

**Trabajo de desarrollo**

**Aplicación médica en python**

**con conexión a PostgreSQL**

**Integrantes: Tirso Alvarado**

**Ignacio Avalos**

**Diego Brito**

**Alberto Olvivares**

**Ruth Sánchez**

**Caso de estudio**

**Introducción**

En el siguiente trabajo se presenta una aplicación en consola desarrollada con Python y el motor de bases de datos PostgreSQL, que simula una ficha de un paciente asociado a entidades como cama, habitación, médico, enfermedad y diagnóstico. El programa permite ingresar datos y también realizar consultas de los datos del paciente a través de la consola, incluyendo el nombre, RUT del paciente, su cama asociada, la habitación, el médico tratante, entre otros. En el desarrollo de esta aplicación, se utilizó Anaconda Navigator como entorno virtual y Jupyter Notebook en su versión 6.5.4 como IDE.

**Desarrollo**

De acuerdo a lo solicitado, se comienza con el diseño de un modelo relacional. Para ello, se definen las claves primarias, las claves foráneas y los tipos de datos.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Tabla** | **Variable** | **Tipo de Dato** | **Relación** | **Restricciones** |
| **Habitacion** | id\_habitacion | SERIAL | Primaria | PRIMARY KEY |
|  | numero\_habitacion | INT | Única | UNIQUE |
| **Cama** | id\_cama | SERIAL | Primaria | PRIMARY KEY |
|  | numero\_cama | INT | - | - |
|  | id\_habitacion | INT | Foránea (Habitacion) | FOREIGN KEY |
| **Paciente** | id\_paciente | SERIAL | Primaria | PRIMARY KEY |
|  | nombre\_paciente | VARCHAR(255) | - | - |
|  | rut | VARCHAR(255) | Única | UNIQUE |
|  | fecha\_ingreso | DATE | - | - |
|  | fecha\_alta | DATE | - | - |
|  | id\_cama | INT | Foránea (Cama) | FOREIGN KEY |
| **Medico** | id\_medico | SERIAL | Primaria | PRIMARY KEY |
|  | nombre\_medico | VARCHAR(255) | - | - |
| **Examen** | id\_examen | SERIAL | Primaria | PRIMARY KEY |
|  | nombre\_examen | VARCHAR(255) | - | - |
|  | tipo\_examen | VARCHAR(255) | - | - |
| **Diagnostico** | id\_diagnostico | SERIAL | Primaria | PRIMARY KEY |
|  | id\_paciente | INT | Foránea (Paciente) | FOREIGN KEY |
|  | id\_medico | INT | Foránea (Medico) | FOREIGN KEY |
|  | comentarios\_diagnostico | TEXT | - | - |
|  | fecha\_diagnostico | DATE | - | - |
| **Enfermedad** | id\_enfermedad | SERIAL | Primaria | PRIMARY KEY |
|  | id\_diagnostico | INT | Foránea (Diagnostico) | FOREIGN KEY |
|  | nombre\_enfermedad | VARCHAR(255) | - | - |
| **Orden** | id\_orden | SERIAL | Primaria | PRIMARY KEY |
|  | id\_paciente | INT | Foránea (Paciente) | FOREIGN KEY |
|  | id\_medico | INT | Foránea (Medico) | FOREIGN KEY |
|  | id\_examen | INT | Foránea (Examen) | FOREIGN KEY |
|  | fecha\_orden | DATE | - | - |
|  | comentarios\_orden | TEXT | - | - |

Tabla 1. Estructura SQL.

**Serial**: Se utilizó para definir un valor único y además incremental. Se hizo uso en todas las ID.

**Primary Key**: Se crearon ID propios para cada entidad. No se utilizó el número de habitación como ID, porque podría haber unicidad *no* asegurada.

**Foránea**: Se utilizaron para establecer vínculos entre entidades.

**Unique**: Se utilizó para asegurar que los valores de la columna sean únicos, por ejemplo, el Rut. Garantiza unicidad.

**INT**: Se asignaron a las variables que son números enteros.

**Vachar**: Para variables que usan caracteres.

De la tabla 1, se crea la Query SQL para PostgreSQL,

CREATE TABLE Habitacion (

id\_habitacion SERIAL PRIMARY KEY,

numero\_habitacion INT UNIQUE

);

CREATE TABLE Cama (

id\_cama SERIAL PRIMARY KEY,

numero\_cama INT,

id\_habitacion INT,

FOREIGN KEY (id\_habitacion) REFERENCES Habitacion(id\_habitacion)

);

CREATE TABLE Paciente (

id\_paciente SERIAL PRIMARY KEY,

nombre\_paciente VARCHAR(255),

rut VARCHAR(255) UNIQUE,

fecha\_ingreso DATE,

fecha\_alta DATE,

id\_cama INT,

FOREIGN KEY (id\_cama) REFERENCES Cama(id\_cama)

);

CREATE TABLE Medico (

id\_medico SERIAL PRIMARY KEY,

nombre\_medico VARCHAR(255)

);

CREATE TABLE Examen (

id\_examen SERIAL PRIMARY KEY,

nombre\_examen VARCHAR(255),

tipo\_examen VARCHAR(255)

);

CREATE TABLE Diagnostico (

id\_diagnostico SERIAL PRIMARY KEY,

id\_paciente INT,

id\_medico INT,

comentarios\_diagnostico TEXT,

fecha\_diagnostico DATE,

FOREIGN KEY (id\_paciente) REFERENCES Paciente(id\_paciente),

FOREIGN KEY (id\_medico) REFERENCES Medico(id\_medico)

);

CREATE TABLE Enfermedad (

id\_enfermedad SERIAL PRIMARY KEY,

id\_diagnostico INT,

nombre\_enfermedad VARCHAR(255),

FOREIGN KEY (id\_diagnostico) REFERENCES Diagnostico(id\_diagnostico)

);

CREATE TABLE Orden (

id\_orden SERIAL PRIMARY KEY,

id\_paciente INT,

id\_medico INT,

id\_examen INT,

fecha\_orden DATE,

comentarios\_orden TEXT,

FOREIGN KEY (id\_paciente) REFERENCES Paciente(id\_paciente),

FOREIGN KEY (id\_medico) REFERENCES Medico(id\_medico),

FOREIGN KEY (id\_examen) REFERENCES Examen(id\_examen)

);

Una vez creada la consulta, se crea la base de datos en Pgadmin 4.

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Se ejecuta la consulta SQL en Pacientes2-> Query Tool

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Para el desarrollo de este proyecto, se trabajará en Jupyter Notebook, desde Anaconda Navigator, con el environment “base”

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

**Creación de script Python**

Para el desarrollo de la aplicación, se utilizó programación sin orientación a objetos. Se separó el programa en 3 archivos. Un archivo de configuración, un script para poblar, y finalmente el script que contiene el menú para consultar los datos del paciente.

**Config.py**

Es el archivo que contiene los datos que permitirán establecer la conexión a la base de datos.

HOST = 'localhost'

DATABASE = 'nombredb'

USER = 'postgres'

PASSWORD = 'pass'

**Poblar.py**

Este script será el encargado de insertar los datos en la base de datos. Puede insertar datos hasta que el usuario indique no seguir ingresando, para ello, se utilizará While True.

**Consultas.py**

Finalmente se crea el archivo que permitirá consultar los datos del paciente e ingresar cambios en la base de datos.

Por practicidad para el trabajo con la base de datos, se creó un archivo de inserciones “insertsql.sql”

**Funcionamiento de la aplicación por consola.**

**Selector de opciones**: Opciones 1,2,3,4,5,6,7

**Texto

Descripción generada automáticamente**

**Opción: 1.** Muestra a todos los pacientes ingresados, su rut, nombre, enfermedad, medico y numero de habitación.

Texto

Descripción generada automáticamente

**Opción 2.** Muestra los detalles de un paciente por rut. En este caso se consultó el paciente con rut 111111111k, que entrega como resultado, el nombre, enfermedad, entre otros.

Texto, Carta

Descripción generada automáticamente

**Opción 3:** Cambiar a un paciente de cama, por ejemplo, al 111111111k, que se encuentra en la cama 5 de la habitación 103.

Texto

Descripción generada automáticamente

Al consultar al paciente, ahora se encuentra en la cama 1 de habitación 103.

Texto

Descripción generada automáticamente

**Opción 4.** Se cambia el médico del paciente 111111111k, por el “Médico 1”

Texto, Carta

Descripción generada automáticamente

Al consultar por el rut, ahora el paciente tiene asignado al “Médico 1”.

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

**Opción 5**. Crear nueva habitación, en este caso se crea la numero 820.

Texto

Descripción generada automáticamente

Se puede ver en la última fila, la habitación 820 creada.

Tabla

Descripción generada automáticamente

**Opción 6:** Crear una cama. Se crea la cama 7, para la habitación 101.

Texto

Descripción generada automáticamente

Aquí se puede ver la cama 7 añadida a la habitacion 101.

Tabla

Descripción generada automáticamente

**Opción 7**. Sale del programa.

**Conclusiones**

El diseño de la base de datos relacional permitió almacenar los datos y darles persistencia en tablas estructuradas para su posterior consulta. En este proceso de creación, se definieron las entidades y las relaciones que se generaron entre ellas. Además, es fundamental diferenciar SERIAL, UNIQUE, claves primarias y claves foráneas, para garantizar la consistencia y la integridad de los datos. El script de Python “poblar.py”, permitió la solicitud de ingreso de datos, aunque no automatizó completamente el proceso. Una solución más automatizada podría haber sido leer los datos, previamente cargados de los pacientes, desde un archivo Excel, CSV, etc., utilizando pandas y estableciendo la conexión a PostgreSQL. Por último, el script de Python “consultas.py”, demostró la versatilidad de este lenguaje para trabajar con bases de datos, facilitando tanto la inserción como la recuperación de información.