Memoria Técnica Práctica 1
Bases de Datos 2
Curso 2024/2025
Universidad de Zaragoza

Diego Mateo Lorente - 873338
Daniel Simón Gayán - 870984
Carlos Solana Melero - 872815

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

Para esta práctica, se optó por utilizar Docker en lugar de instalar cada gestor de bases de datos manualmente en la máquina. Esto se debe a que Docker permite ejecutar cada SGBD en un entorno aislado, evitando conflictos con otros programas y facilitando su configuración.

Una de las principales ventajas de Docker es la rapidez y simplicidad con la que se pueden desplegar los servicios. Gracias a Docker Compose, se definió un archivo (docker-compose.yml) con todos los SGBD necesarios, incluyendo PostgreSQL, Oracle XE, Cassandra, HBase e IBM Db2. Esto permitió levantar todo el entorno con un solo comando, sin necesidad de instalar cada base de datos de forma independiente (que también se puede hacer levantando contenedores concretos).

Otra ventaja importante es la persistencia de datos. Aunque los contenedores pueden eliminarse y recrearse, los datos se almacenan en volúmenes de Docker, asegurando que la información no se pierda entre reinicios.

Además, Docker facilita la configuración de red y acceso remoto. Se expusieron los puertos de los SGBD para poder acceder desde otras aplicaciones o incluso desde otro equipo en la misma red. Esto permitió probar la conectividad sin modificar configuraciones complejas en el sistema operativo.

Instalación y Administración Básica de los SGBD

Comentar versiones de cada SGBD, si se quiere guardar volumen persistente de cada uno, y pruebas de que funciona (acceso a la terminal del SGBD, crear BD, arrancar y parar gestor (entiendo que será el contenedor en sí), pruebas de los distinso usuarios creados (uno puede leer pero no escribir, esas cosas)

“” HACER LOS DEMÁS”

PostgreSQL

Creación SGBD

Se utilizó Docker Compose para desplegar PostgreSQL con la siguiente configuración:

```yaml
postgres:
  image: postgres
  shm_size: 128mb
  environment:
    POSTGRES_USER: postgres
    POSTGRES_PASSWORD: admin
  ports:
    - "5432:5432"
  # Se mantiene el volumen de datos para persistencia
  volumes:
    - postgres-data:/var/lib/postgresql/data
```

Esta configuración expone PostgreSQL en el puerto 5432, establece al usuario administrador (`postgres`) utiliza un volumen persistente para almacenar los datos.

Se ha creado el superusuario postgres con contraseña admin desde el que se realizarán todas las pruebas.

Si se quisiera crear otro manualmente se podría hacer con esta sentencia por ejemplo.

CREATE USER admin_secure WITH PASSWORD 'UnaContraseñaMuySegura123!' SUPERUSER CREATEDB CREATEROLE LOGIN;

Y una vez ejecutada verificar su existencia con el comando \du que muestra los usuarios o probando directamente a conectarse con el comando

docker exec -it p1-postgres-1 psql -U admin_secure -W postgres

Poblado

Una vez estaba activa la bd se comprobó que estaba activa con en comando docker ps (haría falta usar sudo si el usuario no está en el grupo docker)

```
solana@solanad-os:~/Documents/inginf/bases2/p1$ sudo docker ps
CONTAINER ID   IMAGE                COMMAND                  CREATED      STATUS          PORTS                                       NAMES
3e944fac610d   dpage/pgadmin4       "/entrypoint.sh"         2 days ago   Up 3 seconds    443/tcp, 0.0.0.0:5050->80/tcp, [::]:5050->80/tcp   p1-pgadmin-1
ffe282eae455   postgres             "docker-entrypoint.s…"   2 days ago   Up 4 seconds    0.0.0.0:5432->5432/tcp, :::5432->5432/tcp          p1-postgres-1
```

Una vez activa para probar su correcto funcionamiento se ha creado este script para poblar la bd con datos de prueba

```
#!/bin/bash
# Script: docker_postgres_tables.sh
# Objetivo: Ingresar al contenedor p1-postgres-1 y crear (si no existe)
la tabla 'my_table'
#           para luego poblarla con datos de ejemplo, de forma
idempotente.
# Se asume que el contenedor tiene psql disponible.

CONTAINER_NAME="p1-postgres-1"
DATABASE="postgres"
ADMIN_USER="postgres"

echo "Ejecutando script dentro del contenedor '$CONTAINER_NAME' para
crear y poblar tablas en PostgreSQL..."

# Crear tabla si no existe
echo "Creando tabla 'my_table' (si no existe)..."
docker exec -it $CONTAINER_NAME psql -U $ADMIN_USER -d $DATABASE -c
"CREATE TABLE IF NOT EXISTS my_table (
   id SERIAL PRIMARY KEY,
   name VARCHAR(100),
   created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);"
echo "Tabla 'my_table' creada (o ya existía)."

# Insertar datos de ejemplo solo si la tabla está vacía
echo "Insertando datos de ejemplo en 'my_table' (si está vacía)..."
docker exec -it $CONTAINER_NAME psql -U $ADMIN_USER -d $DATABASE -c
"INSERT INTO my_table (name)
SELECT 'Sample Data'
WHERE NOT EXISTS (SELECT 1 FROM my_table);"
echo "Datos insertados (o ya estaban presentes)."
```

El script Bash implementado realiza las siguientes operaciones principales:

- **Conexión al contenedor PostgreSQL:**
  El script establece conexión con el contenedor Docker llamado `p1-postgres-1`, utilizando las credenciales del usuario administrador PostgreSQL.

- **Creación de la tabla (`my_table`):**
  Mediante el comando SQL `CREATE TABLE IF NOT EXISTS`, se asegura que la tabla `my_table` sea creada únicamente si no existe previamente, evitando errores en caso de múltiples ejecuciones.

- **Inserción idempotente de datos:**
  Inserta una fila inicial (`'Sample Data'`) solo si la tabla está vacía. Esto garantiza la no duplicación de datos y permite ejecutar repetidamente el script sin efectos negativos.

Poblado

Se ha implementado un script en Bash denominado `roles.sh`.

El objetivo principal es gestionar eficazmente el control de acceso y seguridad sobre el espacio de datos, creando al menos tres tipos distintos de usuarios con perfiles específicos claramente diferenciados:

- **Un usuario superusuario (administrador)** con todos los privilegios para realizar tareas críticas de administración.
- **Un usuario estándar con privilegios de escritura**, con capacidad para insertar, modificar y eliminar datos en tablas existentes.
- **Un usuario estándar con privilegios únicamente de lectura**, capaz de consultar datos pero sin posibilidad de alterarlos.

Y las pruebas para ver su correcto funcionamiento.

Este documento detalla el trabajo realizado con PostgreSQL desplegado mediante Docker, centrándose específicamente en la creación, configuración y validación automatizada de usuarios con diferentes roles y permisos de acceso sobre el espacio de datos. Para ello se elaboró un script en Bash llamado "docker_postgres_roles.sh", el cual automatiza estas operaciones permitiendo un despliegue repetible, seguro y eficiente.

El script tiene como objetivo principal crear tres tipos distintos de usuarios dentro del servidor PostgreSQL desplegado en Docker:

- Un superusuario con privilegios totales.
- Un usuario estándar con permisos de escritura.
- Un usuario restringido exclusivamente para lectura.

Además, el script ejecuta automáticamente ciertas comprobaciones de interacción sobre una tabla previamente existente ("my_table"), verificando así que los permisos asignados funcionan según lo esperado.

Detalle de la operativa llevada a cabo en el script:

1. Definición de variables iniciales y credenciales seguras

En primer lugar, el script define claramente el nombre del contenedor Docker PostgreSQL (denominado "p1-postgres-1") junto con la base de datos ("postgres") y el usuario administrador ("postgres") que se empleará para ejecutar comandos administrativos.

- Un superusuario denominado "my_superuser"..
- Un usuario estándar con permisos de escritura denominado "my_write_user".
- Un usuario con permisos exclusivamente de lectura denominado "my_read_user".

2. Creación del superusuario con privilegios máximos (administrador)

Posteriormente, se crea un usuario con los máximos privilegios administrativos dentro del servidor PostgreSQL (superusuario). Esto se realiza mediante una consulta SQL que verifica previamente si el rol existe en el catálogo del servidor (tabla "pg_catalog.pg_roles") antes de crearlo. De esta forma, la creación es completamente idempotente: si el usuario ya existe, el script no realiza ningún cambio adicional y no provoca errores al repetirse.

Una vez creado el superusuario ("my_superuser"), se procede a validar que el mismo puede conectarse correctamente al servidor PostgreSQL, ejecutando una consulta sencilla para mostrar información de conexión actual. Esto proporciona confirmación inmediata de que las credenciales generadas funcionan correctamente.

3. Creación del usuario estándar con permisos de escritura sobre las tablas

A continuación, se procede a crear el usuario estándar denominado "my_write_user". Este usuario es diseñado explícitamente para operaciones que modifiquen datos en la base de datos, tales como insertar nuevos registros, actualizarlos o borrarlos. Nuevamente, se emplea una verificación previa que comprueba la no existencia de dicho usuario antes de proceder a su creación.

Al usuario con permisos de escritura se le asignan privilegios específicos sobre todas las tablas existentes en el esquema "public": concretamente privilegios para insertar, actualizar y borrar datos. Adicionalmente, se configura que estos privilegios se asignen automáticamente por defecto a cualquier nueva tabla que se cree en el futuro, asegurando consistencia y simplificando la administración futura del sistema.

Debido a que la tabla "my_table" posee un campo "id" de tipo SERIAL (que genera automáticamente valores mediante una secuencia asociada), es necesario otorgar explícitamente permisos de uso sobre dicha secuencia ("my_table_id_seq"). Esto se realiza mediante la sentencia SQL que concede al usuario permisos de "USAGE" y "SELECT" en dicha secuencia.

3. Creación del usuario estándar con permisos exclusivamente de lectura

A continuación, se procede a crear un usuario diseñado exclusivamente para realizar operaciones de consulta ("SELECT") sobre la base de datos, denominado "my_read_user". Al igual que en los casos anteriores, la creación es idempotente: solo se realiza si el usuario no existe previamente.

A este usuario se le conceden privilegios únicamente para realizar consultas SELECT sobre todas las tablas existentes en el esquema "public". Además, mediante "ALTER DEFAULT PRIVILEGES", se asegura que también tendrá estos permisos automáticamente sobre cualquier tabla que sea creada posteriormente dentro del esquema público. Con ello se garantiza que dicho usuario estará limitado estrictamente a operaciones de lectura, conforme con los requisitos especificados.

3. Bloque de pruebas funcionales para validar la correcta configuración de usuarios y permisos

Tras la creación exitosa de los usuarios y la asignación correspondiente de privilegios, el script realiza una serie de pruebas de validación destinadas a garantizar el correcto funcionamiento de las asignaciones hechas anteriormente.

La prueba inicial consiste en insertar un registro de prueba en la tabla "my_table" mediante el usuario con permisos de escritura ("my_write_user"). Para ello, el script ejecuta una sentencia INSERT que añade un nuevo registro de prueba identificado como "Test insert by write user".

Una vez completada la inserción, se realiza una consulta SELECT empleando al usuario con permisos exclusivos de lectura ("my_read_user"), para verificar que es capaz de leer correctamente el registro recién insertado. El éxito de esta operación confirma que los privilegios asignados a ambos usuarios son correctos, eficaces y están adecuadamente configurados.

Para conectarse a PostgreSQL en remoto desde otro dispositivo se han seguido los siguientes pasos. Primero, en el archivo postgresql.conf, se debe establecer listen_addresses = '*' para que PostgreSQL escuche en todas las interfaces de red. Luego, en el archivo pg_hba.conf, se habilita el acceso remoto añadiendo la línea host all all 0.0.0.0/0 md5, lo que permite conexiones desde cualquier dirección IP usando autenticación por contraseña. Una vez configurado, se ha verificado que el puerto 5432 esté abierto con el comando netstat -tuln | grep 5432. Finalmente, para conectarse desde un cliente externo, se utiliza el comando psql -h [IP_SERVIDOR] -U postgres -d postgres.

Conclusiones

Durante el desarrollo de la práctica, se realizó una comparativa conceptual entre PostgreSQL, IBM DB2 y Oracle Database,que son las más similares, observándose algunas diferencias significativas:

Facilidad de uso y configuración inicial

- **PostgreSQL** ha resultado ser notablemente más sencillo y ágil de configurar y gestionar inicialmente frente a alternativas comerciales como IBM DB2 y Oracle Database.
- El despliegue utilizando Docker y Docker Compose simplifica enormemente la puesta en marcha frente a la configuración tradicional requerida por IBM DB2 y

Oracle, que generalmente implican procesos de instalación más largos y complejos, y configuraciones más detalladas a nivel del sistema operativo.

Administración y gestión de usuarios y roles

- PostgreSQL ofrece una gestión de usuarios y roles más sencilla y transparente. La configuración y asignación de permisos resulta intuitiva y efectiva.
- En contraste, tanto IBM DB2 como Oracle Database suelen diferenciar claramente entre roles y usuarios de forma más rígida, lo cual puede resultar algo más complejo en cuanto a administración inicial. En Oracle, por ejemplo, la gestión de roles y privilegios requiere conocimientos más avanzados sobre esquemas y tablespaces, lo que añade complejidad.

Dificultades encontradas

Durante la realización de esta práctica, prácticamente no se han detectado dificultades relevantes al trabajar con PostgreSQL debido a la claridad en su documentación y la accesibilidad de su comunidad. La gestión de permisos, roles y automatización mediante scripts Bash ha resultado intuitiva y práctica, especialmente gracias al uso de sentencias SQL directas y comandos estándar.

En contraste, experiencias previas con sistemas comerciales como IBM DB2 u Oracle han mostrado que estos suelen ser más exigentes en términos de configuración inicial, administración de roles y licencias, y requieren una curva de aprendizaje inicial considerablemente superior.

Oracle XE

Esta memoria describe detalladamente las actividades realizadas en la práctica de despliegue y gestión de un SGBD Oracle Database Express Edition (XE) utilizando Docker. Se centra especialmente en la configuración básica del entorno, la automatización mediante scripts en Bash para la creación de usuarios y tablas, asignación de permisos y validación de operaciones básicas.

1. Despliegue inicial del SGBD Oracle mediante Docker Compose

La base de datos Oracle XE ha sido desplegada usando Docker Compose con la siguiente configuración:

```
oracle:
```

```
  image: gvenzl/oracle-xe:latest

  ports:

    - "1521:1521"

  environment:

    ORACLE_RANDOM_PASSWORD: "true"

    APP_USER: admin

    APP_USER_PASSWORD: admin

  volumes:

    - oracle-data:/ORCL

  healthcheck:

    test: ["CMD", "healthcheck.sh"]

    interval: 10s

    timeout: 5s

    retries: 10
```

2. Creación inicial del espacio de datos y tablas

Una vez desplegado el entorno Oracle, se ejecutó un script para crear de manera automatizada, idempotente y segura una tabla inicial (`MY_TABLE`) y realizar la inserción inicial de datos de prueba.

```
#!/bin/bash

# Script: docker_oracle_tables.sh

# Objetivo: Ingresar al contenedor p1-oracle-1 y crear (si no existe)
la tabla MY_TABLE

#           para luego poblarla con datos de ejemplo, de forma
idempotente.
```

```
# Se utiliza sqlplus para conectarse a Oracle usando el usuario
APP_USER definido en docker-compose.



CONTAINER_NAME="p1-oracle-1"

# Usamos el usuario de la aplicación definido en docker-compose

ADMIN_USER="admin"

ADMIN_PASSWORD="admin"

# Connect string para Oracle XE (verifica que sea el correcto)

CONNECT_STRING="localhost:1521/XEPDB1"



echo "Ejecutando script dentro del contenedor '$CONTAINER_NAME' para
crear y poblar tablas en Oracle..."



docker exec -it $CONTAINER_NAME bash -c "sqlplus -S
${ADMIN_USER}/${ADMIN_PASSWORD}@${CONNECT_STRING} <<EOF

SET SERVEROUTPUT ON

DECLARE

  table_count NUMBER;

BEGIN

  SELECT COUNT(*) INTO table_count FROM user_tables WHERE table_name =
'MY_TABLE';

  IF table_count = 0 THEN

     DBMS_OUTPUT.PUT_LINE('Creando tabla MY_TABLE...');

     EXECUTE IMMEDIATE 'CREATE TABLE MY_TABLE (

        ID NUMBER GENERATED BY DEFAULT ON NULL AS IDENTITY,

        NAME VARCHAR2(100),

        CREATED_AT TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
```

```
        CONSTRAINT MY_TABLE_PK PRIMARY KEY (ID)

    )';

    DBMS_OUTPUT.PUT_LINE('Tabla MY_TABLE creada.');

  ELSE

    DBMS_OUTPUT.PUT_LINE('La tabla MY_TABLE ya existe.');

  END IF;

END;

/

DECLARE

  cnt NUMBER;

BEGIN

  SELECT COUNT(*) INTO cnt FROM MY_TABLE;

  IF cnt = 0 THEN

    DBMS_OUTPUT.PUT_LINE('Insertando datos de ejemplo en
MY_TABLE...');

    EXECUTE IMMEDIATE 'INSERT INTO MY_TABLE (NAME) VALUES (''Sample
Data'')';

    COMMIT;

    DBMS_OUTPUT.PUT_LINE('Datos insertados.');

  ELSE

    DBMS_OUTPUT.PUT_LINE('MY_TABLE ya contiene datos.');

  END IF;

END;

/

EXIT;

EOF"
```

```
echo "Script docker_oracle_tables.sh completado."
```

Las operaciones realizadas por este script son:

- **Conexión automatizada con Oracle** utilizando `sqlplus` desde dentro del contenedor Docker.
- **Creación idempotente de la tabla MY_TABLE**:
  - Se verifica primero si la tabla existe mediante consultas internas al diccionario de datos (`user_tables`).
  - La tabla se crea únicamente si no existe, con columnas estándar: `ID` (auto-generado), `NAME` y `CREATED_AT`.
- **Inserción idempotente de datos**: Inserta una fila inicial ("Sample Data") sólo si la tabla está vacía, evitando así la duplicación de datos.

3. Creación automatizada de usuarios con distintos permisos

```
#!/bin/bash
# Script: docker_oracle_roles.sh
# Objetivo:
#   - Crear un usuario con permisos de escritura y otro con permisos de solo lectura en Oracle.
#   - Verificar que se puede interactuar con la tabla admin.my_table mediante estos usuarios.
# Se asume que se ejecuta en el contenedor "p1-oracle-1" y que sqlplus está disponible.

CONTAINER_NAME="p1-oracle-1"
# Usamos las credenciales definidas en docker-compose
ADMIN_USER="admin"
ADMIN_PASSWORD="admin"
# Connect string para conectarse a Oracle XE (verifica que sea el correcto)
CONNECT_STRING="localhost:1521/XEPDB1"

# Datos de usuarios (ajustar según políticas de seguridad)
WRITE_USER="MY_WRITE_USER"
WRITE_USER_PASSWORD="StrongWriteUserOracle123!"

READ_USER="MY_READ_USER"
READ_USER_PASSWORD="StrongReadUserOracle123!"
```

```
echo "Ejecutando script en el contenedor '$CONTAINER_NAME' para crear
usuarios de escritura y solo lectura en Oracle (con admin/admin)..."

# Crear usuarios y otorgar privilegios sobre admin.my_table
docker exec -it $CONTAINER_NAME bash -c "sqlplus -S
${ADMIN_USER}/${ADMIN_PASSWORD}@${CONNECT_STRING} <<EOF
SET SERVEROUTPUT ON

-- Crear usuario de escritura si no existe, y otorgar privilegios sobre
admin.my_table
DECLARE
  v_count NUMBER;
BEGIN
  -- Usuario de escritura
  SELECT COUNT(*) INTO v_count FROM all_users WHERE username =
UPPER('$WRITE_USER');
  IF v_count = 0 THEN
     DBMS_OUTPUT.PUT_LINE('Creando usuario de escritura
$WRITE_USER...');
     EXECUTE IMMEDIATE 'CREATE USER $WRITE_USER IDENTIFIED BY
\"$WRITE_USER_PASSWORD\"';
     EXECUTE IMMEDIATE 'GRANT CONNECT, RESOURCE TO $WRITE_USER';
     DBMS_OUTPUT.PUT_LINE('Usuario de escritura $WRITE_USER creado.');
  ELSE
     DBMS_OUTPUT.PUT_LINE('El usuario de escritura $WRITE_USER ya
existe.');
  END IF;

  -- Asignar privilegios a MY_WRITE_USER sobre la tabla admin.my_table
  DBMS_OUTPUT.PUT_LINE('Otorgando privilegios de INSERT, UPDATE, DELETE
a $WRITE_USER sobre admin.my_table...');
  EXECUTE IMMEDIATE 'GRANT INSERT, UPDATE, DELETE ON admin.my_table TO
$WRITE_USER';
END;
/
-- Crear usuario de solo lectura si no existe, y otorgar privilegios
sobre admin.my_table
DECLARE
  v_count NUMBER;
BEGIN
  -- Usuario de solo lectura
```

```
  SELECT COUNT(*) INTO v_count FROM all_users WHERE username =
UPPER('$READ_USER');
  IF v_count = 0 THEN
     DBMS_OUTPUT.PUT_LINE('Creando usuario de solo lectura
$READ_USER...');
     EXECUTE IMMEDIATE 'CREATE USER $READ_USER IDENTIFIED BY
\"$READ_USER_PASSWORD\"';
     EXECUTE IMMEDIATE 'GRANT CONNECT TO $READ_USER';
     DBMS_OUTPUT.PUT_LINE('Usuario de solo lectura $READ_USER
creado.');
  ELSE
     DBMS_OUTPUT.PUT_LINE('El usuario de solo lectura $READ_USER ya
existe.');
  END IF;

  -- Asignar privilegios a MY_READ_USER sobre la tabla admin.my_table
  DBMS_OUTPUT.PUT_LINE('Otorgando privilegios de SELECT a $READ_USER
sobre admin.my_table...');
  EXECUTE IMMEDIATE 'GRANT SELECT ON admin.my_table TO $READ_USER';
END;
/
EXIT;
EOF"

# Bloque de prueba de interacción con la tabla admin.my_table
TEST_VALUE="Test insert by write user"
echo "Insertando registro de prueba con el usuario '$WRITE_USER' en
admin.my_table..."
docker exec -it $CONTAINER_NAME bash -c "sqlplus -S
${WRITE_USER}/${WRITE_USER_PASSWORD}@${CONNECT_STRING} <<EOF
SET SERVEROUTPUT ON
BEGIN
 -- Usamos el esquema explícito admin.my_table
 EXECUTE IMMEDIATE 'INSERT INTO admin.my_table (NAME) VALUES
(''$TEST_VALUE'')';
 COMMIT;
 DBMS_OUTPUT.PUT_LINE('Inserción exitosa por $WRITE_USER en
admin.my_table.');
END;
/
EXIT;
EOF"
```

```
echo "Leyendo registro con el usuario '$READ_USER' desde
admin.my_table..."
docker exec -it $CONTAINER_NAME bash -c "sqlplus -S
${READ_USER}/${READ_USER_PASSWORD}@${CONNECT_STRING} <<EOF
SET SERVEROUTPUT ON
DECLARE
  CURSOR c IS SELECT ID, NAME, CREATED_AT FROM admin.my_table WHERE
NAME = '$TEST_VALUE';
  rec c%ROWTYPE;
BEGIN
  OPEN c;
  LOOP
     FETCH c INTO rec;
     EXIT WHEN c%NOTFOUND;
     DBMS_OUTPUT.PUT_LINE('Fila: ' || rec.ID || ', ' || rec.NAME || ',
' || rec.CREATED_AT);
  END LOOP;
  CLOSE c;
END;
/
EXIT;
EOF"

echo "Prueba de interacción completada en Oracle."
echo "Script docker_oracle_roles.sh finalizado."
```

- **Usuario estándar con permisos de escritura** (`MY_WRITE_USER`):
    - Puede insertar, actualizar y eliminar datos sobre la tabla `admin.my_table`.
    - Se les asignan los roles estándar Oracle (`CONNECT`, `RESOURCE`) necesarios para escritura.
- **Usuario estándar con permisos exclusivamente de lectura** (`MY_READ_USER`):
    - Puede únicamente realizar consultas SELECT sobre la tabla `admin.my_table`.
    - Se crea el usuario y se asignan privilegios adecuados (`CONNECT` y `SELECT`).

Pruebas funcionales automatizadas realizadas por el script:

Este script realiza además operaciones prácticas para verificar automáticamente la asignación correcta de privilegios:

- **Inserción de prueba**: Con el usuario escritor se inserta una fila en la tabla.
- **Consulta de prueba**: El usuario lector intenta leer inmediatamente la fila insertada, comprobando así los permisos asignados.

Estas pruebas garantizan que los usuarios disponen exactamente de los privilegios esperados según su rol.

4. Creación usuario administrador

Debido a que la imagen Docker genera automáticamente la contraseña de SYS/SYSTEM, fue necesario utilizar otro script adicional (`grant_admin.sh`) para extraer esta contraseña del log del contenedor:

```
#!/bin/bash

#

# Script: grant_admin.sh

# Objetivo:

#   1. Buscar la contraseña de SYS y SYSTEM en los logs del contenedor p1-oracle-1.

#   2. Conectarse como SYS usando esa contraseña.

#   3. Otorgar todos los privilegios al usuario admin.



CONTAINER_NAME="p1-oracle-1"

CONNECT_STRING="localhost:1521/XEPDB1"



echo "Obteniendo la contraseña de SYS/SYSTEM desde los logs del contenedor '$CONTAINER_NAME'..."

# Extrae la línea que contiene la contraseña y luego sólo deja la parte de la contraseña

SYS_PASSWORD=$(docker logs "$CONTAINER_NAME" 2>/dev/null \

 | sed -n 's/^ORACLE PASSWORD FOR SYS AND SYSTEM: \(.*\)/\1/p')
```

```
if [ -z "$SYS_PASSWORD" ]; then

 echo "ERROR: No se pudo extraer la contraseña de SYS/SYSTEM. Verifica
los logs manualmente."

 exit 1

fi



echo "Contraseña de SYS/SYSTEM encontrada: $SYS_PASSWORD"



# Otorgar todos los privilegios al usuario admin

echo "Otorgando privilegios a 'admin' usando SYS/$SYS_PASSWORD..."

docker exec -it "$CONTAINER_NAME" bash -c "sqlplus -S
sys/$SYS_PASSWORD@$CONNECT_STRING as sysdba <<EOF

GRANT ALL PRIVILEGES TO admin;

EXIT;

EOF"



echo "Finalizado. Ahora el usuario 'admin' cuenta con todos los
privilegios."
```

- El script obtiene automáticamente la contraseña generada desde los logs del contenedor Docker Oracle.
- Una vez obtenida la contraseña, se conecta como SYSDBA usando `sqlplus` y asigna explícitamente todos los privilegios al usuario administrador (`admin`) que usaremos para operaciones administrativas generales.

Este procedimiento permitió simplificar notablemente la gestión de privilegios en un entorno donde Oracle genera automáticamente las credenciales críticas para SYS y SYSTEM.

Para conectarse en remoto desde otro dispositivo, se han seguido los siguientes pasos. Primero, en el archivo listener.ora, se ha configurado el listener para escuchar en todas las interfaces de red o en una dirección IP específica, asegurando que esté listo para aceptar conexiones externas. Luego, en el archivo tnsnames.ora, se ha definido el servicio de la base de datos con la dirección IP correcta y el puerto predeterminado (1521), lo que permite a los clientes localizar y conectarse a la instancia de Oracle.

Una vez configurado, se ha verificado que el puerto 1521 esté abierto y accesible utilizando el comando netstat -tuln | grep 1521. Esto confirma que el servidor está escuchando en el puerto correcto. Finalmente, para conectarse desde un cliente externo, se ha utilizado la herramienta sqlplus con el comando sqlplus usuario/contraseña@//IP_SERVIDOR:1521/SERVICIO, donde usuario y contraseña son las credenciales de acceso, IP_SERVIDOR es la dirección del servidor, y SERVICIO es el nombre del servicio de la base de datos.

Conclusiones comparativas (Oracle frente a PostgreSQL e IBM DB2)

Facilidad de uso y configuración inicial

Oracle Database, aunque más ágil gracias al uso de Docker frente a métodos tradicionales, resultó claramente más complejo en comparación con PostgreSQL debido a:

- Complejidad en gestión interna (tablespaces, diccionario de datos más extenso).
- Necesidad de gestión adicional de roles y usuarios a nivel del esquema.
- Administración más estricta de privilegios y objetos en comparación con la sencillez proporcionada por PostgreSQL.

En comparación con IBM DB2, Oracle tiene una complejidad similar o ligeramente superior, dado que ambas requieren conocimientos específicos adicionales sobre configuración interna del motor.

Administración de usuarios y roles

Oracle diferencia claramente usuarios y roles, lo cual añade mayor flexibilidad, pero también mayor complejidad administrativa respecto a PostgreSQL, que resulta más intuitivo.

- En Oracle, se requiere especificar explícitamente privilegios y roles estándar (`CONNECT`, `RESOURCE`) al crear usuarios.
- PostgreSQL es más sencillo y directo en este aspecto.

Dificultades encontradas

La principal dificultad en Oracle radicó en la gestión inicial de credenciales automáticas y la comprensión de su modelo de gestión de permisos basado en esquemas, roles y tablespaces, algo que no sucede en PostgreSQL.

Además, la sintaxis PL/SQL utilizada por Oracle añade un grado de complejidad en comparación con SQL estándar que emplea PostgreSQL.

Cassandra

Con respecto a Cassandra, para generar el contenedor con la imagen de dicho SGBD se ha utilizado el siguiente yaml:

```yaml
cassandra:
  image: cassandra:latest
  container_name: cassandra-node
  restart: no
  ports:
    - "9042:9042"
  environment:
    CASSANDRA_CLUSTER_NAME: "MiCluster"
    CASSANDRA_NUM_TOKENS: 256
    CASSANDRA_START_RPC: "true"
  volumes:
    - cassandra-data:/var/lib/cassandra
```

Una vez arrancado el contenedor, a continuación, se va a configurar el SGBD para tener que acceder a Cassandra mediante unas credenciales. Para ello, debemos acceder a la terminal del contenedor de Cassandra:
`docker exec -it cassandra-node bash` (donde cassandra-node es el contenedor de cassandra)

Dentro, se debe modificar un archivo de configuración, en concreto el archivo /etc/cassandra/cassandra.yaml (puede utilizarse cualquier editor de texto. En el caso de prueba se ha instalado nano para la modificación). Dentro, se deben cambiar las siguientes líneas:
`authenthicator: AllowAllAutheticator`
Se ha cambiado por:
`authenticator: PasswordAuthenticator`

`authorizer: AllowAllAuthorizer`
Se ha cambiado por:
`authorizer: CassandraAuthorizer`

Una vez hechos los cambios, se reinicia el contenedor:

```
docker restart cassandra-node
```

Una vez reiniciado, ahora que se ha modificado la configuración, para acceder a Cassandra se debe acceder con unas credenciales concretas. Por defecto, cuando se instala Cassandra, sólo existe un usuario (superusuario) cuyas credenciales son para usuario y contraseña los valores cassandra y cassandra, respectivamente. Para acceder al sistema gestor que está dentro del contenedor, se debe escribir lo siguiente:
`docker exec -it cassandra-node cqlsh -u cassandra -p cassandra` (donde cassandra-node es el contenedor de cassandra)
Una vez hecho, ya se encuentra dentro de Cassandra. Si se desea cambiar la contraseña de un usuario (en este caso solo existe el superusuario) se debe escribir lo siguiente (dentro de Cassandra):

```
ALTER ROLE superadmin WITH PASSWORD = 'NuevaContraseñaSegura!';
```

A continuación, el objetivo es crear la estructura básica del espacio de datos. La estructura de datos dentro de Cassandra es la siguiente:
KEYSPACE -> TABLES -> Filas y Columnas
Para ver todos los Keyspace de Cassandra, se puede utilizar el siguiente comando:

```
DESCRIBE KEYSPACES;
```

Para crear un Keyspace, una vez dentro de Cassandra, se debe escribir lo siguiente:

```
CREATE KEYSPACE mi_keyspace WITH replication = {'class':
'SimpleStrategy', 'replication_factor': 1};
```

Una vez creado el Keyspace, se puede acceder a él de la siguiente manera:

```
USE mi_keyspace
```

Una vez dentro, para ver las tablas que hay dentro de un Keyspace (por defecto no habrá tablas) se puede emplear el siguiente comando:

```
DESCRIBE TABLES
```

A continuación, se va a crear una tabla de prueba para probar el correcto funcionamiento de la estructura. Se va a crear la siguiente tabla:

```
CREATE TABLE usuarios ( id UUID PRIMARY KEY, nombre TEXT, edad INT,
email TEXT );
```

Para probar la tabla, se insertan los siguientes usuarios de prueba:

```
INSERT INTO usuarios (id, nombre, edad, email) VALUES (uuid(),
'Diego Mateo', 20, 'diegomateo56@gmail.com');
INSERT INTO usuarios (id, nombre, edad, email) VALUES (uuid(),
'Carlos Solana', 20, '872815@unizar.es');
```

Una vez insertados, si seleccionamos todas las filas de la tabla usuarios:

```
SELECT * FROM usuarios;
```

Se devuelve la siguiente salida:

```
id                                   | edad | email                 | nombre
--------------------------------------+------+-----------------------+---------------
6d70508f-507a-49ec-a7c9-6dfb19e5db17 |   20 |       872815@unizar.es | Carlos Solana
226ba71e-9609-4085-a63b-3e64a6cff724 |   20 | diegomateo56@gmail.com |   Diego Mateo
```

A continuación, se va a explicar el funcionamiento de los roles. Cassandra proporciona la opción de crear roles que tienen unos permisos concretos. Para el caso de prueba, se van a crear 2 tipos de usuario: lector y escritor. El escritor va a ser capaz de modificar las tablas dentro del Keyspace mi_keyspace, además de poder hacer selecciones (queries). Por otro lado, el lector no será capaz de modificar tablas, pero sí que podrá realizar selecciones.
Para crear un rol, dentro de Cassandra se debe escribir lo siguiente:

```
CREATE ROLE lector WITH PASSWORD = 'lector' AND LOGIN = true;
CREATE ROLE escritor WITH PASSWORD = 'escritor' AND LOGIN = true;
```

Una vez creados los roles, se deben especificar los permisos de cada tipo de usuario:

```
GRANT MODIFY ON KEYSPACE mi_keyspace TO escritor;
GRANT SELECT ON KEYSPACE mi_keyspace TO escritor;
GRANT SELECT ON KEYSPACE mi_keyspace TO lector;
```

Para visualizar todos los roles dentro del SGBD, se puede escribir lo siguiente:

```
SELECT * FROM system_auth.roles;
```

La salida será semejante a esta:

```
 role      | can_login | is_superuser | member_of | salted_hash
-----------+-----------+--------------+-----------+--------------------------------------------------------------
    lector |      True |        False |      null | $2a$10$4E0z5APUGihIv4Xh65fPzOux8vr8rk9assG.kOw5BsZn4g3zQ5rii
 cassandra |      True |         True |      null | $2a$10$pUgRuUnGm3uFPWlP5RVZReuVrGLOwKseb9AcNjuFgbfED1tNTL.x2
  escritor |      True |        False |      null | $2a$10$.PhGExDuFZXPAu8vy9c/o.qKh/hLmxltUlfcGD3GIBlUNv8uwcAc.
```

Una vez cambiados los roles, si prueba a acceder a Cassandra con los 2 roles, probando las inserciones y selecciones, los roles funcionan correctamente.

Por último, se va a configurar el entorno para poder ser accedido de forma remota a través de la red local. Para ello, primero se debe acceder a la IP del equipo. Se puede hacer mediante el siguiente comando:

```
ip a | grep inet
```

Una vez ejecutado, se debe buscar en la salida aquella dirección IP de su máquina local.
Una vez obtenida, se debe modificar el fichero de configuración de Cassandra (/etc/cassandra/cassandra.yaml) y modificar las siguientes líneas:

```
listen_address: 0.0.0.0
broadcast_address: 10.1.160.137          (Poner IP de su equipo)
rpc_address: 0.0.0.0
broadcast_rpc_address: 10.1.160.137      (Poner IP de su equipo)
```

Una vez realizados los cambios, no debe reiniciar el contenedor. Docker cada vez que reinicia un contenedor asigna IPs dinámicas de acceso, por lo que los cambios se pierden.
A continuación, se debe probar que funciona correctamente. Dentro de otro equipo en la misma red local, se debe acceder a Cassandra para probar el funcionamiento del acceso remoto. Para ello, se debe arrancar un contenedor que contenga una imagen de Cassandra y se debe entrar a la terminal. Una vez dentro, ejecute el siguiente comando:

```
cqlsh 10.1.160.137 -u cassandra -p cassandra
```

(Poner IP del equipo remoto, usuario y contraseña del superusuario del equipo al que se quiere conectar)
Una vez ejecutado el comando, se ha accedido correctamente de forma remota al cliente Cassandra del otro equipo, por lo que funciona correctamente el acceso.

Finalmente, se ha realizado el siguiente script para automatizar la prueba de funcionamiento de Cassandra:

```
#!/bin/bash

# 📌 IMPORTANTE: Configuración previa en el archivo de Cassandra
# Para habilitar autenticación y autorización en Cassandra, asegúrate
de cambiar las siguientes líneas
# en el archivo de configuración `/etc/cassandra/cassandra.yaml`:
#
# authenticator: AllowAllAuthenticator  →  Se ha cambiado por:
authenticator: PasswordAuthenticator
# authorizer: AllowAllAuthorizer        →  Se ha cambiado por:
authorizer: CassandraAuthorizer
#
# Después de realizar estos cambios, reinicia Cassandra para que los
ajustes tengan efecto.

# Nombre del contenedor de Cassandra
CONTAINER_NAME="cassandra-node"

echo "🔹 Ejecutando comandos en Cassandra..."

# Ejecutar todos los comandos en un solo bloque para mantener el
contexto del keyspace
docker exec -i $CONTAINER_NAME cqlsh -u cassandra -p cassandra <<EOF
-- Crear el KEYSPACE si no existe
CREATE KEYSPACE IF NOT EXISTS mi_keyspace
WITH replication = {'class': 'SimpleStrategy', 'replication_factor':
1};

USE mi_keyspace;

-- Crear la tabla si no existe
CREATE TABLE IF NOT EXISTS usuarios (
   id INT PRIMARY KEY,
   nombre TEXT,
   edad INT
);

-- Insertar datos de ejemplo
INSERT INTO usuarios (id, nombre, edad) VALUES (1, 'Alice', 25);
INSERT INTO usuarios (id, nombre, edad) VALUES (2, 'Bob', 30);
```

```
INSERT INTO usuarios (id, nombre, edad) VALUES (3, 'Charlie', 22);
INSERT INTO usuarios (id, nombre, edad) VALUES (4, 'David', 27);

-- Verificar si los roles existen antes de crearlos
-- Si el rol 'lector' no existe, lo creamos
CREATE ROLE IF NOT EXISTS lector WITH PASSWORD = 'lector' AND LOGIN =
true;
CREATE ROLE IF NOT EXISTS escritor WITH PASSWORD = 'escritor' AND LOGIN
= true;

-- Asignar permisos solo si los roles existen
GRANT CREATE ON KEYSPACE mi_keyspace TO escritor;
GRANT MODIFY ON KEYSPACE mi_keyspace TO escritor;
GRANT SELECT ON KEYSPACE mi_keyspace TO escritor;
GRANT SELECT ON KEYSPACE mi_keyspace TO lector;

-- Mostrar los datos para verificar que todo esté correcto
SELECT * FROM usuarios;
EOF

echo "✅ Configuración inicial completada."

echo "🔹 Probando acceso con el rol 'lector'..."
docker exec -i $CONTAINER_NAME cqlsh -u lector -p lector <<EOF
USE mi_keyspace;

-- Intentar insertar datos (debería fallar)
INSERT INTO usuarios (id, nombre, edad) VALUES (5, 'Eva', 29);

-- Intentar leer los datos (debería funcionar)
SELECT * FROM usuarios;
EOF

echo "🔹 Probando acceso con el rol 'escritor'..."
docker exec -i $CONTAINER_NAME cqlsh -u escritor -p escritor <<EOF
USE mi_keyspace;

-- Intentar crear una nueva tabla (debería funcionar)
CREATE TABLE IF NOT EXISTS productos (
   id INT PRIMARY KEY,
   nombre TEXT,
   precio DECIMAL
);
```

```
-- Intentar leer los datos de usuarios (debería funcionar)
SELECT * FROM usuarios;
EOF

echo "✅ Pruebas finalizadas."
```

Conclusiones:

Durante el desarrollo de la práctica, se realizó una comparativa conceptual entre **Apache Cassandra**, **PostgreSQL**, **IBM DB2** y **Oracle Database**, observándose algunas diferencias significativas:

Facilidad de uso y configuración inicial

Cassandra ha demostrado ser relativamente sencillo de desplegar utilizando **Docker y Docker Compose**, lo que simplifica la puesta en marcha sin necesidad de configuraciones complejas a nivel del sistema operativo.

Sin embargo, en comparación con bases de datos relacionales como **PostgreSQL**, que también es fácil de configurar, Cassandra requiere modificaciones específicas en su archivo de configuración (`cassandra.yaml`) para habilitar autenticación, autorización y acceso remoto.

Cassandra permite una gestión flexible de usuarios y roles mediante su sistema de autenticación basado en **PasswordAuthenticator** y **CassandraAuthorizer.** En comparación, PostgreSQL ofrece una administración de usuarios más intuitiva, con una diferenciación clara entre roles y permisos.

Dificultades encontradas
Por lo general, no se han encontrado dificultades para la configuración de todo el sistema más allá de saber qué elementos modificar del archivo de configuración. La única dificultad que ha requerido un tiempo para comprender el funcionamiento y hacer que todo funcionara fue la parte del acceso remoto, ya que ha costado saber qué direcciones había que modificar para hacer que el acceso remoto funcione. Además, cuando se terminaba la configuración, al reiniciar el contenedor, se volvían a modificar automáticamente las direcciones IP que se habían modificado anteriormente, cosa que complicó en cierta medida la tarea. Finalmente, se concluyó en que lo más sencillo es no reiniciar el contenedor una vez realizados los cambios para poder probar correctamente el acceso remoto.

Apache Hbase

Para desplegar HBase en Docker, el primer paso fue descargar la imagen oficial utilizando el comando `docker pull dajobe/hbase`. Esto permitió obtener la imagen desde Docker Hub y almacenarla localmente para usarla en la ejecución de un contenedor.

Una vez descargada la imagen, a diferencia de Cassandra que se usó un yml, para hbase se procedió a crear y ejecutar un contenedor directamente basado en ella. Para ello, se utilizó el comando `docker run -d --name hbase_db -p 16010:16010 -p 2181:2181 dajobe/hbase`. Este comando permite iniciar el contenedor en segundo plano, asignarle el nombre hbase_db y mapear los puertos necesarios para su funcionamiento. El puerto 16010 se utilizó para la interfaz web de administración de HBase y el puerto 2181 para el servicio de Zookeeper, el cual es requerido para la coordinación interna de HBase.

Después, se comprobó que este estuviera corriendo correctamente mediante el comando `docker ps` y se observó que aparecía en la lista

Para interactuar con HBase y verificar su correcto funcionamiento, se usó la interfaz web a través de la dirección http://localhost:16010 en un navegador. Desde esta interfaz, se pudo comprobar la presencia del Master y de los Region Servers activos, además de gestionar la creación de tablas y otros parámetros del sistema.

También se realizó una conexión a la shell de HBase dentro del contenedor. Para ello, primero se accedió al contenedor con `docker exec -it hbase_db bash` y, una vez dentro, se inició la shell de HBase ejecutando el comando `hbase shell`. Desde esta interfaz de línea de comandos, se verificó la configuración del sistema con el comando status, que mostró información sobre el estado del Master y los Region Servers. Después, se creó una tabla de prueba con `create 'test_table', 'cf1'`, donde test_table representa el nombre de la tabla y cf1 es la Column Family utilizada para almacenar los datos. Para confirmar que la tabla fue creada exitosamente, se ejecutó el comando `list`, que mostró todas las tablas existentes en la base de datos.

Después de la creación de la tabla, se probó la inserción de datos de prueba. Utilizando la shell de HBase, se ejecutaron los siguientes comandos:

```
put 'test_table', 'row1', 'cf1:name', 'Alice'
put 'test_table', 'row2', 'cf1:name', 'Bob'
put 'test_table', 'row1', 'cf1:age', '30'
put 'test_table', 'row2', 'cf1:age', '25'
```

Estos comandos almacenan datos en la tabla test_table, donde cada fila (row1 y row2) representa un registro con las columnas cf1:name y cf1:age, que contienen los valores correspondientes a nombre y edad.

Para verificar que los datos se insertaron correctamente, se utilizó el comando scan `'test_table'`, que mostró todas las filas y columnas almacenadas en la tabla. Además, para consultar un registro específico, se ejecutó `get 'test_table', 'row1'`, obteniendo así toda la información almacenada en la fila row1.

Si en algún momento se quiere eliminar un dato específico, se puede utilizar los comandos `delete 'test_table', 'row1', 'cf1:name'` para borrar una columna dentro de una fila, o `deleteall 'test_table', 'row1'` para eliminar completamente una fila.

Finalmente, si se desea eliminar la tabla completa, primero se deshabilita con `disable 'test_table'` y luego se elimina con `drop 'test_table'`.

A continuación, se intentó configurar HBase para la creación de usuarios y roles utilizando tanto la autenticación Kerberos como la autenticación simple, pero en ambos casos surgieron errores que no se pudieron resolver.

Con la autenticación simple, se modificaron correctamente los parámetros en el archivo de configuración de HBase para habilitar la seguridad, pero al intentar usar el comando `grant` para asignar permisos, se recibió un error que indicaba que las características de seguridad no estaban disponibles. A pesar de reiniciar HBase y revisar los logs, el problema no fue solucionado.

Por otro lado, con Kerberos también se intentaron las configuraciones necesarias, pero los errores relacionados con la integración de Kerberos no permitieron continuar con la creación de roles y usuarios, fue posible pedirle tickets al sistema kerberos pero se consiguió integrarlo en el HBase.

Finalmente se modificó el fichero de configuración de HBase para que pudiera ser accesible desde otro dispositivo en la misma red local, y al poner esa dirección en el buscador se accedió a la interfaz gráfica de HBase en la que se podían observar las tablas creadas por ejemplo de igual manera que en el dispositivo que se encuentra el contenedor de HBase.

`hbase.master.info.bindAddress`: Establecido en `0.0.0.0` para permitir el acceso a la interfaz web desde cualquier dispositivo en la red.

`hbase.regionserver.bindAddress`: Establecido en `0.0.0.0` para permitir conexiones desde otros dispositivos para acceder a las regiones de HBase.

`hbase.master.rpc.bindAddress`: Establecido en `0.0.0.0` para aceptar peticiones RPC desde otros equipos.

Se han automatizado las tareas realizadas con este script “hbase.sh”

```
#!/bin/bash



CONTAINER_NAME="hbase_db"



# Verificar si el contenedor ya existe

if [ "$(docker ps -aq -f name=^${CONTAINER_NAME}$)" ]; then

   echo "El contenedor $CONTAINER_NAME ya existe."
```

```
    # Si está detenido, lo iniciamos
    if [ "$(docker ps -q -f name=^${CONTAINER_NAME}$)" ]; then
        echo "El contenedor ya está corriendo."
    else
        echo "Iniciando el contenedor $CONTAINER_NAME..."
        docker start $CONTAINER_NAME
        sleep 5  # Esperar un poco para que HBase arranque
    fi
else
    echo "Creando y ejecutando el contenedor $CONTAINER_NAME..."
    docker run -d --name $CONTAINER_NAME harisekhon/hbase
    echo "Esperando a que HBase se inicie..."
    sleep 10
fi
# Ejecutar comandos en la shell de HBase dentro del contenedor
echo "Ejecutando comandos en la shell de HBase..."
docker exec -it hbase_db bash -c "
 echo \"Creando tabla test_table...\";
 echo \"create 'test_table', 'cf1'\" | hbase shell 2>/dev/null;
 echo \"Insertando datos en test_table...\";
 echo \"put 'test_table', 'row1', 'cf1:name', 'Alice'\" | hbase shell
2>/dev/null;
 echo \"put 'test_table', 'row2', 'cf1:name', 'Bob'\" | hbase shell
2>/dev/null;
 echo \"put 'test_table', 'row1', 'cf1:age', '30'\" | hbase shell
2>/dev/null;
```

```
  echo \"put 'test_table', 'row2', 'cf1:age', '25'\" | hbase shell
2>/dev/null;

 echo \"Verificando los datos...\";

 echo \"scan 'test_table'\" | hbase shell 2>/dev/null;

 echo \"Consulta de una fila específica...\";

 echo \"get 'test_table', 'row1'\" | hbase shell 2>/dev/null;

 echo \"Eliminando datos de prueba...\";

  echo \"delete 'test_table', 'row1', 'cf1:name'\" | hbase shell
2>/dev/null;

 echo \"deleteall 'test_table', 'row2'\" | hbase shell 2>/dev/null;

 echo \"Deshabilitando y eliminando la tabla...\";

 echo \"disable 'test_table'\" | hbase shell 2>/dev/null;

 echo \"drop 'test_table'\" | hbase shell 2>/dev/null;

"

echo "Proceso completado."
```

Conclusiones

Durante el desarrollo de esta práctica, se realizó una comparativa entre HBase y otras bases de datos NoSQL y SQL, observándose algunas diferencias significativas:

Facilidad de uso y configuración inicial

HBase requiere una configuración inicial más compleja en comparación con bases de datos relacionales como PostgreSQL. Su despliegue es más complejo, no obstante, el uso de Docker y Docker Compose simplifica el proceso, aunque sigue siendo más exigente en comparación con bases de datos tradicionales.

Administración y gestión de usuarios y roles

A diferencia de bases de datos como PostgreSQL, que ofrecen un sistema de gestión de usuarios y permisos intuitivos, HBase carece de una solución sencilla para control de acceso. La implementación de permisos requiere habilitar **coprocessors** como `AccessController` y, en muchos casos, configurar **Kerberos** para autenticación, lo que aumenta la complejidad de administración en comparación con sistemas que ofrecen

mecanismos de seguridad más directos y accesibles, aunque en este caso como se ha explicado se ha sido incapaz de implementar usuarios debido a complicaciones constantes.

IBM DB2

1. Despliegue del SGBD IBM Db2 mediante Docker Compose

El sistema IBM Db2 fue desplegado usando Docker Compose con la siguiente configuración:

```
db2:
  image: ibmcom/db2
  container_name: db2-server
  privileged: true
  environment:
    DB2INST1_PASSWORD: admin
    DBNAME: testdb
    LICENSE: accept
    BLU: "true"
```

```
    ENABLE_ORACLE_COMPATIBILITY: "false"
    UPDATEAVAIL: "NO"
    TO_CREATE_SAMPLEDB: "false"
  ports:
    - "50000:50000"
  volumes:
    - db2-data:/database
```

2. Poblado

Para validar el entorno Db2 desplegado, se implementó un script automatizado llamado `docker_db2_tables.sh`, que realiza las siguientes tareas:

- **Conexión automática:** Utilizando el comando `db2` dentro del contenedor con el usuario `db2inst1`.
- **Creación de tabla (`MY_TABLE`):**
  - Verifica primero si la tabla existe mediante consulta al diccionario del sistema `syscat.tables`.
  - Crea la tabla únicamente si esta no existe previamente, con campos adecuados (`ID`, `NAME`, `CREATED_AT`), donde `ID` es auto-generado.
- **Inserción de datos:**
  - Inserta una fila inicial (`'Sample Data'`) solo si la tabla está vacía.

Este script facilitó considerablemente las pruebas iniciales, simplificando el proceso de validación del correcto funcionamiento del entorno Db2.

3. Usuarios y roles con diferentes privilegios

- **Usuarios creados:**
  - `my_write_user`: con privilegios para insertar, actualizar y eliminar datos.
  - `my_read_user`: con privilegios únicamente de consulta (lectura).
- **Operaciones del script:**
  - **Creación de usuarios a nivel de sistema operativo:**
    Db2 gestiona usuarios en el sistema operativo, por lo que este script los crea dentro del contenedor si no existen previamente, asignándoles contraseñas seguras.

    - **Asignación de privilegios en Db2:**
      Utilizando el usuario administrador (`db2inst1`), se asignaron explícitamente permisos específicos para:

        - Conectarse a la base de datos (`CONNECT ON DATABASE`).
        - Operar sobre la tabla creada previamente (`MY_TABLE`), otorgando permisos adecuados a cada usuario (escritura o lectura según corresponda).
- **Bloque de pruebas automatizadas:**

    - Se realizó automáticamente la prueba de inserción usando el usuario escritor (`my_write_user`).
    - Seguidamente, se validó la correcta lectura por parte del usuario lector (`my_read_user`).

4. Ejemplo práctico de conexión remota y local con IBM Db2

Durante la práctica también se llevaron a cabo ejemplos manuales prácticos de cómo conectar a la base de datos Db2 desplegada en Docker, incluyendo los siguientes pasos básicos realizados desde la terminal:

- Conexión al contenedor mediante Docker.
- Autenticación con el usuario de instancia `db2inst1`.
- Catalogación manual de una base de datos remota mediante el comando `db2 catalog tcpip node` y posteriormente `db2 catalog db`.
- Ejecución de consultas SQL desde la terminal para validar visualmente la conexión efectiva y la operatividad del entorno.

```
solana@solanad-os:~/Documents/inginf/bases2/p1$ sudo docker exec -it
59b5f64fba9d bash

[root@59b5f64fba9d /]# su - db2inst1

Last login: Fri Feb 21 19:00:03 UTC 2025

[db2inst1@59b5f64fba9d ~]$ db2 catalog tcpip node nodoDB2 remote
192.168.61.158 server 50000

DB20000I  The CATALOG TCPIP NODE command completed successfully.

DB21056W  Directory changes may not be effective until the directory
cache is
```

```
refreshed.

[db2inst1@59b5f64fba9d ~]$ db2 connect to remotedb user db2inst1 using
admin

SQL1013N  The database alias name or database name "REMOTEDB" could not
be

found.  SQLSTATE=42705

[db2inst1@59b5f64fba9d ~]$ db2 catalog db testdb as remotedb at node
nodoDB2

DB20000I  The CATALOG DATABASE command completed successfully.

DB21056W  Directory changes may not be effective until the directory
cache is

refreshed.

[db2inst1@59b5f64fba9d ~]$ db2 connect to remotedb user db2inst1 using
admin


  Database Connection Information


Database server        = DB2/LINUXX8664 11.5.8.0

SQL authorization ID   = DB2INST1

Local database alias   = REMOTEDB


[db2inst1@59b5f64fba9d ~]$ db2 "SELECT * FROM DB2INST1.EMPLEADOS"


ID          NOMBRE
SALARIO

-----------
------------------------------------------------------------------------
---------------------------- ------------
```

```
         1 Juan Pérez
50000.00

         2 Ana Gómez
60000.00

         3 Carlos Martínez
45000.00

         4 Laura Ruiz
55000.00



 4 record(s) selected.



[db2inst1@59b5f64fba9d ~]$
```

5. Conclusiones comparativas

Durante la práctica se han identificado ciertas diferencias significativas entre IBM Db2 y los otros gestores evaluados (PostgreSQL y Oracle):

Facilidad de uso y configuración inicial

- IBM Db2 resultó ser más complejo que PostgreSQL debido a que requiere gestionar usuarios directamente en el sistema operativo y manejar rutas específicas al software y perfiles.

Administración de usuarios y permisos

- En IBM Db2, la gestión de usuarios se realiza sobre usuarios del sistema operativo, lo que implica mayor complejidad frente a PostgreSQL y Oracle, que administran usuarios de manera interna y más intuitiva.
- PostgreSQL proporciona una interfaz mucho más simple para la gestión de usuarios y roles, mientras que Oracle permite mayor control con una complejidad ligeramente superior a Db2.

Dificultades encontradas

- La principal dificultad en Db2 fue gestionar adecuadamente las rutas internas de Db2 en Docker, así como la creación necesaria de usuarios a nivel de sistema operativo, lo que añadió cierta complejidad adicional.

.

Comentarios acerca de las licencias

Las licencias de los sistemas gestores de bases de datos (SGBD) determinan sus condiciones de uso, distribución y modificación. En esta práctica se han utilizado distintos SGBD con licencias que varían entre software libre, freemium y comercial.

PostgreSQL se distribuye bajo la PostgreSQL License, una licencia permisiva que permite su uso, modificación y distribución sin restricciones, lo que lo hace completamente libre para cualquier entorno.

Oracle Database Express Edition (Oracle XE) es gratuito pero con limitaciones en CPU, memoria y almacenamiento. Se puede usar de manera gratis y pruebas como se ha usado en esta práctica pero para utilizarlo de manera profesional será necesario una licencia comercial.

Apache Cassandra y Apache HBase están licenciados bajo la Apache License 2.0, lo que permite su uso, modificación y distribución libremente en cualquier entorno, sin restricciones comerciales.

IBM DB2 Community Edition es una versión gratuita con restricciones en el uso en producción. Tiene limitaciones pero se puede adquirir una licencia profesional.

Esfuerzos invertidos

Nº de horas invertidas en la práctica

	Diego	Carlos	Daniel	Total
Práctica 1	6h	8h	8h	