INFORME DE GUÍA PRÁCTICA

I. PORTADA

Tema:	Taller Docker
Unidad de Organización Curricular:	PROFESIONAL
Nivel y Paralelo:	5to A
Alumnos participantes:	Ases Tibán Jeremy Damián Palate Moreta Kevin Damián Poveda Gómez William Alberto Pullupaxi Chango Daniel
Asignatura:	Gestión de Bases de Datos Distribuidas
Docente:	Ing. Rubén Caiza, Mg.

II. INFORME DE GUÍA PRÁCTICA

2.1 Objetivos

General:
Implementar un contenedor de base de datos PostgreSQL utilizando Docker Compose para facilitar la gestión, despliegue y persistencia de datos en entornos de desarrollo.

Específicos:

- Configurar un servicio de base de datos PostgreSQL en un contenedor Docker con credenciales y base de datos inicial.
- Establecer un volumen persistente para conservar los datos incluso después de detener o eliminar el contenedor.
- Definir una red interna para permitir la comunicación segura entre servicios dentro del entorno Docker.

2.2 Modalidad

Presencial

2.3 Tiempo de duración

Presenciales: 3
No presenciales: 0

2.4 Instrucciones

1.- Crear un archivo denominado docker-compose.yml que contendrá la configuración del servicio PostgreSQL, especificando la imagen, el contenedor, las variables de entorno, los puertos, el volumen y la red.

2.- Abrir una terminal en la carpeta donde se encuentra el archivo y ejecutar el comando docker-compose up -d para iniciar el servicio en segundo plano.

3.- Verificar que el contenedor se haya iniciado correctamente utilizando el comando docker ps.

4.- Conectarse a la base de datos desde una herramienta como pgAdmin o DBeaver, o mediante la terminal, empleando las credenciales y parámetros definidos en el archivo de configuración.

5.- Para detener el servicio, ejecutar el comando docker-compose down. Si se desea eliminar completamente los datos almacenados, se debe utilizar docker-compose down -v.

2.5 Listado de equipos, materiales y recursos

Listado de equipos y materiales generales empleados en la guía práctica:

- **Computadora Portatil**

TAC (Tecnologías para el Aprendizaje y Conocimiento) empleados en la guía práctica:

☐Plataformas educativas
☒Simuladores y laboratorios virtuales
☐Aplicaciones educativas
☐Recursos audiovisuales
☐Gamificación
☐Inteligencia Artificial
Otros (Especifique): _____

2.6 Actividades desarrolladas

1. Preparación

Se crea el archivo Docker-compose.yml para estructurar la aplicación, permitiendo definir, configurar y ejecutar toda nuestra arquitectura multi-contenedor mediante un único archivo docker-compose.yml

```
kevin@Khevin:~/practica_mongo_g3$ cat docker-compose.yml
version: "3.9"
services:
  mongo1:
    image: mongo:7.0
    container_name: mongo1
    ports: ["27017:27017"]
    command: ["mongod", "--replSet=rs0", "--bind_ip_all"]
    volumes: ["mongo1:/data/db"]
  mongo2:
    image: mongo:7.0
    container_name: mongo2
    command: ["mongod", "--replSet=rs0", "--bind_ip_all"]
    volumes: ["mongo2:/data/db"]
  mongo3:
    image: mongo:7.0
    container_name: mongo3
    command: ["mongod", "--replSet=rs0", "--bind_ip_all"]
    volumes: ["mongo3:/data/db"]
volumes:
  mongo1:
  mongo2:
  mongo3:
kevin@Khevin:~/practica_mongo_g3$
```

Fig 1. Contenido del archivo .yml

2. Levantar los contenedores

Se usa el comando docker-compose up -d para iniciar toda nuestra aplicación multi-servicio, tal como está definida en el archivo de configuración.

docker-compose up -d

```
kevin@Khevin:~/practica_mongo_g3$ docker-compose up -d
Creating mongo2 ... done
Creating mongo1 ... done
Creating mongo3 ... done
kevin@Khevin:~/practica_mongo_g3$
```

Fig 2. Contenedor levantado

3. Iniciar el set de replicas

El set de réplicas se inicia para crear un clúster de alta disponibilidad en la base de datos de MongoDB. Esto garantiza que la aplicación no sufra interrupciones y que exista disponibilidad de datos en caso de fallo de uno de ellos.

Fig 3. Inicialización de las replicas

4. Verificar el estado de réplicas

Se usa el comando docker exec -it mongo1 mongosh --eval 'rs.status()' para inspeccionar el estado operativo de las réplicas desde el contenedor primario (mongo1).

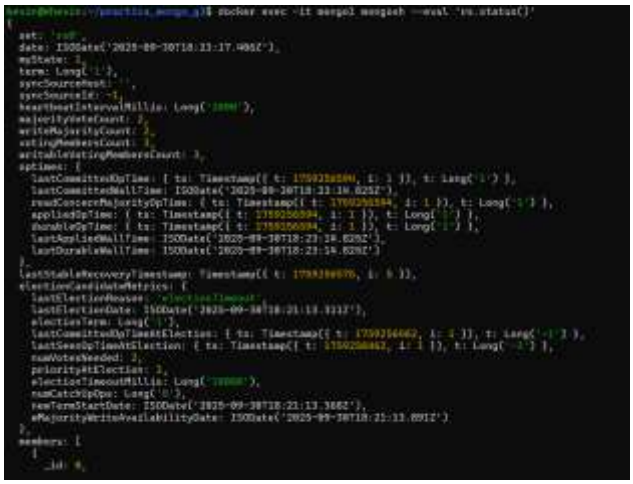

Fig 4. Inspección del estado de las réplicas

5. Importar los datos

Primero se crear los archivos que contendrán los datos en formato JSON.

Fig 5. Listado de archivos .json

Luego cargamos los archivos '.json' en la base de datos

docker exec -i mongo1 mongoimport --db escuela --collection departamentos --file /dev/stdin < departamentos.json

docker exec -i mongo1 mongoimport --db escuela --collection empleados --file /dev/stdin < empleados.json

docker exec -i mongo1 mongoimport --db escuela --collection ventas --file /dev/stdin < ventas.json

Fig 6. Carga de archivos .json

6. Verificación

Se utiliza los comandos para verificar la réplica de datos y probar la alta disponibilidad:

docker exec -it mongo2 mongosh --eval 'db.getMongo().setReadPref("secondary"); db.getSiblingDB("escuela").departamentos.find().pretty()'

docker exec -it mongo2 mongosh --eval 'db.getMongo().setReadPref("secondary"); db.getSiblingDB("escuela").empleados.find().pretty()'

docker exec -it mongo2 mongosh --eval 'db.getMongo().setReadPref("secondary"); db.getSiblingDB("escuela").ventas.find().pretty()'

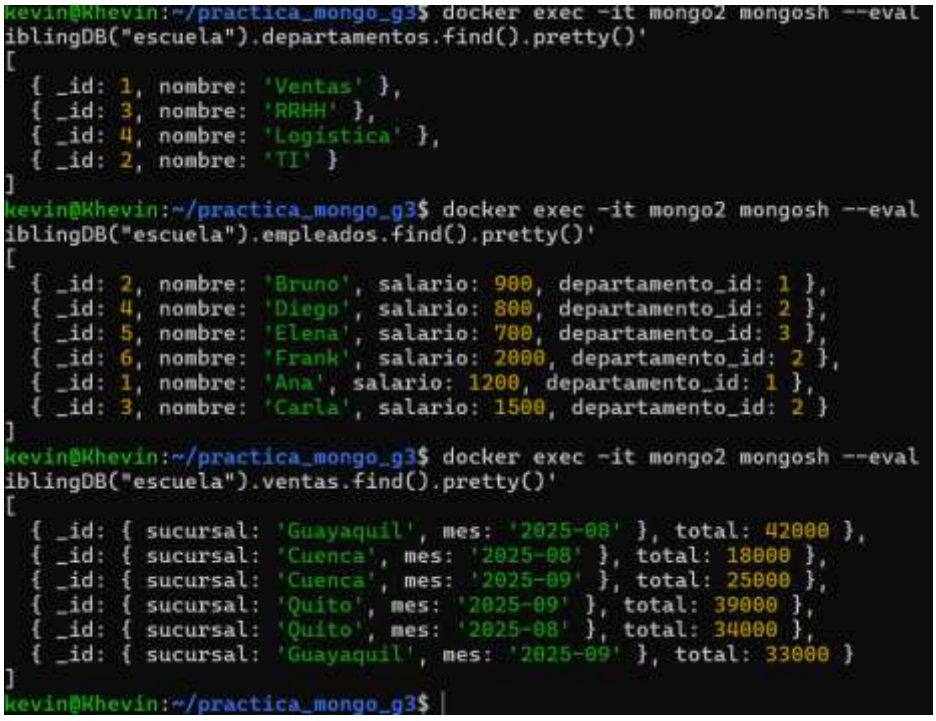

Fig 7. Verificacion de repica de datos

Parte C – Consultas

Primero entrar a mongosh

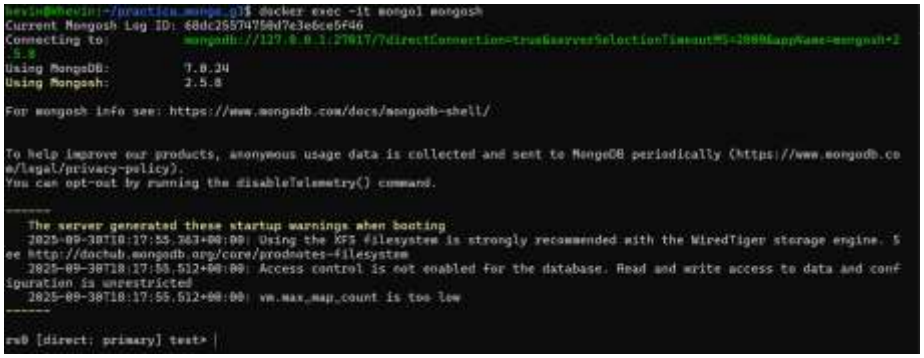

Fig 8. Salida del comando Docker exec it mongo1 mongosh

Luego hacemos uso de la Base de Datos:

Fig 9. Cambio de contexto

1. **Empleados con salario mayor al promedio de la empresa**
   - **Método 1 (ventanas, MongoDB ≥5):** usar $setWindowFields para calcular promEmpresa y luego $match.

```
db.empleados.aggregate([
  // Calculamos el promedio de salario de toda la empresa usando setWindowFields
  {
    $setWindowFields: {
      output: {
        promEmpresa: { $avg: "$salario", window: { documents: ["unbounded", "unbounded"] } }
      }
    }
  },
  // Filtramos los empleados cuyo salario es mayor al promedio
  { $match: { $expr: { $gt: ["$salario", "$promEmpresa"] } } }
])
```

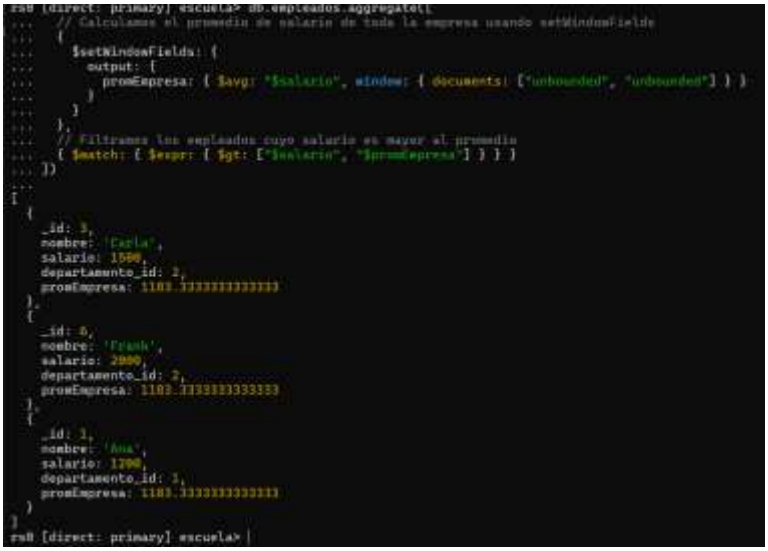

Fig 10. Consulta 1

- **Método 2 (sin ventanas):** calcular promedio con un pipeline y usar ese valor con $expr.

```
// Calculamos el promedio primero
var promedio = db.empleados.aggregate([
  { $group: { _id: null, prom: { $avg: "$salario" } } }
]).toArray()[0].prom;

db.empleados.aggregate([
  { $match: { $expr: { $gt: ["$salario", promedio] } } }
])
```

Fig 11. Consulta 2

2. **Departamentos sin empleados asignados**
   - Usar $lookup de departamentos → empleados y filtrar con $match donde el arreglo resultante tenga tamaño 0.
   - Pista: $lookup + $addFields: {count: {$size: "$empleados"}} + $match: {count: 0}.

```
db.departamentos.aggregate([
  {
    $lookup: {
```

```
    from: "empleados",
    localField: "_id",
    foreignField: "departamento_id",
    as: "empleados"
   }
  },
  {
   $addFields: { count: { $size: "$empleados" } }
  },
  { $match: { count: 0 } },
  { $project: { empleados: 0, count: 0 } } // opcional: solo mostrar datos de
departamento
])
```

Fig 12. Consulta 3

3. **Empleado con salario más alto**
   - Opción A: $sort descendente y $limit: 1.

```
db.empleados.aggregate([
  { $sort: { salario: -1 } },
  { $limit: 1 }
])
```

Fig 13. Consulta 3

   - Opción B: $group + $topN (si está disponible) o $max y luego $match.

```
db.empleados.aggregate([
  {
   $group: {
    _id: null,
    maxSalario: { $max: "$salario" }
   }
  },
  {
   $lookup: {
```

```
      from: "empleados",
      let: { maxSal: "$maxSalario" },
      pipeline: [
        { $match: { $expr: { $eq: ["$salario", "$$maxSal"] } } }
      ],
      as: "empleadoMax"
    }
  },
  { $unwind: "$empleadoMax" },
  { $replaceRoot: { newRoot: "$empleadoMax" } }
])
```

Fig 14. Consulta 4

4. **Para cada empleado, mostrar el salario promedio de su departamento**
   - **Opción con ventanas:** $setWindowFields con partitionBy: "$departamento_id" y {$avg: "$salario"}.

```
db.empleados.aggregate([
  {
   $setWindowFields: {
    partitionBy: "$departamento_id",
    output: {
     promDep: { $avg: "$salario" }
    }
   }
  }
])
```

Fig 15. Consulta 5

- **Opción sin ventanas:** $lookup hacia empleados con let: {dep:"$departamento_id"} y un pipeline que haga $match y $group para obtener el promedio; luego proyectar.

```
db.empleados.aggregate([
  {
   $lookup: {
     from: "empleados",
     let: { depId: "$departamento_id" },
     pipeline: [
       { $match: { $expr: { $eq: ["$departamento_id", "$$depId"] } } },
       { $group: { _id: null, promDep: { $avg: "$salario" } } }
     ],
     as: "promedioDep"
   }
  },
  { $unwind: "$promedioDep" },
  { $addFields: { promDep: "$promedioDep.promDep" } },
  { $project: { promedioDep: 0 } }
])
```

Fig 16. Consulta 6

5. **Departamentos cuyo promedio salarial es mayor al promedio general**
   - Pipeline en dos niveles:
     a) $group por departamento_id para obtener promDep.
     b) Comparar contra promedio global (ventanas o $lookup/doble pipeline).
   - Entregar ambas soluciones si es posible.

```
// Promedio general
var promedioGeneral = db.empleados.aggregate([
  { $group: { _id: null, prom: { $avg: "$salario" } } }
]).toArray()[0].prom;

db.departamentos.aggregate([
  {
   $lookup: {
     from: "empleados",
     localField: "_id",
     foreignField: "departamento_id",
     as: "empleados"
   }
  },
  { $addFields: { promDep: { $avg: "$empleados.salario" } } },
  { $match: { promDep: { $gt: promedioGeneral } } },
  { $project: { empleados: 0 } }
])
```

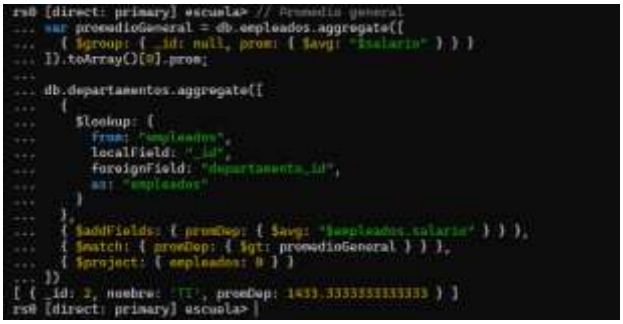

Fig 17. Consulta 7

6. **Ventas: sucursal "top" por mes**
   - A partir de ventas, obtener por **cada mes** la **sucursal con mayor total**.
   - Sugerencia: ordenar por total desc, agrupar por mes y tomar el primero, o usar $group + $topN.

```
db.ventas.aggregate([
  { $sort: { "_id.mes": 1, "total": -1 } },
  {
    $group: {
      _id: "$_id.mes",
      topSucursal: { $first: "$_id.sucursal" },
      total: { $first: "$total" }
    }
  }
])
```

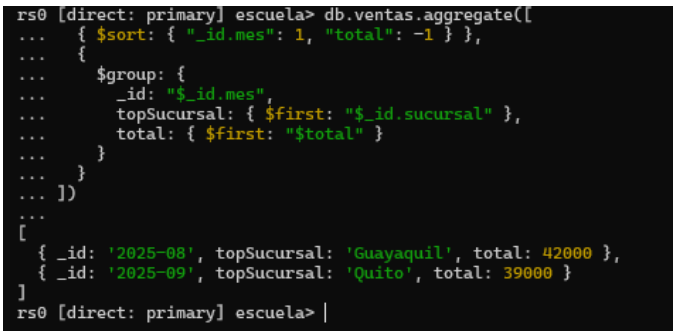

Fig 18. Consulta 8

Parte D – Resiliencia

Verificación de nodos Levantados en otra terminal

Se utiliza el comando docker ps -a para listar todos los contenedores Docker del sistema ya sea que estén en ejecución o no.

Fig 19. Salida del comando Docker ps -a

1. Con el rs0 funcionando, **apaga un nodo**:

   Se utiliza el comando docker stop mongo3 para detener el contenedor mongo3.

Fig 20. Detención del contenedor 3

2. Ejecuta una consulta del punto C y **explica qué ocurre** con el primario/secundario.

Se ejecuta la primera consulta de **Empleados con salario mayor al promedio de la empresa por el Método 1 (ventanas, MongoDB ≥5).**

Fig 21. Ejecución de la consulta 1 nuevamente

Explicación:

- **PRIMARY (mongo1)**: funciona normalmente y atiende la consulta sin problemas, porque todas las escrituras y agregaciones complejas se manejan en el primario.
- **SECONDARY (mongo2)**: sigue replicando desde el primario, porque todavía hay dos nodos operativos: uno **Primario** y el **Secundario.**
- **SECONDARY apagado (mongo3)**: no participa en la replicación mientras está apagado, pero al encenderlo se pone al día con los cambios pendientes desde el PRIMARY automáticamente.

3. Vuelve a **levantar el nodo**:

Se vuelve a levanta el contenedor mediante el comando docker start mongo3.

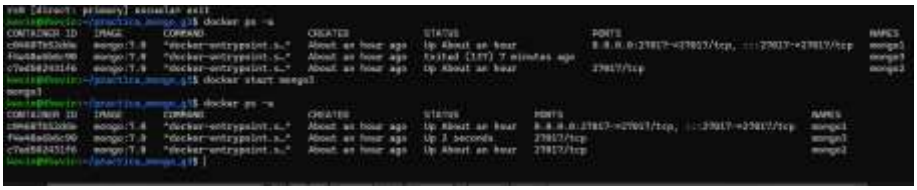

Fig 22. Levantamiento de conetendor mongo3

2.7 Resultados obtenidos

De acuerdo con el tipo de trabajo, se plasmarán los resultados alcanzados en la guía práctica una vez ejecutadas las actividades. Se pueden emplear figuras y tablas las cuales deben ser numeradas.

2.8 Habilidades blandas empleadas en la práctica

☐ Liderazgo
☒ Trabajo en equipo
☐ Comunicación asertiva
☐ La empatía
☐ Pensamiento crítico
☐ Flexibilidad
☐ La resolución de conflictos
☐ Adaptabilidad
☐ Responsabilidad

2.9 Conclusiones

La implementación de PostgreSQL mediante Docker Compose permitió comprender de manera práctica el uso de contenedores para la gestión eficiente de servicios de bases de datos. Esta actividad facilitó la creación de un entorno controlado y reproducible, reduciendo los problemas de configuración manual. El uso de volúmenes demostró la importancia de la persistencia de datos en entornos virtualizados. Asimismo, se verificó la correcta comunicación entre servicios mediante redes internas.

2.10 Recomendaciones

Se recomienda verificar previamente la instalación y correcta configuración de Docker Desktop y Docker Compose antes de iniciar la práctica. Es importante revisar las versiones de las imágenes utilizadas para asegurar compatibilidad y estabilidad del entorno. Asimismo, se sugiere mantener un control adecuado de los volúmenes creados para evitar la acumulación innecesaria de datos. En caso de realizar pruebas adicionales, se aconseja documentar los cambios en el archivo docker-compose.yml para facilitar la replicación del entorno. Finalmente, se recomienda utilizar herramientas como pgAdmin o DBeaver para una mejor administración y visualización de la base de datos.

2.11 Referencias bibliográficas

[1] T. Charboneau, How to Use the Postgres Docker Official Image, Docker, 2022. [En línea]. Disponible en: https://www.docker.com/blog/how-to-use-the-postgres-docker-official-image/

[2] G. Manandhar, Postgres with Docker and Docker Compose: A Step-by-Step Guide for Beginners, 2021. [En línea]. Disponible en: https://geshan.com.np/blog/2021/12/docker-postgres/