LAPORAN PRAKTIKUM ANALISIS BIG DATA

Anggota:

Lion Abdi Marga	121450047
Lia Alyani	121450138
Happy Syahrul Ramadhan	122450013
Eli Dwi Putra Berema	122450064
M. Deriansyah Okutra	122450101

Program Studi Sains Data
Fakultas Sains
Institut Teknologi Sumatera
Lampung Selatan
2025

BAB I
Ringkasan Kebutuhan dari Misi

Desain konseptual untuk data warehouse dalam industri logistik bertujuan menyediakan fondasi analitik yang terstruktur dan terintegrasi, dengan merancang entitas-entitas dimensi utama yang merepresentasikan berbagai entitas bisnis. Struktur dimensi ini sepenuhnya diadopsi dari studi kasus pengelolaan pengiriman logistik dan disesuaikan dengan kebutuhan integrasi sistem seperti ERP, GPS, CRM, serta data eksternal.

1.1 Atribut dan Dimensi

1. **dim_date**
   Menampung atribut temporal yang digunakan untuk mengelompokkan kejadian dan transaksi berdasarkan waktu.

Kolom	**Deskripsi**
date_id (PK)	ID unik untuk setiap tanggal.
date	Tanggal pengiriman atau kejadian terkait pengiriman.

2. **dim_route**
   Mewakili rute-rute pengiriman yang menghubungkan lokasi asal dan tujuan.

Kolom	**Deskripsi**
route_id (PK)	ID unik untuk setiap rute pengiriman.
location_origin	Lokasi asal pengiriman.
location_destination	Lokasi tujuan pengiriman.
planned_distance	Jarak yang direncanakan untuk rute tersebut.

3. **dim_vehicle**
   Mewakili kendaraan logistik yang digunakan dalam proses distribusi.

Kolom	**Deskripsi**
vehicle_id (PK)	ID unik untuk setiap kendaraan.
vehicle_type	Jenis kendaraan (misalnya, truk, van, dll.).
capacity	Kapasitas kendaraan.
gps_device_id	ID perangkat GPS yang terpasang pada kendaraan.
maintenance_status	Status pemeliharaan kendaraan (baik, rusak, dll.).
fuel_type	Jenis bahan bakar kendaraan (misalnya, bensin, diesel).

vehicle_age	Usia kendaraan.

4. **dim_product**

Merepresentasikan barang yang dikirim dalam proses logistik.

Kolom	**Deskripsi**
product_id (PK)	ID unik untuk setiap produk.
product_name	Nama produk
product_category	Kategori produk (misalnya, elektronik, pakaian)
is_sensitive	Menandakan apakah produk sensitif terhadap suhu atau kelembapan atau bantingan.
product_weight	Berat produk.
product_dimensions	Dimensi produk (panjang, lebar, tinggi).

5. **dim_customer**

Berisi data pelanggan penerima barang.

Kolom	**Deskripsi**
customer_id (PK)	ID unik untuk setiap pelanggan.
customer_name	Nama pelanggan.
gender	jenis kelamin customer
date_of_birtday	tgl lahir customer
region	Wilayah atau lokasi geografis pelanggan.

6. **dim_incident**

Menampung informasi insiden selama pengiriman.

Kolom	**Deskripsi**
incident_id (PK)	ID unik untuk setiap insiden.
incident_type	Jenis insiden yang terjadi (kecelakaan, keterlambatan).
description	Deskripsi insiden.

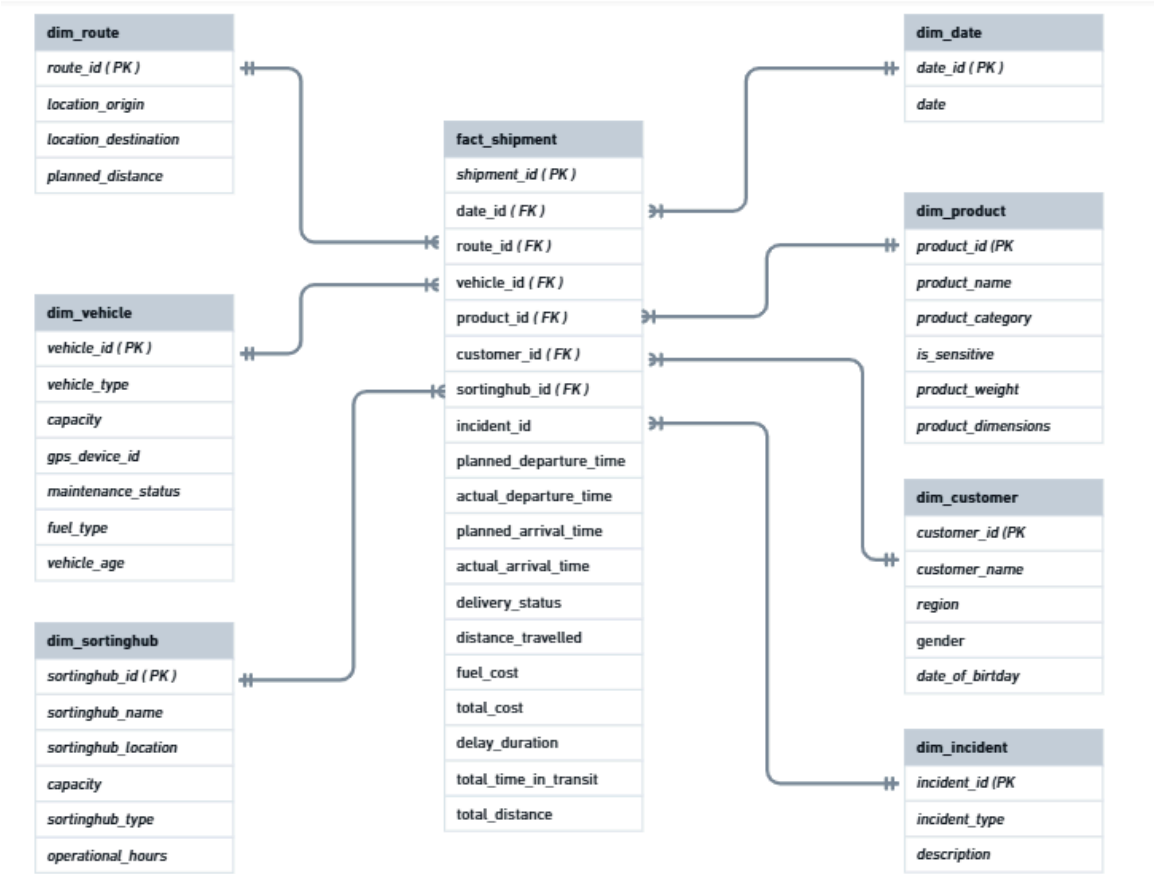
7. **dim_sortinghub**

Mewakili fasilitas penyortiran logistik dalam rantai distribusi.

Kolom	Deskripsi
sortinghub_id (PK)	ID unik untuk setiap hub sortir
sortinghub_name	Nama tempat sortir
sortinghub_location	Lokasi hub sortir.
operational_hours	Jam operasional hub sortir
capacity	Kapasitas hub dalam hal volume atau barang.
sortinghub_type	Jenis hub (misal: pusat distribusi, hub regional).

2.2 Skema Konsetual

Untuk menggambarkan hubungan antar dimensi yang akan mendukung analisis pengiriman, digunakan pendekatan skema bintang (star schema). Dalam pendekatan ini, sebuah pusat entitas utama (tabel fakta) nantinya akan menghubungkan semua dimensi melalui kunci asing (foreign key).



Desain konseptual ini dirancang agar tidak terlalu bergantung pada struktur operasional sumber, tetapi lebih mengarah pada kebutuhan analitik. Oleh karena itu, denormalisasi dilakukan pada

dimensi seperti dim_customer, dim_vehicle, dan dim_product untuk mempercepat proses kueri dan menyederhanakan pengambilan data.

2.3 Lineage Requirement

Dalam implementasi Data Warehouse, data lineage merupakan aspek penting yang menjelaskan asal-usul (source), transformasi, dan tujuan akhir data di seluruh alur pemrosesan. Lineage membantu pengguna dan pengembang memahami bagaimana data di tabel dimensi dibentuk dari sumbernya hingga siap dianalisis dalam DW. Ini sangat penting untuk menjamin integritas, transparansi, dan akuntabilitas data, terutama pada sektor logistik yang memiliki berbagai sumber data yang tersebar (seperti ERP, GPS, CRM, dan sistem pelacakan insiden).

1. Pelacakan Sumber Data: Identifikasi dari sistem mana data berasal (misalnya sistem GPS untuk kendaraan, CRM untuk pelanggan, ERP untuk produk).
2. Transformasi Data: Dokumentasi proses ETL (Extract, Transform, Load) yang dilakukan pada data mentah sebelum masuk ke tabel dimensi. Contohnya, konversi format tanggal, normalisasi status kendaraan, atau klasifikasi sensitivitas produk.
3. Frekuensi Pembaruan: Frekuensi pembaruan tiap dimensi harus terdokumentasi, misalnya data kendaraan diperbarui bulanan, sedangkan data insiden diperbarui harian.
4. Ketertelusuran Atribut: Setiap kolom pada tabel dimensi harus memiliki catatan asal-usul dan logika pembentukannya dari atribut sumber.
5. Versi dan Audit Data: Untuk histori perubahan data yang penting, seperti perubahan status pemeliharaan kendaraan atau pembaruan data pelanggan.

BAB II
Design and Implement Fact Tables

Dalam pengembangan sistem data warehouse untuk mendukung kebutuhan analitik dan pelaporan perusahaan logistik, salah satu komponen utama yang dirancang adalah fact table. Tabel fakta ini merupakan pusat integrasi data numerik yang bersifat kuantitatif dan terhubung ke berbagai tabel dimensi yang menyediakan konteks untuk analisis.

2.1 Identify Measures

Ukuran (measures) dalam tabel fakta, atau yang sering disebut sebagai fakta itu sendiri, merupakan representasi numerik dari peristiwa-peristiwa penting yang terjadi selama proses bisnis berlangsung. Dalam konteks industri logistik, ukuran tersebut mencerminkan performa dan efisiensi pengiriman barang. Contoh ukuran dalam proses pengiriman meliputi:

1. Total Cost (biaya keseluruhan pengiriman),
2. Fuel Cost (biaya bahan bakar),
3. Delay Duration (durasi keterlambatan),
4. Distance Travelled (jarak tempuh aktual).

Ukuran ini dipilih berdasarkan hal-hal yang ingin dipantau atau diperhatikan oleh perusahaan. Oleh karena itu, pemilihan ukuran yang tepat sangat penting agar data yang disimpan dapat dimanfaatkan secara optimal untuk analisis dan pengambilan keputusan.

2.2 Design and Implement Fact Tables

Desain tabel fakta dimulai dengan mendefinisikan proses bisnis utama yang akan dimodelkan. Dalam konteks industri logistik, proses yang dimodelkan adalah pengiriman barang, sehingga tabel fakta utama dinamakan fact_shipment, dengan tipe kolom seperti berikut:

Tipe Kolom	Kolom	Keterangan
Primary Key	shipment_id	sebagai kode unik dari setiap transaksi
Foreign Key	vehicle_id, date_id, route_id, customer_id, product_id	Menghubungkan data pengiriman ke konteks waktu, kendaraan, rute, dll.(Tabel dimensi)
Measure	total_cost, delay_duration, fuel_cost	Nilai-nilai kuantitatif hasil dari proses bisnis
Metadata	created_at, updated_by, batch_id	Mencatat kapan dan oleh siapa data dimasukkan atau diperbarui (proses ETL)

2.3 Create composite keys

Dalam desain data warehouse, composite key digunakan untuk menjamin keunikan setiap baris pada tabel fakta. Composite key dibentuk dari kombinasi beberapa foreign key yang merujuk ke tabel-tabel dimensi. composite key penting untuk menjaga integritas data, mendeteksi duplikat, dan memahami hubungan antar entitas.

Composite Keys yang kami gunakan seperti ini:
1. vehicle_id → kendaraan
2. route_id → rute
3. product_id → produk yang dikirim
4. date_id → tanggal pengiriman
5. customer_id → penerima

Semua tabel dimensi digunakan pada composite keys kali ini, karena Semua dimensi juga relevan dan dibutuhkan untuk analisis performa pengiriman
Code Query:

```
ALTER TABLE fact_shipment
ADD CONSTRAINT pk_fact_shipment_composite
PRIMARY KEY (vehicle_id, route_id, product_id, date_id, customer_id);
```

2.4 Implement additive, semi-additive, and non-additive measures

Dalam perancangan tabel fakta, penting untuk memahami bahwa tidak semua ukuran (measures) dapat diperlakukan secara sama dalam proses agregasi. Berdasarkan sifatnya, ukuran-ukuran dalam tabel fakta dapat diklasifikasikan menjadi tiga kategori utama, yaitu additive, semi-additive, dan non-additive.

1. Additive Measures

Additive measures adalah ukuran yang dapat dijumlahkan melalui semua dimensi. Ini adalah jenis ukuran yang paling umum dan paling mudah digunakan dalam laporan agregasi dan analisis kuantitatif. Additive measures pada fact_shipment sebagai berikut:

- total_cost = dapat dijumlahkan berdasarkan kendaraan, pelanggan, waktu, produk, rute, dsb.
- fuel_cost = dapat dijumlahkan berdasarkan rute, kendaraan, waktu, dan sorting hub.
- distance_travelled = bisa dijumlahkan untuk total jarak pengiriman harian, bulanan, atau per kendaraan.

2. Semi-Additive Measures

Semi-additive measures adalah ukuran yang hanya dapat dijumlahkan pada beberapa dimensi saja, tetapi tidak cocok dijumlahkan terhadap dimensi waktu. Biasanya, ukuran-ukuran ini bersifat snapshot atau mencerminkan suatu nilai pada titik waktu tertentu. Semi-additive measures pada fact shipment, sebagai berikut:

- delay_duration = dirata-ratakan per rute atau kendaraan, tetapi tidak dijumlahkan per minggu atau per bulan secara langsung.
- planned_arrival_time & actual_arrival_time = digunakan untuk menghitung ketepatan waktu, namun tidak dijumlahkan.

3. Non-Additive Measures

Non-additive measures adalah ukuran yang tidak dapat dijumlahkan terhadap dimensi apapun. Umumnya berupa rasio, persentase, atau nilai-nilai turunan dari agregasi lainnya. Non-additive (potensial) pada fact_shipment, sebagai berikut:

- Efisiensi Pengiriman (%) = distance_planned / distance_travelled
- Tingkat Ketepatan Waktu (%) = jumlah pengiriman tepat waktu / total pengiriman
- Rata-rata Keterlambatan = rerata dari delay_duration

2.5 Identify Dimension Table Relationships

Relasi antara tabel fakta dan dimensi membentuk struktur Star Schema, di mana semua tabel dimensi terhubung langsung ke fact_shipment melalui relasi one-to-many (1:M).

Enterprise Business Matrix (EBM) – Perusahaan Logistik:

Business Process	Date	Vehicle	Route	Product	Customer	Incident	Sorting Hub
Pengiriman Barang	✔	✔	✔	✔	✔	✔	✔
Evaluasi Armada		✔				✔	
Analisis Biaya Operasional	✔	✔	✔				
Mitigasi Risiko Insiden	✔	✔	✔		✔	✔	
Segmentasi Pelanggan				✔	✔		
Evaluasi Performa Sorting Hub							✔

2.6 Design a Data Warehouse that Supports Many-to-Many Relationships

Hubungan ini sering muncul dalam skenario bisnis yang kompleks, termasuk dalam industri logistik. Misalnya, satu pelanggan dapat menerima berbagai jenis produk dalam banyak pengiriman yang berbeda, sementara satu produk juga dapat dikirimkan ke banyak pelanggan. Untuk menangani pola relasi seperti ini, data warehouse dirancang dengan menempatkan tabel fakta sebagai jembatan atau penghubung antar dimensi.

Tabel fakta seperti fact_shipment atau fact_sale berperan sebagai penghubung antara dua atau lebih dimensi yang saling berelasi secara M:N. Dalam kasus logistik, tabel ini bisa menghubungkan dim_customer dengan dim_product, di mana masing-masing tidak secara langsung memiliki foreign key satu sama lain, tetapi dapat dikaitkan melalui data transaksi pengiriman. Dengan pendekatan ini, dimensi yang awalnya tidak berelasi langsung dapat dianalisis secara bersilang (cross-analysis), seperti menganalisis produk apa yang paling sering dikirim ke pelanggan tertentu, atau pelanggan mana yang paling sering menerima produk tertentu.

Desain ini memberikan fleksibilitas tinggi dalam pelaporan dan analisis multidimensi karena memungkinkan penggabungan data dari berbagai perspektif. Selain itu, model ini juga mendukung analisis segmentasi, promosi, dan pengambilan keputusan strategis, tanpa harus mengubah struktur inti dari dimensi yang sudah ada. Oleh karena itu, pengelolaan many-to-many relationship melalui tabel fakta merupakan praktik penting dalam membangun data warehouse yang adaptif dan efisien terhadap kebutuhan analitik tingkat lanjut.

Contoh Query Sql:

```
-Produk yang Pernah Dikirim ke Setiap Pelanggan

SELECT DISTINCT
    c.customer_name,
    p.product_name
FROM fact_shipment fs
JOIN dim_customer c ON fs.customer_id = c.customer_id
JOIN dim_product p ON fs.product_id = p.product_id;

-Produk Terpopuler Berdasarkan Jumlah Pengiriman
SELECT
    p.product_name,
    COUNT(*) AS total_shipments
FROM fact_shipment fs
JOIN dim_product p ON fs.product_id = p.product_id
GROUP BY p.product_name
ORDER BY total_shipments DESC;
```

BAB III
Desain dan Implementasi Index

3.1 Desain Skema Index

Tabel fakta adalah pusat dari skema bintang dan menyimpan sebagian besar data. Dalam hal ini, tabel fakta menyimpan informasi terkait pengiriman. Karena tabel fakta seringkali berisi sejumlah besar data, indeksasi sangat penting untuk mempercepat pengambilan data, terutama untuk operasi **join** dan filter.

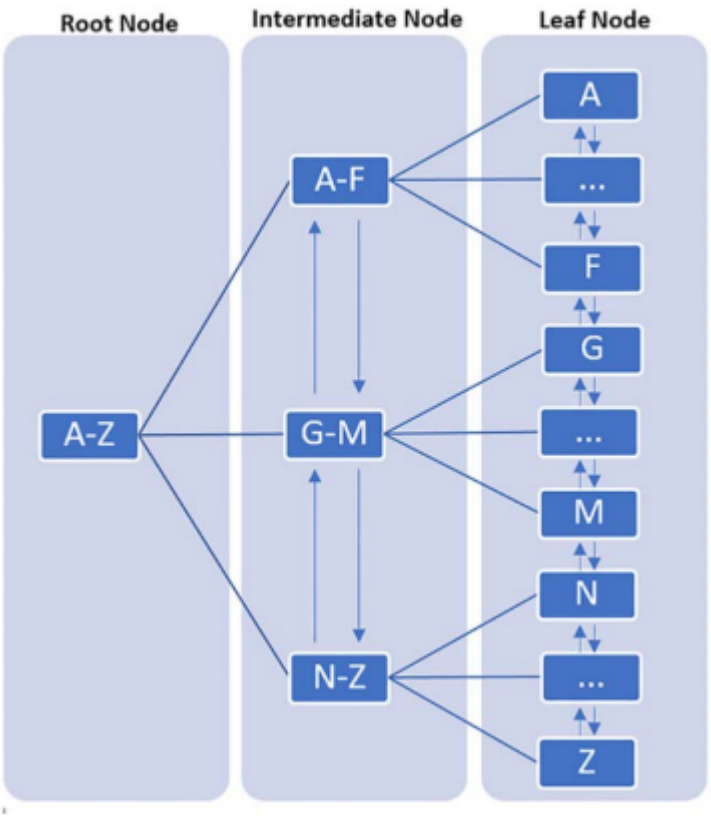
Jadi disini kami menggunakan algoritma B-Tree untuk optimasi pada skema penyimpanan kami :
Skema kami terdiri dari satu tabel fakta (fact_shipment) dan beberapa tabel dimensi (dim_vehicle, dim_product, dll). Query dalam skema ini biasanya melibatkan:

- Join antara tabel fakta dan tabel dimensi
- Filter berdasarkan ID, waktu, lokasi, dll
- Agregasi berdasarkan waktu atau kategori

B-Tree sangat cocok untuk operasi seperti ini karena :

- Efisien dalam pencarian nilai spesifik (misalnya product_id = 100)
- Efektif untuk rentang nilai (misalnya tanggal BETWEEN '2024-01-01' AND '2024-12-31')
- Mendukung operasi sorting dan scanning dengan sangat baik

Skema metode index dengan algoritma B-Tree



3.2 Implementasi Skema index

Implementasi index pada tabel *fact_shipment*

```
-- Membuat indeks utama pada primary key shipment_id (biasanya otomatis jika didefinisikan sebagai PRIMARY KEY)
CREATE UNIQUE INDEX idx_shipment_id ON fact_shipment(shipment_id);

-- Membuat indeks komposit pada semua foreign key untuk mempercepat operasi JOIN
CREATE INDEX idx_fact_shipment_fk ON fact_shipment(route_id, date_id, vehicle_id, product_id, customer_id, sortinghub_id, incident_id);

-- Membuat indeks pada kolom waktu untuk mempercepat pencarian berdasarkan waktu
CREATE INDEX idx_fact_shipment_departure_time ON fact_shipment(planned_departure_time, actual_departure_time);

-- Membuat indeks pada total_cost dan delivery_status
CREATE INDEX idx_fact_cost_status ON fact_shipment(total_cost, delivery_status);
```

Implementasi indeks pada Tabel Dimensi *dim_route*

```
-- Indeks pada primary key route_id
CREATE UNIQUE INDEX idx_route_id ON dim_route(route_id);

-- Indeks pada kolom lokasi rute karena akan sering digunakan untuk pencarian
CREATE INDEX idx_route_location ON dim_route(route_location);
```

Implementasi index pada Tabel Dimensi *dim_vehicle*

```
-- Indeks pada primary key vehicle_id
CREATE UNIQUE INDEX idx_vehicle_id ON dim_vehicle(vehicle_id);

-- Indeks pada fuel_type untuk filter kendaraan berdasarkan jenis bahan bakar
CREATE INDEX idx_vehicle_fuel_type ON dim_vehicle(fuel_type);

-- Indeks pada maintenance_status untuk filter kendaraan aktif atau tidak
CREATE INDEX idx_vehicle_maintenance ON dim_vehicle(maintenance_status);
```

Implementasi index pada Tabel Dimensi *dim_product*

```
-- Indeks pada primary key product_id
CREATE UNIQUE INDEX idx_product_id ON dim_product(product_id);

-- Indeks pada kategori produk
CREATE INDEX idx_product_category ON dim_product(product_category);
```

Implementasi index pada Tabel Dimensi *dim_customer*

```
-- Indeks pada primary key customer_id
CREATE UNIQUE INDEX idx_customer_id ON dim_customer(customer_id);

-- Indeks untuk query berdasarkan wilayah pelanggan
CREATE INDEX idx_customer_region ON dim_customer(region);

-- Indeks jika sering memfilter berdasarkan gender
CREATE INDEX idx_customer_gender ON dim_customer(gender);
```

Implementasi index pada Tabel Dimensi *dim_sortinghub*

```
-- Indeks pada primary key sortinghub_id
CREATE UNIQUE INDEX idx_sortinghub_id ON dim_sortinghub(sortinghub_id);

-- Indeks pada lokasi sorting hub
CREATE INDEX idx_sortinghub_location ON dim_sortinghub(sortinghub_location);
```

Implementasi index pada Tabel Dimensi *dim_incident*

```
-- Indeks pada primary key incident_id
CREATE UNIQUE INDEX idx_incident_id ON dim_incident(incident_id);

-- Indeks pada jenis insiden
CREATE INDEX idx_incident_type ON dim_incident(incident_type);
```

BAB IV
Desain Storage

Perusahaan logistik memerlukan data warehouse untuk mengintegrasikan data pengiriman, rute, armada, insiden, pelanggan, dan produk. Mengingat karakteristik data yang besar dan kompleks, diperlukan desain storage yang mampu mendukung performa kueri analitis, efisiensi storage, serta skalabilitas seiring pertumbuhan data historis.

4.1 Hardware Considerations

Komponen	Jenis Hardware	Kegunaan
Disk	SSD dengan RAID 10	Menyediakan I/O tinggi dan redudansi untuk akses data pengiriman besar
CPU	Minimal 16 core	Mendukung pemrosesanparalel untuk query OLAP dan proses ETL
Memory	128 GB RAM	Memungkinkan proses agregasi data besar secara in-memory

Pengiriman harian dengan ribuan transaksi menyebabkan pertumbuhan data eksponensial. Oleh karena itu, dibutuhkan perangkat keras yang mampu menangani beban baca tinggi dan pemrosesan kueri kompleks dari data pengiriman dan armada.

4.2 Database File Layout

File Tipe	**Lokasi Penyimpanan**
Data Files (.mdf)	SSD 1 (dedicated)
Transaction Logs (.ldf)	SSD 2 (terpisah)
TempDB	SSD 3, dengan multiple data files

Contoh implementasi struktur tabel fakta logistik:

```
CREATE TABLE fact_shipment (
  shipment_id BIGINT IDENTITY(1,1),
  date_id INT,
  route_id INT,
```

```
  vehicle_id INT,
  product_id INT,
  customer_id INT,
  incident_id INT,
  fuel_cost FLOAT,
  total_cost FLOAT,
  delay_duration INT,
  distance_travelled FLOAT
)
WITH (CLUSTERED COLUMNSTORE INDEX);
```

File data, log, dan TempDB dipisahkan agar tidak terjadi I/O contention saat proses ETL dan analisis OLAP dijalankan secara bersamaan.

4.3 Storage Models

Tabel	**Storage Model**
fact_shipment	Clustered Columnstore Index
dim_date	Rowstore Clustered Index
dim_vehicle	Rowstore Clustered Index
dim_route	Rowstore Clustered Index
dim_product	Rowstore Clustered Index
dim_customer	Rowstore Clustered Index
dim_incident	Rowstore Clustered Index
dim_sortinghub	Rowstore Clustered Index

Penggunaan Clustered Columnstore Index pada tabel fakta mempercepat analisis agregasi dan meminimalkan penggunaan storage. Tabel dimensi menggunakan Rowstore karena ukurannya kecil dan lebih cocok untuk lookup cepat.

4.4 Data Compression

Jenis Tabel	**Teknik Kompresi**
fact_shipment	Columnstore Compression (default)
dim_customer, dim_ route	Page Compression
dim_incident, dim_product	Row Compression

Contoh Implementasi:

```
ALTER TABLE dim_customer
REBUILD PARTITION = ALL
WITH (DATA_COMPRESSION = PAGE);
```

Teknik kompresi digunakan untuk mengurangi kebutuhan penyimpanan dan meningkatkan efisiensi cache selama eksekusi query.

4.5 Partitioning Strategy

Tabel	**Strategi Partisi**
fact_shipment	Partisi berdasarkan tahun pengiriman
dim_date	Tidak dipartisi

Contoh partisi berdasarkan year:

```
CREATE PARTITION FUNCTION pf_Year (INT)
AS RANGE LEFT FOR VALUES (2022, 2023, 2024);

CREATE PARTITION SCHEME ps_Year
AS PARTITION pf_Year ALL TO ([PRIMARY]);

CREATE TABLE fact_shipment (
  shipment_id BIGINT,
  year INT,
  total_cost FLOAT,
  ...
) ON ps_Year(year);
```

Partisi digunakan untuk mempercepat query terhadap data terbaru (sliding window) dan mempermudah manajemen data historis.

4.6 Data Distribution (Cloud Option)

Jika data warehouse di-deploy ke cloud seperti Azure Synapse, maka distribusi data dilakukan untuk mempercepat scaling:

```
CREATE TABLE fact_shipment
WITH (
  DISTRIBUTION = HASH(route_id),
  CLUSTERED COLUMNSTORE INDEX
```

```
)
AS SELECT * FROM staging_fact_shipment;
```

Distribusi HASH digunakan untuk menyebarkan data pengiriman antar node secara merata berdasarkan route_id.

4.7 Maintenance & Performance

Jenis	**Tujuan**
Index Rebuild	Menghilangkan fragmantasi index untuk mempercepat akses kueri
Update Statistics	Memperbarui distribusi data agar query plan tetap optimal

Contoh implementasi:

```
ALTER INDEX ALL ON fact_shipment REBUILD WITH (ONLINE = ON);
UPDATE STATISTICS fact_shipment;
```

4.8 Monitoring & Optimization

Monitoring dilakukan menggunakan **Dynamic Management Views (DMV)** di SQL Server untuk mengidentifikasi:

- Query yang boros I/O
- Index yang jarang digunakan
- Fragmentasi index
- Bottleneck pada TempDB atau join kompleks

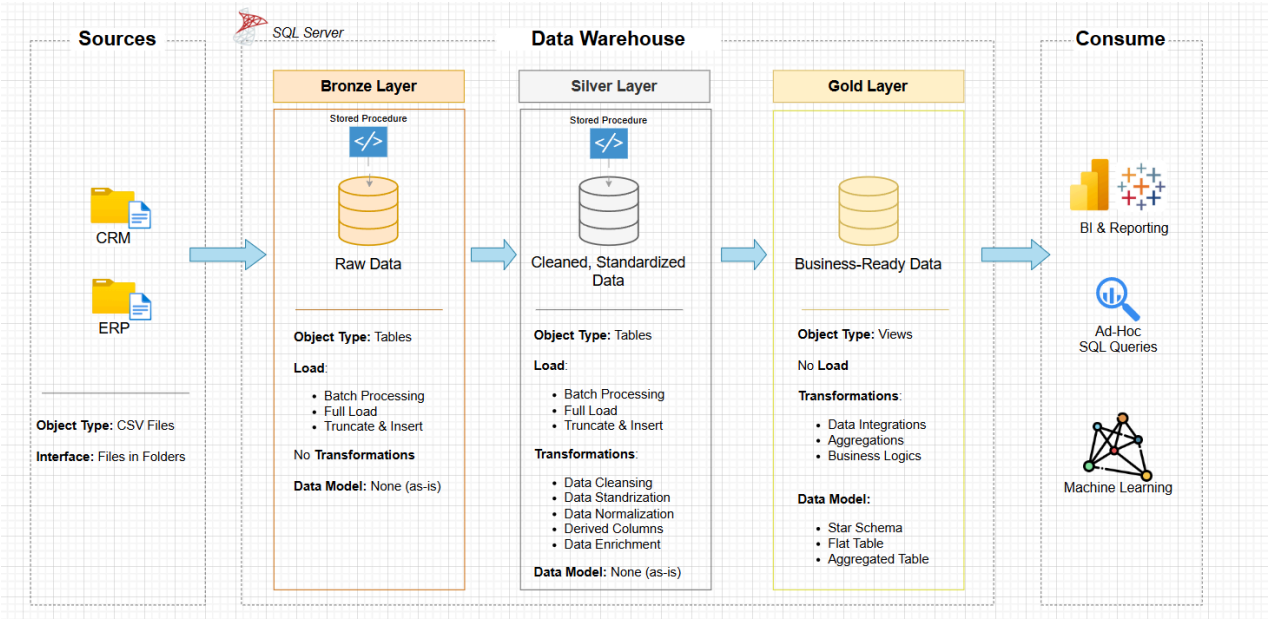
Contoh query:

```
SELECT TOP 10
qs.execution_count,
qs.total_logical_reads,
qs.total_worker_time,
SUBSTRING(qt.text,
  (qs.statement_start_offset/2)+1,
  ((CASE qs.statement_end_offset
    WHEN -1 THEN DATALENGTH(qt.text)
    ELSE qs.statement_end_offset END
```

```
    - qs.statement_start_offset)/2)+1
) AS query_text
FROM sys.dm_exec_query_stats qs
CROSS APPLY sys.dm_exec_sql_text(qs.sql_handle) qt
ORDER BY qs.total_logical_reads DESC;
```

4.9 Desain Arsitektur

Untuk arsitektur data warehouse kami menggunakan Medalian (Medallion Architecture) untuk Data Warehouse. Arsitektur ini merupakan pendekatan yang digunakan untuk mengelola alur data dalam data warehouse dengan pembagian lapisan data yang lebih jelas, yaitu Bronze Layer, Silver Layer, dan Gold Layer. Pendekatan ini membantu memproses dan mengelola data secara bertahap dan sistematis.



Berikut adalah penjelasan tentang bagaimana Arsitektur Medallion dapat diterapkan pada skema penyimpanan data Anda :

1. Bronze Layer (*Raw Data Layer*)

Deskripsi :

- Bronze Layer menyimpan data mentah (raw data) yang diimpor langsung dari sumber sistem seperti file CSV, database sumber, API, atau sumber lain. Data ini belum melalui proses pembersihan atau transformasi.

Implementasi dalam Skema Penyimpanan :

- Data dikumpulkan dalam bentuk yang apa adanya dari sistem sumber, CSV files. Dalam hal ini, data pengiriman dari file CSV diimpor ke dalam SQL Server Database.

- Contoh data mentah di Bronze Layer :
    - Data pengiriman barang dalam format CSV yang mencakup informasi seperti *shipment_id*, *route_id*, *product_id, customer_id, total_cost, dan delivery_time* dalam format asli yang mungkin berisi kesalahan, duplikasi, atau data yang belum terstandarisasi

Fungsi :

- Menyimpan data apa adanya untuk kebutuhan audit, analisis lebih lanjut, dan memastikan bahwa data yang lebih lama dapat diakses kapan saja untuk analisis lebih mendalam atau investigasi.

- Menyediakan data historis yang dapat digunakan untuk mendalami kondisi atau masalah yang terjadi di masa lalu.

2. Silver Layer (*Cleansed Data Layer*)

Deskripsi:

- Silver Layer berfokus pada pembersihan (cleansing), standarisasi, dan normalisasi data agar data tersebut lebih konsisten dan siap untuk digunakan dalam analisis.

Implementasi dalam Skema Penyimpanan :

- Di lapisan ini, data dari Bronze Layer diproses untuk memastikan kualitas data yang lebih baik. Proses yang terjadi di Silver Layer meliputi:

    - Pembersihan Data (*Data Cleaning*): Menghapus data duplikat, memperbaiki format data (misalnya, tanggal yang tidak konsisten, atau nama yang tidak terstandarisasi), dan mengatasi nilai yang hilang.

    - Normalisasi : Menyesuaikan format data seperti menggunakan satuan yang sama untuk waktu, biaya, atau berat.

    - Standarisasi: Misalnya, mengonversi nama produk atau kategori produk agar sesuai dengan standar yang ditetapkan.

Contoh Implementasi di Skema Anda:

- Mengambil data pengiriman yang ada di Bronze Layer (misalnya, *total_cost, product_id, route_id*) dan melakukan transformasi untuk menstandarisasi format nilai-nilai tersebut.
  Misalnya :
    - Menstandarkan delivery_time menjadi format YYYY-MM-DD HH:MM:SS
    - Mengubah harga menjadi nilai yang konsisten (misalnya, dalam satuan mata uang tertentu
    - Menghapus data pengiriman yang duplikat.
      Melakukan normalisasi pada product_category, route_type, dll.

Fungsi :

- Memastikan bahwa data yang disimpan di Silver Layer lebih konsisten dan dapat digunakan untuk analisis yang lebih lanjut, seperti pembuatan laporan atau analisis performa.

- Menyediakan data yang lebih bersih dan lebih terstruktur untuk pengambilan keputusan yang lebih baik.

3. Gold Layer (*Business-Ready Data Layer*)

Deskripsi :

- Gold Layer adalah tempat data yang sudah diproses dan siap digunakan untuk analisis bisnis. Data yang ada di Gold Layer sudah dimodelkan dalam struktur yang sesuai untuk pelaporan dan analitik, seperti star schema atau snowflake schema.

Implementasi dalam Skema Penyimpanan:

- Gold Layer berisi data yang telah dimodelkan untuk memberikan nilai bisnis yang maksimal. Data ini biasanya dikelompokkan dalam star schema untuk membuatnya lebih mudah dianalisis menggunakan alat BI (Business Intelligence) seperti Power BI, Tableau, atau lainnya.

Contoh Implementasi di Skema Anda:

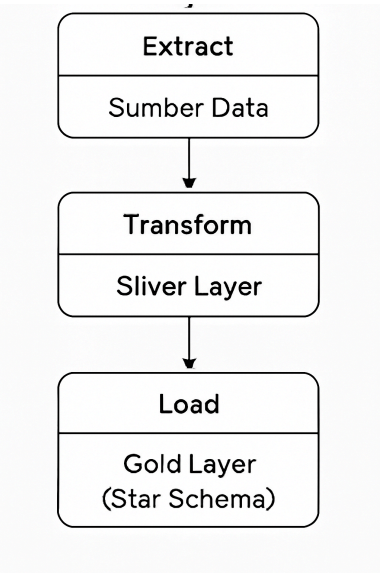
- Star Schema: Data pengiriman akan dimodelkan menggunakan tabel fakta dan dimensi.
    - Tabel Fakta (*fact_shipment*): Menyimpan informasi transaksi terkait pengiriman, misalnya *total_cost, total_delivery_time, shipment_id, product_id, route_id*, dll.
    - Tabel Dimensi:
        - dim_product: Menyimpan informasi terkait produk yang dikirim, seperti *product_id, category, weight*, dll.
        - *dim_customer*: Menyimpan informasi pelanggan, seperti *customer_id, region,* dll.
        - *dim_route*: Menyimpan informasi rute pengiriman, seperti *route_id, route_name,* dll.
        - *dim_vehicle*: Menyimpan informasi kendaraan yang digunakan dalam pengiriman, seperti *vehicle_id, fuel_type*, dll.
        - *dim_date*: Menyimpan informasi tanggal pengiriman untuk analisis berdasarkan waktu.

Fungsi :

- Memberikan data business-ready yang dapat digunakan untuk analisis dan pembuatan laporan berbasis data.

- Memungkinkan business intelligence tools untuk melakukan analisis lebih lanjut, seperti analisis performa pengiriman, analisis biaya, dan pemantauan waktu pengiriman.

4.10 ETL



Alat ETL disini menggunakan SSIS karena kita menggunakan lingkunga data warehouse di Microsoft SQL Server.

SQL Server Integration Services (SSIS) merupakan komponen utama dalam proses Extract, Transform, Load (ETL) dalam lingkungan SQL Server. Dalam proyek ini, SSIS digunakan untuk mengekstrak data dari berbagai sumber, seperti file CSV, laporan Excel dari tim lingkungan. SSIS terdiri dari dua bagian utama, yaitu Control Flow dan Data Flow. Pada Control Flow, kami menyusun urutan tugas-tugas ETL, termasuk menjalankan SQL, skrip PowerShell, atau pengiriman notifikasi. Sedangkan pada Data Flow, kami menggunakan berbagai komponen transformasi, seperti Derived Column, Lookup, dan Conditional Split, untuk membersihkan dan menggabungkan data sebelum memuatnya ke dalam data warehouse. SSIS juga mendukung penjadwalan otomatis melalui SQL Server Agent, memungkinkan proses ETL berjalan secara rutin tanpa memerlukan intervensi manual.

BAB V
Desain dan Implement Partitioned

A. Menentukan Struktur Partisi

1) Partition Function

Partition data berdasarkan kolom tanggal pengiriman (date_id yang merujuk ke dim_date.date). Misalkan ingin menyimpan hanya tiga tahun terakhir (2022, 2023, 2024) di table utama, dan archive sisanya.

```
CREATE PARTITION FUNCTION PF_ShipmentDate_RR(date)
    AS RANGE RIGHT FOR VALUES
        ('2022-01-01',    -- partisi untuk data <2022
        '2023-01-01',     -- partisi untuk data 2022
        '2024-01-01');    -- partisi untuk data 2023
```

Pada RANGE RIGHT, setiap boundary menjadi awal partisi baru.

2) Partition Scheme

Mapping partisi ke filegroup (misal: satu filegroup FG_Shipment):

```
CREATE PARTITION SCHEME PS_ShipmentDate
    AS PARTITION PF_ShipmentDate_RR
    ALL TO (FG_Shipment);
```

Atau untuk sebar merata ke beberapa filegroup (misal FG1-FG4):

```
CREATE PARTITION SCHEME PS_ShipmentDate_Multi
    AS PARTITION PF_ShipmentDate_RR
    TO (FG1, FG2, FG3, FG4);
```

B. Sliding-Window Maintenance

Untuk mempertahankan retensi hanya tiga tahun terakhir, setiap awal tahun jalankan:

1) **SWITCH** partisi tertua (partition 1) ke archive table:

```
ALTER TABLE fact_shipment
    SWITCH PARTITION 1 TO fact_shipment_archive;
```

2) **MERGE** boundary terlama:

```
ALTER PARTITION FUNCTION PF_ShipmentDate_RR()
    MERGE RANGE ('2022-01-01');
```

3) **SPLIT** range untuk tahun baru (misal 2025):

```
ALTER PARTITION FUNCTION PF_ShipmentDate_RR()
    MERGE RANGE ('2022-01-01');
```

C. Partition Elimination

Dengan desain di atas, query yang memfilter date_id otomatis hanya akan membaca partisi yang relevan:

```
SELECT COUNT(*) AS TotalDeliveries
FROM fact_shipment
WHERE date_id BETWEEN '2023-06-01' AND '2023-06-30';
```

Hanya partisi 2023 yang di-scan, partisi lain di eliminate oleh engine.