Introduction to Database Systems Project

Professor:

Teófilo Chambilla Aquino

Group Members:

Leonardo Alfredo Montoya	202110797
Melanie Alexia Cortez	202210100
Sebastián Leonardo Muñico	202210217
Angelo Giovanni Soldi	202210434
Herbert Armando Chumbe	202020041

August, 2023

Contents

1 Requirements

1.1 Introduction

The evaluated company operates two hospital centers (ESSALUD) that have been running for over nine years under the "White Coat" model. This model ensures that all services provided by the hospital complexes are managed privately. To date, this model is considered a success in the country, with results that rival those of private clinics and other purely private institutions.

Over the years, the volume of information handled has grown significantly, which has impacted data management and decision-making processes. For the past few years, the company has focused on ensuring that all decisions are supported by data, which has made this issue increasingly relevant.

This project aims to implement a database model to improve data management and support decision-making processes within the organization.

1.2 General Description of the Problem/Organization/Company

Although the hospital centers have been operating for more than nine years, claim management is currently carried out using a single table where all claim information is stored. This approach lacks flexibility, and as the volume of data has grown over the years, the challenges have become more apparent. Moreover, in the healthcare sector, the information managed is highly sensitive. Hence, the need arises for a more flexible structure to handle the data effectively.

1.3 Need/Uses of the Database

The need for a more flexible database arises from the goal of better managing user information. This will facilitate decision-making, improve the management of active claims, and ensure the timely resolution of issues for the benefit of patients.

Additionally, regulatory institutions such as ESSALUD periodically request information about the status of claims. A well-designed database would streamline the extraction of such data for report generation purposes.

1.4 Current Problem Resolution

1.4.1 How Are Data Stored/Processed Today?

Currently, data are recorded in a system called Hosix, which is stored in the company's data warehouse. SQL Server is used to access all claim-related information.

1.4.2 Data Flow

The hospital's data primarily originate from claims made at the User Service Platform (PAUS). The hospital records the required data to register the claim and progressively adds more information as the claim moves through its process.

1.5 Detailed System Description

1.5.1 Current Information Objects

Data will be uploaded to the new database using Excel files in `.csv` format. Scenarios for testing queries will range from 1,000 to 1,000,000 records, which will be randomly generated while adhering to the constraints outlined in the relational model.

1.5.2 Expected Features and Functionalities

The database system for processing claims in medical centers must be:

- **Structured:** Secure and efficient for storing and retrieving data.
- **User-Friendly:** Allowing efficient searches, filtering, and report generation.
- **Scalable:** Capable of adapting to the medical center's growth.
- **Interoperable:** Compatible with other systems.

These features aim to optimize information management and enhance service quality at the institution.

1.5.3 Existing/Required User Types

- **Patient:** Any person filing a claim at a specific location.
- **Applicant:** The person filing the claim on behalf of the patient, often a family member.

1.5.4 Query and Update Types

The database will primarily answer the following questions:

- What were the top three reasons for claims in 2023?
- Which substantiated claims lacked legal advice?
- In which months were the most measures proposed in 2023?
- How many claims were filed by someone other than the patient?
- What are the top five services with the most claim involvement?
- What is the average time between the start and end of claim traceability?
- Which service has the most claims in each center?
- Which three services played a primary role in claims?
- What is the average time taken by the department to respond to substantiated claims?
- Which location has the most substantiated claims?

1.5.5 Estimated Database Size

The database will store information related to claims across various medical centers. Given the sensitive and extensive data, it is expected to grow significantly. Current estimates indicate the database could eventually occupy several terabytes.

1.6 Project Objectives

(a) Improve claim management and tracking, facilitating case resolution and ensuring adequate attention to patients and applicants.

(b) Optimize the organization and structure of data for efficient access and query execution.

(c) Facilitate the generation of reports and statistics to support decision-making and identify areas of improvement.

(d) Design a scalable and adaptable database to meet future needs.

Table Name	Attribute Size (Bytes)	Record Size (Bytes)
Person	344	344
Phone	50	50
Patient	45	45
Applicant	20	20
Representation	60	60
Claim	44	44
Traceability	372	372
Communication Type	58	58
Reason	58	58
Result	58	58
Action	220	220
Participation	26	26
Service	66	66
Center	66	66
Location	58	58

Table 1: Database Size Estimation

1.7 References

This project is inspired by dissatisfaction with medical services and aims to provide insights into the claim process. References include:

- https://gestion.pe/peru/como-denunciar-una-mala-atencion-en-una-clinica-susalud-medicos-reclamos-nnda-nnlt-noticia/
- https://elcomercio.pe/lima/presentar-queja-clinicas-u-hospitales-144872-noticia/
- https://peru21.pe/lima/ministerio-salud-47-000-reclamos-2017-deficiente-servicio-salud-374194-noticia/
- http://www.scielo.org.pe/scielo.php?script=sci_arttext&pid=S2308-05312020000200246/

1.8 Contingencies

1.8.1 Potential Issues

- The large database size may complicate real-time updates.
- Claims could be erroneously validated or incoherent data might be recorded.

1.8.2 Project Scope and Limitations

Limitations: The project does not include pre-registered complaints resolved before formal registration in the system.
Scope: Covers the formal registration of claims, including resolution measures for substantiated claims.

2 Entity-Relationship Model

2.1 Semantic Rules

- Claims are identified by unique IDs and years.

- Communication types, reasons, and results are uniquely identified by codes and descriptions.
- Traceability processes are recorded with detailed comments, start and end dates.
- Claims have one reason and one result, with only "substantiated" claims having assigned measures.
- Services belong to one center, and centers are tied to specific locations (e.g., Callao, Villa María del Triunfo).

2.2 Entity-Relationship Model

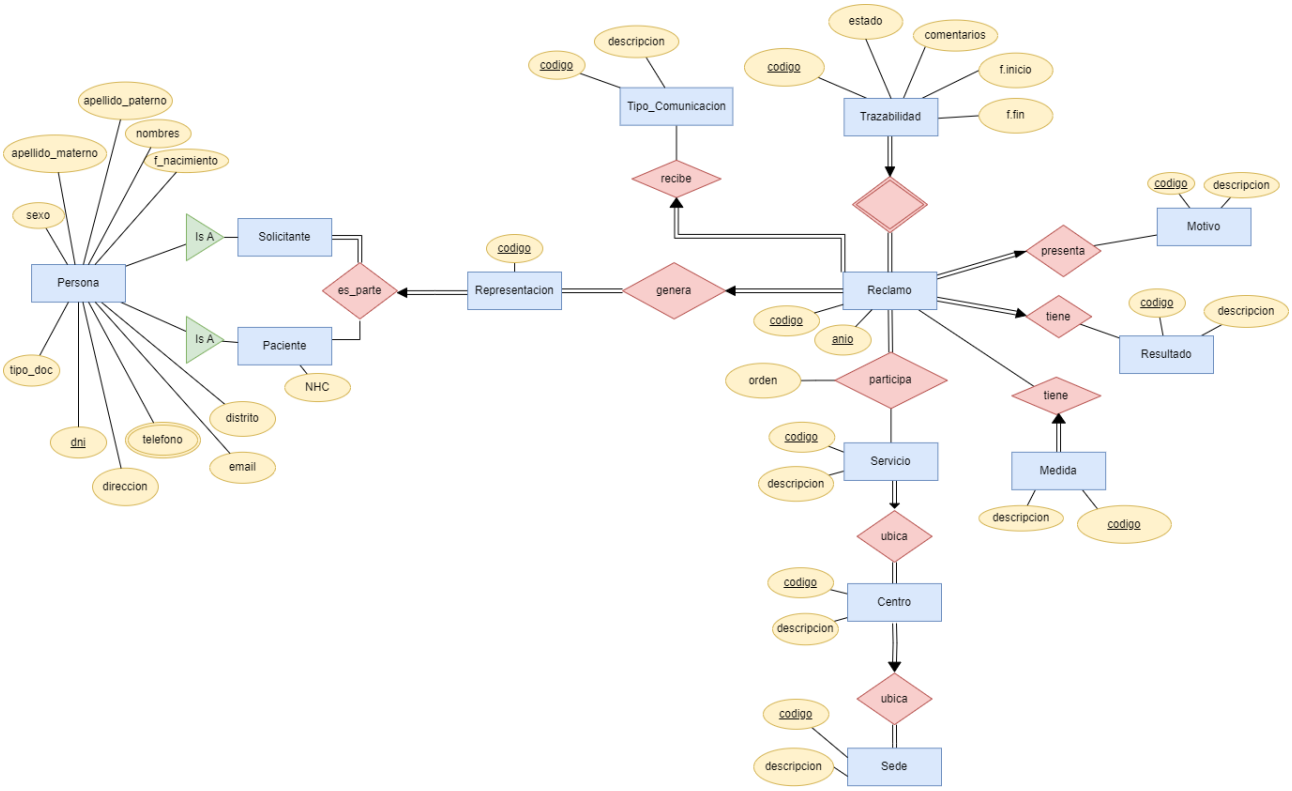

Figure 1: Entity-Relationship Model

2.3 Specifications and Considerations of the Model

1. **Entity: Person**
   Specifications: Contains all relevant information to identify a person.
   Considerations: It is a superclass, and its primary key is 'dni'. There is overlap in its subclasses, as the applicant may also be the same as the patient.

2. **Entity: Applicant**
   Specifications: Refers to the person filing the claim (could be the same as the patient).
   Considerations: It is a subclass of 'Person'.

3. **Entity: Patient**
   Specifications: Could be the person filing the claim and includes an additional attribute, NHC (Medical History Number).
   Considerations: It is a subclass of 'Person'.

4. **Entity: Representation**
   Specifications: Includes the representation code attribute, which is generated for each claim.
   Considerations: This entity stores all instances where an applicant represents a patient. If the applicant is the same as the patient, the patient's information will be stored in both columns.

5. **Entity: Phone**
   Specifications: A multivalued attribute of the 'Person' class.
   Considerations: Each person may have more than one phone number.

6. **Entity: Claim**
   Specifications: Includes two attributes: 'code', which stores the assigned code, and 'year', which stores the year the claim was made.
   Considerations: This entity includes foreign keys for claim code, reason, representation, and communication type.

7. **Entity: Communication Type**
   Specifications: Includes two attributes: 'description', which denotes whether the claim was made in person, virtually, or via phone, and 'code', which represents the description numerically.
   Considerations: It is stored in a claim.

8. **Entity: Traceability**
   Specifications: Contains the process of the claim from its filing to its outcome. It includes five attributes: a numeric primary key ('code') for identification, a 'status' attribute that defines the current stage of the claim process, a 'comment' attribute where observations by the parties involved in the claim are stored, and two date attributes, 'start_date' and 'end_date', which are used to calculate the time between each step of the claim process.
   Considerations: It is a weak entity of the `Claim` class, with its primary keys being the claim `code` and `year`.

9. **Entity: Reason**
   Specifications: Refers to the reason for the claim. It includes a 'description' attribute storing the reason and a 'code' attribute containing a numeric code representing it.
   Considerations: The 'Reason' table is a master table with predefined reasons that encompass others (e.g., Poor Attention, Incorrect Diagnosis).

10. **Entity: Outcome**
    Specifications: Corresponds to the outcome of the claim. The 'description' attribute provides a conclusion for the claim (e.g., substantiated or unsubstantiated), while the primary key 'code' contains a numeric code identifying the description.
    Considerations: Every claim will have an outcome.

11. **Entity: Measure**
    Specifications: Includes two attributes: the primary key ‘code‘ and foreign keys for the service code and the year the claim was communicated.
    Considerations: Each measure is assigned to exactly one claim, and a claim will only have measures if it is substantiated.

12. **Entity: Service**
    Specifications: Refers to the service for which the claim was made. It includes the ‘code‘ attribute identifying the service, a ‘description‘ attribute explaining it, and a foreign key for the center code.
    Considerations: Each service provided by the center is assigned a unique numeric code.

13. **Entity: Center**
    Specifications: Refers to the center where the service that led to the claim was provided (e.g., Hospital or Polyclinic). This information is stored in the ‘description‘ attribute and represented numerically in the primary key ‘code‘.
    Considerations: There are only two possible centers: Hospital and Polyclinic.

14. **Entity: Location**
    Specifications: Refers to the location of the center. Its ‘description‘ attribute contains the district name, identified by the numeric primary key ‘code‘.
    Considerations: There are two possible locations (CLL, VMT).

3 Relational Model

3.1 Relational Model

- **Person**(dni, name, last_name_maternal, last_name_paternal, gender, birth_date, email, address, district, document_type)
- **Phone**(phone, Person.dni)
- **Patient**(Person.dni, NHC)
- **Applicant**(Person.dni)
- **Representation**(code, Person.dniApplicant, Person.dniPatient)
- **Claim**(code, year, Representation.code, Reason.code, Outcome.code, Communication_Type.code)
- **Traceability**(code, Claim.code, Claim.year, status, start_date, end_date, description, comments)
- **Communication_Type**(code, description)
- **Reason**(code, description)
- **Outcome**(code, description)

- **Measure**(code, Claim.code, Claim.year, description)
- **Participates**(Claim.code, Claim.year, Service.code, order)
- **Service**(code, Center.code, description)
- **Center**(code, Location.code, description)
- **Location**(code, description)

3.2 Transformation Specifications

3.2.1 Entities

Since the specifications of the entities have already been described, we will simply list them here. Total entities:

- **Representation**
- **Phone**
- **Claim**
- **Communication_Type**
- **Reason**
- **Outcome**
- **Service**
- **Center**
- **Location**

3.2.2 Superclass/Subclass Entities

- ***Superclass* (Person:)** This superclass includes the attributes `dni`, `name`, `last_name_maternal`, `last_name_paternal`, `gender`, `birth_date`, `email`, `address`, `district`, and `document_type`. The primary key is `dni`, and there are two entities that will inherit this key. In this case, overlap will occur, so the `Person` entity will be transformed into a table.
- ***Subclasses***
  - **Applicant:** Overlap exists because an applicant may also be the patient, or in other cases, the applicant may be a companion of the patient if the patient cannot directly file the claim. A table is created containing the `dni` of the person as both the primary and foreign key.
  - **Patient:** Overlap exists because a patient may also be an applicant. A table is created containing the `dni` of the person as both the primary and foreign key, along with the attribute `NHC`, which represents the patient's medical history number.

3.2.3 Binary Relationships

- **Participates:** This relationship occurs between `Claim` and `Service`. The `Claim` entity has a multiplicity of 1 to n, as one or more services participate in a claim, and at least one service must be related to a claim. On the other hand, the `Service` entity has a multiplicity of 0 to n, as a service may participate in zero or more claims. This means there may be services that do not have any claims associated with them.

3.2.4 Ternary Relationships

No ternary relationships are present in our relational model.

3.3 Data Dictionary

Person				
Field Name	**Data Type**	**PK**	**FK**	**Description**
dni	VARCHAR(20)	X		Unique identifier (DNI) of the person
name	VARCHAR(40)			Name of the person
last_name_maternal	VARCHAR(20)			Maternal last name of the person
last_name_paternal	VARCHAR(20)			Paternal last name of the person
gender	VARCHAR(20)			Gender of the person
birth_date	DATE			Birth date of the person
email	VARCHAR(50)			Email address of the person
address	VARCHAR(100)			Address of the person
district	VARCHAR(40)			District where the person resides
document_type	VARCHAR(15)			Type of document of the person

Table 2: Person Table

Phone				
Field Name	**Data Type**	**PK**	**FK**	**Description**
phone	VARCHAR(30)	X		Phone number of the person
Person.dni	VARCHAR(20)		X	DNI of the person

Table 3: Phone Table

Patient				
Field Name	**Data Type**	**PK**	**FK**	**Description**
Person.dni	VARCHAR(20)	X	X	DNI of the patient
NHC	VARCHAR(15)			Medical history number of the patient

Table 4: Patient Table

Applicant				
Field Name	**Data Type**	**PK**	**FK**	**Description**
Person.dni	VARCHAR(20)	X	X	Applicant's unique identifier (DNI)

Table 5: Applicant Table

Representation				
Field Name	**Data Type**	**PK**	**FK**	**Description**
code	INTEGER	X		Representation code
Person.dniApplicant	VARCHAR(20)		X	Applicant's unique identifier (DNI)
Person.dniPatient	VARCHAR(20)		X	Patient's unique identifier (DNI)

Table 6: Representation Table

Claim				
Field Name	**Data Type**	**PK**	**FK**	**Description**
code	INTEGER	X		Claim code
year	INTEGER	X		Year the claim was filed
Representation.code	INTEGER		X	Representation code
Reason.code	INTEGER		X	Reason code for the claim
Outcome.code	INTEGER		X	Outcome code of the claim
Communication_Type.code	INTEGER		X	Code of the communication type used

Table 7: Claim Table

Traceability				
Field Name	**Data Type**	**PK**	**FK**	**Description**
code	INTEGER	X		Traceability code
Claim.code	INTEGER		X	Claim code
Claim.year	INTEGER		X	Year the claim was filed
start_date	DATE			Start date of the traceability process
end_date	DATE			End date of the traceability process
status	VARCHAR(50)			Current status of the claim
comments	VARCHAR(300)			Comments or observations related to the traceability process

Table 8: Traceability Table

Communication_Type				
Field Name	**Data Type**	**PK**	**FK**	**Description**
code	INTEGER	X		Code for the communication type
description	VARCHAR(50)			Description of the communication type

Table 9: Communication_Type Table

Reason				
Field Name	**Data Type**	**PK**	**FK**	**Description**
code	INTEGER	X		Reason code
description	VARCHAR(50)			Description of the reason

Table 10: Reason Table

Outcome				
Field Name	**Data Type**	**PK**	**FK**	**Description**
code	INTEGER	X		Outcome code
description	VARCHAR(50)			Description of the outcome

Table 11: Outcome Table

Measure				
Field Name	**Data Type**	**PK**	**FK**	**Description**
code	INTEGER	X		Measure code
Claim.code	INTEGER		X	Claim code
Claim.year	INTEGER		X	Year in which the claim was made
description	VARCHAR(200)			Description of the measure

Table 12: Measure Table

Participates				
Field Name	**Data Type**	**PK**	**FK**	**Description**
Claim.code	INTEGER	X	X	Claim code
Claim.year	INTEGER	X	X	Year in which the claim was made
Service.code	INTEGER	X	X	Service code
order	INTEGER			Participation order

Table 13: Participates Table

Service				
Field Name	**Data Type**	**PK**	**FK**	**Description**
code	INTEGER	X		Service code
Center.code	INTEGER		X	Center code
description	VARCHAR(50)			Description of the service

Table 14: Service Table

Center				
Field Name	**Data Type**	**PK**	**FK**	**Description**
code	INTEGER	X		Center code
Location.code	INTEGER		X	Location code
description	VARCHAR(50)			Description of the center

Table 15: Center Table

Location				
Field Name	**Data Type**	**PK**	**FK**	**Description**
code	INTEGER	X		Location code
description	VARCHAR(50)			Description of the location

Table 16: Location Table

4 Database Implementation

4.1 Database Creation

In the following section, we explain how the tables for our relational model were created, along with their respective constraints to ensure the proper functioning of the database.

```
CREATE TABLE Persona(
    dni VARCHAR(20),
    nombre VARCHAR(40),
    apellido_paterno VARCHAR(20),
    apellido_materno VARCHAR(20),
    sexo VARCHAR(20),
    fecha_nacimiento DATE,
    email VARCHAR(50),
    direccion VARCHAR(100)
);

CREATE TABLE Telefono(
    numero VARCHAR(10),
    dni VARCHAR(20)
);

CREATE TABLE Paciente(
    dni VARCHAR(20),
    NHC VARCHAR(15)
);

CREATE TABLE Solicitante(
    dni VARCHAR(20)
);

CREATE TABLE Representacion(
    codigo INTEGER,
    dni_solicitante VARCHAR(20),
    dni_paciente VARCHAR(20)
);

CREATE TABLE Reclamo(
    codigo INTEGER,
    anio INTEGER,
    representacion INTEGER,
    motivo INTEGER,
    resultado INTEGER,
    comunicacion INTEGER
);

CREATE TABLE Trazabilidad(
    codigo INTEGER,
    reclamo INTEGER,
    anio INTEGER,
    fecha_inicio DATE,
    fecha_fin DATE,
    estado VARCHAR(50)
);

CREATE TABLE Tipo_comunicacion(
    codigo INTEGER,
    descripcion VARCHAR(50)
);

CREATE TABLE Motivo(
    codigo INTEGER,
```

```
    descripcion VARCHAR(50)
);

CREATE TABLE Resultado(
    codigo INTEGER,
    descripcion VARCHAR(50)
);

CREATE TABLE Medida(
    codigo INTEGER,
    reclamo INTEGER,
    anio INTEGER
);

CREATE TABLE Participa(
    reclamo INTEGER,
    anio INTEGER,
    servicio INTEGER
);

CREATE TABLE Servicio(
    codigo INTEGER,
    centro INTEGER,
    descripcion VARCHAR(50)
);

CREATE TABLE Centro(
    codigo INTEGER,
    sede INTEGER,
    descripcion VARCHAR(50)
);

CREATE TABLE Sede(
    codigo INTEGER,
    descripcion VARCHAR(50)
);

--Persona
ALTER TABLE Persona ADD CONSTRAINT dni_persona_pk PRIMARY KEY (dni);
ALTER TABLE Persona ADD CONSTRAINT chk_validar_dni check(length(dni)=8);
ALTER TABLE Persona ADD CONSTRAINT persona_email_unique UNIQUE (email);

--Telefono
ALTER TABLE Telefono ADD CONSTRAINT telefono_pk PRIMARY KEY (numero);
ALTER TABLE Telefono ADD CONSTRAINT p_dni_fk FOREIGN KEY (dni) REFERENCES
Persona (dni);
ALTER TABLE Telefono ADD CONSTRAINT chk_validar_numero check(LEFT(numero,1)='9' and LENGTH(
    numero)=9);
ALTER TABLE Telefono ADD CONSTRAINT chk_validar_dni check(length(dni)=8);

--Paciente
ALTER TABLE Paciente ADD CONSTRAINT paciente_dni_pk PRIMARY KEY (dni);
ALTER TABLE Paciente ADD CONSTRAINT paciente_dni_fk FOREIGN KEY (dni) REFERENCES
Persona(dni);
ALTER TABLE Paciente ADD CONSTRAINT paciente_nhc_unique UNIQUE(NHC);
ALTER TABLE Paciente ADD CONSTRAINT chk_validar_NHC check(length(NHC)=7);
ALTER TABLE Paciente ADD CONSTRAINT chk_validar_dni check(length(dni)=8);

--Solicitante
ALTER TABLE Solicitante ADD CONSTRAINT p_dni_pk PRIMARY KEY (dni);
ALTER TABLE Solicitante ADD CONSTRAINT p_dni_fk FOREIGN KEY (dni) REFERENCES
Persona(dni);
ALTER TABLE Solicitante ADD CONSTRAINT chk_validar_dni check(length(dni)=8);

--Representacion
```

```
ALTER TABLE Representacion ADD CONSTRAINT r_codigo_pk PRIMARY KEY (codigo);
ALTER TABLE Representacion ADD CONSTRAINT r_dnisolicitante_fk FOREIGN KEY (dni_solicitante)
REFERENCES Persona(dni);
ALTER TABLE Representacion ADD CONSTRAINT r_dnipaciente_fk FOREIGN KEY (dni_paciente)
REFERENCES Persona(dni);
ALTER TABLE Representacion ALTER COLUMN codigo ADD GENERATED ALWAYS AS IDENTITY;
ALTER TABLE Representacion ADD CONSTRAINT chk_validar_dnisolicitante check(length(
    dni_solicitante)=8);
ALTER TABLE Representacion ADD CONSTRAINT chk_validar_dnipaciente check(length(dni_paciente)
    =8);


--Tipo_comunicacion
ALTER TABLE Tipo_comunicacion ADD CONSTRAINT tc_codigo_pk PRIMARY KEY(codigo);

--Motivo
ALTER TABLE Motivo ADD CONSTRAINT m_codigo_pk PRIMARY KEY(codigo);

--Resultado
ALTER TABLE Resultado ADD CONSTRAINT re_codigo_pk PRIMARY KEY (codigo);

--Sede
ALTER TABLE Sede ADD CONSTRAINT s_codigo_pk PRIMARY KEY (codigo);

--Centro
ALTER TABLE Centro ADD CONSTRAINT c_codigo_pk PRIMARY KEY (codigo);
ALTER TABLE Centro ADD CONSTRAINT c_sede_codigo_fk FOREIGN KEY (sede)
REFERENCES Sede(codigo);

--Servicio
ALTER TABLE Servicio ADD CONSTRAINT se_codigo_pk PRIMARY KEY (codigo);
ALTER TABLE Servicio ADD CONSTRAINT se_centro_codigo_fk FOREIGN KEY (centro)
REFERENCES Centro(codigo);

--Reclamo
ALTER TABLE Reclamo ADD CONSTRAINT rec_pk PRIMARY KEY (codigo,anio);
ALTER TABLE Reclamo ADD CONSTRAINT rec_representacion_codigo_fk FOREIGN KEY (representacion)
REFERENCES Representacion(codigo);
ALTER TABLE Reclamo ADD CONSTRAINT rec_motivo_codigo_fk FOREIGN KEY (motivo)
REFERENCES Motivo(codigo);
ALTER TABLE Reclamo ADD CONSTRAINT rec_resultado_codigo_fk FOREIGN KEY (resultado)
REFERENCES Resultado(codigo);
ALTER TABLE Reclamo ADD CONSTRAINT rec_tipo_comunicacion_fk FOREIGN KEY (comunicacion)
REFERENCES Tipo_comunicacion(codigo);
ALTER TABLE Reclamo ALTER COLUMN codigo ADD GENERATED ALWAYS AS IDENTITY;
ALTER TABLE Reclamo ALTER COLUMN representacion SET NOT NULL;
ALTER TABLE Reclamo ALTER COLUMN representacion ADD GENERATED ALWAYS AS IDENTITY;
ALTER TABLE Reclamo ADD CONSTRAINT rec_codrep_unique UNIQUE (representacion);

--Trazabilidad
ALTER TABLE Trazabilidad ADD CONSTRAINT t_codigo_pk PRIMARY KEY (codigo);
ALTER TABLE Trazabilidad ADD CONSTRAINT t_reclamo_codigo_fk FOREIGN KEY (reclamo,anio)
REFERENCES Reclamo(codigo,anio);
ALTER TABLE Trazabilidad ALTER COLUMN codigo ADD GENERATED ALWAYS AS IDENTITY;
ALTER TABLE Trazabilidad ADD CONSTRAINT chk_validar_fechas check(fecha_inicio <= fecha_fin);
ALTER TABLE Trazabilidad ADD CONSTRAINT p_recorden_unique UNIQUE (reclamo,anio,estado);

--Medida
ALTER TABLE Medida ADD CONSTRAINT me_codigo_pk PRIMARY KEY (codigo);
ALTER TABLE Medida ADD CONSTRAINT me_reclamo_codigo_fk FOREIGN KEY (reclamo,anio)
REFERENCES Reclamo(codigo,anio);
ALTER TABLE Medida ALTER COLUMN codigo ADD GENERATED ALWAYS AS IDENTITY;

--Participa
```

```sql
ALTER TABLE Participa ADD CONSTRAINT p_reclamo_codigo_pk PRIMARY KEY (reclamo,anio,servicio)
    ;
ALTER TABLE Participa
    ADD CONSTRAINT p_reclamo_fk FOREIGN KEY (reclamo,anio) REFERENCES Reclamo(codigo,anio),
    ADD CONSTRAINT p_servicio_fk FOREIGN KEY (servicio) REFERENCES Servicio(codigo);
```

4.2 Data Loading

The data for the master tables (`center`, `outcome`, `location`, `service`, `communication_type`) was inserted based on information derived from real-world cases.

4.3 Simulation of Missing Data

For generating missing data, Python scripts were used with the support of libraries such as `datetime`, `faker`, and `psycopg2`. This approach helped generate highly consistent data that adhered to the various constraints imposed by our tables. As we know, the queries needed to be conducted in four contexts: 1k, 10k, 100k, and 1M records. Using Python for data generation allowed for seamless scaling of the number of generated records.
Additionally, the `measure` table and one column of the `claim` table were populated through the following triggers.

```sql
CREATE OR REPLACE FUNCTION agregarResultado() RETURNS TRIGGER AS $$
BEGIN
    IF EXISTS(SELECT * FROM reclamo r
            WHERE r.codigo = NEW.reclamo
            AND r.anio = NEW.anio
            AND NEW.estado = 'NOTIFICADO'
            AND NEW.fecha_fin IS NOT NULL)
    THEN UPDATE reclamo r SET resultado = ROUND(RANDOM() + 1)
            WHERE r.codigo = NEW.reclamo
            AND r.anio = NEW.anio;
    END IF;
    RETURN NEW;
    END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trigger_agregarResultado
AFTER INSERT OR UPDATE ON trazabilidad
FOR EACH ROW EXECUTE PROCEDURE agregarResultado();

CREATE OR REPLACE FUNCTION agregarMedida()
RETURNS trigger AS $$
DECLARE
    cantidad INT := ROUND(RANDOM()*5+0.5);
BEGIN
    IF (NEW.resultado = 1) THEN
    FOR i IN 1..cantidad
    LOOP
        INSERT INTO medida (reclamo, anio) VALUES (NEW.codigo,NEW.anio);
    END LOOP;
    END IF;
RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trigger_crear_medida
AFTER INSERT OR UPDATE ON reclamo
FOR EACH ROW
EXECUTE FUNCTION agregarMedida();
```

5 Optimization and Experimentation

This section describes a series of tests designed to evaluate the performance of our database. These tests are based on three SQL queries of varying complexity. It is important to highlight that these experiments will be conducted using different data volumes: 1K, 10K, 100K, and 1M records. Furthermore, we will perform these tests in two distinct scenarios: one without indexes and the other with indexes. This approach allows us to determine whether the presence of indexes has a positive effect on optimizing database processes.

5.1 SQL Queries for the Experiment

5.1.1 Description of the Selected Queries

Query 1

- **Question:** How many valid claims were generated each year where the reason starts with the letter 'A,' the patient and the applicant are not the same person, and no more than one service was involved?
- **Justification:** Understanding the quantity and nature of claims filed on behalf of others helps identify patterns and trends in complaints. This information can assist in focusing improvements in specific areas of service and monitoring the effectiveness of claim resolution strategies over time.

Query 2

- **Question:** What is the most frequent reason for valid claims resolved within 30 days during the first half of 2023?
- **Justification:** Identifying the primary reason for claims resolved quickly provides insights into which hospital areas are efficiently managing issues. This information can guide the improvement of protocols, development of new claim management strategies, and proactive anticipation of potential problems.

Query 3

- **Question:** What is the number of valid claims with no more than one registered measure for each year, month, and communication type?
- **Justification:** Analyzing the number of claims resolved with a single measure, categorized by year, month, and communication type, offers a comprehensive view of the effectiveness of resolution actions and their relationship with communication methods. This perspective can drive optimization in claim management and communication strategies.

5.2 SQL Query Implementation

Query 1

```
SELECT anio, COUNT(CONCAT(codigo,anio)) AS cantidad
FROM (
    SELECT r.codigo,r.anio FROM
    (SELECT codigo,anio
    FROM reclamo
    WHERE resultado = 1
        AND motivo IN
        (SELECT codigo
        FROM motivo
        WHERE substr(descripcion,1,1) = 'A')
        AND representacion IN
            (SELECT codigo
            FROM representacion
```

```
            WHERE dni_paciente <> dni_solicitante)) AS r
    INNER JOIN
        (SELECT reclamo,anio
        FROM participa
        GROUP BY reclamo,anio
        HAVING COUNT(CONCAT(reclamo,anio)) =  1) AS p
    ON r.codigo = p.reclamo AND r.anio = p.anio) AS reclamos
GROUP BY anio;
```

Query 2

```
SELECT m.descripcion AS Motivo,tmp_ConteoMotivo.conteo
 FROM motivo m
     INNER JOIN
        (SELECT motivo AS codigo, COUNT(motivo) AS conteo
        FROM reclamo
        WHERE resultado = 1
          AND codigo IN (
            SELECT reclamo
            FROM (
                SELECT reclamo, anio, EXTRACT (DAYS FROM AGE(fecha_fin, fecha_inicio)) AS
    dias
                FROM (
                    SELECT reclamo, anio, fecha_inicio, fecha_fin
                    FROM trazabilidad
                    WHERE anio = 2023 AND (fecha_fin IS NOT NULL AND fecha_fin <= '
    2023-06-30')) AS t
                WHERE reclamo IN (
                    SELECT reclamo
                    FROM trazabilidad
                    WHERE anio = 2023
                      AND (fecha_fin IS NOT NULL AND fecha_fin <= '2023-06-30')
                    GROUP BY reclamo
                    HAVING COUNT(reclamo) = 5)) AS tmp_TiempoReclamo
                GROUP BY reclamo
                HAVING SUM(dias) <= 30)
        GROUP BY motivo) AS tmp_ConteoMotivo USING (codigo)
 ORDER BY conteo DESC
 LIMIT 1;
```

Query 3

```
SELECT tmp_Detalle.anio,EXTRACT(MONTH FROM fecha_fin) AS mes,tmp_Detalle.TipoComunicacion,
    COUNT(CONCAT(tmp_Detalle.codigo,tmp_Detalle.anio)) AS Cantidad
FROM trazabilidad t
INNER JOIN
    (SELECT tc.descripcion AS TipoComunicacion, tmp_Comunicacion.codigo,anio
    FROM tipo_comunicacion tc
    INNER JOIN
        (SELECT codigo,anio,comunicacion
        FROM reclamo
        WHERE resultado = 1
          AND CONCAT(codigo,anio) IN
            (SELECT DISTINCT CONCAT(reclamo,anio)
            FROM medida
            GROUP BY reclamo,anio
            HAVING COUNT(CONCAT(reclamo,anio)) = 1)) AS tmp_Comunicacion
    ON tmp_Comunicacion.comunicacion = tc.codigo) AS tmp_Detalle
ON tmp_Detalle.codigo = t.codigo AND tmp_Detalle.anio = t.anio
GROUP BY tmp_Detalle.anio, mes,tmp_Detalle.TipoComunicacion
ORDER BY tmp_Detalle.anio,mes;
```

5.3 Experiment Methodology

Prior to starting the tests, we will set up four project instances, each containing a different number of tuples: 1,000 (1K), 10,000 (10K), 100,000 (100K), and 1,000,000 (1M). To analyze the efficiency of the three formulated queries, a uniform procedure will be applied for each project instance: the queries will be executed in scenarios both with and without indexes.

5.3.1 Test Without Indexes

This test will be conducted four times, and the average of these observations will be used for comparison. Before implementing the following procedure, default indexes provided by PostgreSQL must be disabled. This can be done using the following command:

```
SET enable_mergejoin TO OFF ;
SET enable_hashjoin TO OFF ;
SET enable_bitmapscan TO OFF ;
SET enable_sort TO OFF ;
```

The execution plan will be obtained using the pgAdmin 4 tool. We will also use the `VACUUM` command on each table to clear the cache every time the query is executed:

```
VACUUM centro;
VACUUM medida;
VACUUM motivo;
VACUUM paciente;
VACUUM participa;
VACUUM persona;
VACUUM reclamo;
VACUUM representacion;
VACUUM resultado;
VACUUM sede;
VACUUM servicio;
VACUUM solicitante;
VACUUM telefono;
VACUUM tipo_comunicacion;
VACUUM trazabilidad;
```

Finally, we will use the `EXPLAIN ANALYZE` command alongside the query to obtain the corresponding execution plan and evaluate its performance:

```
EXPLAIN (ANALYZE,BUFFERS)
```

5.3.2 Test With Indexes

This experiment will also be conducted four times, with the average of the observations used for comparison. Before proceeding with the process, suggested indexes will be generated to enhance query efficiency, and the execution plan will be obtained using the pgAdmin tool.

Similar to the tests without indexes, the `VACUUM` command will be used to clear the cache of each table, and the previously disabled default indexes will be reactivated:

```
VACUUM centro;
VACUUM medida;
VACUUM motivo;
VACUUM paciente;
VACUUM participa;
VACUUM persona;
VACUUM reclamo;
VACUUM representacion;
VACUUM resultado;
VACUUM sede;
VACUUM servicio;
VACUUM solicitante;
```

```
VACUUM telefono;
VACUUM tipo_comunicacion;
VACUUM trazabilidad;

SET enable_mergejoin TO ON ;
SET enable_hashjoin TO ON ;
SET enable_bitmapscan TO ON ;
SET enable_sort TO ON ;
```

5.4 Query Optimization

5.4.1 Indexes Proposed for Query 1

```
-- Primera Consulta
CREATE INDEX index_representacion ON representacion USING hash(codigo);
CREATE INDEX index_reclamo ON reclamo USING hash(representacion);
```

For the first query, a hash join is used. This approach is effective as each claim is associated with a single representation, allowing for exact matching. A hash is applied to both, enabling efficient one-to-one lookups.

5.4.2 Indexes Proposed for Query 2

```
-- Segunda Consulta
CREATE INDEX index_trazabilidad ON trazabilidad USING btree(fecha_fin);
```

For the second query, a range search is performed to identify claims resolved within the first half of 2023. Using a B-Tree index optimizes the query by facilitating the search for dates between 01-01-2023 and 30-06-2023.

5.4.3 Indexes Proposed for Query 3

```
-- Indices por defecto
```

For this query, the indexes generated from the primary key assignments are utilized, as seen in the case of the claim and communication type tables.

5.5 Time Measurement

5.5.1 Without Indexes

QUERY 1				
Execution (ms)	**1k**	**10k**	**100k**	**1M**
1	108.870	71.372	185.123	4908.759
2	110.067	68.073	179.612	4811.423
3	96.559	51.978	180.431	4688.640
4	110.605	47.633	186.805	4081.072
Average	106.525	59.764	182.993	4622.4735
Standard Deviation	6.684	11.712	3.043	110.304

Table 17: Query 1 without indexes

QUERY 2				
Execution (ms)	**1k**	**10k**	**100k**	**1M**
1	349.170	521.696	1139.119	8735.086
2	350.245	547.986	1206.105	8526.298
3	318.156	526.108	1190.019	7986.034
4	325.198	521.989	1124.042	7503.829
Average	335.692	529.448	1164.821	8187.812
Standard Deviation	16.442	12.524	39.424	554.566

Table 18: Query 2 without indexes

QUERY 3				
Execution (ms)	**1k**	**10k**	**100k**	**1M**
1	437.750	1613.69	13614.716	870037.980
2	435.294	1903.50	13682.236	881396.742
3	431.614	1858.36	13701.981	728800.802
4	433.268	1771.89	13701.981	1108735.396
Average	433.961	1786.86	13675.228	897242.730
Standard Deviation	2.648	127.709	41.400	157154.939

Table 19: Query 3 without indexes

5.5.2 With Indexes

QUERY 1				
Execution (ms)	**1k**	**10k**	**100k**	**1M**
1	8.781	39.124	108.296	2146.958
2	8.418	37.331	108.671	1481.302
3	7.775	31.477	103.070	1954.710
4	8.656	32.888	117.170	1772.298
Average	8.407	35.205	109.467	1838.817
Standard Deviation	0.448	3.612	5.053	283.210

Table 20: Query 1 with indexes

QUERY 2				
Execution (ms)	**1k**	**10k**	**100k**	**1M**
1	4.517	13.873	134.150	8428.673
2	4.379	22.264	126.620	7828.081
3	4.559	24.684	141.791	4081.568
4	4.465	13.621	133.776	2345.733
Average	4.480	18.610	134.084	5671.014
Standard Deviation	0.077	5.703	6.197	2934.931

Table 21: Query 2 with indexes

QUERY 3				
Execution (ms)	**1k**	**10k**	**100k**	**1M**
1	5.631	30.021	88.798	6542.494
2	5.341	20.930	85.641	4871.446
3	5.127	19.881	86.596	4618.429
4	5.287	22.262	86.232	4699.169
Average	5.220	23.274	86.817	5182.885
Standard Deviation	0.210	4.602	1.193	912.528

Table 22: Query 3 with indexes

5.6 Analysis and Discussion

For the first query, the implementation of indexes resulted in a notable improvement. Execution times significantly decreased across all data sizes. This suggests that Query 1 performs a search that directly benefits from indexing.

The second query also showed improvement with the use of indexes; however, the difference in the 1M dataset was not as pronounced as in Query 1. This might indicate that the efficiency of the index could be affected by the growing amount of data, possibly due to the data structure or the complexity of the query.

Finally, the third query exhibited the greatest improvement with the implementation of indexes. The reduction in execution time was exceptionally significant, especially in the 1M dataset, where the average time decreased by a factor of over 100.

Therefore, while indexes generally improve query time, their impact may vary depending on the nature of the query and the data structure. A careful analysis is recommended to determine where to effectively implement indexes.

6 Conclusions

1. The design of entity-relationship and relational models lays the foundation for the database structure that represents the business process.

2. The tables underwent a normalization process, in addition to generating constraints and implementing triggers and stored procedures to maintain data integrity.

3. Increasing the level of complexity highlights more noticeably how execution times are reduced with the use of indexes, as well as how they are impacted as the number of tuples increases.

7 Annex

7.1 Normalization of the Database

In this section, it will be demonstrated that our database satisfies the Third Normal Form (3NF).

Person(dni,name, last_name_maternal, last_name_paternal, gender, birth_date, email, address, district, doc_type)

Person = $\{\text{dni} \rightarrow name, last\_name\_maternal, last\_name\_paternal, gender, birth\_date, email, address, district, doc\_type\}$

dni is a superkey in the set of functional dependencies. Therefore, it satisfies the Third Normal Form (3NF).

Phone(phone, Person.dni)

Phone = $\{\text{phone} \rightarrow dni\}$

phone is a superkey in the set of functional dependencies. Therefore, it satisfies the Third Normal Form (3NF).

Patient(Person.dni,NHC)

Patient = {dni $\rightarrow NHC$}
$dni is a superkey in the set of functional dependencies. Therefore, it satisfies the Third Normal Form (3NF)$.

Representation(code,
Person.dniRequester,Person.dniPatient)

Representation = {code $\rightarrow dniRequester, dniPatient$}

code is a superkey in the set of functional dependencies. Therefore, it satisfies the Third Normal Form (3NF).

Claim(code,year,Representation.code,Reason.code,Result.code,Communication_Type.code)

Claim = {code, year $\rightarrow representation, reason, result, communication\_type$}

code(claim) and year are superkeys in the set of functional dependencies. Therefore, it satisfies the Third Normal Form (3NF).

Traceability(code,Claim.code,Claim.year,status, start_date, end_date, description, comments)

Traceability = {code $\rightarrow claim, year, start\_date, end\_date, description, comments$}

code is a superkey in the set of functional dependencies. Therefore, it satisfies the Third Normal Form (3NF).

Communication_Type(code,description)

Communication_Type = {code $\rightarrow description$}

code is a superkey in the set of functional dependencies. Therefore, it satisfies the Third Normal Form (3NF).

Reason(code,description)

Reason = {code $\rightarrow description$}

Code is a superkey in the set of functional dependencies. Therefore, it satisfies the Third Normal Form (3NF).

Result(code,description)

Result = {code $\rightarrow description$}

code is a superkey in the set of functional dependencies. Therefore, it satisfies the Third Normal Form (3NF).

Measure(code,Claim.code,Claim.year,description)

Measure = {code $\rightarrow$ *claim*, *year*, *description*}

code is a superkey in the set of functional dependencies. Therefore, it satisfies the Third Normal Form (3NF).

Participates(Claim.code,Claim.year,Service.code,order)

Participates = {claim, year, service $\rightarrow$ *order*}

claim(code), year, and service are superkeys in the set of functional dependencies. Therefore, it satisfies the Third Normal Form (3NF).

Service(code,Center.code,description)

Service = {code $\rightarrow$ *center*, *description*}

code is a superkey in the set of functional dependencies. Therefore, it satisfies the Third Normal Form (3NF).

Center(code,Headquarter.code,description)

Center = {code $\rightarrow$ *headquarter*, *description*}

code is a superkey in the set of functional dependencies. Therefore, it satisfies the Third Normal Form (3NF).

7.2 Physical Model

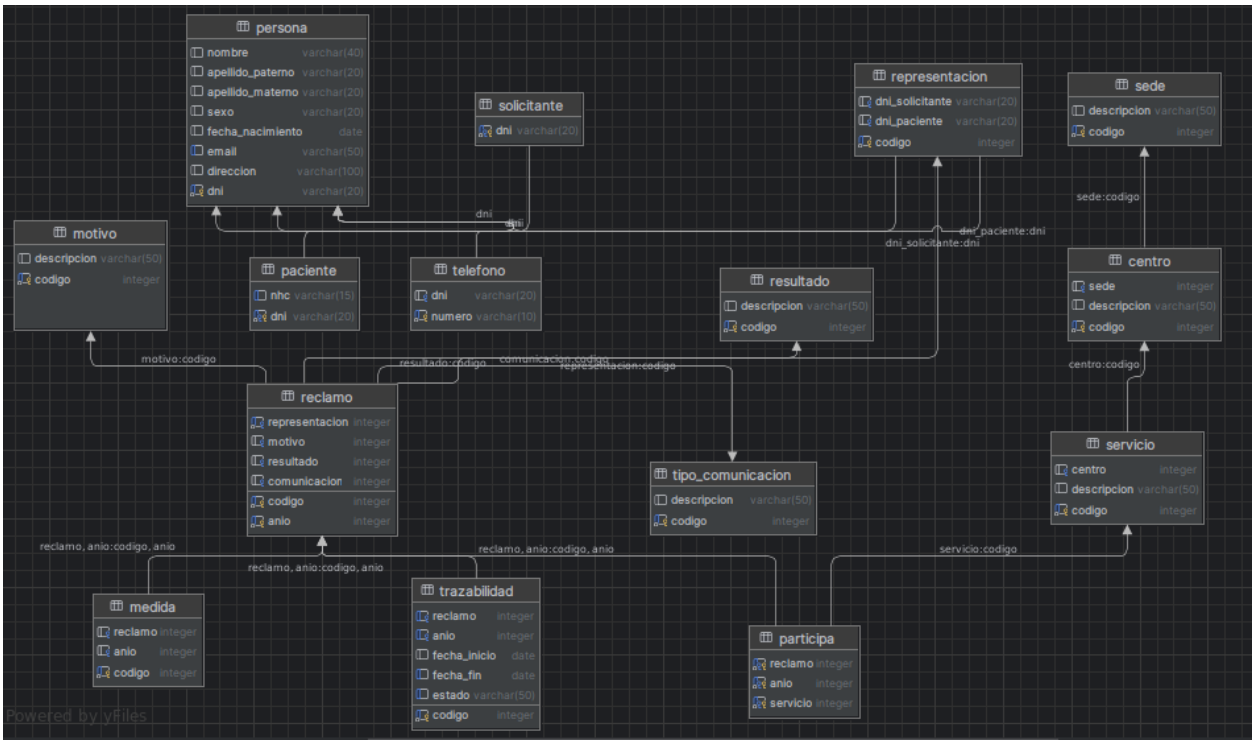

Figure 2: Physical Model

7.3 Experimentation Videos

`https://drive.google.com/drive/folders/14Qyign2R0Cs4Zn9vK78vzD6zdvAq8ZTt?usp=sharing`

7.4 Additional Question

What will be the operational complexity if we scale the data above one million? Perform a comparison with the amount of data in the previous section. Is the Client-Server architecture sufficient to process millions of records?

The operational complexity of scaling data beyond one million increases due to factors such as processing time and resource usage. Without an efficient scaling strategy, response times may degrade, complicating data analysis for decision-making. Comparatively, the increase in data relative to the previous amount will escalate the workload and the resources required to process and analyze this data, potentially resulting in a decline in efficiency. Regarding the Client-Server architecture, it may be sufficient if the server has adequate resources. However, it might be advisable to migrate to a distributed database architecture to enhance scalability and performance when handling millions of records.