.csv'
DELIMITER ';'
CSV HEADER;

COPY Usuario
FROM 'C:\Users\57320\Downloads\Usuario.csv'
DELIMITER ';'
CSV HEADER;

COPY Rol
FROM 'C:\Users\57320\Downloads\Rol.csv'
DELIMITER ';'
CSV HEADER;

COPY Permisos
FROM 'C:\Users\57320\Downloads\Permisos.csv'
DELIMITER ';'
CSV HEADER;
```

0.7. Consultas

En las consultas probamos que la creación de nuestras tablas y las inserciones estuvieran correctas, empezando a hacer consultas simples, medias y avanzadas las cuales se mostraran a continuación con lo que produce respectivamente:

0.7.1. Consultas Básicas

- La primera consulta que hicimos fue una básica mostrando todos los datos de la tabla Usuario

```
--Muestra todos los datos de usuario
SELECT*
FROM usuario U;
```

	id [PK] integer	nombre character varying (100)	correo character varying (100)	clave_usuario integer	archivo_guardados integer	no_rol integer	no_permisos integer
1	1	Lucía Ardila	lucia.ardila@gmail.com	123	3	1	1
2	2	Valentina Herrera	valentina.herrera@gmail.com	456	7	1	1
3	3	Cesar Segura	cesar.segura@gmail.com	6925	11	1	1
4	4	Sara Sánchez	sara.sanchez@gmail.com	[null]	1	2	2
5	5	Oscar Pérez	oscar.perez@gmail.com	[null]	70	4	4
6	6	Julieta Ponce	julieta.ponce@gmail.com	[null]	34	3	3
7	7	Adriana Hernández	adriana.hernandez@gmail.com	[null]	28	2	2
8	8	María Gonzales	maria.gonzales@gmail.com	[null]	90	3	3
9	9	Mauricio García	mauricio.garcia@gmail.com	934	23	1	1
10	10	Jose Fernández	jose.fernandez@gmail.com	[null]	56	4	4
11	11	Camilo Quintana	camilo.quintana@gmail.com	[null]	39	3	3
12	12	Sofia Castillo	sofia.castillo@gmail.com	[null]	78	2	2
13	13	Juan José Acosta	juanj.acosta@gmail.com	[null]	5	4	4
14	14	Sergio Quintero	sergio.quintero@gmail.com	789	100	1	1
15	15	Laura Camacho	laura.camacho@gmail.com	[null]	130	3	3
16	16	Jose Luis Peña	josel.peña@gmail.com	[null]	45	2	2
17	17	Juana Contreras	juana.contreras@gmail.com	[null]	143	3	3
18	18	Mariana Palacios	mariana.palacios@gmail.com	546	77	1	1
19	19	Carlos Salazar	carlos.salazar@gmail.com	7645	89	1	1
20	20	Sol Amaya	sol.amaya@gmail.com	989	2	1	1
21	21	karen Murcia	karen.murcia@gmail.com	34	83	1	1
22	22	Andrés Bernal	andres.bernal@gmail.com	[null]	20	2	2
23	23	Gabriela Torres	gabriela.torres@gmail.com	[null]	14	2	2
24	24	Sofia Echeverry	sofia.echeverry@gmail.com	[null]	21	3	3
25	25	Julian Diaz	julian.diaz@gmail.com	[null]	5	4	4
26	26	Juan Contreras	juan.contreras@gmail.com	111	60	1	1
27	27	Samuel Montoya	samuel.montoya@gmail.com	[null]	3	3	3
28	28	Santiago Marin	santiago.marin@gmail.com	[null]	20	2	2

0.7.2. Consultas usando JOINS

- Luego hicimos una consulta uniendo tablas, en este caso uniendo la tabla Usuario con la tabla Rol, queriendo mostrar el nombre del usuario con su respectivo Rol

```
--Muestra el rol que tiene cada usuario
SELECT U.nombre, R.rol
FROM Usuario U JOIN Rol R ON U.no_rol = R.no_rol;
```

	nombre character varying (100)	rol character varying (100)
1	Lucía Ardila	Administrador
2	Valentina Herrera	Administrador
3	Cesar Segura	Administrador
4	Sara Sánchez	Editor
5	Oscar Pérez	comentarista
6	Julieta Ponce	Lector
7	Adriana Hernández	Editor
8	María Gonzales	Lector
9	Mauricio García	Administrador
10	Jose Fernández	comentarista
11	Camilo Quintana	Lector
12	Sofia Castillo	Editor
13	Juan José Acosta	comentarista
14	Sergio Quintero	Administrador
15	Laura Camacho	Lector
16	Jose Luis Peña	Editor
17	Juana Contreras	Lector
18	Mariana Palacios	Administrador
19	Carlos Salazar	Administrador
20	Sol Amaya	Administrador
21	karen Murcia	Administrador
22	Andrés Bernal	Editor
23	Gabriela Torres	Editor
24	Sofia Echeverry	Lector
25	Julian Diaz	comentarista
26	Juan Contreras	Administrador
27	Samuel Montoya	Lector
28	Santiago Marin	Editor

- Mostramos a continuación el nombre del usuario junto con el número de archivos que tiene cada uno, para esto utilizamos los JOINS para poder unir las tablas así poder juntas estos dos atributos generando la subconsulta.

```
--Subconsulta para saber el num de archivos que tiene un usuario
SELECT A.nombre, B.num_archivos
FROM (SELECT*
     FROM Usuario u JOIN usuario_carpeta uc1 ON u.id = uc1.id_usuario) AS A
     JOIN
     (SELECT*
     FROM datos_carpeta dc JOIN carpeta c ON dc.codigo_carpeta = c.codigo) AS B
     ON A.codigo_carpeta = B.codigo;
```

	nombre character varying (100)	num_archivos integer
1	Valentina Herrera	3
2	Cesar Segura	3
3	María Gonzales	1
4	Lucía Ardila	2
5	Adriana Hernández	2
6	Juana Contreras	4
7	Laura Camacho	4
8	Sofia Castillo	4
9	Sara Sánchez	4
10	Jose Luis Peña	1
11	Sergio Quintero	1
12	Camilo Quintana	1
13	Mauricio García	1
14	Mariana Palacios	1
15	Juan José Acosta	1
16	Lucía Ardila	1
17	Oscar Pérez	2
18	Valentina Herrera	2
19	Jose Fernández	4
20	Julieta Ponce	1

- Tambien hicimos una consulta con Joins buscando que nos diera el código y nombre de los usuarios que actualizaron un archivo en el año 2022 y fuera de tipo cpp.

```
--Codigo y nombre de los usuarios que actualizacion un archivo en el año 2022 y era de tipo cpp

SELECT U.id, U.nombre
FROM Usuario U INNER JOIN Usuario_archivo UA ON (U.id = UA.Id_usuario)
               INNER JOIN Archivo A ON (A.codigo = UA.codigo_archivo)
               INNER JOIN Datos_archivo D ON (D.Codigo_Archivo = A.codigo)
where extract(year from D.fecha) = '2022' and D.tipo_archivo = 'cpp';
```

0.7.3. Vistas

- Ahora creamos vistas, en la primera vista, creamos una para conocer los administradores de la base de datos

```sql
--VISTAS-----------------------------------------
--muestra todos los admins de la base de datos
CREATE VIEW admins AS (
SELECT*
FROM Usuario U
WHERE no_rol = 1
);
-- materialización de la vista
SELECT*
FROM admins
```

	id integer	nombre character varying (100)	correo character varying (100)	clave_usuario integer	archivo_guardados integer	no_rol integer	no_permisos integer
1	1	Lucía Ardila	lucia.ardila@gmail.com	123	3	1	1
2	2	Valentina Herrera	valentina.herrera@gmail.com	456	7	1	1
3	3	Cesar Segura	cesar.segura@gmail.com	6925	11	1	1
4	9	Mauricio García	mauricio.garcia@gmail.com	934	23	1	1
5	14	Sergio Quintero	sergio.quintero@gmail.com	789	100	1	1
6	18	Mariana Palacios	mariana.palacios@gmail.com	546	77	1	1
7	19	Carlos Salazar	carlos.salazar@gmail.com	7645	89	1	1
8	20	Sol Amaya	sol.amaya@gmail.com	989	2	1	1
9	21	karen Murcia	karen.murcia@gmail.com	34	83	1	1
10	26	Juan Contreras	juan.contreras@gmail.com	111	60	1	1
11	39	Alejandra Serrano	alejandra.serrano@gmail.com	101	19	1	1
12	47	Camila Herrera	camila.herrera@gmail.com	569	15	1	1

- En la segunda creamos una vista para mostrar el peso que tiene una carpeta como se muestra a continuación:

```sql
--mostrar el peso de una carpeta
CREATE VIEW peso_carpeta AS(
SELECT C.nombre, DC.peso_carpeta
FROM Carpeta C JOIN Datos_carpeta DC ON C.codigo = DC.codigo_carpeta);

--materializando vista
SELECT*
FROM peso_carpeta;
```

	nombre character varying (100)	peso_carpeta integer
1	Descargas	20
2	Macc	11
3	Universidad	1
4	Colegio	258
5	Personal	31
6	Calculo	20
7	Fotos	140
8	Clase	8
9	2022	62
10	Defensa	26
11	Proyecto	140
12	Finales	1
13	Matematicas	62
14	Videos	258
15	Fotos	20
16	Descargas	62
17	Documentos	258
18	R	62
19	Python	258
20	Viaje	20

0.7.4. Funciones

- Creamos luego una función, la cual nos sirve para recuperar datos mas específicos de las tablas sacando en este caso la clave que tiene un usuario.

```
--FUNCTION-----------------------------------------------------------------
--Funcion que retorna la clave de un usuario
CREATE OR REPLACE FUNCTION retornarclave(nombre_usuario VARCHAR(45))
    RETURNS INTEGER
    LANGUAGE 'plpgsql'
AS $$
DECLARE
    clave INTEGER;
BEGIN
    clave := (SELECT clave_usuario
              FROM usuario
              WHERE nombre = $1);
    RETURN clave;
END;
$$
SELECT retornarclave('Carlos Salazar');
```

	retornarclave integer
1	7645

0.7.5. Triggers

- En esta parte creamos un trigger o disparador que nos permite conocer que es lo que ocurre cuando se presenta una acción, en nuestro caso creamos uno que diera una alerta cuando alguien inserta un dato en la tabla Archivo.

```
--Se tiene un Trigger que se dispara cuando alguien inserta una fila en la tabla Archivo
--Este registra en un log previamente creado, el codigo, su nombre y
--el usuario actual que hizo la insercion y el instante en que se hizo.

DROP TABLE IF EXISTS Log_Inserciones_Archivo;
CREATE TABLE Log_Inserciones_Archivo(
    codigo INTEGER,
    nombre VARCHAR (100),
    user_name VARCHAR (100),
    fecha TIMESTAMP WITHOUT TIME ZONE
);

CREATE OR REPLACE FUNCTION inserciones_Archivo()
    RETURNS TRIGGER
    LANGUAGE 'plpgsql'
AS $$
BEGIN
    INSERT INTO Log_Inserciones_Archivo(codigo, nombre, user_name, fecha)
        VALUES (NEW.codigo, NEW.nombre, CURRENT_USER, now());
    RETURN NEW;
END;
$$

CREATE TRIGGER registrar_Log_Inserciones_Archivo
    AFTER INSERT ON public.Archivo
    FOR EACH ROW
        EXECUTE FUNCTION inserciones_Archivo();


SELECT *
FROM Log_Inserciones_Archivo;
```

0.7.6. Roles, Usuarios y Privilegios

- Aquí designamos que algunos usuarios pudieran tener todos los privilegios, siendo estos de insertar, actualizar y eliminar.

```
--ROLES, USUARIOS Y PRIVILEGIOS-----------------------------------------
--admins
CREATE ROLE admin WITH
LOGIN;
CREATE USER Lucia_Ardila WITH
PASSWORD '123';
GRANT ALL PRIVILEGES ON DATABASE "Google Drive" TO lucia_Ardila;

CREATE USER Cesar_Segura WITH
PASSWORD '6925';
GRANT ALL PRIVILEGES ON DATABASE "Google Drive" TO Cesar_segura;

CREATE USER valentina_Herrera WITH
PASSWORD '456';
GRANT ALL PRIVILEGES ON DATABASE "Google Drive" TO valentina_Herrera;
```

0.7.7. Consultas Recursivas

- Para esta consulta designamos mostrar la jerarquía mostrando que van administradores, luego editores, comentaristas y de ultimas lectores, para esto creamos una columna para poder determinar los sub alternos de los administradores.

```
--RECURSIÓN-----------------------------------------------------------------------------------------------------
-- jerarquía: la jerarquía en la base de datos es de admin, editor, comentarista y por último lector
--creando una columna para de terminar los sub alternos de los admin
ALTER TABLE usuario
    ADD COLUMN administrador INTEGER;

--ingresando los datos
INSERT INTO usuario (id, nombre, correo, clave_usuario, archivo_guardados, no_rol, no_permisos,  administrador)
    VALUES (1 , 'Lucía Ardila'      , 'lucia.ardila@gmail.com'      , 123  , 3   , 1 , 1 , NULL),
           (2 , 'Valentina Herrera' , 'valentina.herrera@gmail.com' , 456  , 7   , 1 , 1 , NULL),
           (3 , 'Cesar Segura'      , 'cesar.segura@gmail.com'      , 6925 , 11  , 1 , 1 , NULL),
           (4 , 'Sara Sánchez'      , 'sara.sanchez@gmail.com'      , NULL , 1   , 2 , 2 , 1),
           (5 , 'Oscar Pérez'       , 'oscar.perez@gmail.com'       , NULL , 70  , 4 , 4 , 18),
           (6 , 'Julieta Ponce'     , 'julieta.ponce@gmail.com'     , NULL , 34  , 3 , 3 , 19),
           (7 , 'Adriana Hernández' , 'adriana.hernandez@gmail.com' , NULL , 28  , 2 , 2 , 18),
           (8 , 'María Gonzales'    , 'maria.gonzales@gmail.com'    , NULL , 90  , 3 , 3 , 21),
           (9 , 'Mauricio García'   , 'mauricio.garcia@gmail.com'   , 934  , 23  , 1 , 1 , NULL),
           (10, 'Jose Fernández'    , 'jose.fernandez@gmail.com'    , NULL , 56  , 4 , 4 , 19),
           (11, 'Camilo Quintana'   , 'camilo.quintana@gmail.com'   , NULL , 39  , 3 , 3 , 20),
           (12, 'Sofia Castillo'    , 'sofia.castillo@gmail.com'    , NULL , 78  , 2 , 2 , 14),
           (13, 'Juan José Acosta'  , 'juanj.acosta@gmail.com'      , NULL , 5   , 4 , 4 , 2),
           (14, 'Sergio Quintero'   , 'sergio.quintero@gmail.com'   , 789  , 100 , 1 , 1 , NULL),
           (15, 'Laura Camacho'     , 'laura.camacho@gmail.com'     , NULL , 130 , 3 , 3 , 1),
           (16, 'Jose Luis Peña'    , 'josel.peña@gmail.com'        , NULL , 45  , 2 , 2 , 2),
           (17, 'Juana Contreras'   , 'juana.contreras@gmail.com'   , NULL , 143 , 3 , 3 , 18),
           (18, 'Mariana Palacios'  , 'mariana.palacios@gmail.com'  , 546  , 77  , 1 , 1 , NULL),
           (19, 'Carlos Salazar'    , 'carlos.salazar@gmail.com'    , 7645 , 89  , 1 , 1 , NULL),
           (20, 'Sol Amaya'         , 'sol.amaya@gmail.com'         , 0989 , 2   , 1 , 1 , NULL),
           (21, 'karen Murcia'      , 'karen.murcia@gmail.com'      , 0034 , 83  , 1 , 1 , NULL),
           (22, 'Andrés Bernal'     , 'andres.bernal@gmail.com'     , NULL , 20  , 2 , 2 , NULL);
```

```
--creando función recursiva para mostrar los sub alternos de mariana palacios con codigo 18
WITH RECURSIVE sub_alternos AS(
  SELECT*
  FROM usuario
  WHERE administrador = 18
  UNION
  SELECT U.*
  FROM usuario U INNER JOIN sub_alternos S ON S.id = U.id
)
SELECT*
FROM sub_alternos
```

	id integer	nombre character varying (100)	correo character varying (100)	clave_usuario integer	archivo_guardados integer	no_rol integer	no_permisos integer	administrador integer
1	5	Oscar Pérez	oscar.perez@gmail.com	[null]	70	4	4	18
2	7	Adriana Hernández	adriana.hernandez@gmail.com	[null]	28	2	2	18
3	17	Juana Contreras	juana.contreras@gmail.com	[null]	143	3	3	18

0.7.8. Ranking

- Seguimos con las consultas creando un ranking el cual nos permite mirar de forma ascendente el número de permisos que tiene cada usuario.

```
--RANKING-----------------------------------------------------------------
--Mostrar de forma Ascendente del número de roles que tiene un ususario
SELECT U.nombre, P.no_permisos, DENSE_RANK() OVER (
       ORDER BY P.permisos ASC) AS ranking_permisos
FROM Usuario U JOIN permisos P ON U.no_permisos = P.no_permisos
```

	nombre character varying (100)	no_permisos integer	ranking_permisos bigint
1	Jose Fernández	4	1
2	Julian Diaz	4	1
3	Juliana Sopo	4	1
4	Juan José Acosta	4	1
5	Oscar Pérez	4	1
6	Santiago Peña	4	1
7	Manuela Tovar	4	1
8	Mariana Ramirez	4	1
9	Valentina Rojas	4	1
10	Susana Gonzales	2	2
11	Sara Sánchez	2	2
12	Adriana Hernández	2	2
13	Sofia Castillo	2	2
14	Jose Luis Peña	2	2
15	Andrés Bernal	2	2
16	Gabriela Torres	2	2
17	Santiago Marin	2	2
18	Laura Gamboa	2	2
19	Valentina Cantor	2	2
20	Sofia Sanchez	2	2
21	Julian Amador	2	2
22	Maria Comas	2	2
23	Paula Gomez	2	2
24	Sofia Echeverry	3	3
25	Camilo Quintana	3	3
26	Samuel Montoya	3	3
27	Anibal Velazquez	3	3

- La segunda consulta de ranking fue en crear un rango para mostrar el usuario que tiene la mayor cantidad de numero de archivos.

```
SELECT A.nombre, B.num_archivos, DENSE_RANK() OVER(
       ORDER BY B.num_archivos DESC) AS rankig_num_archivos
FROM (SELECT*
     FROM Usuario U JOIN Usuario_carpeta UA ON U.id = UA.id_usuario) AS A
     JOIN
     (SELECT*
     FROM carpeta C JOIN Datos_carpeta DC ON C.codigo = DC.codigo_carpeta) AS B
     ON A.codigo_carpeta = B.codigo
```

	nombre character varying (100)	num_archivos integer	rankig_num_archivos bigint
1	Laura Camacho	4	1
2	Jose Fernández	4	1
3	Sofia Castillo	4	1
4	Sara Sánchez	4	1
5	Juana Contreras	4	1
6	Valentina Herrera	3	2
7	Cesar Segura	3	2
8	Adriana Hernández	2	3
9	Valentina Herrera	2	3
10	Oscar Pérez	2	3
11	Lucía Ardila	2	3
12	Mariana Palacios	1	4
13	Julieta Ponce	1	4
14	María Gonzales	1	4
15	Mauricio García	1	4
16	Camilo Quintana	1	4
17	Juan José Acosta	1	4
18	Sergio Quintero	1	4
19	Jose Luis Peña	1	4
20	Lucía Ardila	1	4

0.7.9. Transacciones

- Creamos tambien unas consultas de transacciones las cuales representan cualquier cambio que se haga en la base de datos, proporcionando fiabilidad en la consulta además de poder recuperarse fácilmente ante algún error. En este caso nuestra transaccion nos indica el numero de archivos, es decir cuando un usuario elimina un archivo o sube 3 archivos, igual

cabe añadir que esto se puede modificar dependiendo de lo que ocurra con el usuario.

```
--TRANSACCIONES-------------------------------------
--transacción para el numero de archivos
BEGIN;
--si el usuario elimina un archivo
    UPDATE Usuario
    SET archivo_guardados = archivo_guardados - 1
    WHERE id = 3;
    --el usuario sube 3 archivos
    UPDATE Usuario
    SET archivo_guardados = archivo_guardados + 3
    WHERE id = 9;

--confirmar la transacción
COMMIT;

--Mostrar los datos de los usuarios
SELECT u.id, U.nombre, U.archivo_guardados
FROM usuario U;
```

	id [PK] integer	nombre character varying (100)	archivo_guardados integer
1	1	Lucía Ardila	3
2	2	Valentina Herrera	7
3	4	Sara Sánchez	1
4	5	Oscar Pérez	70
5	6	Julieta Ponce	34
6	7	Adriana Hernández	28
7	8	María Gonzales	90
8	10	Jose Fernández	56
9	11	Camilo Quintana	39
10	12	Sofia Castillo	78
11	13	Juan José Acosta	5
12	14	Sergio Quintero	100
13	15	Laura Camacho	130
14	16	Jose Luis Peña	45
15	17	Juana Contreras	143
16	18	Mariana Palacios	77
17	19	Carlos Salazar	89
18	20	Sol Amaya	2
19	21	karen Murcia	83
20	22	Andrés Bernal	20
21	23	Gabriela Torres	14
22	24	Sofia Echeverry	21
23	25	Julian Diaz	5
24	26	Juan Contreras	60
25	27	Samuel Montoya	3
26	28	Santiago Marin	20
27	29	Laura Garnboa	42
28	30	Valentina Cantor	1

0.8. Plotly

A través de Plotly la cual es un dashboard, donde podemos crear diagramas de barras, puntos, circulares que nos permiten tener una mejor visualización de las tablas de la base de datos. Para lograr esto, nos ayudamos del lenguaje de programación Python.

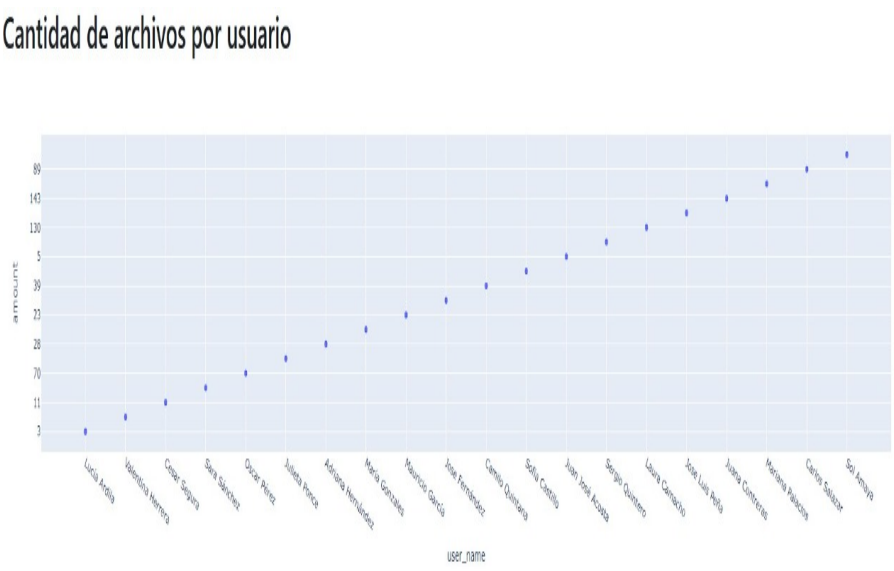
Esto lo realizamos a partir del siguiente código en Python:

```python
import psycopg2


class Connection:

    def __init__(self):
        self.connection = None

    def openConnection(self):
        try:
            self.connection = psycopg2.connect(user="postgres",
                                               password="123456789",
                                               database="Google Drive",
                                               host="localhost",
                                               port="5432")

        except Exception as e:
            print(e)

    def closeConnection(self):
        self.connection.close()
```

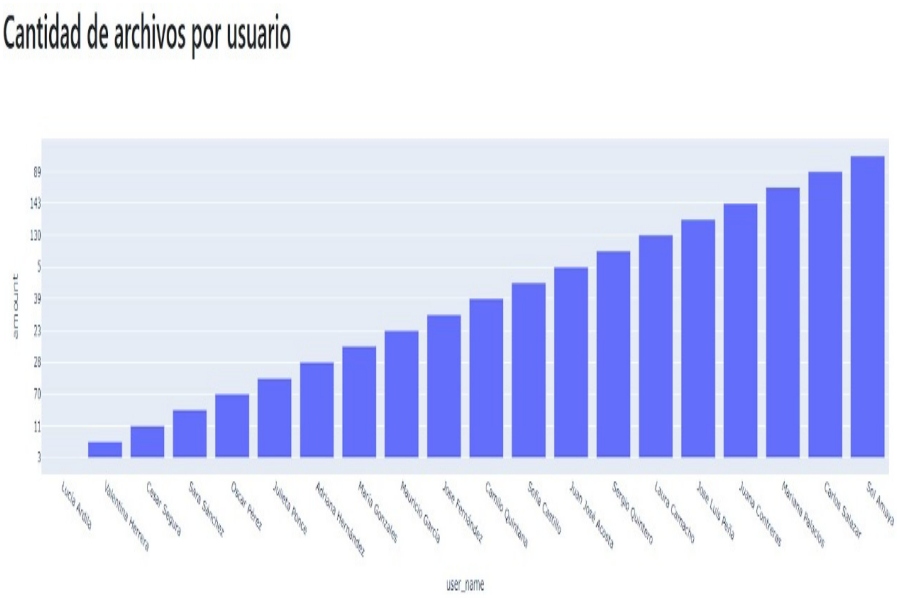
```python
import dash
from dash import dcc
from dash import html
import plotly.express as px
import pandas as pd
from conection import Connection
import conSQL as sql

external_stylesheets = ["https://cdn.jsdelivr.net/npm/bootstrap@5.0.0-beta3/dist/css/bootstrap.min.css"]

# Inicializacion app dash
app = dash.Dash(__name__, external_stylesheets=external_stylesheets)


# Casos por pais
con = Connection()
con.openConnection()
query = pd.read_sql_query(sql.IndArchivos(), con.connection)
con.closeConnection()
dfCases = pd.DataFrame(query, columns=["user_name", "amount"])

# Grafico barras
figBarCases = px.bar(dfCases.head(20), x="user_name", y="amount")

# Layout
app.layout = html.Div(children=[
    html.H1(children='Cantidad de archivos por usuario'),
    dcc.Graph(
        id='barArchporUsr',
        figure=figBarCases
    )
])

if __name__ == '__main__':
    app.run_server(debug=True)
```

```
app = Flask(__name__)

try:
    conn = psycopg2.connect(
        host = hostname,
        dbname = database,
        user = username,
        password = pwd,
        port = port_id)

    cur = conn.cursor()

    @app.route('/')
    def index():
        return render_template('index.html')

    @app.route('/new_user', methods={'POST'})
    def submit():
        if request.method == 'POST':
            user = request.form['usuario']
            space = request.form['correo']
            passwd = request.form['clave']
            if user == '' or space == '':
                return render_template('index.html', message='Por favor llenar los campos')
            else:
                insertion = 'INSERT INTO Usuario (nombre_usuario, correo, clave) VALUES (%s, %s, %s, %s)'
                values = [(user, space, passwd)]
                cur.execute(insertion,values)


    conn.close()

except Exception as error:
    print(error)

finally:
    if conn is not None:
        conn.close()

if __name__ == '__main__':
    app.run()
```

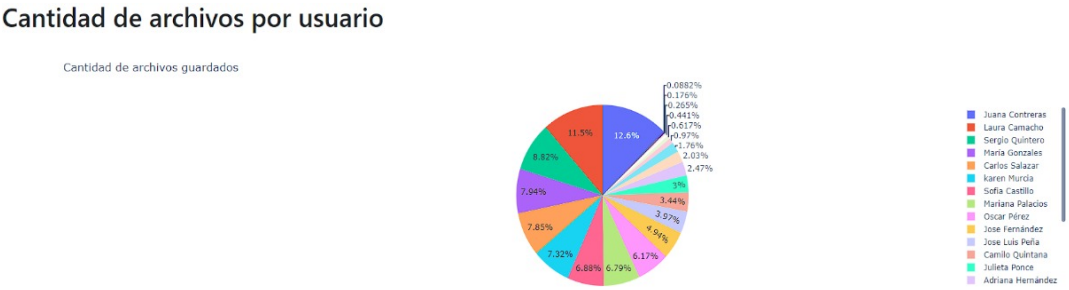
- La primera gráfica que construimos fue una de puntos la cual nos muestra la cantidad de archivos que tiene cada usuario



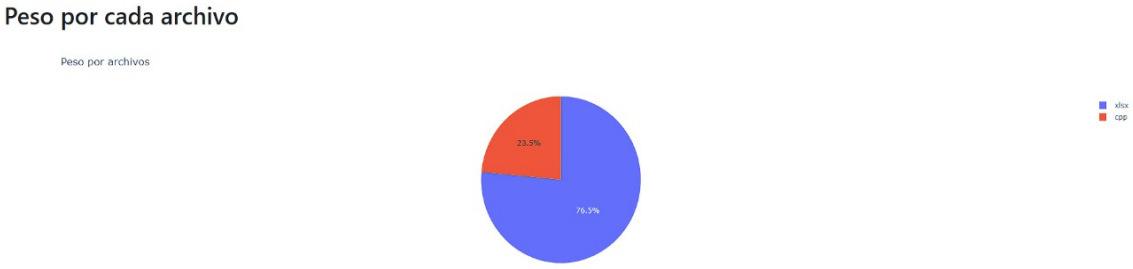
- También quisimos ver la cantidad de archivos que tienen los usuarios pero con un diagrama de barras complementando la visualización de nuestro primer diagrama



- También quisimos ver la cantidad de archivos que tienen los usuarios pero con un diagrama de barras complementando la visualización de nuestro primer diagrama



- En este gráfico se puede observar la cantidad de archivos que un usuario posee y la cantidad de espacio que estos ocupan tanto en tamaño como porcentualmente.


Peso por cada archivo
Peso por archivos
23.5%
76.5%
xlsx
cpp