Task 5: Database & Storage Optimization

This task focuses on optimizing **PostgreSQL or MongoDB** databases to improve **performance, scalability, and efficiency**. The goal is to ensure **faster queries, reduced storage costs, and better reliability**.

📌 Overview

Database optimization is essential for:
✅ **Faster Query Execution** (reducing response times).
✅ **Efficient Storage Utilization** (avoiding redundant data).
✅ **Scalability** (handling large datasets with minimal performance loss).

I will focus on **PostgreSQL optimization**, but the same principles apply to **MongoDB**.

📌 1. PostgreSQL Optimization Techniques

🔹 Step 1: Indexing for Faster Queries

🔸 Why?

Indexes speed up searches by reducing the amount of data scanned.

🔹 How to Apply?

Check slow queries:
sql
CopyEdit
```
EXPLAIN ANALYZE SELECT * FROM users WHERE email = 'test@example.com';
```

1.

Create an Index:
sql
CopyEdit
```
CREATE INDEX idx_users_email ON users(email);
```

2.

Verify Improvement:
sql
CopyEdit
```sql
EXPLAIN ANALYZE SELECT * FROM users WHERE email = 'test@example.com';
```

3.

✅ **Indexing reduced query execution time significantly!**

🔹 Step 2: Optimize Queries

🔸 Why?

Poorly written queries cause **slow performance & high CPU usage**.

🔹 How to Apply?

Avoid SELECT *: Fetch only required columns.
sql
CopyEdit
```sql
SELECT id, name FROM users WHERE email = 'test@example.com';
```

1.

Use Joins Efficiently:
sql
CopyEdit
```sql
SELECT users.name, orders.amount
FROM users
INNER JOIN orders ON users.id = orders.user_id
WHERE users.email = 'test@example.com';
```

2.
3. **Use Query Caching**: Store **frequently accessed data** in memory.

✅ **Optimized queries improve response time and reduce database load.**

🔹 Step 3: Partitioning Large Tables

🔸 Why?

When dealing with **millions of records**, searching a **single huge table** slows down performance.

🔹 How to Apply?

Create a Partitioned Table:
sql
CopyEdit
```
CREATE TABLE orders (
    id SERIAL PRIMARY KEY,
    order_date DATE NOT NULL,
    amount NUMERIC
) PARTITION BY RANGE (order_date);
```

1.

Create Partitions:
sql
CopyEdit
```
CREATE TABLE orders_2024 PARTITION OF orders
FOR VALUES FROM ('2024-01-01') TO ('2024-12-31');
```

2.

✅ **Partitioning improves query speed by narrowing the search range.**

🔹 Step 4: Storage Optimization Strategies

🔸 Why?

Efficient storage reduces **costs and improves performance**.

🔹 How to Apply?

✅ **Enable Compression** for large text fields:

sql
CopyEdit
```
ALTER TABLE logs SET STORAGE EXTERNAL;
```

✅ **Delete Unused Data**: Archive or remove **stale data** regularly.
✅ **Use JSONB Instead of JSON** for faster queries in PostgreSQL.

sql
CopyEdit
```
ALTER TABLE events ALTER COLUMN metadata SET DATA TYPE JSONB;
```

✅ **Storage optimization keeps the database lightweight and efficient.**

📌 2. MongoDB Optimization Techniques (If Used)

If using **MongoDB**, apply these techniques:
✅ **Use Indexes**:

sh
CopyEdit
```
db.users.createIndex({ email: 1 })
```

✅ **Optimize Queries**:

sh
CopyEdit
```
db.users.find({ email: "test@example.com" }).explain("executionStats")
```

✅ **Use Sharding for Scalability**:

sh
CopyEdit
```
sh.enableSharding("myDatabase")
```

📌 3. Conclusion

✅ **Indexing speeds up queries.**
✅ **Partitioning improves scalability.**
✅ **Storage optimization reduces database size & costs.**

🚀 **Now, the database is fully optimized for high performance!** 🎉