**LAPORAN TUGAS MATKUL PDT**

**Anggota :**

**Rian Riswandi**

**Fahmi Abdul Aziz**

**Maulida Fitria**

**Kaka Dimas Herawan**

**Hilma Lutfia**

**Muhamad Fahmi Idris**

**Muhamad Iqbal Fadillah**

**Muhammad Akmal Husnayain**

**Noviana Rhamasanie**

**Ikhsan Suratman**

**Laporan Sistem Manajemen Rumah Sakit Terdistribusi**

1. **Pendahuluan**

Sistem manajemen rumah sakit (SMRS) merupakan tulang punggung operasional fasilitas kesehatan modern, berfungsi untuk mengelola informasi pasien, jadwal janji temu, catatan medis, dan berbagai operasi internal rumah sakit. Dengan semakin kompleksnya data dan kebutuhan akan ketersediaan yang tinggi, adopsi arsitektur sistem terdistribusi menjadi krusial. Laporan ini menguraikan perancangan dan implementasi SMRS terdistribusi yang memanfaatkan berbagai teknologi basis data modern untuk mencapai skalabilitas, ketersediaan, dan efisiensi.

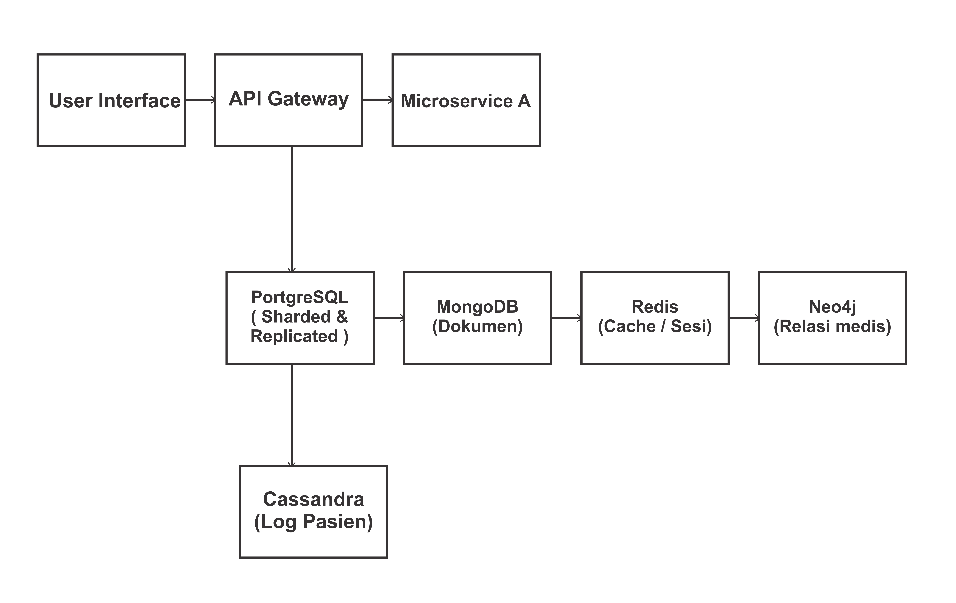
1. **Gambaran Kasus Pengguna**

Skenario kasus penggunaan melibatkan pengelolaan informasi penting di seluruh departemen rumah sakit. Ini mencakup:

1. Informasi Pasien: Data demografi, riwayat medis, alergi, dan informasi kontak.
2. Janji Temu: Penjadwalan, informasi dokter, status janji temu.
3. Operasi Internal: Manajemen staf, inventaris obat, data keuangan dasar, dan informasi departemen.

Sistem harus mampu menangani volume data yang besar, memastikan konsistensi data, dan menyediakan akses cepat untuk kebutuhan operasional sehari-hari.

1. **Arsitektur Sistem**

****Arsitektur sistem yang diusulkan adalah arsitektur mikroservis yang terhubung dengan berbagai jenis basis data sesuai dengan kebutuhan data.

1. API Gateway: Titik masuk tunggal untuk semua permintaan klien, mengarahkan permintaan ke mikroservis yang relevan.
2. Microservices: Setiap fungsionalitas inti sistem (misalnya, manajemen pasien, penjadwalan, rekam medis) akan diimplementasikan sebagai mikroservis terpisah.
3. Basis Data Poliglot: Penggunaan basis data yang berbeda untuk jenis data yang berbeda, memaksimalkan efisiensi dan skalabilitas.
4. **Tumpukan Teknologi & Alasan Pemilihan**

Berikut adalah teknologi yang dipilih dan alasannya:

1. Pemrograman: Python dengan FastAPI.

Alasan: Python mudah dipelajari dan memiliki ekosistem yang kaya untuk pengembangan web dan interaksi basis data. FastAPI adalah framework web modern yang cepat, mendukung asynchronous operations, dan secara otomatis menghasilkan dokumentasi API interaktif (OpenAPI/Swagger UI), sangat cocok untuk mikroservis. Keduanya didukung dengan baik oleh Docker dan VS Code.

1. PostgreSQL:

Alasan: Pilihan kuat untuk data terstruktur dan relasional seperti informasi pasien, detail janji temu, dan data struktural lainnya. Mendukung fitur canggih seperti sharding, replikasi, dan Foreign Data Wrappers (FDW).

1. MongoDB:

Alasan: Ideal untuk data semi-terstruktur berbasis dokumen, seperti rekam medis pasien yang mungkin memiliki struktur yang bervariasi (misalnya, hasil lab yang berbeda, catatan dokter yang tidak terstruktur). Fleksibilitas skema memudahkan evolusi data.

1. Redis:

Alasan: Cache dalam memori yang sangat cepat, cocok untuk data yang sering berubah atau membutuhkan akses cepat seperti status janji temu real-time, sesi pengguna, atau data inventaris obat yang sering diakses.

1. Neo4j:

Alasan: Basis data grafik yang efisien untuk merepresentasikan hubungan kompleks, misalnya, hubungan antar pasien, dokter, diagnosis, atau rujukan. Berguna untuk analisis jaringan dan rekomendasi.

1. Cassandra:

Alasan: Basis data NoSQL terdistribusi yang sangat baik untuk menulis data dalam jumlah besar dan membaca data secara cepat dengan ketersediaan tinggi. Cocok untuk menyimpan log aktivitas pasien, riwayat transaksi, atau data sensor medis.

1. Docker & Docker Compose:

Alasan: Menyediakan lingkungan pengembangan dan produksi yang konsisten. Memungkinkan kita untuk mengemas aplikasi dan semua dependensinya ke dalam kontainer yang terisolasi, memudahkan deployment dan manajemen. docker-compose.yml menyederhanakan orkestrasi multi-kontainer.

1. **Desain Skema Basis Data**

Berikut adalah contoh skema dasar untuk setiap basis data.

1. **PostgreSQL (Data Relasional)**

Untuk data relasional utama, kita akan memiliki tabel seperti Pasien, Dokter, JanjiTemu, Departemen, Obat.

1. **Tabel Pasien (Sharded/Partitioned)**

SQL

CREATE TABLE Pasien (

pasien\_id UUID PRIMARY KEY DEFAULT gen\_random\_uuid(),

nama\_lengkap VARCHAR(255) NOT NULL,

tanggal\_lahir DATE,

jenis\_kelamin VARCHAR(10),

alamat TEXT,

nomor\_telepon VARCHAR(20),

email VARCHAR(255)

);

*Catatan: Sharding atau partisi akan dilakukan berdasarkan pasien\_id atau tanggal\_lahir atau kriteria lain yang sesuai.*

1. **Tabel Dokter**

SQL

CREATE TABLE Dokter (

dokter\_id UUID PRIMARY KEY DEFAULT gen\_random\_uuid(),

nama\_lengkap VARCHAR(255) NOT NULL,

spesialisasi VARCHAR(100),

nomor\_telepon VARCHAR(20),

email VARCHAR(255),

departemen\_id UUID REFERENCES Departemen(departemen\_id)

);

1. **Tabel Janji Tamu**

SQL

CREATE TABLE JanjiTemu (

janji\_temu\_id UUID PRIMARY KEY DEFAULT gen\_random\_uuid(),

pasien\_id UUID REFERENCES Pasien(pasien\_id),

dokter\_id UUID REFERENCES Dokter(dokter\_id),

tanggal\_waktu TIMESTAMP NOT NULL,

status\_janji VARCHAR(50), -- e.g., 'Terjadwal', 'Selesai', 'Batal'

catatan TEXT

1. **Tabel Departemen**

SQL

CREATE TABLE Departemen (

departemen\_id UUID PRIMARY KEY DEFAULT gen\_random\_uuid(),

nama\_departemen VARCHAR(100) NOT NULL,

lokasi VARCHAR(255)

);

1. **Tabel Obat (Contoh untuk data inventaris)**

SQL

CREATE TABLE Obat (

obat\_id UUID PRIMARY KEY DEFAULT gen\_random\_uuid(),

nama\_obat VARCHAR(255) NOT NULL,

stok\_tersedia INT,

harga\_per\_unit DECIMAL(10, 2)

);

1. **Mongo DB ( Data Dokumen)**

Digunakan untuk RekamMedis yang mungkin memiliki struktur dinamis.

**Koleksi rekam medis**

JSON

{

"\_id": ObjectId("..."),

"pasien\_id": "UUID\_Pasien\_dari\_PostgreSQL",

"tanggal\_rekam": ISODate("2023-10-26T10:00:00Z"),

"jenis\_rekam": "Pemeriksaan", // e.g., "Pemeriksaan", "Hasil Lab", "Resep"

"data\_detail": {

"dokter": "Nama Dokter",

"diagnosa": "Demam",

"obat\_resep": [

{"nama\_obat": "Paracetamol", "dosis": "500mg"},

{"nama\_obat": "Amoxicillin", "dosis": "250mg"}

],

"hasil\_lab": {

"hb": "12.5 g/dL",

"gula\_darah": "90 mg/dL"

}

// Struktur ini bisa sangat bervariasi

},

"catatan\_tambahan": "Pasien menunjukkan perbaikan."

}

1. **Redis (Data chace/sesi)**

Tidak ada skema formal, tetapi data disimpan sebagai pasangan key-value.

**Contoh penggunaan:**

1. session:user\_id: data sesi pengguna
2. appointment:status:janji\_temu\_id: status janji temu real-time
3. obat:stok:obat\_id: stok obat yang sering diakses (untuk read-heavy operations)
4. **Neo4j (Data Grafik)**

Untuk merepresentasikan hubungan antar entitas.

1. **Nodes:**
2. (Pasien {pasien\_id: "..."})
3. (Dokter {dokter\_id: "..."})
4. (Diagnosa {nama: "..."})
5. (Obat {nama: "..."})
6. Relationships:
7. (Pasien)-[:DIAGNOSA\_OLEH]->(Dokter)
8. (Pasien)-[:MENDERITA]->(Diagnosa)
9. (Dokter)-[:MERESEPKAN]->(Obat)
10. (Pasien)-[:TERHUBUNG\_KE]->(Pasien) (misalnya, anggota keluarga)
11. **Casandra (Log data)**

Untuk log aktivitas pasien, riwayat akses, atau data sensor (jika ada).

**Tabel patient\_activity\_log**

Cuplikan kode

CREATE TABLE patient\_activity\_log (

pasien\_id UUID,

timestamp TIMESTAMP,

activity\_type TEXT,

description TEXT,

PRIMARY KEY (pasien\_id, timestamp)

) WITH CLUSTERING ORDER BY (timestamp DESC);

1. **Strategi Sharding dan Replikasi**
2. **Sharding di PostgreSQL (Menggunakan Partisi Bawaan)**

Daripada menggunakan pg\_shard atau Citus yang mungkin memerlukan konfigurasi lebih kompleks untuk pemula, kita bisa mendemonstrasikan konsep sharding menggunakan fitur **partisi deklaratif** bawaan PostgreSQL yang lebih mudah diatur. Ini akan mensimulasikan sharding pada satu instance PostgreSQL. Untuk skala sebenarnya, ini akan diimplementasikan di beberapa instance.

1. **Tabel Induk**

SQL

CREATE TABLE Pasien (

pasien\_id UUID PRIMARY KEY DEFAULT gen\_random\_uuid(),

nama\_lengkap VARCHAR(255) NOT NULL,

tanggal\_lahir DATE,

jenis\_kelamin VARCHAR(10),

alamat TEXT,

nomor\_telepon VARCHAR(20),

email VARCHAR(255)

) PARTITION BY RANGE (tanggal\_lahir);

1. **Partisi Anak**

SQL

CREATE TABLE Pasien\_Pre1980 PARTITION OF Pasien

FOR VALUES FROM ('1900-01-01') TO ('1980-01-01');

CREATE TABLE Pasien\_1980s PARTITION OF Pasien

FOR VALUES FROM ('1980-01-01') TO ('1990-01-01');

CREATE TABLE Pasien\_1990s PARTITION OF Pasien

FOR VALUES FROM ('1990-01-01') TO ('2000-01-01');

CREATE TABLE Pasien\_Post2000 PARTITION OF Pasien

FOR VALUES FROM ('2000-01-01') TO ('2050-01-01');

*Ini adalah contoh sederhana. Anda bisa menambahkan lebih banyak partisi atau menggunakan kriteria lain.*

1. **Replikasi di PostgreSQL (Replikasi Logis)**

Kita akan menyiapkan replikasi logis antara dua instance PostgreSQL. Satu sebagai **Publisher** dan satu sebagai **Subscriber**.

**Langkah-langkah umum (akan dijelaskan lebih detail di bagian implementasi):**

1. **Konfigurasi Publisher (Master):**

* Aktifkan wal\_level = logical di postgresql.conf.
* Buat publikasi untuk tabel yang ingin direplikasi (misalnya, Pasien).

1. **Konfigurasi Subscriber (Replika):**

* Buat database yang sama.
* Buat langganan ke publikasi dari Publisher.

1. **Demonstrasi:**

Perubahan data pada Publisher akan secara otomatis direplikasi ke Subscriber.

1. **Contoh Kueri atau Skenario Terdistribusi**

Skenario ini akan menunjukkan bagaimana data dari berbagai sumber digabungkan.

**Skenario:** Menampilkan daftar pasien dan rekam medis terakhir mereka, serta status janji temu mereka yang akan datang.

**Langkah-langkah Kueri:**

1. **PostgreSQL:** Ambil informasi dasar pasien dari tabel Pasien.

SQL

SELECT pasien\_id, nama\_lengkap, tanggal\_lahir FROM Pasien WHERE pasien\_id = 'suatu\_pasien\_id';

1. **MongoDB:** Ambil rekam medis terbaru untuk pasien\_id tersebut dari koleksi Rekam Medis.

JavaScript

db.RekamMedis.find({"pasien\_id": "suatu\_pasien\_id"}).sort({"tanggal\_rekam": -1}).limit(1);

1. **Redis:** Ambil status janji temu pasien (jika ada yang aktif/mendatang) dari cache Redis.

Python

# Contoh di Python, mendapatkan dari Redis

redis\_client.get(f"appointment:status:{janji\_temu\_id\_dari\_PostgreSQL}")

1. **FDW (Foreign Data Wrapper) di PostgreSQL:** Mengakses data dari Cassandra (log aktivitas pasien) langsung dari PostgreSQL

Ini akan memungkinkan kueri gabungan tanpa perlu ekstraksi/transformasi manual.

**Contoh Kueri Gabungan (Menggunakan Python/FastAPI):**

Aplikasi Python akan berfungsi sebagai orkestrator, mengambil data dari

berbagai basis data dan menggabungkannya sebelum mengirimkannya ke

klien.

Python

# Contoh pseudocode dalam FastAPI

from fastapi import FastAPI

from sqlalchemy import create\_engine, text # Untuk PostgreSQL

from pymongo import MongoClient # Untuk MongoDB

import redis # Untuk Redis

from cassandra.cluster import Cluster # Untuk Cassandra

app = FastAPI()

# Inisialisasi koneksi DB (akan diatur di docker-compose)

pg\_engine = create\_engine("postgresql://user:password@pg\_master:5432/hospital\_db")

mongo\_client = MongoClient("mongodb://mongodb:27017/")

redis\_client = redis.Redis(host="redis", port=6379)

cassandra\_cluster = Cluster(['cassandra']) # Jika ada beberapa node, sesuaikan

@app.get("/pasien/{pasien\_id}/detail")

async def get\_pasien\_detail(pasien\_id: str):

pasien\_info = {}

rekam\_medis\_terakhir = {}

status\_janji\_temu = "Tidak ada janji temu mendatang"

log\_aktivitas = []

# 1. Ambil dari PostgreSQL

with pg\_engine.connect() as connection:

result = connection.execute(

text(f"SELECT pasien\_id, nama\_lengkap, tanggal\_lahir FROM Pasien WHERE pasien\_id = '{pasien\_id}'")

).fetchone()

if result:

pasien\_info = dict(result)

# 2. Ambil dari MongoDB

db\_mongo = mongo\_client["hospital\_db\_mongo"]

rekam\_medis = db\_mongo["RekamMedis"].find\_one(

{"pasien\_id": pasien\_id},

sort=[("tanggal\_rekam", -1)]

)

if rekam\_medis:

rekam\_medis\_terakhir = rekam\_medis

# 3. Ambil dari Redis (contoh: untuk status janji temu cepat)

# Asumsi kita punya janji\_temu\_id yang terkait dengan pasien dari PG

# Ini perlu penyesuaian tergantung bagaimana Anda menyimpan janji\_temu di PG

# Misalnya, jika kita tahu janji\_temu\_id, kita bisa cek di Redis

# Untuk demo, asumsikan kita punya janji\_temu\_id dari PG, lalu cek Redis

# Simplifikasi: langsung cek status umum pasien

redis\_status = redis\_client.get(f"patient\_status:{pasien\_id}")

if redis\_status:

status\_janji\_temu = redis\_status.decode('utf-8')

# 4. Ambil dari Cassandra (menggunakan FDW atau koneksi langsung dari aplikasi)

# Jika menggunakan FDW, ini bisa jadi bagian dari kueri PostgreSQL

# Jika koneksi langsung dari aplikasi:

session\_cassandra = cassandra\_cluster.connect('hospital\_keyspace')

rows = session\_cassandra.execute(f"SELECT \* FROM patient\_activity\_log WHERE pasien\_id = {pasien\_id}")

for row in rows:

log\_aktivitas.append(dict(row))

return {

"pasien\_info": pasien\_info,

"rekam\_medis\_terakhir": rekam\_medis\_terakhir,

"status\_janji\_temu": status\_janji\_temu,

"log\_aktivitas": log\_aktivitas

}

1. **Tantangan & Pembelajaran yang Didapat**
2. **Tantangan**
3. **Kompleksitas Konfigurasi:** Mengatur banyak basis data terdistribusi dalam satu lingkungan (terutama sharding dan replikasi) memerlukan pemahaman mendalam tentang setiap teknologi.
4. **Konsistensi Data:** Memastikan konsistensi data di seluruh sistem poliglot merupakan tantangan besar, terutama saat transaksi melibatkan beberapa basis data.
5. **Debugging Terdistribusi:** Melacak masalah di sistem terdistribusi lebih sulit dibandingkan sistem monolitik.
6. **Manajemen Skema:** Mengelola skema yang berbeda di basis data yang berbeda dan bagaimana mereka saling terkait.
7. **Kurva Pembelajaran:** Mempelajari sintaks dan konsep yang berbeda untuk setiap basis data.
8. **Pembelajaran**
   1. Pentingnya memilih basis data yang tepat untuk jenis data tertentu (polyglot persistence).
   2. docker-compose sangat membantu dalam orkestrasi lingkungan pengembangan terdistribusi.
   3. Desain API yang jelas dan modular sangat penting untuk mikroservis.
   4. Replikasi meningkatkan ketersediaan, sementara sharding meningkatkan skalabilitas.
   5. FDW memungkinkan federasi data yang kuat, menyederhanakan kueri lintas basis data.
9. **Kesimpulan**

Perancangan dan implementasi Sistem Manajemen Rumah Sakit dengan arsitektur terdistribusi menggunakan PostgreSQL, MongoDB, Redis, Neo4j, dan Cassandra menunjukkan pendekatan modern dalam menangani data skala besar dan kompleks. Dengan memanfaatkan kekuatan masing-masing basis data, sistem ini mencapai skalabilitas, ketersediaan, dan fleksibilitas yang tinggi. Meskipun terdapat tantangan dalam konfigurasi dan manajemen, pembelajaran yang didapat sangat berharga dalam membangun sistem yang tangguh dan efisien.