Escuela de
Ingeniería y Arquitectura
Universidad Zaragoza

Universidad de Zaragoza

Facultad de Ingeniería

Memoria Práctica 1

Bases de Datos 2

Miércoles A 9:00-11:00

Sergio Isla Lasheras - 873983

Irene Pascual Albericio - 871627

Athanasios Usero Samarás - 839543

Febrero de 2025

Contents

1 Introduction

El objetivo de la presente práctica es el de instalar y configurar los diversos sistemas gestores de bases de datos con los que se trabajará posteriormente en las sesiones prácticas de la asignatura. Pero además, se aprovechará la ocasión para introducirse en las particularidades (aunque a grandes rasgos) de cada sistema gestor: como se gestionan los usuarios, roles y credenciales, como se estructura el espacio de trabajo, construcción semántica y sintáctica de las bases de datos y operaciones de consulta y manipulación básicas; así como sistema de puertos y conectividad externa.

Pero antes de proceder a la configuración de cada gestor, se llevó a cabo un proceso de consenso para elegir la mejor, o aunque fuese la menos arriesgada, plataforma de despliegue. La distribución del trabajo y los esfuerzos dedicados por cada integrante figuran en el apartado Esfuerzos invertidos. Asimismo, se probará la conexión externa para cada gestor por medio de la plataforma *Dbeaver*.
Los SGBD que se van a instalar y configurar son:

- **PostgreSQL**: Sistema de gestión de bases de datos relacional orientado a objetos y de código abierto. Es muy potente, dado que cumple con los estándares SQL, a la par que proporciona una gran flexibilidad y extensibilidad.
- **Oracle XE**: SGBD gratuito de Oracle, que proporciona una funcionalidad reducida de la versión empresarial de la misma. De todos modos, se muestra una opción acertada para el aprendizaje de Oracle y los modelos relacionales, además de que permite acceder a funcionalidades avanzadas dado que es totalmente compatible con PL/SQL.
- **Apache Cassandra**: SGBD NoSQL distribuido de código abierto diseñada para ser fácilmente escalable y tolerante a fallos, lo que la convierte en una elección extendida cuando se trabaja sobre una red con grandes volúmenes de datos. Además, no trabaja sobre SQL, sino CLQ, lo que podrá ser enriquecedor por el hecho de forzar a paliar algunas limitaciones del lenguaje (como la inexistencia de JOINS) con las funcionalidades de las bases de datos distribuidas.
- **Apache Hbase**: SGBD NoSQL orientada a columnas, diseñada también para manejar grandes volúmenes de datos distribuidos con una gran escalabilidad. Parece presentar algunas similitudes con Cassandra, pero ambas difieren en sus arquitecturas y escenarios de uso (por ejemplo, Cassandra emplea una arquitecturan *peer-to-peer* mientras que Hbase una arquitectura principal-secundaria).
- **IBM DB2**: SGBD relacional ampliamente empleado por empresas que trabajan con datos complejos. Esto se debe a que proporciona un alto rendimiento trabajando sobre SQL pero además soporta modelos híbridos incorporando otros NoSQL (Json, por columnas, etc.)

2 Configuración del entorno de trabajo (máquina virtual / uso de Docker)

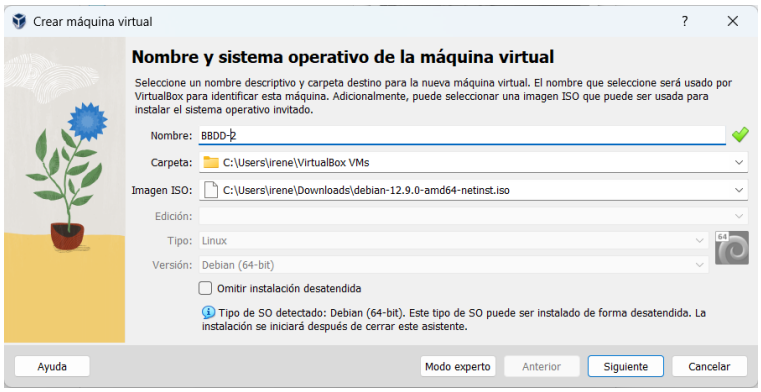
La fase inicial de elaboración de la práctica ocupó la decisión de *cómo* y *dónde* desplegar los gestores. La primera decisión fue la de emplear contenedores (*dockers*). De este modo, cada gestor se encontraría aislado en su propio entorno, salvándose de interferencias inoportunas. No solo esto, sino que la rapidez y sencillez del despliegue de los contenedores, así como su consistencia e independencia de entornos la hacían idónea, dada la pluralidad de integrantes del grupo y gestores con los que trabajar.

El uso de contenedores, ahora bien, comportó otra decisión de configuración. Como los sistemas Windows pueden ocasionar problemas a la hora de desplegar algunos gestores sobre contenedores, se optó por instalar una máquina virtual sobre la que instalar los contenedores. Tal como se mencionará más adelante, se optará por instalar una imagen de Linux de 64 bits completa, en vez de máquinas más ligeras como *Turnkey Linux*. La razón es la misma que la de instalar la propia máquina virtual: la búsqueda de un sistema estable, flexible y con garantías.

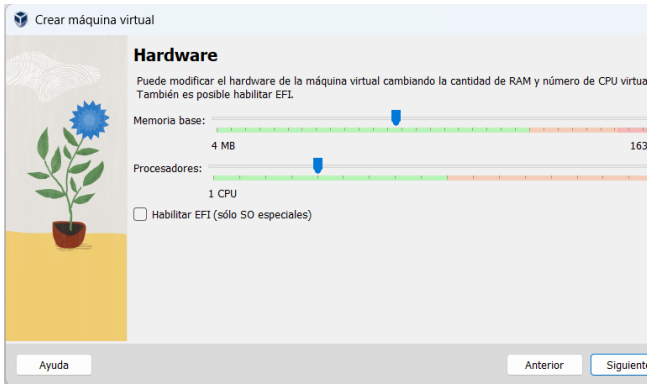
2.1 Instalación de la máquina virtual

La máquina virtual se desplegará sobre *Virtual Box*. Como se encuentra disponible en todas las máquinas del laboratorio, no se detallará su instalación. La imagen de Linux a utilizar será la última versión proporcionada por *Debian*, en nuestro caso la versión 12.9.0 de 64 bits.

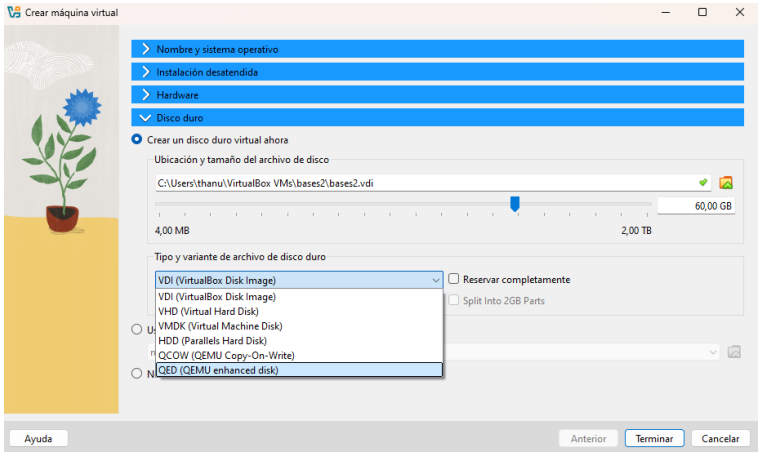
En cuanto a las características de la máquina virtual, se optó por un hardware de 5 procesadores y aproximadamente 6.3 GB, así como un discor duro de 60GB; lo que permitirá holgadamente ejecutar el sistema operativo sin problemas (como mucho, demasiado holgadamente).
Se incluyen algunas imagenes ilustrativas del proceso (también se tuvo que configurar el NAT y cambiar el idioma del teclado, pero son procesos que ya vienen explicados en el material de apoyo):



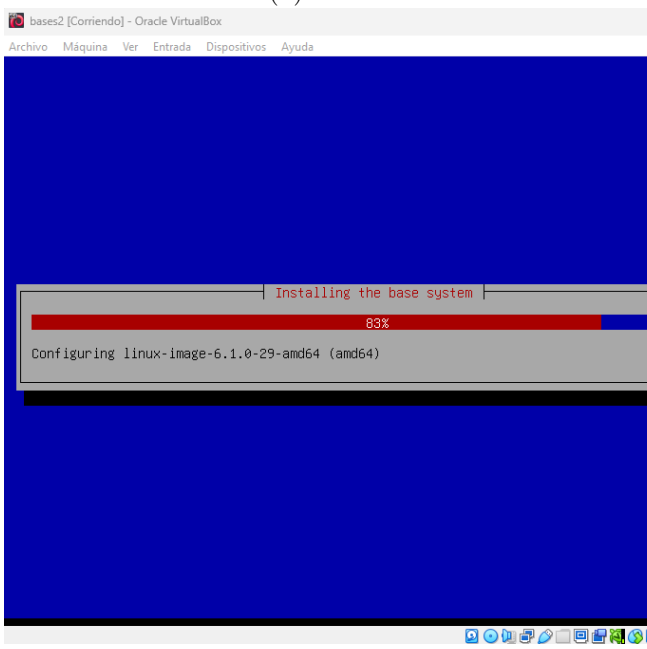
(a) Importar imagen



(b) conf. Hardware



(c) Disco duro



(d) Instalación Debian

Figure 1: Instalación de la máquina virtual Debian

2.2 Instalación de *Docker*

El primer paso será instalar *Docker* en Debian. Para ello, será necesario seguir los siguientes pasos:

```
# Actualizar paquetes para asegurar que el sistema esté actualizado.
sudo apt update
sudo apt upgrade -y
#Instalar las dependencias necesarias para poder acceder a repositorios y descargar ima
sudo apt install ca-certificates curl gnupg lsb-release
# Añadir clave GPG de Docker (para verificar autenticidad de paquetes)
curl -fsSL https://download.docker.com/linux/debian/gpg | sudo gpg --dearmor -o /usr/sh
# Añadir repositorio oficial de Docker
echo \
"deb [arch=$(dpkg --print-architecture) signed-by=/usr/share/keyrings/docker-archive-ke
# Instalar Docker Engine
sudo apt update
sudo apt install docker-ce docker-ce-cli containerd.io
```

Una vez se ha instalado Docker, puede ser (como en el caso del presente grupo), que *Docker Compose* no venga incluido en sus herramientas (en todo caso, comprobar con `sudo docker compose version`). En ese caso, se instalará de la siguiente manera:

```
# Traer del repositorio oficial
sudo curl -L "https://github.com/docker/compose/releases/latest/download/docker-compose
$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose
# Dar permisos de ejecución
sudo chmod +x /usr/local/bin/docker-compose
```

3 Instalación y Administración Básica de los SGBD

Para automatizar el despliegue de los contendores, se ha creado un archivo `docker-compose.yml`,el cual permitirá lanzar los contenedores fácilmente a todos los integrantes del equipo:

```yaml
version: '3.8'

services:
  postgres:
    image: postgres:latest
    container_name: postgres-db
    environment:
      POSTGRES_USER: admin
      POSTGRES_PASSWORD: admin123
    ports:
      - "5432:5432"
    volumes:
      - pg_data:/var/lib/postgresql/data

  oracle:
    image: container-registry.oracle.com/database/express:latest
    container_name: oracle-xe
    environment:
      ORACLE_PWD: oracle123
    ports:
      - "1521:1521"
      - "5500:5500"
    volumes:
      - oracle_data:/opt/oracle/oradata

  cassandra:
    image: cassandra:latest
    container_name: cassandra-db
    environment:
      CASSANDRA_CLUSTER_NAME: "MyCluster"
    ports:
      - "9042:9042"
    volumes:
      - cassandra_data:/var/lib/cassandra

  hbase:
    image: dajobe/hbase
    container_name: hbase-db
    ports:
      - "16010:16010"
      - "9090:9090"
    volumes:
      - hbase_data:/opt/hbase-data

  db2:
    image: ibmcom/db2
    container_name: db2-db
    environment:
      DB2INST1_PASSWORD: db2inst1
      LICENSE: accept
    ports:
      - "50000:50000"
    volumes:
      - db2_data:/database

volumes:
  pg_data:
  oracle_data:
  cassandra_data:
  hbase_data:
  db2_data:
```

En él, se estructuran claramente los contenedores para cada gestor. Nótese que en algunos casos se han incluido en *environmen* credenciales de usuarios. En todo caso, por motivos didácticos posteriormente se han modificado. Además, se ha decidido integrar volúmenes para asegurar la persistencia de los datos y se han configurado los puertos (que se probarán con Dbeaver). Por otra parte, se lleva a cabo un manejo explícito de las versiones a instalar. Se trató de que fuesen las más recientes, o las oficiales de *Docker*.

3.1 PostgreSQL

3.1.1 Inicializar el contenedor

La imagen de PostgreSQL instalada es la última versión oficial disponible en Docker Hub, en este caso la versión *PostgreSQL 17.3*.

Terminal

```
azzale@vbox:~$ psql --version
PostgreSQL 17.3 (Debian 17.3-1.pgdg120+1) on x86_64-pc-linux-gnu,
compiled by gcc (Debian 12.2.0-14) 12.2.0, 64-bit

(END) Cancel request sent
log file:
```

La inicialización del contenedor PostgreSQL se realiza mediante:

```
# Levantar contenedor
docker compose up -d postgres
```

Posteriormente, se hicieron las comprobaciones pertinentes para comprobar la versión, y ademas validar que el contenedor está debidamente inicializado. Si el proceso ha salido correctamente se debe obtener una salida así:

Terminal

```
azzale@vbox:~$ docker ps
CONTAINER ID   IMAGE             COMMAND                  CREATED      STATUS
↪  PORTS                                        NAMES
42877b1f3e93   postgres:latest   "docker-entrypoint.s..."   5 days ago   Up 48 minutes
↪  0.0.0.0:5432->5432/tcp, :::5432->5432/tcp   postgres-db
```

En cuanto al contenedor en sí, este, además de definir la versión a instalar, instancia ya además los puertos para conexiones remotas y define variables para crear un administrador (aunque luego se probará a modificar desde dentro del gestor)

Además, se debe mencionar que existen dos maneras de interactuar con el gestor: ejecutar comandos desde la interfaz interactiva de PostgreSQL o ejecutar comandos directamente desde fuera:

```
# 1. ACCEDER A INTERFAZ INTERACTIVA
#   (ejemplo para entrar como administrador)
docker exec -it postgres-db psql -U

# 2. EJECUTAR DIRECTAMENTE DESDE  FUERA
```

```
docker exec -it postgres-db psql -U admin -c "comando SQL"

# 3. PROCEDIMIENTO PARA EJECUTAR SCRIPT
# EJECUTANDO DESDE DENTRO DEL CONTENEDOR NO ES
# NECESARIO AUTENTICAR
docker cp <script.sql> postgres-db:/tmp/<script.sql>
docker exec -it postgres-db psql -U <usuario> -d practicas_bd -f /tmp/<script>
```

En todos los casos donde se trabaje con scripts, se mostrará la ejecución con comandos desde fuera, pero si se trata de comandos aislados se mostrará la ejecución desde dentro de la interfaz por simplicidad.

3.1.2 Creación de la estructura básica del espacio de la base de datos

PostgreSQL, como instancia, no es más que un cluster que permite trabajar con distintas bases de datos aisladas entre sí. Cada base de datos será un conjunto independiente de objetos, los cuales a su vez se agrupan lógicamente en **squemas**. POr defecto, toda báse de datos tiene el esquema *public*. Los objetos (tablas, vistas, etc) serán familiares, al seguir el estándar SQL y haber sido trabajados en la asignatura *Bases de Datos I*.

En nuestro caso, se creará una base de datos para la asignatura, e inicialmente se trabajará con el esquema por defecto, y gestionando los permisos a nivel de base y no de esquema:

crear_Espacio.sq

```
DROP DATABASE IF EXISTS practicas_bd;
CREATE DATABASE practicas_bd;

\c practicas_bd; -- Conecta a la base de datos practicas_bd
                 -- automáticamente
```

3.1.3 Asignación de roles

En PostgreSQL, los usuarios y roles existen a nivel de cluster. Es decir, son compartidos por todas las bases de datos. Así, son los permisos los que permiten controlar el acceso a recursos, bien a nivel de base o de esquemas. Además, se debe incidir en que PostgreSQL establece una diferencia entre usuarios y roles. Por una parte, un rol es una entidad que permite agrupar permisos, mientras que un usuario es un rol básico con permiso para iniciar sesión; el cual además puede heredar permisos de otros roles. Por tanto, nosotros trabajaremos a nivel de roles, instanciando usuarios específicos con esos roles.

El proceso de creación de usuarios y roles es el siguiente:

crear_Usuarios.sq

– Conectar a la base de datos específica practicas_bd – Eliminar usuarios y roles existentes si existen DROP OWNED BY lector CASCADE; DROP OWNED BY escritor CASCADE; DROP OWNED BY admin CASCADE; DROP ROLE IF EXISTS rol_lector; DROP ROLE IF EXISTS rol_escritor; DROP ROLE IF EXISTS admin;
– 1. Crear superusuario (admin) con todos los permisos CREATE ROLE admin WITH LOGIN SUPERUSER CREATEDB CREATEROLE PASSWORD 'admin123';
– 2. Crear rol lector, con solo permisos de lectura CREATE ROLE rol_lector; GRANT CONNECT ON DATABASE practicas_bd TO rol_lector; GRANT USAGE ON SCHEMA public TO rol_lector; GRANT SELECT ON ALL TABLES IN SCHEMA public TO rol_lector; ALTER DEFAULT PRIV-

ILEGES IN SCHEMA public GRANT SELECT ON TABLES TO rol_lector;
–3. Crear rol escritor, con permisos de lectura, escritura y modificación CREATE ROLE rol_escritor; GRANT CONNECT ON DATABASE practicas_bd TO rol_escritor; GRANT USAGE ON SCHEMA public TO rol_escritor; GRANT SELECT, INSERT, UPDATE ON ALL TABLES IN SCHEMA public TO rol_escritor; ALTER DEFAULT PRIVILEGES IN SCHEMA public GRANT SELECT, INSERT, UPDATE ON TABLES TO rol_escritor;
–4. Crear usuario lector con login y contraseña CREATE USER lector WITH PASSWORD 'lector123'; GRANT rol_lector TO lector; – 5. Crear usuario escritor con login y contraseña CREATE USER escritor WITH PASSWORD 'escritor123'; GRANT rol_escritor TO escritor;

Se debe recordar que para ejecutar scripts desde dentro del docker solo se necesita especificar usuario, no contraseña, pues la autenticación viene dada por el mismo; pero cuando alguien se conecte remotamente necesitará ingresar la contraseña correcta para poder acceder como uno (y solo uno de ellos) usuarios definidos.

Terminal

```
azzale@vbox:~/Desktop/postgreSQL$ docker exec -it postgres-db psql -U admin -W
Password:
psql (17.3 (Debian 17.3-1.pgdg120+1))
Type "help" for help.

admin=#
```

Para comprobar que la creación ha sido correcta, podemos ejecutar el comando
du para observar la tabla de usuarios o
dp para observar los permisos de un usuario concreto.

Terminal

```
azzale@vbox:~/Desktop/postgreSQL$ docker exec -it postgres-db psql -U admin -d practicas\_bd -c "\du"
                             List of roles
 Role name   |                         Attributes
-------------+------------------------------------------------------------
 admin       | Superuser, Create role, Create DB, Replication, Bypass RLS
 escritor    |
 lector      |
 rol_escritor | Cannot login
 rol_lector  | Cannot login

azzale@vbox:~/Desktop/postgreSQL$ docker exec -it postgres-db psql -U admin -d practicas\_bd -c "\dp"
                                      Access privileges
 Schema |      Name      |   Type   |   Access privileges    | Column privileges | Policies
--------+----------------+----------+------------------------+-------------------+----------
 public | medicos        | table    | admin=arwdDxtm/admin  +|                   |
        |                |          | lector_rol=r/admin    +|                   |
        |                |          | rol_lector=r/admin    +|                   |
        |                |          | rol_escritor=arw/admin |                   |
 public | pacientes      | table    | admin=arwdDxtm/admin  +|                   |
        |                |          | lector_rol=r/admin    +|                   |
        |                |          | rol_lector=r/admin    +|                   |
        |                |          | rol_escritor=arw/admin |                   |
 public | pruebas        | table    | admin=arwdDxtm/admin  +|                   |
        |                |          | lector_rol=r/admin    +|                   |
        |                |          | rol_lector=r/admin    +|                   |
        |                |          | rol_escritor=arw/admin |                   |
 public | pruebas_id_seq | sequence | admin=rwU/admin        |                   |
(4 rows)
```

Por último, mencionar que la verificación de permisos se realizó en la fase de creación, poblado y cosultas, tratando de realizar diferentes acciones para cada usuario. Y efectivamente, todos funcionaban correctamente, como se puede ver a continuación:

Terminal

```
azzale@vbox:~/Desktop/postgreSQL$ docker exec -it postgres-db psql -U lector -d practicas\_bd -f /tmp/insertarTablas.sql
psql:/tmp/insertarTablas.sql:3: ERROR:  permission denied for table medicos
psql:/tmp/insertarTablas.sql:7: ERROR:  permission denied for table pacientes
psql:/tmp/insertarTablas.sql:11: ERROR:  permission denied for table pacientes
psql:/tmp/insertarTablas.sql:15: ERROR:  permission denied for table pruebas
psql:/tmp/insertarTablas.sql:18: ERROR:  permission denied for table pacientes
```

3.1.4 Conectividad remota

El docker-compose configuraba el puerto 5432 para exponer PostgreSQL al exterior. Antes que nada, hay que redirigir el correspondiente puerto en Virtual Box, tal como se establece en el guión de prácticas. Con solo esto, cualquier conexión será redirigida correctamente al puerto de PostgreSQL.

La conexión con el mismo se probará con la herramienta *Dbeaver*.Para ello, se escogerá la opción "*Nueva conexión*", y dentro de ella "*PostgreSQL*". En cuanto a la configuración, se dejará como Host la red local, se inserta en puerto *5432*, como base de datos *practicas_bd*; y en usuario y contraseña las credenciales del usuario con el que se quiera establecer la conexión.

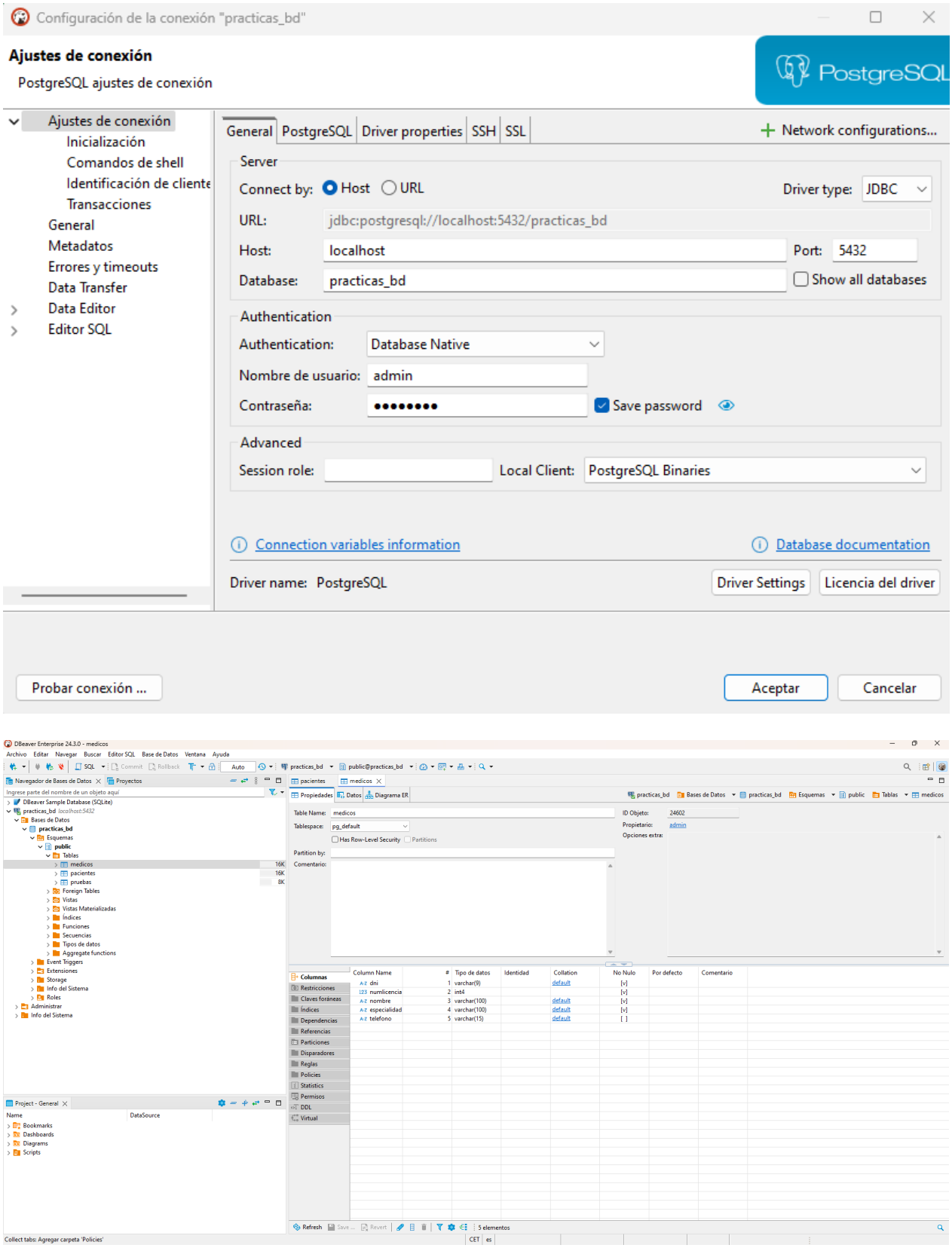


Figure 2: Conexión a PostgreSQL con DBeaver

3.2 IBM DB2

A continuación, indicamos los pasos seguidos para cada una de las tareas solicitadas a realizar:

3.2.1 Creación de un superusuario, con credenciales seguras, y verificación de que podemos conectarnos con dicho usuario.

Lo primero que hemos realizado ha sido ejecutar una terminal dentro del contenedor con la imagen del SGBD IBM DB2 ejecutando el siguiente comando:

```
# Verifica que el contenedor db2-db está corriendo
docker ps | grep db2
```

Una vez hemos verificado que el contenedor está corriendo, lo próximo a realizar es ejecutar una terminal dentro del contenedor, a la cual accedemos por primera vez con el usuario *root*.

```
# Ejecutar una terminal dentro del contenedor
docker exec -it db2 bash
```

A continuación, se procede a crear el usuario *admin* cuya contraseña va a ser igual al nombre.

```
# Crear el usuario admin
useradd -m -d /home/admin -s /bin/bash admin
# Configurar contraseña del superusuario
echo "admin:admin" | chpasswd
```

Una vez realizados estos pasos, hay que cambiar al usuario que viene definido en el docker (es decir, *db2inst1*) y entrar a la consola del SGBD donde vamos a crear una base de datos, llamada *ibm_bbdd*, donde se le concederán todos los permisos al usuario creado anteriormente para convertirlo en superusuario.

```
# Cambiar de usuario
su - db2inst1
# Entrar a la consola del SGBD
db2start
# Crear la base de datos ibm_bbdd
db2 create database ibm_bbdd
# Conectar a la base de datos
db2 connect to ibm_bbdd
```

Tras crear la base de datos en el SGBD y conectarnos a ella, ahora vamos a crear un rol que va a representar al superusuario dentro de dicha base de datos y le vamos a dar al usuario que hemos creado anteriormente dicho rol.

```
# Crear rol superusuario
db2 create role superadmin
# Conceder permisos a dicho rol para que sea superusuario
db2 GRANT DBADM,SECADM,DATAACCESS,ACCESSCTRL ON DATABASE TO ROLE superadmin
# Asociar al usuario ibm\_db2_admin el rol de superadmin
db2 GRANT ROLE superadmin TO USER admin
# Conceder permiso al usuario admin para conectarse a la base de datos
db2 GRANT CONNECT ON DATABASE TO USER admin
```

Tras haber realizado todos los comandos descritos anteriormente, ahora al ejecutar el comando *db2 connect to ibm_bbdd user admin using admin* nos podemos conectar sin problemas a la base de datos que hemos creado.

3.2.2 Creación de la estructura básica del espacio de datos

Para este apartado se ha creado una serie de ficheros *.sql* que definen la creación de la base de datos, las inserciones, las consultas SQL y la eliminación de la misma.
Para la ejecución de dichos ficheros en IBM es necesario introducir el siguiente comando:

```
# Ejecutar un fichero .sql en IBM
# Donde pone fichero, estaría el nombre del fichero que se desea ejecutar
db2 -tvf /tmp/{fichero}.sql
```

En el fichero *create_database.sql* se ha creado la base de datos **GESTION** y dentro de esta el schema **MEDICA**, donde estarán definidas las tablas. A continuación, se muestra el contenido del fichero:

```
-- Crear la Base de Datos
CREATE DATABASE GESTION;
CONNECT TO GESTION;

-- Crear el Esquema
CREATE SCHEMA MEDICA;
SET CURRENT SCHEMA MEDICA;

-- Crear tabla de médicos
CREATE TABLE MEDICA.MEDICOS (
    dni VARCHAR(9) NOT NULL PRIMARY KEY,
    numLicencia INTEGER NOT NULL UNIQUE,
    nombre VARCHAR(100) NOT NULL,
    especialidad VARCHAR(100) NOT NULL,
    telefono VARCHAR(15)
);

-- Crear tabla de pacientes
CREATE TABLE MEDICA.PACIENTES (
    dni VARCHAR(9) NOT NULL PRIMARY KEY,
    nss INTEGER NOT NULL UNIQUE,
    nombre VARCHAR(100),
    telefono VARCHAR(15)
);

-- Crear tabla de pruebas médicas
CREATE TABLE MEDICA.PRUEBAS (
    id INTEGER GENERATED ALWAYS AS IDENTITY PRIMARY KEY,
    dni_medico VARCHAR(9) NOT NULL,
    dni_paciente VARCHAR(9) NOT NULL,
    tipo_prueba VARCHAR(100),
    fecha DATE,
    resultado VARCHAR(200),
    FOREIGN KEY (dni_medico) REFERENCES MEDICA.MEDICOS(dni),
    FOREIGN KEY (dni_paciente) REFERENCES MEDICA.PACIENTES(dni)
);

-- Verificar las tablas creadas
LIST TABLES FOR SCHEMA MEDICA;

-- Confirmar la estructura de las tablas
DESCRIBE TABLE MEDICA.MEDICOS;
DESCRIBE TABLE MEDICA.PACIENTES;
DESCRIBE TABLE MEDICA.PRUEBAS;
```

```

-- Desconectar de la base de datos
CONNECT RESET;

```

Para confirmar que se ha generado correctamente las tablas, el esquema y la base de datos se han realizado los siguientes comandos:

```
# Conectar a la base de datos creada por el script
db2 CONNECT TO GESTION
# Listar las tablas creadas dentro del esquema de la base de datos
db2 LIST TABLES FOR SCHEMA MEDICA
```

Tras ejecutar el último comando, hemos podido visualizar la correcta creación de las tres tablas.

Para la inserción de datos se ha empleado el siguiente fichero *insert_data.sql*

```
-- Conectar a la base de datos
CONNECT TO GESTION;

-- Insertar en la tabla MEDICOS evitando duplicados
MERGE INTO MEDICA.MEDICOS AS T
USING (VALUES ('12345678A', 98765, 'Dr. Juan Pérez', 'Cardiología', 600123456)) AS S(DNI,
↪  NUMLICENCIA, NOMBRE, ESPECIALIDAD, TELEFONO)
ON T.DNI = S.DNI AND T.NUMLICENCIA = S.NUMLICENCIA
WHEN NOT MATCHED THEN
    INSERT (DNI, NUMLICENCIA, NOMBRE, ESPECIALIDAD, TELEFONO)
    VALUES (S.DNI, S.NUMLICENCIA, S.NOMBRE, S.ESPECIALIDAD, S.TELEFONO);

MERGE INTO MEDICA.MEDICOS AS T
USING (VALUES ('87654321B', 12366, 'Dr. Ana Gómez', 'Neurología', 611987654)) AS S(DNI,
↪  NUMLICENCIA, NOMBRE, ESPECIALIDAD, TELEFONO)
ON T.DNI = S.DNI AND T.NUMLICENCIA = S.NUMLICENCIA
WHEN NOT MATCHED THEN
    INSERT (DNI, NUMLICENCIA, NOMBRE, ESPECIALIDAD, TELEFONO)
    VALUES (S.DNI, S.NUMLICENCIA, S.NOMBRE, S.ESPECIALIDAD, S.TELEFONO);

-- Insertar en la tabla PACIENTES evitando duplicados
MERGE INTO MEDICA.PACIENTES AS T
USING (VALUES ('11111111A', 1000001, 'Carlos López', 654123988)) AS S(DNI, NSS, NOMBRE,
↪  TELEFONO)
ON T.DNI = S.DNI AND T.NSS = S.NSS
WHEN NOT MATCHED THEN
    INSERT (DNI, NSS, NOMBRE, TELEFONO)
    VALUES (S.DNI, S.NSS, S.NOMBRE, S.TELEFONO);

MERGE INTO MEDICA.PACIENTES AS T
USING (VALUES ('22222222B', 1000002, 'María Fernández', 623987654)) AS S(DNI, NSS, NOMBRE,
↪  TELEFONO)
ON T.DNI = S.DNI AND T.NSS = S.NSS
WHEN NOT MATCHED THEN
    INSERT (DNI, NSS, NOMBRE, TELEFONO)
    VALUES (S.DNI, S.NSS, S.NOMBRE, S.TELEFONO);

MERGE INTO MEDICA.PACIENTES AS T
```

```
    USING (VALUES ('33333333C', 1000003, 'Pedro Sánchez', 698741236)) AS S(DNI, NSS, NOMBRE,
    ↪  TELEFONO)
    ON T.DNI = S.DNI AND T.NSS = S.NSS
    WHEN NOT MATCHED THEN
        INSERT (DNI, NSS, NOMBRE, TELEFONO)
        VALUES (S.DNI, S.NSS, S.NOMBRE, S.TELEFONO);

    MERGE INTO MEDICA.PACIENTES AS T
    USING (VALUES ('44444444D', 1000004, 'Lucía Rodríguez', 677852963)) AS S(DNI, NSS, NOMBRE,
    ↪  TELEFONO)
    ON T.DNI = S.DNI AND T.NSS = S.NSS
    WHEN NOT MATCHED THEN
        INSERT (DNI, NSS, NOMBRE, TELEFONO)
        VALUES (S.DNI, S.NSS, S.NOMBRE, S.TELEFONO);

    -- Insertar en la tabla PRUEBAS evitando duplicados
    MERGE INTO MEDICA.PRUEBAS AS T
    USING (VALUES ('11111111A', '12345678A', 'Electrocardiograma', DATE('2024-02-15'), 'Normal'))
    ↪  AS S(DNI_PACIENTE, DNI_MEDICO, TIPO_PRUEBA, FECHA, RESULTADO)
    ON T.DNI_PACIENTE = S.DNI_PACIENTE AND T.DNI_MEDICO = S.DNI_MEDICO
    WHEN NOT MATCHED THEN
        INSERT (DNI_PACIENTE, DNI_MEDICO, TIPO_PRUEBA, FECHA, RESULTADO)
        VALUES (S.DNI_PACIENTE, S.DNI_MEDICO, S.TIPO_PRUEBA, S.FECHA, S.RESULTADO);

    MERGE INTO MEDICA.PRUEBAS AS T
    USING (VALUES ('22222222B', '12345678A', 'Ecografía', DATE('2024-02-10'), 'Sin anomalías'))
    ↪  AS S(DNI_PACIENTE, DNI_MEDICO, TIPO_PRUEBA, FECHA, RESULTADO)
    ON T.DNI_PACIENTE = S.DNI_PACIENTE AND T.DNI_MEDICO = S.DNI_MEDICO
    WHEN NOT MATCHED THEN
        INSERT (DNI_PACIENTE, DNI_MEDICO, TIPO_PRUEBA, FECHA, RESULTADO)
        VALUES (S.DNI_PACIENTE, S.DNI_MEDICO, S.TIPO_PRUEBA, S.FECHA, S.RESULTADO);

    MERGE INTO MEDICA.PRUEBAS AS T
    USING (VALUES ('33333333C', '87654321B', 'Resonancia magnética', DATE('2024-02-18'), 'Lesión
    ↪  detectada en L3-L4')) AS S(DNI_PACIENTE, DNI_MEDICO, TIPO_PRUEBA, FECHA, RESULTADO)
    ON T.DNI_PACIENTE = S.DNI_PACIENTE AND T.DNI_MEDICO = S.DNI_MEDICO
    WHEN NOT MATCHED THEN
        INSERT (DNI_PACIENTE, DNI_MEDICO, TIPO_PRUEBA, FECHA, RESULTADO)
        VALUES (S.DNI_PACIENTE, S.DNI_MEDICO, S.TIPO_PRUEBA, S.FECHA, S.RESULTADO);

    MERGE INTO MEDICA.PRUEBAS AS T
    USING (VALUES ('44444444D', '87654321B', 'Análisis de sangre', DATE('2024-02-15'), 'Niveles
    ↪  normales')) AS S(DNI_PACIENTE, DNI_MEDICO, TIPO_PRUEBA, FECHA, RESULTADO)
    ON T.DNI_PACIENTE = S.DNI_PACIENTE AND T.DNI_MEDICO = S.DNI_MEDICO
    WHEN NOT MATCHED THEN
        INSERT (DNI_PACIENTE, DNI_MEDICO, TIPO_PRUEBA, FECHA, RESULTADO)
        VALUES (S.DNI_PACIENTE, S.DNI_MEDICO, S.TIPO_PRUEBA, S.FECHA, S.RESULTADO);

    -- Desconectar de la base de datos
    CONNECT RESET;
```

Para probar las consultas en este gestor, se han hecho consultas simples sobre listar los datos de cada una de las tablas de la base de datos que se describen en el fichero *querys.sql*:

```
-- Conectar a la base de datos GESTION
CONNECT TO GESTION;

-- Consultar y listar médicos si hay registros
SELECT * FROM MEDICA.MEDICOS
WHERE (SELECT COUNT(*) FROM MEDICA.MEDICOS) > 0;

-- Consultar y listar pacientes si hay registros
SELECT * FROM MEDICA.PACIENTES
WHERE (SELECT COUNT(*) FROM MEDICA.PACIENTES) > 0;

-- Consultar y listar pruebas si hay registros
SELECT * FROM MEDICA.PRUEBAS
WHERE (SELECT COUNT(*) FROM MEDICA.PRUEBAS) > 0;

-- Desconectar de la base de datos
CONNECT RESET;
```

Para eliminar todos los datos, tablas, esquema y base de datos se ha empleado el fichero *delete_database.sql*:

```
-- Conectar a la base de datos GESTION
CONNECT TO GESTION;

-- Eliminar tablas si existen
DROP TABLE MEDICA.MEDICOS IF EXISTS;
DROP TABLE MEDICA.PACIENTES IF EXISTS;
DROP TABLE MEDICA.PRUEBAS IF EXISTS;

-- Eliminar el esquema si existe
DROP SCHEMA MEDICA RESTRICT;

-- Desconectarse de la base de datos
CONNECT RESET;

-- Eliminar la base de datos GESTION
drop database GESTION;
```

3.2.3 Creación de usuarios y roles con distinto acceso sobre los elementos del espacio de datos

Además del rol superusuario que se ha creado en el primer apartado, se han añadido dos roles más: uno que permite consultar, insertar y actualizar (que hemos llamado writer) y otro que solo permite consultar (que hemos llamado reader). Lo primero de todo ha sido añadir dos usuarios al sistema:

```
# Crear el usuario writer
useradd -m -d /home/writer -s /bin/bash writer
# Crear el usuario reader
useradd -m -d /home/reader -s /bin/bash reader
# Configurar contraseña del usuario writer
echo "writer:writer" | chpasswd
# Configurar contraseña del usuario reader
echo "reader:reader" | chpasswd
```

Una vez creados, hay que acceder nuevamente a la terminal del contenedor y configurar los roles.

```
docker exec -it contenedor_db2 bash
su - db2inst1
db2 CONNECT TO GESTION
# Crear y configurar rol writerrole
db2 CREATE ROLE writerrole
db2 GRANT CONNECT ON DATABASE TO writerrole
db2 GRANT SELECT, INSERT, UPDATE ON MEDICA.MEDICOS TO writerrole
db2 GRANT SELECT, INSERT, UPDATE ON MEDICA.PACIENTES TO writerrole
db2 GRANT SELECT, INSERT, UPDATE ON MEDICA.PRUEBAS TO writerrole
# Crear y configurar rol readerrole
db2 CREATE ROLE readerrole
db2 GRANT CONNECT ON DATABASE TO readerrole
db2 GRANT SELECT ON MEDICA.MEDICOS TO readerrole
db2 GRANT SELECT ON MEDICA.PACIENTES TO readerrole
db2 GRANT SELECT ON MEDICA.PRUEBAS TO readerrole
# Asignar los roles a los respectivos usuarios creados
db2 GRANT CONNECT ON DATABASE TO USER writer
db2 GRANT ROLE writerrole TO writer
db2 GRANT CONNECT ON DATABASE TO USER reader
db2 GRANT ROLE readerrole TO reader
```

3.2.4 Problemas encontrados

En este SGBD nos hemos encontrado con complicaciones para realizar el script que genere los usuarios del sistema debido a que los comandos tardan mucho en ejecutarse y da problemas a la hora de realizar muchas instrucciones seguidas.

3.3 Oracle

Instalar Oracle Database en Docker facilita la gestión de bases de datos ofreciendo un entorno consistente y aislado. Este método utiliza Docker para correr Oracle en un contenedor, permitiendo una configuración y despliegue rápidos. Antes de la instalación, se requiere descargar Oracle Instant Client y SQL*Plus, ajustar las variables de entorno y obtener la imagen adecuada de Docker. Este proceso simplifica la operación de Oracle, haciendo su manejo más accesible y eficiente para usuarios de todos los niveles.

3.3.1 Preparación del Entorno Oracle en Docker

Asegúrate de que el contenedor de Oracle está operativo y listo para la conexión.

```
docker ps | grep oracle-xe
```

Si el contenedor no está corriendo, inícialo desplegando de nuevo en base al `docker-compose.yml`.

3.3.2 Configuración del Cliente Oracle Instant en la Máquina Local

Instala y configura el cliente Oracle para permitir conexiones locales y manipulación de la base de datos.

```
#Instalar dependencias necesarias
sudo apt update
sudo apt install alien libaio1 wget
```

```
#Descargar Oracle Instant Client
wget https://download.oracle.com/otn_software/linux/instantclient/2370000/instantclient-basic-linux
    ↪ .x64-23.7.0.25.01.zip
```

```
#Crear directorio y descomprimir el archivo
mkdir -p ~/instantclient_23_7
unzip instantclient-basic-linux.x64-23.7.0.25.01.zip -d ~/instantclient_23_7
```

3.3.3 Configurar las variables de entorno

Edita el archivo `.bashrc` y añade las siguientes líneas:

```
export ORACLE_HOME=$HOME/instantclient_23_7/instantclient_23_7
export LD_LIBRARY_PATH=$ORACLE_HOME
export PATH=$PATH:$ORACLE_HOME
source ~/.bashrc
```

3.3.4 Descargar y Configurar SQL*Plus

Instala SQL*Plus para ejecutar comandos SQL desde la línea de comandos.

```
#Descargar SQL*Plus
wget https://download.oracle.com/otn_software/linux/instantclient/2370000/instantclient-sqlplus-
    ↪ linux.x64-23.7.0.25.01.zip
```

```
#Descomprimir SQL*Plus
unzip instantclient-sqlplus-linux.x64-23.7.0.25.01.zip -d ~/instantclient_23_7
```

3.3.5 Conectar a Oracle Usando SQL*Plus

En Oracle Database, el usuario `sys` es uno de los usuarios más importantes y tiene privilegios de superusuario. Se le considera el usuario administrador principal y es utilizado para realizar tareas de administración de alto nivel.

Establece una conexión inicial con la base de datos.

```
docker exec -it oracle-xe bash
```

3.3.6 La contraseña/password, debe ser la misma que hemos introducido en el `docker-file.yml`.

```
sqlplus sys/password@//localhost:1521/XE as sysdba
```

3.3.7 Redirección de puertos

Para gestionar y asegurar la correcta redirección de puertos en la configuración de una base de datos Oracle usando DBeaver Enterprise, se debe descargar e instalar la aplicacion desde el sitio oficial de DBeaver, accesible a través del enlace *DBeaver Enterprise*.

Posteriormente, tras el proceso de registro o inicio de sesión en DBeaver, se establece una nueva conexión y se procede a especificar los detalles necesarios para conectar con la base de datos Oracle que se ha creado.

Estos detalles necesarios, incluyen: el host, usualmente localhost, el puerto estándar 1521 de Oracle, el nombre del servicio de la base de datos, que en este caso fue XEPDB1, etc.

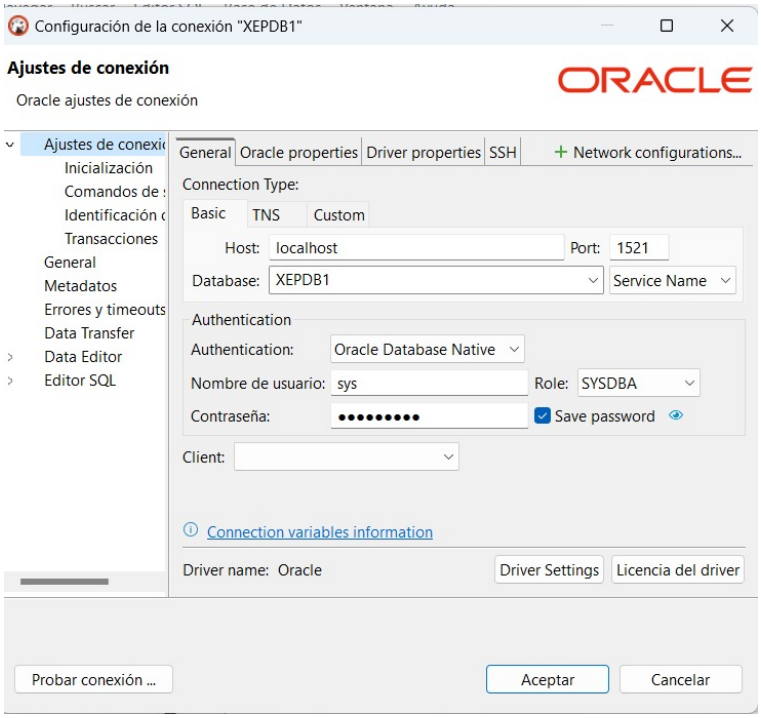


Figure 3: Instalación de la máquina virtual Debian

Una vez configurados estos parámetros, se realiza una prueba de conexión, desde "Probar conexión...", y si funciona, se pulsa sobre "Aceptar" y se establece la conexión.

Finalmente, ya tenemos accesible desde DBeaver la base de datos, y se puede modificar, crear, eliminar, consultar... sobre esta base.

3.4 Apache Cassandra

3.4.1 Inicializar el contenedor

La imagen de Apache Cassandra instalada es la última versión oficial disponible en Docker Hub, en este caso la versión *Cassandra 5.0.3*. Esta se muestra siempre que se accede a la línea de comandos de Cassandra:

Terminal

```
azzale@vbox:~/Desktop/cassandra$ docker exec -it cassandra-db cqlsh -u cassandra -p
↪  cassandra

Warning: Using a password on the command line interface can be insecure.
Recommendation: use the credentials file to securely provide the password.

Connected to MyCluster at 127.0.0.1:9042
[cqlsh 6.2.0 | Cassandra 5.0.3 | CQL spec 3.4.7 | Native protocol v5]
Use HELP for help.
cassandra@cqlsh>
```

Para levantar Cassandra, se ejecutará en el siguiente comando desde el mismo directorio donde se encuentra el *docker-compose*: La inicialización del contenedor Cassandra se realiza mediante:

```
# Levantar contenedor
docker compose up -d cassandra-db
```

Posteriormente, se hicieron las comprobaciones pertinentes para comprobar la versión, y ademas validar que el contenedor está debidamente inicializado. Si el proceso ha salido correctamente se debe obtener una salida así:

Terminal

```
azzale@vbox:~/Desktop/cassandra$ sudo docker ps
CONTAINER ID   IMAGE              COMMAND                  CREATED      STATUS
↪ PORTS
↪ NAMES
7abd9c2803f3   cassandra:latest   "docker-entrypoint.s..."   5 days ago   Up 7 seconds
↪ 7000-7001/tcp, 7199/tcp, 9160/tcp, 0.0.0.0:9042->9042/tcp, :::9042->9042/tcp
↪ cassandra-db
```

Además, se debe mencionar que existen dos maneras de interactuar con el gestor: ejecutar comandos desde la interfaz interactiva de Cassandra o ejecutar comandos directamente desde fuera:

```
# 1. ACCEDER A INTERFAZ INTERACTIVA
docker exec -it cassandra-db cqlsh -u <usuario> -p <contraseña>

# 2. EJECUTAR DIRECTAMENTE DESDE FUERA
docker exec -it cassandra-db cqlsh -u <usuario> -p <contraseña> "comando CQL"

# 3. PROCEDIMIENTO PARA EJECUTAR SCRIPT
docker cp <script.cql> cassandra-db:/<script.cql>
docker exec -i cassandra-db cqlsh -u <usuario> -p <contraseña> -f /<script.cql>
```

En todos los casos donde se trabaje con scripts, se mostrará la ejecución con comandos desde fuera, pero si se trata de comandos aislados se mostrará la ejecución desde dentro de la interfaz por simplicidad. De hecho, la interfaz de Cassandra esta diseñada como una consola altamente interactiva, por lo que en caso de no ejecutar scripts será la principal herramienta a usar.

3.4.2 Creación de la estructura básica del espacio de la base de datos

Con Cassandra no se habla de bases de datos como tal, sino de *keyspaces*. Ahora ya se trabaja, en este sentido, a un mayor grado de organización (distribuido), donde los *keyspaces* no solo se almacenan tablas, sino también estrategias de replicación. Así, con el objetivo de proporcionar una mayor tolerancia a fallos, los datos se pueden hallar replicados en el espacio de la base de datos.
Se creará, de este modo, un *keyspace* sencillo con el que trabajar por el momento, con una estrategia básica y sin copias adicionales de datos (es decir, con un solo nodo):

crear_Espacio.cql

Por su parte, la validación de que se ha creado correctamente se sigue de este modo:

Terminal

```
azzale@vbox:~/Desktop/cassandra$ docker exec -it cassandra-db cqlsh -u cassandra -p
↪  cassandra

Warning: Using a password on the command line interface can be insecure.
Recommendation: use the credentials file to securely provide the password.

Connected to MyCluster at 127.0.0.1:9042
[cqlsh 6.2.0 | Cassandra 5.0.3 | CQL spec 3.4.7 | Native protocol v5]
Use HELP for help.
cassandra@cqlsh> DESCRIBE KEYSPACES

practicas_ks  system_auth         system_schema  system_views
system        system_distributed  system_traces  system_virtual_schema

cassandra@cqlsh> DESCRIBE KEYSPACE practicas_ks;

CREATE KEYSPACE practicas_ks WITH replication = {'class': 'SimpleStrategy',
↪  'replication_factor': '1'}  AND durable_writes = true;
```

3.4.3 Asignación de roles

Apache Cassandra, o al menos la versión instalada, tiene la autenticación desactivada, asi como la asignación específica de permisos. Por lo tanto, el primer paso para poder crear distintos y usuarios fue modificando las pertinententes propiedades del archivo *cassandra.yaml* del contenedor. Los pasos a seguir son los siguientes:

```
# 1 Copiar fuera del contenedor y abrir el fichero
docker cp cassandra-db:/etc/cassandra/cassandra.yaml ./cassandra.yaml
nano cassandra.yaml

# 2. MODIFICAR LAS LÍNEAS "authenticator" y "authorizer"
authenticator: PasswordAuthenticator
authorizer: CassandraAuthorizer

# 3. Copiar el fichero modificado y reiniciar contenedor para aplicar cambios
docker cp ./cassandra.yaml cassandra-db:/etc/cassandra/cassandra.yaml
docker restart cassandra-db
```

Ahora, al intentar acceder a la interfaz de cassandra sin autenticarse se obtendrá un error:

Terminal

```
azzale@vbox:~/Desktop/cassandra$ docker exec -it cassandra-db cqlsh
Connection error: ('Unable to connect to any servers', {'127.0.0.1:9042':
↪  AuthenticationFailed('Remote end requires authentication')})
```

Por lo tanto, para acceder por primera vez, se necesitará acceder con el superusuario por defecto, con nombre y contraseña *cassandra*.

Una vez se ha configurado correctamente el gestor, conviene mencioa los roles y usuarios Cassandra. Realmente, Cassandra no diferencia entre usuario y rol, de manera que un rolo puede actuar como usuario si tiene permiso login o como un conjunto de permisos (lo que es rol en otros gestores) si no lo tiene.
Puestos en contexto, el código para crear los roles exigidos por el guión son los siguientes:

crear_Espacio.cql

Si listamos ahora los roles existentes desde la interfaz de Cassandra:

Terminal

```
admin@cqlsh>  LIST ROLES ;

 role      | super | login | options | datacenters
-----------+-------+-------+---------+-------------
     admin |  True |  True |        {} |         ALL
 cassandra |  True |  True |        {} |         ALL
  escritor | False |  True |        {} |         ALL
    lector | False |  True |        {} |         ALL

(4 rows)
admin@cqlsh> LIST ALL PERMISSIONS OF escritor
   ... ;

 role     | username | resource                | permission
----------+----------+-------------------------+------------
 escritor | escritor | <keyspace practicas_ks> |     MODIFY

(1 rows)
```

Obsérvese que sigue presente el rol cassandra, el cual es además superusuario. Lo que se ha decidido en este caso es mantenerlo pero modificar su contraseña, para cubrirse ante la inoportuna eliminación involuntaria del rol *admin*. Además, otro detalle importante es que el rol escritor se ha definido con el rol escritor cuenta con el permiso *MODIFY*. En Cassandra, los permisos de *INSERT*, *UPDATE* y *DELETE* se engloban en un solo permiso, por lo que para impedir que el escritor pueda eliminar datos de tablas se tendría que añadir código a nivel de aplicación.
Al igual que con el resto de gestores, se probó a conectarse con los distintos gestores y a realizar operaciones para las que cuentan y no cuentan con permisos:

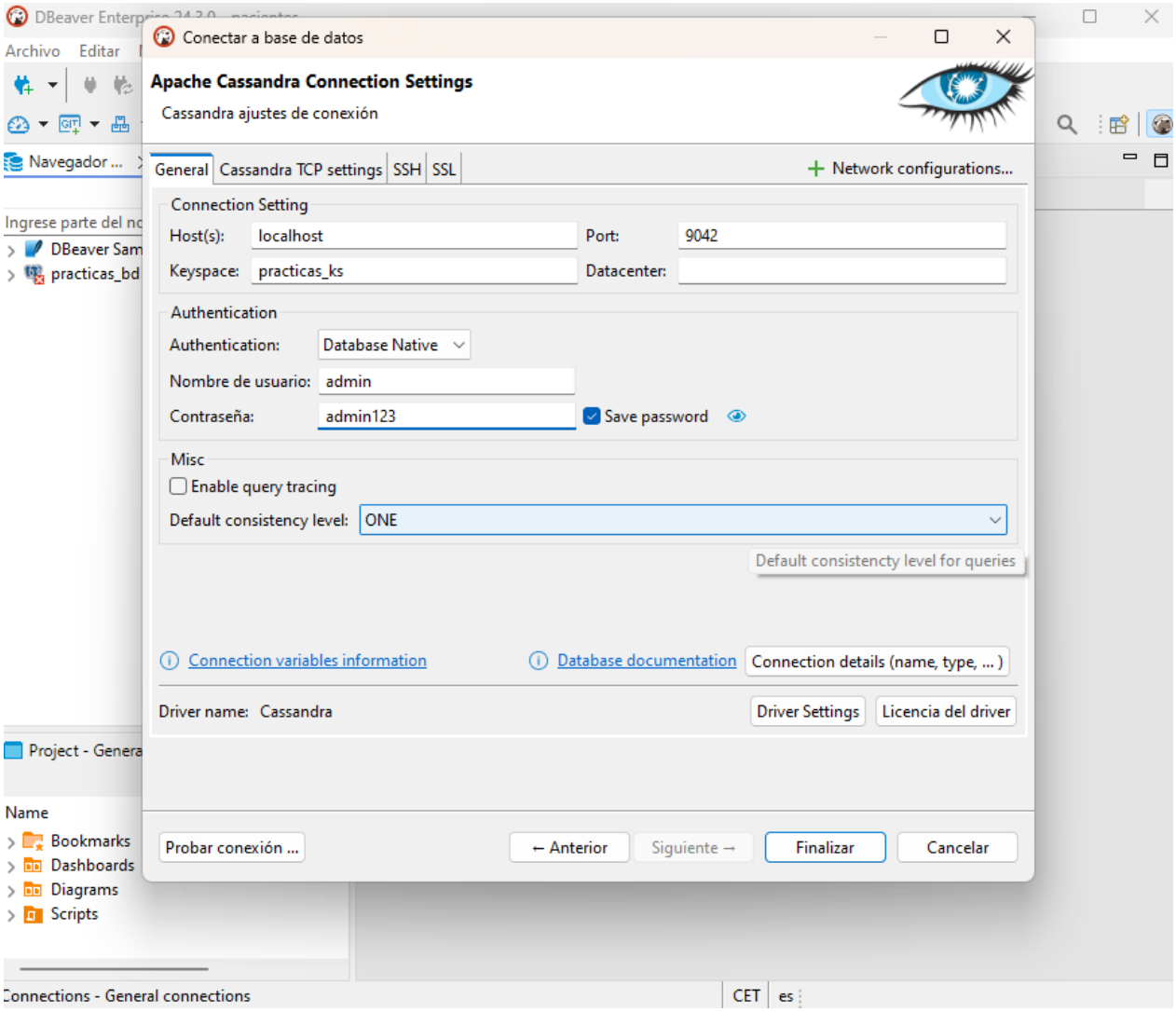
Terminal

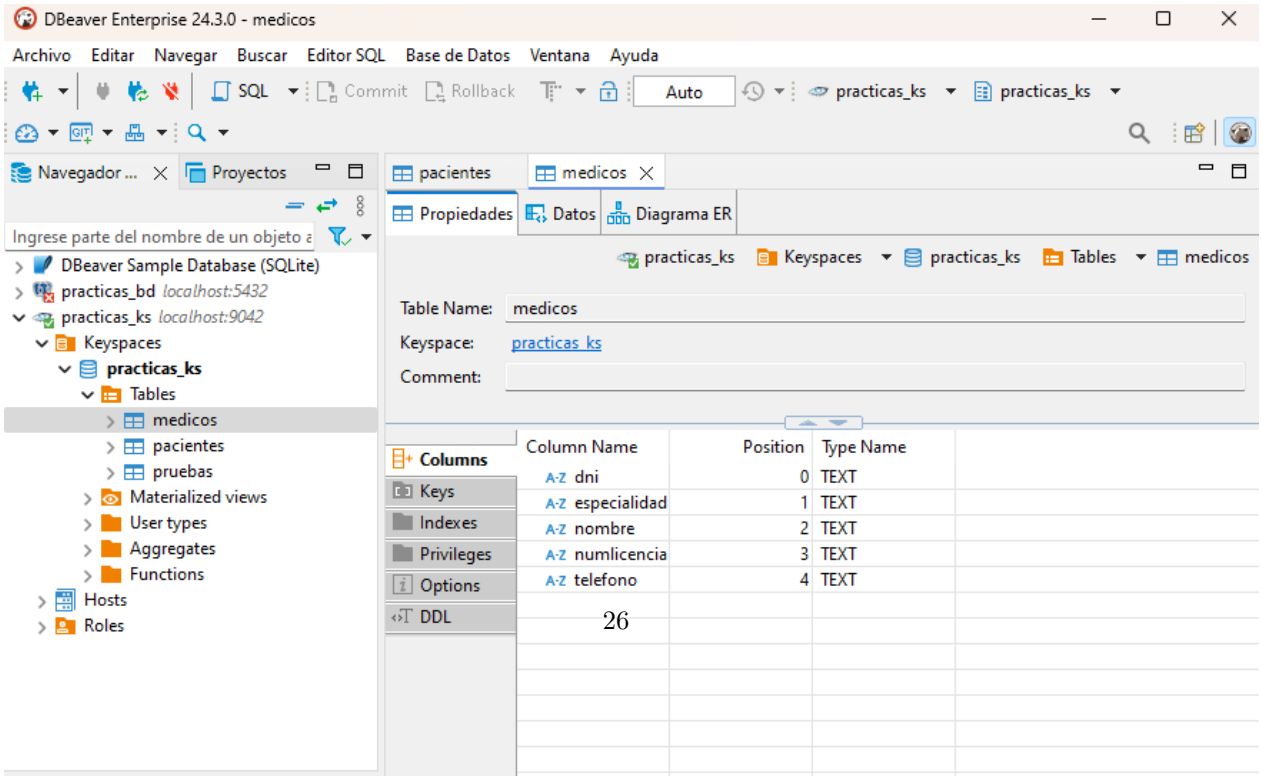
```
azzale@vbox:~/Desktop/cassandra$ docker exec -i cassandra-db cqlsh -u lector -p lector123
↪  -f /insertarTablas.cql

Warning: Using a password on the command line interface can be insecure.
Recommendation: use the credentials file to securely provide the password.

/insertarTablas.cql:3:Unauthorized: Error from server: code=2100 [Unauthorized]
↪  message="User lector has no MODIFY permission on <table practicas_ks.pruebas> or any
↪  of its parents"
/insertarTablas.cql:4:Unauthorized: Error from server: code=2100 [Unauthorized]
↪  message="User lector has no MODIFY permission on <table practicas_ks.medicos> or any
↪  of its parents"
/insertarTablas.cql:5:Unauthorized: Error from server: code=2100 [Unauthorized]
↪  message="User lector has no MODIFY permission on <table practicas_ks.pacientes> or any
↪  of its parents"
/insertarTablas.cql:9:Unauthorized: Error from server: code=2100 [Unauthorized]
↪  message="User lector has no MODIFY permission on <table practicas_ks.medicos> or any
↪  of its parents"
/insertarTablas.cql:12:Unauthorized: Error from server: code=2100 [Unauthorized]
↪  message="User lector has no MODIFY permission on <table practicas_ks.medicos> or any
↪  of its parents"
/insertarTablas.cql:16:Unauthorized: Error from server: code=2100 [Unauthorized]
↪  message="User lector has no MODIFY permission on <table practicas_ks.pacientes> or any
↪  of its parents"
/insertarTablas.cql:19:Unauthorized: Error from server: code=2100 [Unauthorized]
↪  message="User lector has no MODIFY permission on <table practicas_ks.pacientes> or any
↪  of its parents"
/insertarTablas.cql:29:Unauthorized: Error from server: code=2100 [Unauthorized]
↪  message="User lector has no MODIFY permission on <table practicas_ks.pruebas> or any
↪  of its parents"

```

3.4.4 Conectividad remota

El docker-compose configuraba el puerto 9042 para exponer Cassandra al exterior. Antes que nada, hay que redirigir el correspondiente puerto en Virtual Box, tal como se establece en el guión de prácticas. Con solo esto, cualquier conexión será redirigida correctamente al puerto de Cassandra.

La conexión con el mismo se probará con la herramienta *Dbeaver*.Para ello, se escogerá la opción "*Nueva conexión*", y dentro de ella "*Cassandra*". En cuanto a la configuración, se dejará como Host la red local, se inserta en puerto *9042*, como base de datos *practicas_ks*; y en usuario y contraseña las credenciales del usuario con el que se quiera establecer la conexión.


Conectar a base de datos
Apache Cassandra Connection Settings
Cassandra ajustes de conexión
General
Cassandra TCP settings
SSH
SSL
Network configurations...
Connection Setting
Host(s): localhost
Port: 9042
Keyspace: practicas_ks
Datacenter:
Authentication
Authentication: Database Native
Nombre de usuario: admin
Contraseña: admin123
Save password
Misc
Enable query tracing
Default consistency level: ONE
Default consistency level for queries
Connection variables information
Database documentation
Connection details (name, type, ...)
Driver name: Cassandra
Driver Settings
Licencia del driver
Probar conexión ...
← Anterior
Siguiente →
Finalizar
Cancelar


DBeaver Enterprise 24.3.0 - medicos
Archivo Editar Navegar Buscar Editor SQL Base de Datos Ventana Ayuda
Navegador ... Proyectos
pacientes
medicos
Propiedades
Datos
Diagrama ER
practicas_ks Keyspaces practicas_ks Tables medicos
Table Name: medicos
Keyspace: practicas_ks
Comment:
DBeaver Sample Database (SQLite)
practicas_bd localhost:5432
practicas_ks localhost:9042
Keyspaces
practicas_ks
Tables
medicos
pacientes
pruebas
Materialized views
User types
Aggregates
Functions
Hosts
Roles
Columns
Keys
Indexes
Privileges
Options
DDL
Column Name Position Type Name
dni 0 TEXT
especialidad 1 TEXT
nombre 2 TEXT
numlicencia 3 TEXT
telefono 4 TEXT

3.5 HBase

3.5.1 Introducción

HBase es un sistema de bases de datos NoSQL escalable basado en HDFS, diseñado para almacenar grandes volúmenes de datos distribuidos. En esta documentación, se describirá el proceso de instalación y configuración de HBase, incluyendo los intentos de integración con Kerberos y la posterior decisión de prescindir de esta tecnología debido a dificultades técnicas.

3.5.2 Intento de Integración con Kerberos

Para garantizar una autenticación segura en HBase, se intentó la integración de Kerberos como sistema de gestión de identidades. El objetivo era proporcionar una autenticación basada en tickets para controlar el acceso a los datos. Sin embargo, debido a múltiples problemas técnicos, se decidió desestimar Kerberos y proceder con una configuración sin autenticación centralizada.

3.5.3 Configuración Inicial de Kerberos

La instalación de Kerberos en el host se realizó con los siguientes comandos:

```
sudo apt update
sudo apt install -y krb5-kdc krb5-admin-server krb5-user
```

A continuación, se configuró el archivo `/etc/krb5.conf`:

```
[libdefaults]
default_realm = HBASE.LOCAL
dns_lookup_realm = false
dns_lookup_kdc = false
ticket_lifetime = 24h
renew_lifetime = 7d
forwardable = true

[realms]
HBASE.LOCAL = {
    kdc = localhost
    admin_server = localhost
}

[domain_realm]
.hbase.local = HBASE.LOCAL
hbase.local = HBASE.LOCAL
```

Se procedió a inicializar la base de datos de Kerberos:

```
sudo krb5_newrealm
```

Se crearon los principales para HBase Master y RegionServer:

```
sudo kadmin.local -q "addprinc -randkey hbase-master@HBASE.LOCAL"
sudo kadmin.local -q "addprinc -randkey hbase-regionserver@HBASE.LOCAL"
```

Y se generaron los keytabs:

```
sudo kadmin.local -q "ktadd -k /etc/hbase/keytabs/hbase-master.keytab hbase-master@HBASE.LOCAL"
sudo kadmin.local -q "ktadd -k /etc/hbase/keytabs/hbase-regionserver.keytab hbase-
    ↪ regionserver@HBASE.LOCAL"
```

Estos archivos fueron copiados a los contenedores de HBase:

```
docker cp /etc/hbase/keytabs/hbase-master.keytab hbase-db:/etc/hbase/keytabs/hbase-master.keytab
docker cp /etc/hbase/keytabs/hbase-regionserver.keytab hbase-db:/etc/hbase/keytabs/hbase-
    ↪ regionserver.keytab
```

A pesar de la configuración, surgieron diversos problemas que impidieron la correcta autenticación en HBase:

3.5.4 Problema con los Keytabs

Al ejecutar `klist -kt` dentro del contenedor de HBase, se obtuvo el error:

```
klist: Unsupported key table format version number while starting keytab scan
```

Se intentó regenerar los keytabs y copiarlos nuevamente al contenedor sin éxito.

3.5.5 Errores en la Autenticación de HBase

Al ejecutar `kinit` y listar las regiones en HBase, se obtuvo el error:

```
1 ERROR: No valid credentials provided (Mechanism level: Failed to find any Kerberos tgt)
```

Tras varios intentos, se decidió desestimar la configuración de Kerberos.

3.5.6 Desestimación de Kerberos

Dado que los problemas con Kerberos dificultaban el desarrollo y consumían demasiado tiempo, se optó por eliminar la configuración de seguridad y proceder sin autenticación:

3.5.7 1. Eliminación de Kerberos

```
docker-compose down
sudo rm -rf /etc/hbase/keytabs
sudo rm -rf /var/lib/krb5kdc
sudo apt-get remove --purge krb5-kdc krb5-admin-server
sudo apt-get autoremove
```

3.5.8 2. Modificación de la configuración de HBase

Se editó el archivo hbase-site.xml para cambiar la autenticación:

```
hbase.security.authentication
simple

hbase.security.authorization
false
```

Luego, se copió el archivo actualizado dentro del contenedor de HBase:

```
docker cp ~/bd2/hbase-config/hbase-site.xml hbase-db:/opt/hbase/conf/hbase-site.xml
```

3.5.9 3. Reinicio de Servicios

```
docker restart hbase-db
```

3.5.10 4. Verificación de la Configuración

Para verificar que HBase funcionaba sin Kerberos, se ejecutaron los siguientes comandos:

```
docker exec -it hbase-db hbase shell
```

Se intentó crear la tabla ‘hbase:acl‘ para la gestión de permisos, pero se observó que **ya existía**, lo que indicaba que la configuración de permisos estaba activa:

```
hbase(main):023:0* create 'hbase:acl', {NAME => 'f', VERSIONS => 1}
ERROR: Table already exists: hbase:acl!
```

Para confirmar su existencia, se escaneó su contenido:

```
hbase(main):024:0> scan 'hbase:acl'
ROW COLUMN+CELL
0 row(s)
Took 0.0219 seconds
```

Finalmente, se verificó el estado del sistema:

```
hbase(main):025:0> status 'detailed'
```

Lo importante aquí fue notar la presencia de:

```
master coprocessors: [AccessController]
```

Esto confirmó que **el sistema ya tenía habilitados los permisos sin necesidad de Kerberos**.

3.5.11 Conclusión

La configuración de HBase sin Kerberos permitió evitar problemas de autenticación y facilitar el acceso. La configuración final es sin autenticación, con la gestión de permisos mediante hbase:acl.

4 Generación de Datos y Pruebas

La fase de generación de Datos y pruebas sirvió, además para asegurar una funcionalidad y compatibilidad básica con los gestores de datos instalados, para familiarizarse un poco con la semántica de cada gestor (lenguaje). Así, se opta por un esquema conceptual sencillo, correspondiente al ejemplo más básico de base de datos de pruebas médicas diseñado en clase (existen médicos, pacientes, y pruebas asociadas a ambos). Así, se trabaja con conceptos básicos como claves primarias, claves foráneas, no nulos, etc.

4.1 PostgreSQL

La creación de tablas en PostgreSQL no supuso grnades complicaciones ni suscitó un gran asombro, ya que cumple con estándares SQL con los que el equipo ya se encontraba familiarizado. La posibilidad de incluir funcionalidades más complejas, como el soporte a OO mediante herencia se descartó al considerar que se aleja de los objetivos de esta práctica (ya se trabajará en las posteriores).

creacion_tablas.sql

```sql
-- Limpiar claramente objetos anteriores (si existen) para evitar conflictos
DROP TABLE IF EXISTS pruebas CASCADE;
DROP TABLE IF EXISTS pacientes CASCADE;
DROP TABLE IF EXISTS medicos CASCADE;

-- Crear tablas para gestión médica claramente desde cero

-- Tabla de médicos
CREATE TABLE medicos (
    dni VARCHAR(9) PRIMARY KEY,
    numLicencia INTEGER NOT NULL UNIQUE,
    nombre VARCHAR(100) NOT NULL,
    especialidad VARCHAR(100) NOT NULL,
    telefono VARCHAR(15)
);

-- Tabla de pacientes
CREATE TABLE pacientes (
    dni VARCHAR(9) PRIMARY KEY,
    nss INTEGER NOT NULL UNIQUE,
    nombre VARCHAR(100),
    telefono VARCHAR(15)
);

-- Tabla de pruebas médicas
CREATE TABLE pruebas (
    id SERIAL PRIMARY KEY,
    dni_medico VARCHAR(9) NOT NULL REFERENCES medicos(dni),
    dni_paciente VARCHAR(9) NOT NULL REFERENCES pacientes(dni),
    tipo_prueba VARCHAR(100),
    fecha DATE,
    resultado VARCHAR(200)
);
```

Se verificó además que las tablas se crearon correctamente:

Terminal

```
azzale@vbox:~/Desktop/postgreSQL$ docker exec -it postgres-db psql -U
           admin -d practicas_bd -c "\dt"
         List of relations
 Schema |   Name    | Type  | Owner
--------+-----------+-------+-------
 public | medicos   | table | admin
 public | pacientes | table | admin
 public | pruebas   | table | admin
(3 rows)

```

Además, se crearon scripts sencillos para la escritura y consulta básica de las tablas, los cuales se probaron con los distintos usuarios existentes.

insertarTablas.sql

```sql
-- Script para insertar y modificar datos con el usuario escritor
\connect practicas_bd

-- Limpiar tablas antes de insertar datos
DELETE FROM pruebas;
DELETE FROM pacientes;
DELETE FROM medicos;

-- Insertar datos en la tabla medicos
INSERT INTO medicos (dni, numlicencia, nombre, especialidad, telefono)
VALUES ('12345678A', 12345, 'Dr. Izquierdo', 'M.Familia', '876000111');

-- Insertar datos en la tabla pacientes
INSERT INTO pacientes (dni, nss, nombre, telefono)
VALUES ('87654321B', 987654321, 'Ada Byron', '699654321');

-- Insertar datos en la tabla pacientes
INSERT INTO pacientes (dni, nss, nombre, telefono)
VALUES ('87654321X', 987654322, 'Nadie', '699654322');

-- Insertar datos en la tabla pruebas
INSERT INTO pruebas (id_medico, id_paciente, tipo_prueba, fecha, resultado)
VALUES ('12345678A', '87654321B', 'muestra de orina', '2025-02-17', 'DATO1 : X');

-- Modificar datos (cambiar número de teléfono del paciente)
UPDATE pacientes SET telefono = '655888999' WHERE dni = '87654321B';

```

consultas.sql

```
-- Script para consultar datos con el usuario lector
\connect practicas_bd
-- Consultar todos los médicos
SELECT * FROM medicos;

-- Consultar todos los pacientes
SELECT * FROM pacientes where nombre 'Ada Byron';
```

4.2 Apache Cassandra

La manipulación de tablas con Cassandra suponen (o supondrán) un cambio de paradigma con respecto al anterior trabajo con SQL. En sentido, las tablas no están pensadas para hacer JOINS, por lo que no hay relaciones (claves foráneas) entre tablas. Cassandra realmente está orientada a consultas específicas, por lo que las relaciones en el modelo conceptual se transformarán en desnormalizaciones en la medida en la que se necesite. A continuación se muestra la prueba de creación de tablas:

creacionTablas.cql

```
-- Limpiar tablas previamente si ya existen
DROP TABLE IF EXISTS practicas_ks.pruebas;
DROP TABLE IF EXISTS practicas_ks.medicos;
DROP TABLE IF EXISTS practicas_ks.pacientes;
USE practicas_ks;

-- Crear tabla 'medicos'
CREATE TABLE medicos (
    dni TEXT PRIMARY KEY,
    numLicencia TEXT,
    nombre TEXT,
    especialidad TEXT,
    telefono TEXT
);

-- Crear tabla 'pacientes'
CREATE TABLE pacientes (
    dni TEXT PRIMARY KEY,
    nss TEXT,
    nombre TEXT,
    telefono TEXT
);

-- Crear tabla 'pruebas'
CREATE TABLE pruebas (
    id UUID PRIMARY KEY,
    dni_medico TEXT,
    nombre_medico TEXT,     -- Nombre médico desnormalizado
    dni_paciente TEXT,
    nombre_paciente TEXT,   -- Nombre paciente desnormalizado
    tipo_prueba TEXT,
    fecha TIMESTAMP,
    resultado TEXT
);

-- Verificar creación
DESCRIBE TABLES;

```

Obsérvese como en este caso se ha producido una desnormalización en la tabla de pruebas para incluir los nombres del médico y paciente. Esto puede ser así porque se considerará habitual consultar el nombre de los pacientes y médicos, por poner un ejemplo.

Además, se realizarón pruebas para la inserción y consulta de las mismas.

insertarTablas.cql

```
-- Limpiar  datos de tablas
TRUNCATE TABLE practicas_ks.pruebas;
TRUNCATE TABLE practicas_ks.medicos;
TRUNCATE TABLE practicas_ks.pacientes;

-- Insertar datos en tabla 'medicos'
INSERT INTO practicas_ks.medicos (dni, numLicencia, nombre, especialidad, telefono)
VALUES ('12345678A', 'MED123', 'Dr. López', 'Cardiología', '600123456');

INSERT INTO practicas_ks.medicos (dni, numLicencia, nombre, especialidad, telefono)
VALUES ('87654321B', 'MED456', 'Dra. Martínez', 'Neurología', '699654321');

-- Insertar datos en tabla 'pacientes'
INSERT INTO practicas_ks.pacientes (dni, nss, nombre, telefono)
VALUES ('11111111C', 'NSS001', 'Ana Gómez', '655555555');

INSERT INTO practicas_ks.pacientes (dni, nss, nombre, telefono)
VALUES ('22222222D', 'NSS002', 'Carlos Pérez', '644444444');

-- Insertar datos en tabla 'pruebas'
INSERT INTO practicas_ks.pruebas (
    id, dni_medico, nombre_medico, dni_paciente, nombre_paciente, tipo_prueba, fecha,
↪   resultado
) VALUES (
    uuid(),
    '12345678A', 'Dr. López',
    '11111111C', 'Ana Gómez',
    'Electrocardiograma', toTimestamp(now()), 'Normal'
);

```

consultas.cql

Nótese que en la segunda consulta se ha especificado la cláusula ALLOW FILTERING. Esto se debe a que *dni_medico* no forma parte primaria de la tabla, y por lo tanto la consulta debe "forzarse" ya que su rendimiento no es idóneo. Realmente la mejor solución no es la cláusula, sino añadir como clave primaria de la tabla pruebas también el dni.

4.3 Operaciones Básicas en Oracle

Configuraciones iniciales, creación de usuarios y tablespaces.

4.3.1 Creación de superusuarios:

En Oracle, el término "usuario común" no implica que tenga privilegios limitados o normales. Se refiere específicamente a que el usuario puede acceder y operar en todas las bases de datos pluggables (PDBs) dentro de una base de datos contenedora (CDB) en una configuración multitenant.

Cuando un "usuario común" como superadmin recibe el rol de DBA, se convierte en un superusuario debido a los privilegios administrativos extensos que este rol le permite, como: control total sobre aspectos críticos y administrativos de la base de datos, gestionar usuarios, configurar la seguridad, manejar copias de seguridad, etc.

```
#Creación del superusuario
CREATE USER c##superadmin IDENTIFIED BY Admin1234;
GRANT DBA TO c##superadmin;
```

4.3.2 Creación de tablespaces

Un tablespace en Oracle es un contenedor de almacenamiento que guarda los datos. Es esencial configurar tablespaces antes de crear usuarios, ya que determinan dónde se almacenarán los datos de estos.

Antes de crear un tablespace, es muy importante seleccionar el contenedor adecuado para asegurarse de que los datos se almacenen en la ubicación correcta, especialmente en configuraciones multitenant.

El proceso comienza con la verificación de las PDBs disponibles:

```
-- Mostrar todas las PDBs disponibles en la instancia de Oracle
SHOW PDBS;

CON_ID CON_NAME OPEN MODE RESTRICTED
------- ------------------------------ ---------- ----------
    2 PDB$SEED READ ONLY NO
    3 XEPDB1 READ WRITE NO
```

Una vez identificada la PDB deseada, cambiamos la sesión a esa PDB específica:

```
-- Cambiar la sesión al contenedor XEPDB1
ALTER SESSION SET CONTAINER=XEPDB1;
```

A continuación, se muestra cómo crear un tablespace denominado `myworkspace1`:

```
CREATE TABLESPACE myworkspace DATAFILE 'myworkspace1.dbf' SIZE 100M AUTOEXTEND ON NEXT 10M MAXSIZE
    ↪ UNLIMITED;
```

4.3.3 Creación de usuarios de escritura y lectura:

Primero, creamos el usuario `writer`. Este proceso se realiza mediante un comando SQL que especifica una contraseña para el usuario y el tablespace predeterminado donde se almacenarán los objetos que `writer` cree.

El comando es el siguiente:

```
CREATE USER writer IDENTIFIED BY writerPass DEFAULT TABLESPACE myworkspace1;
```

Este paso es crucial porque establece las credenciales de `writer` y define dónde se guardarán físicamente los datos que el usuario maneje, facilitando una organización lógica y eficiente del espacio en la base de datos.

Al crear el usuario `writer`, también necesitamos asignarle privilegios básicos que le permitirán operar dentro de la base de datos: realizar funciones básicas necesarias, modificar cualquier tabla, crear secuencias para autoincrementos, definir vistas y crear sinónimos.

```
-- Privilegios inciales:
GRANT CREATE SESSION, CREATE TABLE TO writer;

-- Privilegios adicionales:
GRANT ALTER ANY TABLE, CREATE SEQUENCE, CREATE VIEW, CREATE SYNONYM TO writer;
```

Luego, procedemos a crear el usuario `reader`, similar a `writer`, pero con privilegios más limitados adecuados para un perfil que solo requiere leer datos:

```
CREATE USER reader IDENTIFIED BY readerPass DEFAULT TABLESPACE myworkspace1;
```

Para `reader`, solo otorgamos el privilegio de CREATE SESSION, que es suficiente para que acceda a la base de datos sin permitirle ni modificarla ni crear nuevos objetos:

```
GRANT CREATE SESSION TO reader;
```

Cuando se termine el siguiente apartado de "Creación de tablas" que hay a continuación, se deben ejecutar estos comandos, para poder otorgar correctamente los privilegios de lectura al usuario lector de las tablas creadas:

```
-- Conexión como SYSDBA
CONNECT sys/oracle123@//localhost:1521/XEPDB1 AS SYSDBA;

-- Otorgamos privilegios
GRANT SELECT ON WRITER.medicos TO reader;
GRANT SELECT ON WRITER.pacientes TO reader;
GRANT SELECT ON WRITER.pruebas TO reader;
```

Para realizar la creación de usuarios, también se puede realizar mediante este script:

```
#!/bin/bash

CONTAINER_NAME="oracle-xe"
SYSDBA_USER="sys"
SYSDBA_PASS="oracle123"
DB_NAME="XEPDB1"

# Función para ejecutar SQL
run_sql() {
    local sql_command=$1
    docker exec -i "$CONTAINER_NAME" bash -c "echo \"$sql_command\" | sqlplus -S
    ↪  $SYSDBA_USER/$SYSDBA_PASS@\"$DB_NAME\" AS SYSDBA"
}

# Función para verificar existencia de usuario
user_exists() {
    local username=$1
    local sql_query="SET HEADING OFF;
SET FEEDBACK OFF;
SELECT username FROM dba_users WHERE username = UPPER('$username');"

    local result=$(run_sql "$sql_query" | tr -d '[:space:]')

    [ -n "$result" ] && return 0 || return 1
}

# Crear usuario solo si no existe
create_user() {
    local user=$1
    local pass=$2
    local privileges=$3

    if ! user_exists "$user"; then
```

```
            echo "Creando usuario $user..."
            run_sql "CREATE USER $user IDENTIFIED BY $pass DEFAULT TABLESPACE myworkspace1;"
            run_sql "$privileges"
            echo "Usuario $user creado exitosamente"
        else
            echo "El usuario $user YA EXISTE. No se realiza ninguna acción."
        fi
    }

    # --- Ejecución principal ---
    create_user "writer" "writerPass" "
    GRANT CREATE SESSION, CREATE TABLE TO writer;
    GRANT ALTER ANY TABLE, CREATE SEQUENCE, CREATE VIEW, CREATE SYNONYM TO writer;
    "

    create_user "reader" "readerPass" "
    GRANT CREATE SESSION TO readerjuju;
    "
```

4.3.4 Creación e inserción de tablas

Para la creación de tablas, se ejecuta un archivo de script SQL denominado "CreacionTablasOracle", que contiene todos los comandos necesarios para definir las tablas y las relaciones entre ellas de médicos, pacientes y pruebas.

El contenido del script "CreacionTablasOracle.sql" incluye:

```
-- Conectar al usuario writer con su contraseña
CONNECT writer/writerPass@localhost:1521/XEPDB1;
SHOW USER;

-- Eliminar las tablas si ya existen, y si no, no se hace nada
BEGIN
    EXECUTE IMMEDIATE 'DROP TABLE pruebas CASCADE CONSTRAINTS';
EXCEPTION
    WHEN OTHERS THEN
        NULL;
END;
/

BEGIN
    EXECUTE IMMEDIATE 'DROP TABLE pacientes CASCADE CONSTRAINTS';
EXCEPTION
    WHEN OTHERS THEN
        NULL;
END;
/

BEGIN
    EXECUTE IMMEDIATE 'DROP TABLE medicos CASCADE CONSTRAINTS';
EXCEPTION
    WHEN OTHERS THEN
        NULL;
END;
/
```

```
-- Crear la tabla de médicos
CREATE TABLE medicos (
    dni VARCHAR2(9) PRIMARY KEY,
    numLicencia NUMBER NOT NULL UNIQUE,
    nombre VARCHAR2(100) NOT NULL,
    especialidad VARCHAR2(100) NOT NULL,
    telefono VARCHAR2(15)
);

-- Crear la tabla de pacientes
CREATE TABLE pacientes (
    dni VARCHAR2(9) PRIMARY KEY,
    nss NUMBER NOT NULL UNIQUE,
    nombre VARCHAR2(100),
    telefono VARCHAR2(15)
);

-- Crear la tabla de pruebas médicas
CREATE TABLE pruebas (
    id NUMBER GENERATED ALWAYS AS IDENTITY PRIMARY KEY,
    dni_medico VARCHAR2(9) NOT NULL,
    dni_paciente VARCHAR2(9) NOT NULL,
    tipo_prueba VARCHAR2(100),
    fecha DATE,
    resultado VARCHAR2(200),
    CONSTRAINT fk_medico FOREIGN KEY (dni_medico) REFERENCES medicos(dni),
    CONSTRAINT fk_paciente FOREIGN KEY (dni_paciente) REFERENCES pacientes(dni)
);
```

Para ejecutar este script utilizando SQL*Plus, se introduce el siguiente comando:

```
nano CreacionTablasOracle.sql
docker cp ~/bd2/oracle/CreacionTablasOracle.sql oracle-xe:/opt/oracle
docker exec -it oracle-xe bash
sqlplus writer/writerPass@localhost:1521/XEPDB1
@/opt/oracle/CreacionTablasOracle.sql
```

Tras crear las tablas, hay que conectarse al sistema como el usuario `sys` utilizando privilegios de SYSDBA. Este nivel de acceso es esencial para realizar ajustes administrativos de alto nivel que afectan la estructura y los límites de almacenamiento de la base de datos.

```
CONNECT sys/oracle123@//localhost:1521/XEPDB1 AS SYSDBA;
```

Una vez establecida la conexión, se altera la cuota de almacenamiento del usuario `writer` para asignarle una cuota ilimitada en el tablespace myworkspace1. Esto asegura que writer tenga suficiente espacio para almacenar datos y crear objetos sin enfrentar restricciones de espacio.

```
ALTER USER writer QUOTA UNLIMITED ON myworkspace1;
```

Posteriormente, hay que conectarse de nuevo como `writer` para realizar operaciones de inserción de datos en las tablas. Este paso lo llevamos a cabo mediante un script denominado "InsercionTablasOracle".

El contenido del script "InsercionTablasOracle.sql" incluye:

```
-- Conexión como usuario writer y con su contraseña
CONNECT writer/writerPass@localhost:1521/XEPDB1;

SET SERVEROUTPUT ON;
```

```
-- Comprobación de existencia de la tabla 'medicos' antes de insertar
BEGIN
    -- Intentar acceder a la tabla 'medicos' en el esquema 'WRITER' con EXECUTE IMMEDIATE
    BEGIN
        EXECUTE IMMEDIATE 'SELECT COUNT(*) FROM WRITER.medicos';
        DBMS_OUTPUT.PUT_LINE('La tabla MEDICOS existe en el esquema WRITER.');

        -- Insertar en la tabla 'medicos'
        EXECUTE IMMEDIATE 'INSERT INTO WRITER.medicos (dni, numLicencia, nombre, especialidad,
            ↪ telefono)
                           SELECT ''12345678A'', 98765, ''Dr. Juan Pérez'', ''Cardiología'', ''
                               ↪ 600123456''
                           FROM dual
                           WHERE NOT EXISTS (SELECT 1 FROM WRITER.medicos WHERE dni = ''12345678A''
                               ↪ )';

        EXECUTE IMMEDIATE 'INSERT INTO WRITER.medicos (dni, numLicencia, nombre, especialidad,
            ↪ telefono)
                           SELECT ''87654321B'', 12366, ''Dra. Ana Gómez'', ''Neurología'', ''
                               ↪ 611987654''
                           FROM dual
                           WHERE NOT EXISTS (SELECT 1 FROM WRITER.medicos WHERE dni = ''87654321B''
                               ↪ )';
    EXCEPTION
        WHEN OTHERS THEN
            DBMS_OUTPUT.PUT_LINE('La tabla MEDICOS no existe en el esquema WRITER. No se realizar
                ↪ án inserciones.');
    END;
END;
/

-- Comprobación de existencia de la tabla 'pacientes' antes de insertar
BEGIN
    -- Intentar acceder a la tabla 'pacientes' en el esquema 'WRITER' con EXECUTE IMMEDIATE
    BEGIN
        EXECUTE IMMEDIATE 'SELECT COUNT(*) FROM WRITER.pacientes';
        DBMS_OUTPUT.PUT_LINE('La tabla PACIENTES existe en el esquema WRITER.');

        -- Insertar en la tabla 'pacientes'
        EXECUTE IMMEDIATE 'INSERT INTO WRITER.pacientes (dni, nss, nombre, telefono)
                           SELECT ''11111111A'', 1000001, ''Carlos López'', ''654123987''
                           FROM dual
                           WHERE NOT EXISTS (SELECT 1 FROM WRITER.pacientes WHERE dni = ''11111111A
                               ↪ '')';

        EXECUTE IMMEDIATE 'INSERT INTO WRITER.pacientes (dni, nss, nombre, telefono)
                           SELECT ''22222222B'', 1000002, ''María Fernández'', ''623987654''
                           FROM dual
                           WHERE NOT EXISTS (SELECT 1 FROM WRITER.pacientes WHERE dni = ''22222222B
                               ↪ '')';

        EXECUTE IMMEDIATE 'INSERT INTO WRITER.pacientes (dni, nss, nombre, telefono)
                           SELECT ''33333333C'', 1000003, ''Pedro Sánchez'', ''698741236''
                           FROM dual
                           WHERE NOT EXISTS (SELECT 1 FROM WRITER.pacientes WHERE dni = ''33333333C
                               ↪ '')';
```

```
        EXECUTE IMMEDIATE 'INSERT INTO WRITER.pacientes (dni, nss, nombre, telefono)
                           SELECT ''44444444D'', 1000004, ''Lucía Rodríguez'', ''677852963''
                           FROM dual
                           WHERE NOT EXISTS (SELECT 1 FROM WRITER.pacientes WHERE dni = ''44444444D
                               ↪ '')';
    EXCEPTION
        WHEN OTHERS THEN
            DBMS_OUTPUT.PUT_LINE('La tabla PACIENTES no existe en el esquema WRITER. No se
                ↪ realizarán inserciones.');
    END;
END;
/

-- Comprobación de existencia de la tabla 'pruebas' antes de insertar
BEGIN
    -- Intentar acceder a la tabla 'pruebas' en el esquema 'WRITER' con EXECUTE IMMEDIATE
    BEGIN
        EXECUTE IMMEDIATE 'SELECT COUNT(*) FROM WRITER.pruebas';
        DBMS_OUTPUT.PUT_LINE('La tabla PRUEBAS existe en el esquema WRITER.');

        -- Insertar en la tabla 'pruebas'
        EXECUTE IMMEDIATE 'INSERT INTO WRITER.pruebas (dni_medico, dni_paciente, tipo_prueba,
            ↪ fecha, resultado)
                           SELECT ''12345678A'', ''11111111A'', ''Electrocardiograma'', TO_DATE(''
                               ↪ 2024-02-15'', ''YYYY-MM-DD''), ''Normal''
                           FROM dual
                           WHERE NOT EXISTS (SELECT 1 FROM WRITER.pruebas WHERE dni_medico = ''
                               ↪ 12345678A'' AND dni_paciente = ''11111111A'' AND tipo_prueba = '
                               ↪ 'Electrocardiograma'' AND fecha = TO_DATE(''2024-02-15'', ''YYYY
                               ↪ -MM-DD''))';

        EXECUTE IMMEDIATE 'INSERT INTO WRITER.pruebas (dni_medico, dni_paciente, tipo_prueba,
            ↪ fecha, resultado)
                           SELECT ''12345678A'', ''22222222B'', ''Ecografía'', TO_DATE(''2024-02-10
                               ↪ '', ''YYYY-MM-DD''), ''Sin anomalías''
                           FROM dual
                           WHERE NOT EXISTS (SELECT 1 FROM WRITER.pruebas WHERE dni_medico = ''
                               ↪ 12345678A'' AND dni_paciente = ''22222222B'' AND tipo_prueba = '
                               ↪ 'Ecografía'' AND fecha = TO_DATE(''2024-02-10'', ''YYYY-MM-DD'')
                               ↪ )';

        EXECUTE IMMEDIATE 'INSERT INTO WRITER.pruebas (dni_medico, dni_paciente, tipo_prueba,
            ↪ fecha, resultado)
                           SELECT ''87654321B'', ''33333333C'', ''Resonancia magnética'', TO_DATE('
                               ↪ '2024-02-18'', ''YYYY-MM-DD''), ''Lesión detectada en L3-L4''
                           FROM dual
                           WHERE NOT EXISTS (SELECT 1 FROM WRITER.pruebas WHERE dni_medico = ''
                               ↪ 87654321B'' AND dni_paciente = ''33333333C'' AND tipo_prueba = '
                               ↪ 'Resonancia magnética'' AND fecha = TO_DATE(''2024-02-18'', ''
                               ↪ YYYY-MM-DD''))';

        EXECUTE IMMEDIATE 'INSERT INTO WRITER.pruebas (dni_medico, dni_paciente, tipo_prueba,
            ↪ fecha, resultado)
                           SELECT ''87654321B'', ''44444444D'', ''Análisis de sangre'', TO_DATE(''
                               ↪ 2024-02-20'', ''YYYY-MM-DD''), ''Niveles normales''
```

```
                        FROM dual
                        WHERE NOT EXISTS (SELECT 1 FROM WRITER.pruebas WHERE dni_medico = ''
                            ↪ 87654321B'' AND dni_paciente = ''44444444D'' AND tipo_prueba = '
                            ↪ 'Análisis de sangre'' AND fecha = TO_DATE(''2024-02-20'', ''YYYY
                            ↪ -MM-DD''))';
    EXCEPTION
        WHEN OTHERS THEN
            DBMS_OUTPUT.PUT_LINE('La tabla PRUEBAS no existe en el esquema WRITER. No se realizar
                ↪ án inserciones.');
    END;
END;
/
```

Para ejecutar este script utilizando SQL*Plus, utilizamos el siguiente comando:

```
nano InsercionTablasOracle.sql
docker cp ~/bd2/oracle/InsercionTablasOracle.sql oracle-xe:/opt/oracle
docker exec -it oracle-xe bash
sqlplus writer/writerPass@localhost:1521/XEPDB1
@/opt/oracle/InsercionTablasOracle.sql
```

4.3.5 IBM DB2

4.3.6 Consultas

En el apartado de consultas, el usuario `reader` lleva a cabo una serie de operaciones para acceder a la información contenida en las tablas medicos, pacientes y pruebas que fueron previamente creadas y pobladas por el usuario `writer`.

El proceso inicia con la conexión a la base de datos Oracle usando las credenciales del usuario `reader`.

```
CONNECT reader/readerPass@localhost:1521/XEPDB1;
```

Para realizar las consultas han sido agrupadas todas ellas en un script llamado "ConsultasOracle".
El contenido del script "ConsultasOracle.sql" incluye:

```
-- Conexión como usuario reader con su contraseña
CONNECT reader/readerPass@localhost:1521/XEPDB1;

-- Comprobación de existencia de la tabla 'medicos' antes de mostrar los datos
DECLARE
    v_exists NUMBER;
BEGIN
    SELECT COUNT(*) INTO v_exists
    FROM all_tables
    WHERE table_name = 'MEDICOS' AND owner = 'WRITER';

    IF v_exists > 0 THEN
        EXECUTE IMMEDIATE 'SELECT * FROM WRITER.medicos';
    END IF;
END;
/

-- Comprobación de existencia de la tabla 'pacientes' antes de mostrar los datos
DECLARE
    v_exists NUMBER;
BEGIN
    SELECT COUNT(*) INTO v_exists
```

```
    FROM all_tables
    WHERE table_name = 'PACIENTES' AND owner = 'WRITER';

    IF v_exists > 0 THEN
        EXECUTE IMMEDIATE 'SELECT * FROM WRITER.pacientes';
    END IF;
END;
/

-- Comprobación de existencia de la tabla 'pruebas' antes de mostrar los datos
DECLARE
    v_exists NUMBER;
BEGIN
    SELECT COUNT(*) INTO v_exists
    FROM all_tables
    WHERE table_name = 'PRUEBAS' AND owner = 'WRITER';

    IF v_exists > 0 THEN
        EXECUTE IMMEDIATE 'SELECT * FROM WRITER.pruebas';
    END IF;
END;
/

-- Comprobación de existencia de la tabla 'pacientes' antes de realizar la búsqueda
DECLARE
    v_exists NUMBER;
BEGIN
    SELECT COUNT(*) INTO v_exists
    FROM all_tables
    WHERE table_name = 'PACIENTES' AND owner = 'WRITER';

    IF v_exists > 0 THEN
        EXECUTE IMMEDIATE 'SELECT * FROM WRITER.pacientes WHERE nombre LIKE ''%María%''';
    END IF;
END;
/

-- Comprobación de existencia de la tabla 'pruebas' antes de realizar la consulta por fecha
DECLARE
    v_exists NUMBER;
BEGIN
    SELECT COUNT(*) INTO v_exists
    FROM all_tables
    WHERE table_name = 'PRUEBAS' AND owner = 'WRITER';

    IF v_exists > 0 THEN
        EXECUTE IMMEDIATE 'SELECT * FROM WRITER.pruebas WHERE fecha = TO_DATE(''2024-02-15'', ''
            ↪ YYYY-MM-DD'')';
    END IF;
END;
/

-- Comprobación de existencia de las tablas 'medicos', 'pacientes' y 'pruebas' antes de realizar
    ↪ la consulta detallada
DECLARE
    v_exists_medicos NUMBER;
```

```
    v_exists_pacientes NUMBER;
    v_exists_pruebas NUMBER;
BEGIN
    SELECT COUNT(*) INTO v_exists_medicos
    FROM all_tables
    WHERE table_name = 'MEDICOS' AND owner = 'WRITER';

    SELECT COUNT(*) INTO v_exists_pacientes
    FROM all_tables
    WHERE table_name = 'PACIENTES' AND owner = 'WRITER';

    SELECT COUNT(*) INTO v_exists_pruebas
    FROM all_tables
    WHERE table_name = 'PRUEBAS' AND owner = 'WRITER';

    IF v_exists_medicos > 0 AND v_exists_pacientes > 0 AND v_exists_pruebas > 0 THEN
        EXECUTE IMMEDIATE '
            SELECT m.nombre AS medico, p.nombre AS paciente, pr.tipo_prueba, pr.fecha, pr.
                ↪ resultado
            FROM WRITER.pruebas pr
            JOIN WRITER.medicos m ON pr.dni_medico = m.dni
            JOIN WRITER.pacientes p ON pr.dni_paciente = p.dni';
    END IF;
END;
/
```

Para ejecutar este script, se debe ejecutar este comando:

```
sqlplus reader/readerPass@localhost:1521/XEPDB1 @ConsultasOracle.sql
```

4.4 HBase

Tras la desestimación de Kerberos, se procedió a la configuración de HBase sin autenticación centralizada. En esta sección se describen las operaciones realizadas para gestionar el acceso de usuarios, la creación de espacios, tablas y la ejecución de consultas.

4.4.1 Creación superusuarios:

El primer paso consistió en otorgar permisos administrativos al usuario ‘admin‘. Esto se logró mediante el siguiente comando en HBase Shell:

```
hbase(main):028:0* grant 'admin', 'RWCXA'
Took 0.4414 seconds
```

Donde los permisos otorgados fueron:

```
    {R} - Read (Lectura)
    {W} - Write (Escritura)
    {C} - Create (Creación)
    {X} - Execute (Ejecución)
    {A} - Admin (Administración)
```

Este comando habilitó al usuario ‘admin‘ con privilegios totales sobre la base de datos de HBase.

4.4.2 Creación de tablespaces

En HBase, un espacio de nombres (namespace) permite organizar las tablas en una estructura similar a los esquemas en SQL. Se creó un espacio de nombres específico:

```
hbase(main):033:0* create_namespace 'espacioHBase'
Took 0.2981 seconds
```

Para verificar la correcta creación del namespace, se ejecutó:

```
hbase(main):034:0> list_namespace
NAMESPACE
espacioHBase
default
hbase
3 row(s)
Took 0.0593 seconds
```

El namespace ‘espacioHBase‘ apareció correctamente en la lista, junto con los espacios de nombres ‘default‘ y ‘hbase‘, confirmando su creación exitosa.

4.4.3 Creación de usuarios de escritura y lectura

Se crearon usuarios con diferentes roles de acceso. El usuario ‘escritor‘ recibió permisos de lectura y escritura sobre el namespace ‘espacioHBase‘:

```
hbase(main):038:0* grant 'escritor', 'RW', '@espacioHBase'
Took 0.0568 seconds
```

Mientras que el usuario ‘lector‘ recibió únicamente permisos de lectura:

```
hbase(main):039:0> grant 'lector', 'R', '@espacioHBase'
Took 0.0511 seconds
```

Para verificar los permisos, se ejecutó:

```
hbase(main):040:0> scan 'hbase:acl'
ROW COLUMN+CELL
 @espacioHBase column=l:escritor, timestamp=1740387002806, value=RW
 @espacioHBase column=l:lector, timestamp=1740387008040, value=R
 hbase:acl column=l:admin, timestamp=1740386869990, value=RWXCA
2 row(s)
Took 0.0290 seconds
```

Aquí se pudo confirmar que ‘admin‘ tenía todos los permisos (‘RWXCA‘), ‘escritor‘ tenía ‘RW‘, y ‘lector‘ tenía únicamente ‘R‘.

La creación de usuarios también se puede realizar mediante este script:

```
1  #!/bin/bash
2
3  # Función para verificar si un usuario existe
4  check_user_exists() {
5    local user=$1
6    # Escanear la tabla hbase:acl y buscar el usuario en la columna 'l'
7    local result=$(docker exec -i hbase-db hbase shell <<EOF
8  scan 'hbase:acl', {COLUMNS => 'l'}
9  EOF
10   )
```

```
    # Buscar el patrón "l:<usuario>" en la salida
    if echo "$result" | grep -q "l:$user"; then
      echo "El usuario '$user' ya existe."
      return 0  # Usuario encontrado
    else
      echo "El usuario '$user' no existe."
      return 1  # Usuario no encontrado
    fi
  }

  # Función para crear un usuario
  create_user() {
    local user=$1
    local role=$2
    local namespace=$3

    # Comprobamos si el usuario ya existe
    if ! check_user_exists "$user"; then
      echo "Creando el usuario '$user' con el rol '$role' en el espacio de nombres
      ↪  '$namespace'..."

      # Crear el usuario
      docker exec -i hbase-db hbase shell <<EOF
  grant '$user', '$role', '@$namespace'
  EOF

      if [ $? -eq 0 ]; then
        echo "Usuario '$user' creado con éxito."
      else
        echo "Error al crear el usuario '$user'." >&2
        return 1
      fi
    fi
  }

  # Crear usuarios
  create_user 'admin' 'RWXCA' 'espacioHBase'
  create_user 'escritor' 'RW' 'espacioHBase'
  create_user 'hiho' 'R' 'espacioHBase'
```

4.4.4 Creación e inserción de tablas

Se crearon tres tablas dentro del espacio de nombres ‘espacioHBase‘ para almacenar información médica:

```
    medicos: Para almacenar información de los médicos.
    pacientes: Para registrar a los pacientes.
    pruebas: Para almacenar pruebas médicas realizadas.
```

Para crear las tablas, se ejecutó un script, el cual además mostraba al final las tablas creadas:

```
drop 'medicos'
drop 'pacientes'
```

```
drop 'pruebas'

# Crear la tabla de médicos
create 'medicos', {NAME => 'info', VERSIONS => 1}

# Crear la tabla de pacientes
create 'pacientes', {NAME => 'info', VERSIONS => 1}

# Crear la tabla de pruebas médicas
create 'pruebas', {NAME => 'info', VERSIONS => 1}

# Verificar que las tablas se crearon
list
```

Para ejecutar este script, utilizamos el siguiente comando:

```
nano CreacionTablasHBase.hbase
cat CreacionTablasHBase.hbase | docker exec -i hbase-db hbase shell
```

A continuación, se llevó a cabo la inserción en las tablas que previamente se han creado. Esto también se realizó mediante un script:

```
    #!/bin/bash

    # Función para obtener la lista de tablas
    get_tables() {
      docker exec -i hbase-db hbase shell "list"
    }

    tables=$(get_tables)

    # Verificar si las tablas 'medicos', 'pacientes', y 'pruebas' existen antes de insertar
    if [[ "$tables" == *"medicos"* ]]; then
      echo "La tabla 'medicos' ya existe, insertando datos..."
      docker exec -i hbase-db hbase shell "put 'medicos', '12345678A', 'info:numLicencia',
      ↪  '98765'"
      docker exec -i hbase-db hbase shell "put 'medicos', '12345678A', 'info:nombre', 'Dr. Juan
      ↪  Pérez'"
      docker exec -i hbase-db hbase shell "put 'medicos', '12345678A', 'info:especialidad',
      ↪  'Cardiología'"
      docker exec -i hbase-db hbase shell "put 'medicos', '12345678A', 'info:telefono',
      ↪  '600123456'"
    else
      echo "La tabla 'medicos' no existe, no se insertarán datos."
    fi

    # Verificar lo mismo para 'pacientes' y 'pruebas'
    if [[ "$tables" == *"pacientes"* ]]; then
      echo "La tabla 'pacientes' ya existe, insertando datos..."
      docker exec -i hbase-db hbase shell "put 'pacientes', '11111111A', 'info:nss', '1000001'"
    else
      echo "La tabla 'pacientes' no existe, no se insertarán datos."
    fi

    if [[ "$tables" == *"pruebas"* ]]; then
```

```
    echo "La tabla 'pruebas' ya existe, insertando datos..."
    docker exec -i hbase-db hbase shell "put 'pruebas', '1', 'info:dni_medico', '12345678A'"
  else
    echo "La tabla 'pruebas' no existe, no se insertarán datos."
  fi
```

Este script inserta correctamente todos los datos en cada tabla, y para ejecutarlo, se debe ejecutar este comando:

```
nano InsercionTablasHBase.sh
chmod +x InsercionTablasHBase.sh
./InsercionTablasHBase.sh
```

4.4.5 Consultas

Una vez creadas las tablas y agregados los datos, se realizaron consultas para verificar la información almacenada.

De nuevo, las consultas fueron ejecutadas mediante un script. El script de las consultas:

```
#!/bin/bash

# Función para obtener la lista de tablas
get_tables() {
  docker exec -i hbase-db hbase shell "list"
}

tables=$(get_tables)

# Verificar si la tabla 'medicos' existe antes de hacer la consulta
if [[ "$tables" == *"medicos"* ]]; then
  echo "La tabla 'medicos' existe, mostrando todas las filas..."
  docker exec -i hbase-db hbase shell "scan 'medicos'"
else
  echo "La tabla 'medicos' no existe, no se puede realizar la consulta."
fi

# Verificar si la tabla 'pacientes' existe antes de hacer la consulta
if [[ "$tables" == *"pacientes"* ]]; then
  echo "La tabla 'pacientes' existe, mostrando todas las filas..."
  docker exec -i hbase-db hbase shell "scan 'pacientes'"
else
  echo "La tabla 'pacientes' no existe, no se puede realizar la consulta."
fi

# Verificar si la tabla 'pruebas' existe antes de hacer la consulta
if [[ "$tables" == *"pruebas"* ]]; then
  echo "La tabla 'pruebas' existe, mostrando todas las filas..."
  docker exec -i hbase-db hbase shell "scan 'pruebas'"
else
  echo "La tabla 'pruebas' no existe, no se puede realizar la consulta."
fi

# Consulta para buscar información de un paciente específico
if [[ "$tables" == *"pacientes"* ]]; then
```

```
    echo "Buscando información de un paciente específico..."
    docker exec -i hbase-db hbase shell "scan 'pacientes', {FILTER =>
    ↪ \"SingleColumnValueFilter('info', 'nombre', =, 'substring:María')\"}"
  else
    echo "La tabla 'pacientes' no existe, no se puede realizar la consulta."
  fi

  # Consulta para obtener las pruebas realizadas en una fecha específica
  if [[ "$tables" == *"pruebas"* ]]; then
    echo "Buscando pruebas realizadas en una fecha específica..."
    docker exec -i hbase-db hbase shell "scan 'pruebas', {FILTER =>
    ↪ \"SingleColumnValueFilter('info', 'fecha', =, 'binary:2024-02-15')\"}"
  else
    echo "La tabla 'pruebas' no existe, no se puede realizar la consulta."
  fi

  # Consulta para obtener los detalles de las pruebas médicas
  if [[ "$tables" == *"pruebas"* ]]; then
    echo "Mostrando detalles de las pruebas médicas..."
    docker exec -i hbase-db hbase shell "scan 'pruebas'"
  else
    echo "La tabla 'pruebas' no existe, no se puede realizar la consulta."
  fi

```

Para poder ejecutar el script de consultas, se debe ejecutar el siguiente comando:

```
nano ConsultasHBase.sh
chmod +x ConsultasHBase.sh
./ConsultasHBase.sh
```

5 Comentarios Acerca de las Licencias

A continuacion se detallan los aspectos mas relevantes de las licencias de los SGBD mencionados.

5.1 PostgreSQL

- **Tipo de licencia**: Licencia PostgreSQL (similar a BSD/MIT)
- **Caracteristicas**:
  - Codigo abierto para uso comercial.
  - Sin obligacion de compartir modificaciones.
- **Fragmento clave**:

```
Permission to use, copy, modify, and distribute this software [...]
without fee, and without a written agreement is hereby granted...
```

- **Implicaciones**:
  - Adecuado para proyectos flexibles sin coste.
  - Soporte profesional opcional (de pago).

5.2 Oracle XE

- **Tipo de licencia**: Propietaria (gratuita con limites)
- **Caracteristicas**:
  - 12GB almacenamiento maximo.
  - Limitado a 2 CPUs.
- **Fragmento clave**:

```
Oracle Database XE may be used for free for development [...]
(subject to resource constraints)
```

- **Implicaciones**:
  - Requiere licencia comercial para produccion.
  - Auditorias de cumplimiento.

5.3 Apache Cassandra

- **Tipo de licencia**: Apache 2.0
- **Caracteristicas**:
  - Modificaciones permitidas.
  - Atribucion obligatoria.
- **Fragmento clave**:

```
You must give recipients a copy of this License...
```

- **Implicaciones**:
  - Compatible con software propietario.
  - Requiere mencion de copyright.

5.4 Apache HBase

- **Tipo de licencia**: Apache 2.0
- **Fragmento clave**:

```
Redistributions must reproduce copyright notice...
```

- **Implicaciones**:
  - Mismas condiciones que Cassandra.
  - Debe incluir disclaimer (declaracion legal) original.

5.5 IBM DB2

- **Tipo de licencia**: Propietaria (suscripcion)
- **Caracteristicas**:
  - Community Edition con limitaciones.
  - Modelo por nucleos CPU.
- **Fragmento clave**:

```
The Program may not be used for production [...]
unless IBM has received payment...
```

- **Implicaciones**:
  - Costes escalables en produccion.
  - Verificacion de licencias periodica.

5.6 Resumen Comparativo

- **Codigo abierto**:
  - PostgreSQL/Cassandra/Apache HBase: Estos sistemas son ejemplos de bases de datos de codigo abierto. Son libres para redistribuir. Esto significa que puedes usar estos sistemas de basos de datos sin necesidad de pagar licencias u obtener permiso de una entidad propietaria.
  - Apache Cassandra: Atribucion requerida, es decir, que son bases de datos de codigo abierto y gratuitas, pero si se hace uso de ella, se debe dar credito a la autoria original.

- **Propietarias**:
  - Oracle XE/IBM DB2: Son bases de datos comerciales, lo cual significa que estan controladas por empresas propietarias (Oracle Corporation para Oracle y IBM para DB2). Estas bases de datos comerciales, suelen tener costos variables asociados con el escalado del sistema (a medida que la base de datos crece, ya sea en terminos de usuarios, volumen de datos o funcionalidades adicionales, los costos de uso pueden aumentar.

6 Esfuerzos invertidos

El proceso de decisión del entorno y plataforma de despliegue se realizó conjuntamente.

Table 1: Distribución de horas por integrante

Tarea	**Sergio Isla**	**Irene Pascual**	**Athanasios Usero**
Instalación SGBD	3 horas	3 horas	3 horas
Configuración de Docker	0.5 horas	0.25 horas	2 horas
Configurar archivos de configuración	0.5 horas	2 horas	1 horas
Desarrollo de scripts	6 horas	7 horas	6.5 horas
Configuración acceso remoto	1 horas	0.5 horas	1 horas
Creación de tablas/espacios	1 horas	0.5 horas	1 horas
Gestión de usuarios	6 horas	2 horas	3 horas
Inserciones de datos	2 horas	1 horas	1 horas
Consultas	1 horas	0.5 horas	0.5 horas
Documentación y memoria	3 horas	3 horas	5.25 horas
Resolución de problemas	4 horas	8 horas	5 horas
Total por integrante	27.5 horas	27.5 horas	29.5 horas

7 Referencias

A continuación se detallan las fuentes y recursos utilizados para la realización de esta práctica:

- **Oficial PostgreSQL Documentation**:
  `https://www.postgresql.org/docs/`
  Documentación oficial de PostgreSQL, utilizada para consultar la sintaxis de comandos SQL, configuración de roles y permisos.
- **Oracle XE Documentation**:
  `https://docs.oracle.com/en/database/oracle/oracle-database/`
  Guía oficial de Oracle Database Express Edition (XE), empleada para la instalación, configuración y administración básica.
- **Apache Cassandra Documentation**:
  `https://cassandra.apache.org/doc/latest/`
  Documentación oficial de Apache Cassandra, útil para entender la creación de keyspaces, tablas y la configuración de seguridad.
- **Apache HBase Documentation**:
  `https://hbase.apache.org/book.html`
  Guía oficial de Apache HBase, consultada para la configuración de namespaces, tablas y la gestión de permisos.
- **IBM DB2 Documentation**:
  `https://www.ibm.com/docs/en/db2/`
  Documentación oficial de IBM DB2, utilizada para la creación de bases de datos, esquemas y la gestión de usuarios.
- **Docker Documentation**:
  `https://docs.docker.com/`
  Recursos oficiales de Docker, empleados para la creación y configuración de contenedores para cada SGBD.
- **DBeaver Documentation**:
  `https://dbeaver.io/docs/`
  Guía oficial de DBeaver, utilizada para configurar conexiones remotas a los SGBD.
- **Stack Overflow**:
  `https://stackoverflow.com/`
  Foro de preguntas y respuestas utilizado para resolver problemas técnicos específicos durante la instalación y configuración.
- **GitHub Repositories**:
  `https://github.com/`
  Repositorios de GitHub consultados para encontrar scripts y configuraciones de ejemplo para los SGBD.

Nota: Todas las referencias fueron consultadas durante el desarrollo de la práctica para garantizar la correcta instalación, configuración y uso de los sistemas gestores de bases de datos.