

Base de datos para la gestión de peticiones, quejas y reclamos (PQR) en empresas prestadoras de salud (EPS) en Colombia.

Sebastian Moncada A.¹, Johan Sanchez R.²

¹⁻²Facultad de Ingeniería y Ciencias Basicas,
Universidad Central

Maestria en Analítica de Datos

Curso de Bases de Datos

Bogotá, Colombia

¹jmoncadaa@ucentral.edu.com, ²jsanchezr26@ucentral.edu.co

November 25, 2023

Contents

1	Introducción	3
2	Características del proyecto	3
2.1	Titulo del proyecto de investigación	3
2.2	Objetivo general	3
2.2.1	Objetivos especificos	3
2.3	Alcance	4
2.4	Pregunta de investigación	4
2.5	Hipotesis	5
3	Reflexiones sobre el origen de datos e información	6
3.1	¿Cual es el origen de los datos e información?	6
3.2	¿Cuales son las consideraciones legales o eticas del uso de la información?	6
3.3	¿Cuales son los retos de la información y los datos que utilizara en la base de datos en terminos de la calidad y la consolidación?	7
3.4	¿Que espera de la utilización de un sistema de Bases de Datos para su proyecto?	7

4	Diseño del Modelo de Datos del SMBD (Sistema Manejador de Bases de Datos)	8
4.1	Características del SMBD (Sistema Manejador de Bases de Datos) para el proyecto	8
4.2	Diagrama modelo de datos	8
4.3	Imágenes de la Base de Datos	8
4.4	Código SQL - lenguaje de definición de datos (DDL)	10
4.5	Código SQL - Manipulación de datos (DML)	12
4.6	Código SQL + Resultados: Vistas	14
4.7	Código SQL + Resultados: Triggers	15
4.8	Código SQL + Resultados: Funciones	15
4.9	Código SQL + Resultados: procedimientos almacenados	16
5	Bases de Datos No-SQL	17
5.1	Diagrama Bases de Datos No-SQL	17
5.2	SMBD utilizado para la Base de Datos No-SQL	18
5.3	Conclusiones modelos de datos	27
6	Aplicación de ETL (Extract, Transform, Load) y Bodega de Datos (<i>Tercera entrega</i>)	28
6.1	Ejemplo de aplicación de ETL y Bodega de Datos (<i>Tercera entrega</i>)	28
6.2	Automatización de Datos (<i>Tercera entrega</i>)	28
6.3	Integración de Datos (<i>Tercera entrega</i>)	28
7	Proximos pasos	30
8	Lecciones aprendidas	31
9	Bibliografía	32

1 Introducción

Las Empresas Promotoras de Salud (EPS) desempeñan un papel crucial en el ecosistema de la atención médica, actuando como intermediarios entre los pacientes y los proveedores de servicios de salud. En este contexto, la gestión eficaz de Peticiones, Quejas y Reclamos (PQR) se ha convertido en un pilar esencial para garantizar la satisfacción de los usuarios y la mejora constante de los servicios de salud.

Este proyecto se centra en abordar la imperante necesidad de optimizar la gestión de PQR en las EPS mediante la creación de una base de datos integral. Esta base de datos no solo funcionará como un almacén de información, sino como un sistema completo que permitirá un seguimiento minucioso de cada caso, desde su registro inicial hasta su resolución. Este enfoque no solo agilizará los procesos internos de las EPS, sino que también facilitará la toma de decisiones informadas al proporcionar una visión completa de las preocupaciones de los usuarios.

El proyecto abordará diversos aspectos técnicos esenciales, como el diseño meticuloso de la base de datos, la definición de relaciones entre tablas, entre otros. La seguridad de los datos ocupará un lugar prioritario en la planificación, garantizando la confidencialidad y la integridad de la información de los usuarios. A lo largo de este informe, se describirán los pasos necesarios para llevar a cabo este proyecto, las tecnologías a implementar, los beneficios esperados y su importancia en el contexto en constante evolución de la atención médica.

2 Características del proyecto

2.1 Título del proyecto de investigación

Proyecto de Bases de datos para la gestión de peticiones, quejas y reclamos (PQR) mediante NLP en empresas prestadoras de salud (EPS) en Colombia.

2.2 Objetivo general

Este proyecto tiene como objetivo principal la creación y desarrollo de una base de datos integral y eficiente para la gestión de las PQR presentadas por los usuarios de las EPS. La implementación de esta base de datos permitirá una administración más ágil y efectiva para el procesamiento de lenguaje natural que permitirá clasificar dichas PQRS y así contribuir a la mejora continua de los servicios de salud ofrecidos.

2.2.1 Objetivos específicos

- Definir una arquitectura de base de datos que permita un almacenamiento organizado y rápido acceso a la información de PQR de las EPS.

- Crear una base de datos que pueda adaptarse a cambios futuros en la gestión de PQR y a un aumento en la cantidad de datos sin comprometer el rendimiento.
- facilitar la recopilación, búsqueda y análisis de PQR para una gestión más rápida y efectiva de los casos, lo que conducirá a una atención al cliente mejorada.
- Facilitar la identificación de patrones y tendencias en las PQR, lo que permitirá a las EPS tomar medidas proactivas para resolver problemas y mejorar la calidad de la atención médica.

2.3 Alcance

Este proyecto se concentra en el diseño exhaustivo de la estructura de la base de datos para la gestión de Peticiones, Quejas y Reclamos (PQR) en las Empresas Promotoras de Salud (EPS). Su alcance abarca los siguientes aspectos clave:

- Diseño de la Base de Datos: Se desarrollará un diseño completo de la base de datos que comprende la creación de tablas, definición de campos y atributos, y establecimiento de relaciones entre tablas para capturar y organizar de manera eficiente la información relacionada con las PQR.
- Identificación de Campos Relevantes: Se definirán los campos esenciales para registrar datos cruciales, como la información del usuario (nombre, contacto), la fecha de presentación de la PQR, la categoría o tipo de solicitud, el estado de resolución y cualquier otra información relevante.
- Normalización de Datos: Se aplicarán principios de normalización de bases de datos para evitar la redundancia de datos y garantizar la integridad y coherencia de la información almacenada.
- Especificación de Restricciones de Integridad: Se establecerán restricciones de integridad para mantener la coherencia de los datos, como claves primarias, foráneas y restricciones de unicidad.
- Documentación del Diseño: Se proporcionará una documentación detallada que describe la estructura de la base de datos, las tablas, campos, relaciones y restricciones, facilitando la comprensión y el mantenimiento del sistema en el futuro.

2.4 Pregunta de investigación

¿Cual es el impacto de la implementación de una base de datos eficiente en la gestión de para su posterior procesamiento con NLP?

2.5 Hipotesis

La implementación exitosa de una base de datos específica para la gestión de PQR en las EPS conlleva una reducción significativa en los tiempos de respuesta a las solicitudes de los usuarios.

Esta hipótesis parte de la premisa de que al implementar una base de datos diseñada especialmente para la gestión de PQR en las EPS, se pueden agilizar los procesos de atención a los usuarios. La lógica subyacente es que la base de datos proporcionará a los empleados de las EPS un acceso más rápido a la información relevante, lo que les permitirá responder a las solicitudes de los usuarios de manera más eficiente. Además, al mejorar la organización de los datos y su accesibilidad, se espera que se reduzcan los tiempos de búsqueda y procesamiento, lo que tendrá un impacto positivo en la satisfacción de los usuarios, quienes experimentarán una atención más oportuna y efectiva.

3 Reflexiones sobre el origen de datos e información

En el ámbito de la atención médica, la gestión eficiente de las PQRS (Peticiones, Quejas, Reclamos y Sugerencias) es esencial para garantizar la satisfacción de los afiliados y mantener la calidad de los servicios. En este contexto, los datos utilizados para evaluar y mejorar la atención provienen de un histórico de PQRS que ha sido recibido por una EPS.

Es importante destacar que las PQRS pueden variar dependiendo de la ciudad o región a la que pertenezcan los afiliados. Esta variabilidad geográfica se debe a una serie de factores, incluyendo las condiciones de salud prevalentes en la zona, la disponibilidad de recursos médicos y las particularidades socioeconómicas de la población local. Por lo tanto, una comprensión precisa de la geolocalización de las PQRS es esencial para abordarlas de manera efectiva.

Entre las PQRS más comunes, destacan las relacionadas con la necesidad de oxígeno o la solicitud de atención médica domiciliaria. Estas solicitudes son críticas, ya que a menudo están vinculadas a la salud y el bienestar de los afiliados. La respuesta oportuna y adecuada a tales solicitudes es fundamental para garantizar la seguridad y el cuidado de los pacientes.

La gestión efectiva de las PQRS no solo es un requisito para brindar una atención de calidad, sino que también tiene implicaciones legales. Si no se atienden de manera adecuada y oportuna, las PQRS pueden escalar a tutelas, un proceso legal que permite a los afiliados buscar la protección de sus derechos a través de la intervención de una autoridad judicial. Esto no solo puede ser costoso y llevar tiempo, sino que también puede dañar la reputación de la EPS y socavar la confianza de los afiliados.

3.1 ¿Cual es el origen de los datos e información?

Los datos anonimizados se generaron sintéticamente a partir de PQRS encontradas en línea, que simulan quejas de usuarios sobre servicios de una EPS. Además, se incorporó información demográfica para mejorar la clasificación demográfica de las PQRS y optimizar su análisis.

3.2 ¿Cuales son las consideraciones legales o eticas del uso de la información?

Para salvaguardar la privacidad y cumplir con las regulaciones legales, ciertos registros han sido anonimizados en este conjunto de datos, que incluye información de personas que solicitan servicios. Además, gran parte de los datos ha sido generada sintéticamente y adaptada para los fines del proyecto. Es esencial respetar la confidencialidad de esta información y evitar su divulgación.

3.3 ¿Cuales son los retos de la información y los datos que utilizara en la base de datos en terminos de la calidad y la consolidación?

El desafío incluye la tarea crucial de mapear y validar la información antes de cargarla en la base de datos. Esto es fundamental para prevenir problemas de rendimiento y lectura, especialmente cuando se enfrentan caracteres especiales o codificaciones incompatibles. Además, debemos mantener la confidencialidad de los datos en todo momento.

3.4 ¿Que espera de la utilización de un sistema de Bases de Datos para su proyecto?

Se anticipa una mejora significativa en la organización de la información en el sistema de la base de datos, lo que permitirá consultas más eficientes gracias a la implementación de un modelo relacional adecuado. Además, esta optimización allana el camino hacia la posibilidad de una integración web para el consumo y la visualización más accesible de las PQRS.

4 Diseño del Modelo de Datos del SMBD (Sistema Manejador de Bases de Datos)

4.1 Características del SMBD (Sistema Manejador de Bases de Datos) para el proyecto

Nuestra base de datos está diseñada para gestionar una alta carga transaccional en tiempo real, ya que se espera que reciba una gran variedad de consultas. Además, se debe garantizar que el proceso de carga masiva de información, independientemente de la cantidad, se realice de manera eficiente para que la información esté disponible en el menor tiempo posible.

Por estas razones, hemos optado por utilizar PostgreSQL como nuestro Sistema de Gestión de Bases de Datos (SMBD). PostgreSQL ha demostrado ser una elección sólida que satisface las necesidades mencionadas anteriormente. Además, su reputación en la industria y el constante respaldo de la comunidad lo han posicionado como uno de los SMBD más populares y confiables. Esta elección se basa en la capacidad de PostgreSQL para brindar un rendimiento óptimo tanto en transacciones en tiempo real como en cargas masivas de datos, lo que garantiza que nuestra base de datos cumpla con los estándares más exigentes en términos de disponibilidad y eficiencia.

4.2 Diagrama modelo de datos

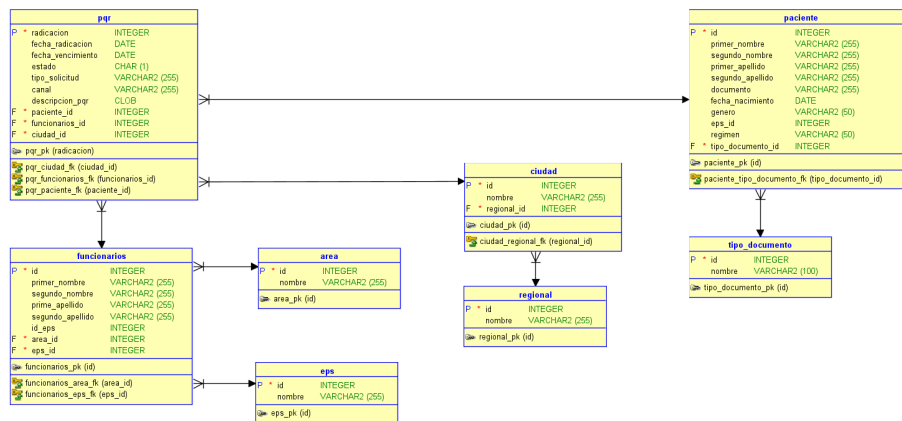


Figure 1: Modelo entidad relación.

4.3 Imágenes de la Base de Datos

Después de llevar a cabo el proceso de construcción de la base de datos y de realizar las operaciones de inserción de datos, hemos obtenido resultados

sumamente prometedores que arrojan luz sobre la eficacia y el potencial del proyecto. Este esfuerzo significativo ha dado lugar a una serie de hallazgos clave.

En cuanto a los resultados de las operaciones de inserción, podemos afirmar con satisfacción que se han completado de manera exitosa y sin errores significativos. Esto demuestra la eficiencia de nuestras metodologías y la capacidad de nuestro equipo para llevar a cabo tareas complejas de gestión de datos.

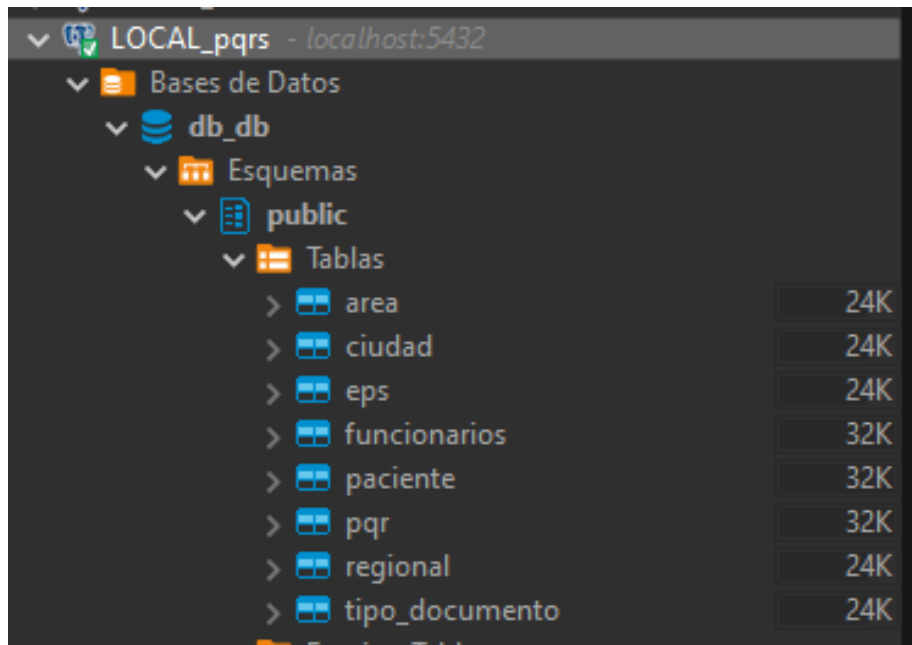


Tabla	Tamaño
area	24K
ciudad	24K
eps	24K
funcionarios	32K
paciente	32K
pqr	32K
regional	24K
tipo_documento	24K

Figure 2: Lista de tablas.

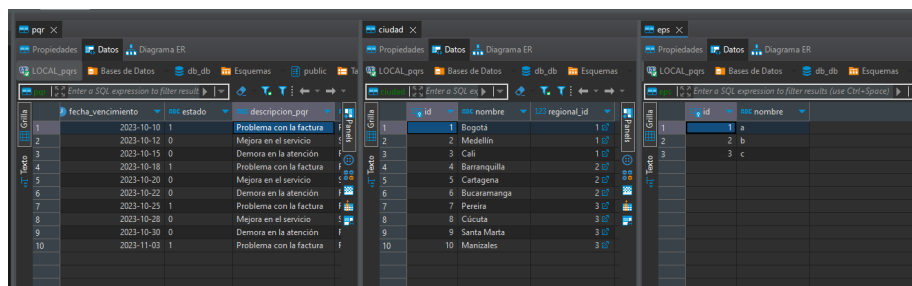


Tabla	ID	Nombre	regional_id
pqr	1	Problema con la factura	1
	2	Mejora en el servicio	1
	3	Demora en la atención	1
	4	Problema con la factura	2
	5	Mejora en el servicio	2
	6	Demora en la atención	2
	7	Problema con la factura	3
	8	Mejora en el servicio	3
	9	Demora en la atención	3
	10	Problema con la factura	3
ciudad	1	Bogotá	1
	2	Medellín	1
	3	Cali	1
	4	Barranquilla	2
	5	Cartagena	2
	6	Bucaramanga	2
	7	Pereira	3
	8	Cúcuta	3
	9	Santa Marta	3
	10	Manizales	3
eps	1	a	
	2	b	
	3	c	

Figure 3: Algunas tablas con datos.

4.4 Código SQL - lenguaje de definición de datos (DDL)

```
CREATE TABLE area (  
    id      INTEGER NOT NULL,  
    nombre  VARCHAR2(255)  
)  
ALTER TABLE area ADD CONSTRAINT area_pk PRIMARY KEY ( id );
```

```
CREATE TABLE ciudad (  
    id          INTEGER NOT NULL,  
    nombre      VARCHAR2(255),  
    regional_id INTEGER NOT NULL  
)  
ALTER TABLE ciudad ADD CONSTRAINT ciudad_pk PRIMARY KEY ( id );
```

```
CREATE TABLE eps (  
    id      INTEGER NOT NULL,  
    nombre  VARCHAR2(255)  
)  
ALTER TABLE eps ADD CONSTRAINT eps_pk PRIMARY KEY ( id );
```

```
CREATE TABLE funcionarios (  
    id              INTEGER NOT NULL,  
    primer_nombre   VARCHAR2(255),  
    segundo_nombre  VARCHAR2(255),  
    prime_apellido  VARCHAR2(255),  
    segundo_apellido VARCHAR2(255),  
    id_eps          INTEGER,  
    area_id         INTEGER NOT NULL,  
    eps_id          INTEGER NOT NULL  
)  
ALTER TABLE funcionarios ADD CONSTRAINT funcionarios_pk PRIMARY KEY ( id );
```

```
CREATE TABLE paciente (  
    id              INTEGER NOT NULL,  
    primer_nombre   VARCHAR2(255),  
    segundo_nombre  VARCHAR2(255),  
    primer_apellido VARCHAR2(255),  
    segundo_apellido VARCHAR2(255),  
    documento       VARCHAR2(255),  
    fecha_nacimiento DATE,
```

```

        genero          VARCHAR2(50),
        eps_id          INTEGER,
        regimen         VARCHAR2(50),
        tipo_documento_id INTEGER NOT NULL
    )
ALTER TABLE paciente ADD CONSTRAINT paciente_pk PRIMARY KEY ( id );

CREATE TABLE pqr (
    radicacion          INTEGER NOT NULL,
    fecha_radicacion    DATE,
    fecha_vencimiento    DATE,
    estado              CHAR(1),
    tipo_solicitud       VARCHAR2(255),
    canal               VARCHAR2(255),
    descripcion_pqr      CLOB,
    paciente_id          INTEGER NOT NULL,
    funcionarios_id      INTEGER NOT NULL,
    ciudad_id            INTEGER NOT NULL
)
ALTER TABLE pqr ADD CONSTRAINT pqr_pk PRIMARY KEY ( radicacion );

CREATE TABLE regional (
    id      INTEGER NOT NULL,
    nombre  VARCHAR2(255)
)
ALTER TABLE regional ADD CONSTRAINT regional_pk PRIMARY KEY ( id );

CREATE TABLE tipo_documento (
    id      INTEGER NOT NULL,
    nombre  VARCHAR2(100)
)
ALTER TABLE tipo_documento ADD CONSTRAINT tipo_documento_pk PRIMARY KEY ( id );

ALTER TABLE ciudad
    ADD CONSTRAINT ciudad_regional_fk FOREIGN KEY ( regional_id )
        REFERENCES regional ( id )
        NOT DEFERRABLE;

ALTER TABLE funcionarios
    ADD CONSTRAINT funcionarios_area_fk FOREIGN KEY ( area_id )
        REFERENCES area ( id )

```

```

NOT DEFERRABLE;

ALTER TABLE funcionarios
  ADD CONSTRAINT funcionarios_eps_fk FOREIGN KEY ( eps_id )
    REFERENCES eps ( id )
  NOT DEFERRABLE;

ALTER TABLE paciente
  ADD CONSTRAINT paciente_tipo_documento_fk FOREIGN KEY ( tipo_documento_id )
    REFERENCES tipo_documento ( id )
  NOT DEFERRABLE;

ALTER TABLE pqr
  ADD CONSTRAINT pqr_ciudad_fk FOREIGN KEY ( ciudad_id )
    REFERENCES ciudad ( id )
  NOT DEFERRABLE;

ALTER TABLE pqr
  ADD CONSTRAINT pqr_funcionarios_fk FOREIGN KEY ( funcionarios_id )
    REFERENCES funcionarios ( id )
  NOT DEFERRABLE;

ALTER TABLE pqr
  ADD CONSTRAINT pqr_paciente_fk FOREIGN KEY ( paciente_id )
    REFERENCES paciente ( id )
  NOT DEFERRABLE;

```

4.5 Código SQL - Manipulación de datos (DML)

```

--Insert Eps
INSERT INTO public.eps (id,nombre) VALUES (1,'Compensar')
INSERT INTO public.eps (id,nombre) VALUES (2,'Sanitas')
INSERT INTO public.eps (id,nombre) VALUES (3,'Nueva EPS')

--Insert area
INSERT INTO public.area (id,nombre) VALUES (1,'Atencion al cliente')
INSERT INTO public.area (id,nombre) VALUES (2,'Medicina')
INSERT INTO public.area (id,nombre) VALUES (3,'Administrativa')

--Insert document-type
INSERT INTO public.tipo_documento (id,nombre) VALUES (1,'CC')
INSERT INTO public.tipo_documento (id,nombre) VALUES (2,'TI')
INSERT INTO public.tipo_documento (id,nombre) VALUES (3,'PA')

--Regional

```

```

INSERT INTO public.regional (id,nombre) VALUES (1,'Bogota')
INSERT INTO public.regional (id,nombre) VALUES (2,'Barranquilla')
INSERT INTO public.regional (id,nombre) VALUES (3,'Sur Occidente')

--Ciudades
INSERT INTO public.ciudad (id,nombre,regional_id) VALUES (1,'Bogotá',1)
INSERT INTO public.ciudad (id,nombre,regional_id) VALUES (2,'Medellín',1)
INSERT INTO public.ciudad (id,nombre,regional_id) VALUES (3,'Cali',1)
INSERT INTO public.ciudad (id,nombre,regional_id) VALUES (4,'Barranquilla',2)
INSERT INTO public.ciudad (id,nombre,regional_id) VALUES (5,'Cartagena',2)
INSERT INTO public.ciudad (id,nombre,regional_id) VALUES (6,'Bucaramanga',2)
INSERT INTO public.ciudad (id,nombre,regional_id) VALUES (7,'Pereira',3)
INSERT INTO public.ciudad (id,nombre,regional_id) VALUES (8,'Cúcuta',3)
INSERT INTO public.ciudad (id,nombre,regional_id) VALUES (10,'Manizales',3)

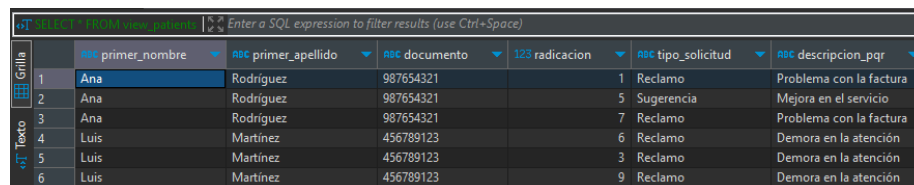
--Funcionarios
INSERT INTO funcionarios (id, primer_nombre, segundo_nombre,
prime_apellido, segundo_apellido, area_id, eps_id)
VALUES (1, 'Juan', 'Carlos', 'Gómez', 'López', 1, 1);
INSERT INTO funcionarios (id, primer_nombre, segundo_nombre,
prime_apellido, segundo_apellido, area_id, eps_id)
VALUES (2, 'Ana', 'María', 'Rodríguez', 'Pérez', 2, 1);
INSERT INTO funcionarios (id, primer_nombre, segundo_nombre,
prime_apellido, segundo_apellido, area_id, eps_id)
VALUES (3, 'Luis', 'Felipe', 'Martínez', 'García', 1, 2);
INSERT INTO funcionarios (id, primer_nombre, segundo_nombre,
prime_apellido, segundo_apellido, area_id, eps_id)
VALUES (4, 'María', 'Isabel', 'Fernández', 'Ramírez', 2, 2);

--Pacientes
INSERT INTO paciente (id, primer_nombre, segundo_nombre, primer_apellido,
segundo_apellido, documento, fecha_nacimiento, genero, eps_id, regimen,
tipo_documento_id)
VALUES (1, 'Juan', 'Carlos', 'Gómez', 'López', '123456789', '1990-05-15',
'Masculino', 1, 'Contributivo', 1);
INSERT INTO paciente (id, primer_nombre, segundo_nombre, primer_apellido,
segundo_apellido, documento, fecha_nacimiento, genero, eps_id, regimen,
tipo_documento_id)
VALUES (2, 'Ana', 'María', 'Rodríguez', 'Pérez', '987654321', '1985-08-20',
'Femenino', 2, 'Subsidiado', 2);
INSERT INTO paciente (id, primer_nombre, segundo_nombre, primer_apellido,
segundo_apellido, documento, fecha_nacimiento, genero, eps_id, regimen,
tipo_documento_id)
VALUES (3, 'Luis', 'Felipe', 'Martínez', 'García', '456789123', '1992-03-10',
'Masculino', 3, 'Contributivo', 3);

```

```
--PQRS
INSERT INTO pqr (radicacion, fecha_radicacion, fecha_vencimiento, estado,
tipo_solicitud, canal, descripcion_pqr, paciente_id, funcionarios_id, ciudad_id)
VALUES (1, '2023-09-10', '2023-10-10', 1, 'Reclamo', 'Correo',
'Problema con la factura', 2, 1, 3);
INSERT INTO pqr (radicacion, fecha_radicacion, fecha_vencimiento, estado,
tipo_solicitud, canal, descripcion_pqr, paciente_id, funcionarios_id, ciudad_id)
VALUES (2, '2023-09-12', '2023-10-12', 0, 'Sugerencia', 'Teléfono',
'Mejora en el servicio', 1, 3, 2);
INSERT INTO pqr (radicacion, fecha_radicacion, fecha_vencimiento, estado,
tipo_solicitud, canal, descripcion_pqr, paciente_id, funcionarios_id, ciudad_id)
VALUES (3, '2023-09-15', '2023-10-15', 0, 'Reclamo',
'Chat en Línea', 'Demora en la atención', 3, 2, 1);
INSERT INTO pqr (radicacion, fecha_radicacion, fecha_vencimiento, estado,
tipo_solicitud, canal, descripcion_pqr, paciente_id, funcionarios_id, ciudad_id)
VALUES (4, '2023-09-18', '2023-10-18', 1, 'Reclamo', 'Correo',
'Problema con la factura', 1, 1, 3);
```

4.6 Código SQL + Resultados: Vistas



	primer_nombre	primer_apellido	documento	radicacion	tipo_solicitud	descripcion_pqr
1	Ana	Rodriguez	987654321	1	Reclamo	Problema con la factura
2	Ana	Rodriguez	987654321	5	Sugerencia	Mejora en el servicio
3	Ana	Rodriguez	987654321	7	Reclamo	Problema con la factura
4	Luis	Martinez	456789123	6	Reclamo	Demora en la atención
5	Luis	Martinez	456789123	3	Reclamo	Demora en la atención
6	Luis	Martinez	456789123	9	Reclamo	Demora en la atención

Figure 4: Resultado View.

```
create or replace
view view_patients as
select
p.primer_nombre,
p.primer_apellido,
p.documento,
p2.radicacion,
p2.tipo_solicitud,
p2.descripcion_pqr
from
```

```

paciente p
inner join pqr p2 on
p2.paciente_id = p.id
order by
p.primer_nombre

SELECT * FROM view_patients;

```

4.7 Código SQL + Resultados: Triggers

```

CREATE TRIGGER insert_eps
AFTER INSERT ON eps
FOR EACH ROW
EXECUTE FUNCTION create_eps_funcionario(1);

```

4.8 Código SQL + Resultados: Funciones

```

CREATE OR REPLACE FUNCTION resolve_pqr(id_pqr integer)
RETURNS integer AS $$
BEGIN
    UPDATE public.pqr set estado = 1 where id = id_pqr;
END;
$$ LANGUAGE plpgsql;

```

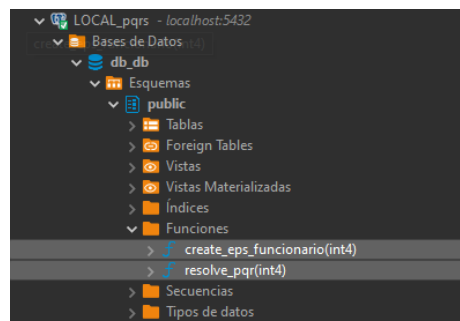


Figure 5: Resultado functions.

```

CREATE OR REPLACE FUNCTION create_eps_funcionario(id_eps integer)
RETURNS integer AS $$
BEGIN

```

```

        INSERT INTO funcionarios (id, primer_nombre, segundo_nombre, prime_apellido,
        segundo_apellido, area_id, eps_id)
VALUES (1, 'Lider', '', 'EPS', '', 1, 1);
END;
$$ LANGUAGE plpgsql;

```

4.9 Código SQL + Resultados: procedimientos almacenados

```

CREATE OR REPLACE FUNCTION pqrs_by_employee(employee_id INT)
RETURNS TABLE (
    id INT,
    nombre VARCHAR(255),
    pqr VARCHAR(255)
) AS $$
BEGIN
    RETURN QUERY
    SELECT f.ID, f.Nombre, pqr.PQR
    FROM funcionario f
    INNER JOIN pqr ON pqr.funcionario_id = f.id
    WHERE f.ID = employee_id;
END;
$$ LANGUAGE plpgsql;

```

Name	Value
Updated Rows	0
Query	<pre> CREATE OR REPLACE FUNCTION pqrs_by_employee(employee_id INT) RETURNS TABLE (id INT, nombre VARCHAR(255), pqr VARCHAR(255)) AS \$\$ BEGIN RETURN QUERY SELECT f.ID, f.Nombre, pqr.PQR FROM funcionario f INNER JOIN pqr ON pqr.funcionario_id = f.id WHERE f.ID = employee_id; END; \$\$ LANGUAGE plpgsql </pre>
Start time	Sat Oct 07 08:56:40 COT 2023

Figure 6: Resultado procedimiento almacenado.

5 Bases de Datos No-SQL

5.1 Diagrama Bases de Datos No-SQL

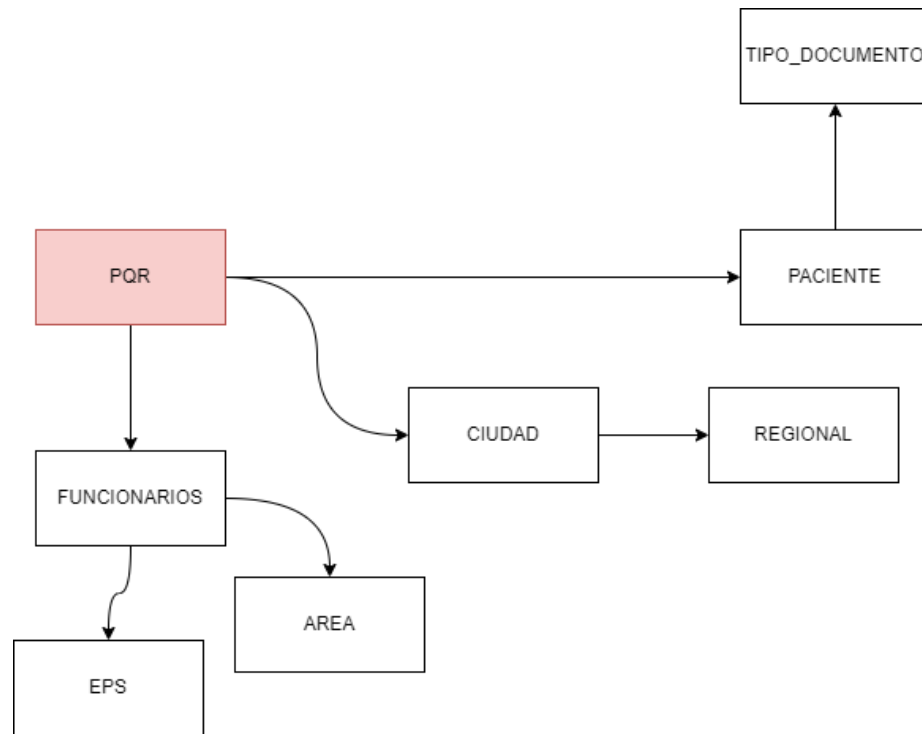


Figure 7: Modelo Conceptual.

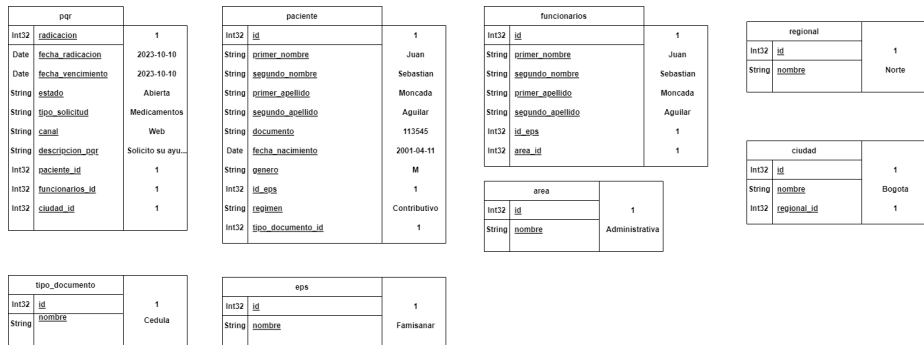


Figure 8: Modelo Físico.

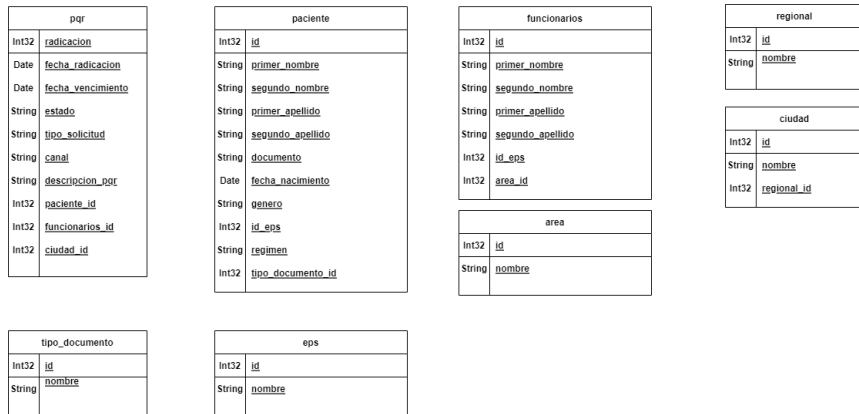


Figure 9: Modelo Lógico.

5.2 SMBD utilizado para la Base de Datos No-SQL

JUAN SEBASTIAN'S ORD - 2025-10-28 > PROJECT 0 > DATABASES

ClusterO

VERSION: 6.0.11 | REGION: AWS N. Virginia (us-east-1)

Overview Real Time Metrics Collections Search Profiler Performance Advisor Online Archive Cmd Line Tools

DATABASES: 1 COLLECTIONS: 8

Visualize Your Data Refresh

Create Database

Search Namespaces

projecto_pqr

LOGICAL DATA SIZE: 36KB | STORAGE SIZE: 44KB | INDEX SIZE: 44KB | TOTAL COLLECTIONS: 8

Create Collection

Collection Name	Documents	Logical Data Size	Avg Document Size	Storage Size	Indexes	Index Size	Avg Index Size
area	3	16KB	57B	20KB	1	20KB	20KB
ciudad	3	199B	67B	20KB	1	20KB	20KB
eps	0	0B	0B	4KB	1	4KB	4KB
funcionarios	0	0B	0B	4KB	1	4KB	4KB
paciente	0	0B	0B	4KB	1	4KB	4KB
pqr	0	0B	0B	4KB	1	4KB	4KB
regional	0	0B	0B	4KB	1	4KB	4KB
tipo_documento	0	0B	0B	4KB	1	4KB	4KB

Figure 10: Creación BD no relacional con MongoDB.

```
1 !pip install pymongo

2 from pymongo.mongo_client import MongoClient
3 from pymongo.server_api import ServerApi
4 uri = "mongodb+srv://jmoncadaa:a33KlgtHjcx7Nu2@cluster0.a2998sn.mongodb.net/?retryWrites=true&w=majority"
5 # Create a new client and connect to the server
6 client = MongoClient(uri, server_api=ServerApi('1'))
7 # Send a ping to confirm a successful connection
8 try:
9     client.admin.command('ping')
10    print("Pinged your deployment. You successfully connected to MongoDB!")
11 except Exception as e:
12    print(e)
```

Pinged your deployment. You successfully connected to MongoDB!

Figure 11: Conexión remota a MongoDB.

```
1 # Seleccionar la base de datos
2 base_de_datos = client.projecto_pqr
3
4 # Seleccionar la colección
5 coleccion = base_de_datos.area
6
7 # Datos que deseas insertar (una lista de documentos)
8 nuevos_documentos = [
9     {
10         "id": 1,
11         "nombre": "Administrativa",
12         # Agrega más campos según sea necesario
13     },
14     {
15         "id": 2,
16         "nombre": "Gestion",
17         # Agrega más campos según sea necesario
18     },
19     {
20         "id": 3,
21         "nombre": "Atencion al cliente",
22         # Agrega más campos según sea necesario
23     }
24 ]
25
26 # Insertar los documentos en la colección
27 resultado = coleccion.insert_many(nuevos_documentos)
28
29 # Imprimir los IDs de los documentos recién insertados
30 print(f"IDs de los nuevos documentos: {resultado.inserted_ids}")
31
32 # Cerrar la conexión
33 client.close()
```

IDs de los nuevos documentos: [ObjectId('654f7f1656a46d1d54ad913e'), ObjectId('654f7f1656a46d1d54ad913f'), ObjectId('654f7f1656a46d1d54ad9140')]

Figure 12: Carga masiva de datos de Area.

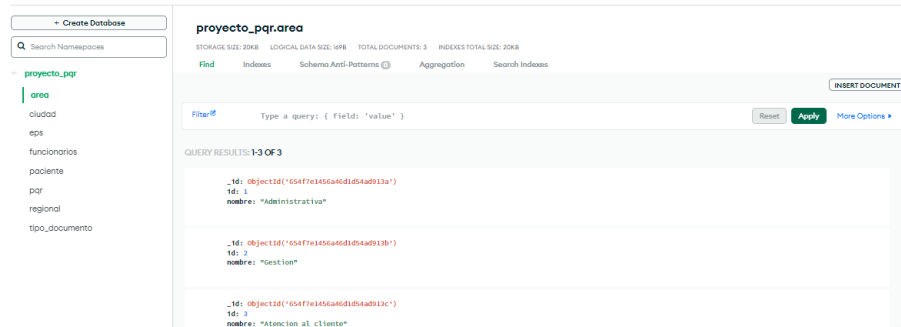


Figure 13: Resultado datos de Area.



Figure 14: Carga masiva de datos de Ciudad.

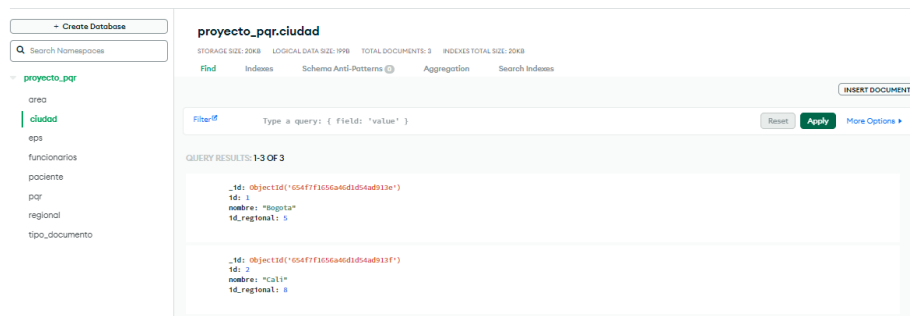


Figure 15: Resultado carga de datos de Ciudad.

```

1 # Seleccionar la base de datos
2 base_de_datos = client.proyecto_pqr
3
4 # Seleccionar la colección
5 coleccion = base_de_datos.eps
6
7 # Datos que deseas insertar (una lista de documentos)
8 nuevos_documentos = [
9     {
10         "id": 1,
11         "nombre": "Sanitas",
12         # Agrega más campos según sea necesario
13     },
14     {
15         "id": 2,
16         "nombre": "Compensar",
17         # Agrega más campos según sea necesario
18     },
19     {
20         "id": 3,
21         "nombre": "Nueva EPS",
22         # Agrega más campos según sea necesario
23     }
24 ]
25
26 # Insertar los documentos en la colección
27 resultado = coleccion.insert_many(nuevos_documentos)
28
29 # Imprimir los IDs de los documentos recién insertados
30 print(f"IDs de los nuevos documentos: {resultado.inserted_ids}")
31
32 # Cerrar la conexión
33 client.close()

```

IDs de los nuevos documentos: [ObjectId('654f83b556a46d1d54ad9142'), ObjectId('654f83b556a46d1d54ad9143'), ObjectId('654f83b556a46d1d54ad9144')]

Figure 16: Carga masiva de datos de EPS.

The screenshot shows the MongoDB Compass interface. On the left, a sidebar lists databases and collections. The 'proyecto_pqr' database is selected, and the 'eps' collection is highlighted. The main area displays the 'proyecto_pqr.eps' collection details, including storage and document statistics. Below this, a query results section shows three documents inserted into the collection. Each document contains an '_id' (ObjectId), an 'id' (numeric), and a 'nombre' (string).

Document	_id	id	nombre
1	ObjectId('654f83b556a46d1d54ad9142')	1	Sanitas
2	ObjectId('654f83b556a46d1d54ad9143')	2	Compensar
3	ObjectId('654f83b556a46d1d54ad9144')	3	Nueva EPS

Figure 17: Resultado carga de datos de Eps.

```

1 # Seleccionar la base de datos
2 base_de_datos = client.proyecto_pqr
3
4 # Seleccionar la colección
5 coleccion = base_de_datos.funcionarios
6
7 # Datos que deseas insertar (una lista de documentos)
8 nuevos_documentos = [
9     {
10         "id": 1,
11         "primer_nombre": "Falcon",
12         "segundo_nombre": "Teofilo",
13         "primer_apellido": "Garcia",
14         "segundo_apellido": "Gutierrez",
15         "id_eps": 5,
16         "area_id": 1,
17         # Agrega más campos según sea necesario
18     },
19     {
20         "id": 2,
21         "primer_nombre": "Roger",
22         "segundo_nombre": "Rafael",
23         "primer_apellido": "Federer",
24         "segundo_apellido": "Nadal",
25         "id_eps": 5,
26         "area_id": 1,
27         # Agrega más campos según sea necesario
28     },
29     {
30         "id": 3,
31         "primer_nombre": "Jose",
32         "segundo_nombre": "Gabriel",
33         "primer_apellido": "Saramago",
34         "segundo_apellido": "Marquez",
35         "id_eps": 5,
36         "area_id": 1,
37         # Agrega más campos según sea necesario
38     }
39 ]

```

Figure 18: Carga masiva de datos de Funcionarios.

The screenshot shows the MongoDB Compass interface. On the left, a sidebar lists the database structure: 'proyecto_pqr' (expanded) contains 'area', 'ciudad', 'eps', 'funcionarios' (selected), 'paciente', 'pqr', 'regional', and 'tipo_documento'. The main panel displays the 'proyecto_pqr.funcionarios' collection. At the top, it shows statistics: STORAGE SIZE: 20KB, LOGICAL DATA SIZE: 500B, TOTAL DOCUMENTS: 3, INDEXES TOTAL SIZE: 20KB. Below this are tabs for 'Find', 'Indexes', 'Schema And Patterns', 'Aggregation', and 'Search Indexes'. The 'Find' tab is active, showing a query filter bar with 'Filter' and 'Type a query: { field: 'value' }'. Below the filter bar, it says 'QUERY RESULTS: 1-3 OF 3'. The results show three documents:

```

{
  "_id": ObjectId("654f85485ea46d8d54ad9146"),
  "id": 1,
  "primer_nombre": "Falcon",
  "segundo_nombre": "Teofilo",
  "primer_apellido": "Garcia",
  "segundo_apellido": "Gutierrez",
  "id_eps": 5,
  "area_id": 1
}
{
  "_id": ObjectId("654f85485ea46d8d54ad9147"),
  "id": 2,
  "primer_nombre": "Roger",
  "segundo_nombre": "Rafael",
  "primer_apellido": "Federer",
  "segundo_apellido": "Nadal",
  "id_eps": 5,
  "area_id": 1
}
{
  "_id": ObjectId("654f85485ea46d8d54ad9148"),
  "id": 3,
  "primer_nombre": "Jose",
  "segundo_nombre": "Gabriel",
  "primer_apellido": "Saramago",
  "segundo_apellido": "Marquez",
  "id_eps": 5,
  "area_id": 1
}

```

Figure 19: Resultado carga de datos de Funcionarios.

```

1 # Seleccionar la base de datos
2 base_de_datos = client.proyecto_pqr
3
4 # Seleccionar la colección
5 coleccion = base_de_datos.paciente
6
7 # Datos que deseas insertar (una lista de documentos)
8 nuevos_documentos = [
9     {
10         "id": 1,
11         "primer_nombre": "Johan",
12         "primer_apellido": "Sanchez",
13         "segundo_apellido": "Rojas",
14         "documento": "548698",
15         "fecha_nacimiento": "1990-10-11",
16         "genero": "NB",
17         "id_eps": 5,
18         "regimen": "Contributivo",
19         "tipo_documento_id": 2,
20         # Agrega más campos según sea necesario
21     },
22     {
23         "id": 2,
24         "primer_nombre": "Juan",
25         "segundo_nombre": "Sebastian",
26         "primer_apellido": "Moncada",
27         "segundo_apellido": "Aguilar",
28         "documento": "548665",
29         "fecha_nacimiento": "2000-08-11",
30         "genero": "M",
31         "id_eps": 2,
32         "regimen": "Contributivo",
33         "tipo_documento_id": 1,
34         # Agrega más campos según sea necesario
35     },
36     {
37         "id": 3,

```

Figure 20: Carga masiva de datos de Pacientes.

The screenshot shows the MongoDB Atlas interface for the 'proyecto_pqr' database. The 'paciente' collection is selected, showing 3 documents. The query results display the following data:

Document ID	Fields
1	id: 1, primer_nombre: "Johan", primer_apellido: "Sanchez", segundo_apellido: "Rojas", documento: "548698", fecha_nacimiento: "1990-10-11", genero: "NB", id_eps: 5, regimen: "Contributivo", tipo_documento_id: 2
2	id: 2, primer_nombre: "Juan", segundo_nombre: "Sebastian", primer_apellido: "Moncada", segundo_apellido: "Aguilar", documento: "548665", fecha_nacimiento: "2000-08-11", genero: "M", id_eps: 2, regimen: "Contributivo", tipo_documento_id: 1
3	id: 3

Figure 21: Resultado carga de datos de Pacientes.

```

1 # Seleccionar la base de datos
2 base_de_datos = client.proyecto_pqr
3
4 # Seleccionar la colección
5 coleccion = base_de_datos.pqr
6
7 # Datos que deseas insertar (una lista de documentos)
8 nuevos_documentos = [
9     {
10         "radicacion": 1,
11         "fecha_radicacion": datetime(2023, 5, 30),
12         "fecha_vencimiento": datetime(2023, 6, 30),
13         "estado": "Abierta",
14         "tipo_solicitud": "Medicamentos",
15         "canal": "Web",
16         "descripcion_pqr": "Solicito la entrega de mis medicamentos para la tension",
17         "id_paciente": "125698",
18         "id_funcionario": 5,
19         "id_ciudad": 2,
20         # Agrega más campos según sea necesario
21     },
22     {
23         "radicacion": 2,
24         "fecha_radicacion": datetime(2023, 6, 30),
25         "fecha_vencimiento": datetime(2023, 7, 30),
26         "estado": "Abierta",
27         "tipo_solicitud": "Medicamentos",
28         "canal": "Telefono",
29         "descripcion_pqr": "Solicito la entrega de mis medicamentos para la migraña",
30         "id_paciente": "125698",
31         "id_funcionario": 3,
32         "id_ciudad": 4,
33         # Agrega más campos según sea necesario
34     },
35     {
36         "radicacion": 3,
37         "fecha_radicacion": datetime(2023, 6, 30),
38         "fecha_vencimiento": datetime(2023, 7, 30),

```

Figure 22: Carga masiva de datos de PQR.

Figure 23: Resultado carga de datos de PQR.


```

1 # Seleccionar la base de datos
2 base_de_datos = client.proyecto_pqr
3
4 # Seleccionar la colección
5 coleccion = base_de_datos.regional
6
7 # Datos que deseas insertar (una lista de documentos)
8 nuevos_documentos = [
9     {
10         "id": 1,
11         "nombre": "Occidente",
12         # Agrega más campos según sea necesario
13     },
14     {
15         "id": 2,
16         "nombre": "Norte",
17         # Agrega más campos según sea necesario
18     },
19     {
20         "id": 3,
21         "nombre": "Sur",
22         # Agrega más campos según sea necesario
23     }
24 ]
25
26 # Insertar los documentos en la colección
27 resultado = coleccion.insert_many(nuevos_documentos)
28
29 # Imprimir los IDs de los documentos recién insertados
30 print(f"IDs de los nuevos documentos: {resultado.inserted_ids}")
31
32 # Cerrar la conexión
33 client.close()

```

Figure 24: Carga masiva de datos de Regional.

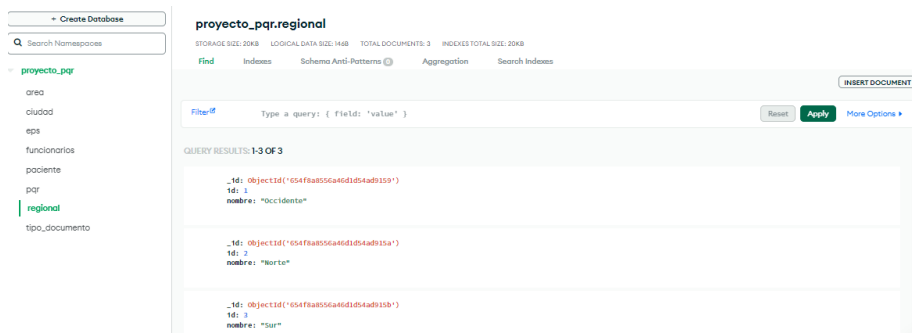


Figure 25: Resultado carga de datos de Regional.

```

1 # Seleccionar la base de datos
2 base_de_datos = client.proyecto_pqr
3
4 # Seleccionar la colección
5 coleccion = base_de_datos.tipo_documento
6
7 # Datos que deseas insertar (una lista de documentos)
8 nuevos_documentos = [
9     {
10         "id": 1,
11         "nombre": "Cedula",
12         # Agrega más campos según sea necesario
13     },
14     {
15         "id": 2,
16         "nombre": "TI",
17         # Agrega más campos según sea necesario
18     },
19     {
20         "id": 3,
21         "nombre": "Pasaporte",
22         # Agrega más campos según sea necesario
23     }
24 ]
25
26 # Insertar los documentos en la colección
27 resultado = coleccion.insert_many(nuevos_documentos)
28
29 # Imprimir los IDs de los documentos recién insertados
30 print(f"IDs de los nuevos documentos: {resultado.inserted_ids}")
31
32 # Cerrar la conexión
33 client.close()

```

IDs de los nuevos documentos: [ObjectId('654f8b0556a46d1d54ad915d'), ObjectId('654f8b0556a46d1d54ad915e'), ObjectId('654f8b0556a46d1d54ad915f')]

Figure 26: Carga masiva de datos de tipo de documento.

Figure 27: Resultado carga de datos de tipo de documento.

5.3 Conclusiones modelos de datos

Al realizar un análisis más detenido de los dos tipos de modelos, se evidencia que el modelo relacional emerge como una elección más adecuada para la implementación de un sistema de gestión de PQRS (Peticiones, Quejas, Reclamos y Sugerencias). Esta preferencia se fundamenta en las intrincadas relaciones que el modelo relacional puede establecer entre diversas entidades, permitiendo una representación estructurada y coherente de la información.

El modelo relacional, al basarse en tablas y establecer vínculos claros entre los datos, facilita la organización y recuperación eficiente de la información relacionada con las PQRS. La capacidad para definir claves foráneas y establecer relaciones entre tablas proporciona una estructura robusta que refleja fielmente la complejidad de las interacciones en un sistema de gestión de PQRS.

6 Aplicación de ETL (Extract, Transform, Load) y Bodega de Datos (*Tercera entrega*)

6.1 Ejemplo de aplicación de ETL y Bodega de Datos (*Tercera entrega*)

En el proceso de extracción, la información inicial de las PQRS se alojaba en un archivo .XLSX. Para llevar a cabo esta extracción, se empleó Python con la biblioteca Pandas para manipular eficientemente este archivo Excel. Una vez que los datos se habían convertido en dataframes, se procedió a realizar subsets correspondientes a las entidades definidas en el modelo de la base de datos.

Este enfoque permitió una manipulación precisa y estructurada de la información inicial. Python y Pandas no solo facilitaron la extracción de datos del archivo .XLSX, sino que también posibilitaron la creación de subsets específicos que se ajustaban a las entidades previamente delineadas en el modelo de la base de datos.

Simultáneamente a la manipulación de los datos, se llevó a cabo un proceso integral de transformación. Este proceso permitió la depuración de la información al eliminar caracteres especiales, enlaces y otros posibles inconvenientes derivados de la codificación de los datos iniciales. Esta fase de transformación fue crucial para garantizar la integridad y calidad de los datos.

Como resultado de este proceso, se generaron nuevos archivos Excel, uno destinado a cada entidad previamente definida. Estos archivos ahora depurados y organizados están listos para ser cargados eficientemente en las respectivas tablas de la base de datos. Este enfoque exhaustivo no solo aseguró la limpieza de los datos, sino que también preparó la información de manera específica para cada entidad, facilitando así su posterior incorporación en las tablas correspondientes.

6.2 Automatización de Datos (*Tercera entrega*)

Debido al constante flujo de nuevas PQRS a diario, se ha decidido implementar un Script en python que facilite la lectura y procesamiento de archivos Excel. Este Script utilizará diversas funciones para llevar a cabo la limpieza de datos, generando al final un nuevo archivo Excel para cada entidad de la base de datos. Estos archivos se almacenarán en una carpeta específica.

Dentro de la base de datos, se ha programado una tarea que monitorea una carpeta en una fuente local donde se depositan los nuevos archivos Excel. Esta tarea permite la carga automática de los datos en las tablas correspondientes una vez que los archivos son detectados en esa carpeta.

6.3 Integración de Datos (*Tercera entrega*)

Una vez que la información de los archivos Excel se ha cargado en la base de datos, es crucial realizar la inserción en cada tabla de manera secuencial. Este enfoque secuencial busca prevenir errores relacionados con las claves foráneas entre las diversas tablas que conforman la base de datos.

Por lo tanto, el primer paso sería la inserción en las tablas de Tipo documento, regional, área y EPS. El siguiente paso sería insertar datos en las tablas de ciudad, funcionario y paciente. Finalmente, el último paso consistiría en la inserción en la tabla PQR junto con sus respectivas claves foráneas.

7 Proximos pasos

Hasta el momento, se han alcanzado los objetivos establecidos para la creación de la base de datos. No obstante, es importante destacar que este logro forma parte de un proceso más amplio: posibilitar la clasificación de las PQRS mediante el uso de Procesamiento del Lenguaje Natural (NLP). Esta implementación permitirá categorizar eficientemente las PQRS, contribuyendo así a la optimización del proceso de gestión y resolución.

8 Lecciones aprendidas

La manipulación y organización efectiva de los datos desempeña un papel fundamental en la gestión de las PQRS (Peticiónes, Quejas, Reclamos y Sugerencias). Identificar relaciones sólidas entre los datos resulta crucial para optimizar la velocidad de obtención de la información. En este contexto, diversos motores de bases de datos ofrecen soluciones adaptables a distintos problemas. Por ejemplo, MySQL y PostgreSQL destacan como opciones robustas para bases de datos relacionales, mientras que DynamoDB y MongoDB son excelentes alternativas para bases de datos no relacionales.

La gestión de PQRS se ve especialmente desafiada por la volatilidad de la información, especialmente cuando se trata de textos extensos. En este contexto, la eficiencia en la obtención de datos se vuelve primordial, ya que la rapidez en acceder a la información se convierte en un factor clave para garantizar una gestión efectiva de las PQRS.

9 Bibliografía

- Ordóñez, M. P. Z., Ríos, J. R. M., Castillo, F. F. R. (2017). Administración de bases de datos con PostgreSQL. Ingeniería y Tecnología. 3Ciencias. ISBN: 9788494668463. Recuperado de <https://books.google.com.co/books?id=5-mkDgAAQBAJ>
- Ácid Carrillo, S., Marín Ruiz, N., Medina Rodríguez, J. M., Pons Capote, O., Vila Miranda, A. (2008). Introducción a los sistemas de bases de datos. Ediciones Paraninfo, S.A. ISBN: 9788497325158. Recuperado de <https://books.google.com.co/books?id=HmnHeZ1wsvwC>
- Sarasa, A., Cabezuelo, A. S., e-libro, Corp. (2016). Introducción a las bases de datos NoSQL usando MongoDB. Elibro Catedra. Editorial UOC. ISBN: 9788491162667. Recuperado de <https://books.google.com.co/books?id=ntoEkAEACAAJ>