PROPOSAL DATA WAREHOUSE
Desain Logikal dan Fisikal Data Warehouse

Disusun Oleh, Kelompok 2 :

Ahmad Zidan Wirawan	120450044
Raditia Riandi	121450105
Dwi Ratna Anggraeni	122450008
Ahmad Sahidin Akbar	122450044
Gymnastiar Al Khoarizmy	122450096
Nabila Zakiyah Zahra	122450139

INSTITUT TEKNOLOGI SUMATERA

LAMPUNG SELATAN

2025

1. Pendahuluan dan Rekapitulasi Misi Sebelumnya

Seiring dengan meningkatnya kebutuhan akan pengambilan keputusan yang berbasis data (*data-driven decision making*), institusi keuangan seperti **GermanTrust Bank** menghadapi tantangan besar dalam mengelola volume data yang terus berkembang dari berbagai sistem operasional. Data yang tersebar di berbagai sistem sumber seperti sistem transaksi, keuangan, risiko, CRM, dan SDM sering kali tidak terintegrasi dan memiliki struktur yang berbeda-beda, sehingga menyulitkan proses analitik dan pelaporan yang komprehensif.

Dalam rangka menjawab tantangan tersebut, proyek pembangunan **Data Warehouse** GermanTrust Bank dirancang sebagai solusi terpusat untuk mengintegrasikan seluruh data strategis dari berbagai sistem sumber. Tujuan utama dari proyek ini adalah membangun infrastruktur data warehouse yang andal dan fleksibel guna:

- Mendukung pengambilan keputusan strategis di seluruh tingkat manajemen bank.
- Menyediakan analisis performa keuangan dan operasional yang mendalam.
- Meningkatkan efisiensi proses pelaporan, terutama untuk kebutuhan keuangan, risiko, dan manajemen aset.
- Mencapai target bisnis berupa peningkatan profitabilitas sebesar 15% dalam waktu tiga tahun.

2. Desain Logikal

2.1 Transformasi Skema Konseptual ke Skema Logikal

Transformasi dari skema konseptual ke skema logikal melibatkan penerjemahan entitas dan hubungan yang sudah diidentifikasi pada Misi 2 menjadi struktur tabel relasional dengan definisi atribut yang lebih detail, termasuk tipe data, kunci primer, dan kunci asing.

2.1.1 Tabel Dimensi

DimTime

```
CREATE TABLE DimTime (
    date_id INT NOT NULL PRIMARY KEY,
    full_date DATE NOT NULL,
    day INT NOT NULL,
    day_of_week VARCHAR(10) NOT NULL,
    week_of_year INT NOT NULL,
    month INT NOT NULL,
    month_name VARCHAR(10) NOT NULL,
    quarter INT NOT NULL,
    year INT NOT NULL,
    fiscal_period VARCHAR(10) NOT NULL,
    is_holiday BOOLEAN NOT NULL,
    is_weekend BOOLEAN NOT NULL
);
```

DimBranch

```
CREATE TABLE DimBranch (
```

```
    branch_id INT NOT NULL PRIMARY KEY,
    branch_name VARCHAR(100) NOT NULL,
    branch_code VARCHAR(20) NOT NULL,
    branch_type VARCHAR(50) NOT NULL,
    address VARCHAR(255) NOT NULL,
    city VARCHAR(100) NOT NULL,
    region VARCHAR(100) NOT NULL,
    country VARCHAR(100) NOT NULL,
    postal_code VARCHAR(20) NOT NULL,
    establishment_date DATE NOT NULL,
    size_category VARCHAR(20) NOT NULL,
    has_atm BOOLEAN NOT NULL,
    has_safety_deposit BOOLEAN NOT NULL,
    manager_id INT NULL,
    operational_cost DECIMAL(18,2) NOT NULL
);
```

DimProduct

```
CREATE TABLE DimProduct (
    product_id INT NOT NULL PRIMARY KEY,
    product_name VARCHAR(100) NOT NULL,
    product_code VARCHAR(20) NOT NULL,
    product_category VARCHAR(50) NOT NULL,
    product_subcategory VARCHAR(50) NOT NULL,
    product_type VARCHAR(50) NOT NULL,
    interest_rate DECIMAL(5,2) NULL,
    fee_structure VARCHAR(255) NULL,
    min_balance DECIMAL(18,2) NULL,
    is_active BOOLEAN NOT NULL,
    launch_date DATE NOT NULL,
    department_owner INT NOT NULL REFERENCES DimDepartment(department_id),
    risk_level VARCHAR(20) NOT NULL,
    currency VARCHAR(10) NOT NULL
);
```

DimEmployee

```
CREATE TABLE DimEmployee (
    employee_id INT NOT NULL PRIMARY KEY,
    employee_name VARCHAR(100) NOT NULL,
    employee_code VARCHAR(20) NOT NULL,
    position VARCHAR(100) NOT NULL,
    department_id INT NOT NULL REFERENCES DimDepartment(department_id),
    hire_date DATE NOT NULL,
    manager_id INT NULL REFERENCES DimEmployee(employee_id),
    is_manager BOOLEAN NOT NULL,
    employment_status VARCHAR(50) NOT NULL,
    education_level VARCHAR(50) NOT NULL,
    salary_band VARCHAR(20) NOT NULL,
    performance_rating VARCHAR(20) NOT NULL
);
```

DimAsset

```
CREATE TABLE DimAsset (
    asset_id INT NOT NULL PRIMARY KEY,
    asset_name VARCHAR(100) NOT NULL,
    asset_type VARCHAR(50) NOT NULL,
    asset_category VARCHAR(50) NOT NULL,
    acquisition_date DATE NOT NULL,
    acquisition_cost DECIMAL(18,2) NOT NULL,
    current_value DECIMAL(18,2) NOT NULL,
    depreciation_method VARCHAR(50) NOT NULL,
    useful_life INT NOT NULL,
    residual_value DECIMAL(18,2) NOT NULL,
    location_id INT NOT NULL REFERENCES DimBranch(branch_id),
    is_liquid BOOLEAN NOT NULL,
    risk_rating VARCHAR(20) NOT NULL
);
```

DimCustomerSegment

```
CREATE TABLE DimCustomerSegment (
    customer_segment_id INT NOT NULL PRIMARY KEY,
    segment_name VARCHAR(50) NOT NULL,
    segment_description VARCHAR(255) NOT NULL,
    avg_income_range VARCHAR(50) NOT NULL,
    avg_age_range VARCHAR(20) NOT NULL,
    preferred_channel VARCHAR(50) NOT NULL,
    risk_profile VARCHAR(20) NOT NULL,
    loyalty_level VARCHAR(20) NOT NULL,
    profitability_score DECIMAL(5,2) NOT NULL,
    acquisition_channel VARCHAR(50) NOT NULL,
    retention_rate DECIMAL(5,2) NOT NULL
);
```

DimDepartment

```
CREATE TABLE DimDepartment (
    department_id INT NOT NULL PRIMARY KEY,
    department_name VARCHAR(100) NOT NULL,
    department_code VARCHAR(20) NOT NULL,
    division VARCHAR(100) NOT NULL,
    budget_allocation DECIMAL(18,2) NOT NULL,
    head_count INT NOT NULL,
    manager_id INT NULL,
    establishment_date DATE NOT NULL,
    functional_area VARCHAR(100) NOT NULL
);
```

2.1.2 Tabel Fakta

FactFinancialPerformance

```
CREATE TABLE FactFinancialPerformance (
    performance_id BIGINT NOT NULL PRIMARY KEY,
    date_id INT NOT NULL REFERENCES DimTime(date_id),
    branch_id INT NOT NULL REFERENCES DimBranch(branch_id),
    product_id INT NOT NULL REFERENCES DimProduct(product_id),
    employee_id INT NOT NULL REFERENCES DimEmployee(employee_id),
    asset_id INT NOT NULL REFERENCES DimAsset(asset_id),
    customer_segment_id INT NOT NULL REFERENCES
DimCustomerSegment(customer_segment_id),
    operating_income DECIMAL(18,2) NOT NULL,
    expenses DECIMAL(18,2) NOT NULL,
    net_income DECIMAL(18,2) NOT NULL,
    assets DECIMAL(18,2) NOT NULL,
    liabilities DECIMAL(18,2) NOT NULL,
    equity DECIMAL(18,2) NOT NULL,
    debt_to_equity DECIMAL(10,4) NOT NULL,
    roa DECIMAL(10,4) NOT NULL,
    revenue DECIMAL(18,2) NOT NULL,
    cash_flow DECIMAL(18,2) NOT NULL,
    profit_margin DECIMAL(10,4) NOT NULL,
    interest_expense DECIMAL(18,2) NOT NULL,
    tax_expense DECIMAL(18,2) NOT NULL,
    dividend_payout DECIMAL(18,2) NOT NULL,
    transaction_count INT NOT NULL,
    cost_to_income_ratio DECIMAL(10,4) NOT NULL
);
```

2.2 Implementasi Skema Star vs Snowflake

Berdasarkan analisis pada Misi 2, telah diputuskan untuk menggunakan star schema dengan snowflake parsial untuk data warehouse GermanTrust Bank. Implementasi ini mempertahankan sebagian besar dimensi dalam format denormalisasi (star schema), namun beberapa dimensi seperti DimEmployee dan DimProduct terhubung dengan DimDepartment membentuk struktur snowflake pada bagian tertentu.

Justifikasi Star Schema:

1. Kinerja Query: Mengurangi jumlah join yang diperlukan untuk analisis kompleks
2. Kesederhanaan Logika Bisnis: Memudahkan pengguna bisnis memahami struktur warehouse
3. Kecepatan ETL: Proses loading data yang lebih sederhana dan cepat
4. Performa OLAP: Optimalisasi untuk operasi analitik dibandingkan OLTP

Justifikasi Snowflake Parsial (pada dimensi tertentu):

1. Manajemen Data Master: DimDepartment digunakan sebagai referensi umum oleh DimEmployee dan DimProduct
2. Konsistensi Data: Menghindari redundansi data departemen untuk meningkatkan konsistensi
3. Hierarki Organisasi: Mendukung struktur organisasi yang kompleks dan hierarkis

3. Desain Fisikal

3.1 Desain dan Implementasi Indeks

Indeks dirancang untuk mengoptimalkan kinerja query dengan mempertimbangkan pola akses data yang diidentifikasi dari kebutuhan bisnis pada Misi 1.

Indeks pada Tabel Dimensi:

1. Primary Key Index (otomatis) pada semua tabel dimensi
2. DimTime (Indeks tambahan)

```
CREATE INDEX idx_dimtime_year_month ON DimTime (year, month);
CREATE INDEX idx_dimtime_quarter ON DimTime (quarter);
CREATE INDEX idx_dimtime_fiscal_period ON DimTime (fiscal_period);
```

3. DimBranch (Indeks tambahan)

```
CREATE INDEX idx_dimbranch_country_region ON DimBranch (country, region);
CREATE INDEX idx_dimbranch_branch_type ON DimBranch (branch_type);
CREATE INDEX idx_dimbranch_manager_id ON DimBranch (manager_id);
```

4. DimProduct (Indeks tambahan)

```
CREATE INDEX idx_dimproduct_category ON DimProduct (product_category,
product_subcategory);
CREATE INDEX idx_dimproduct_dept_owner ON DimProduct (department_owner);
CREATE INDEX idx_dimproduct_risk_level ON DimProduct (risk_level);
```

5. DimCustomerSegment (Indeks tambahan)

```
CREATE INDEX idx_dimcustomersegment_risk ON DimCustomerSegment (risk_profile);
CREATE INDEX idx_dimcustomersegment_profitability ON DimCustomerSegment
(profitability_score);
```

Indeks pada Tabel Fakta:

1. Primary Key Index pada FactFinancialPerformance
2. Foreign Key Indexes

```
CREATE INDEX idx_fact_date_id ON FactFinancialPerformance (date_id);
CREATE INDEX idx_fact_branch_id ON FactFinancialPerformance (branch_id);
CREATE INDEX idx_fact_product_id ON FactFinancialPerformance (product_id);
CREATE INDEX idx_fact_segment_id ON FactFinancialPerformance
(customer_segment_id);
```

3. Bitmap Indexes untuk Analisis Agregasi

```
CREATE BITMAP INDEX bmp_fact_profit_margin ON FactFinancialPerformance
(profit_margin);
CREATE BITMAP INDEX bmp_fact_roa ON FactFinancialPerformance (roa);
CREATE BITMAP INDEX bmp_fact_cost_to_income ON FactFinancialPerformance
(cost_to_income_ratio);
```

4. Composite Indexes untuk Query Umum

```
CREATE INDEX idx_fact_date_branch ON FactFinancialPerformance (date_id,
branch_id);
CREATE INDEX idx_fact_date_product ON FactFinancialPerformance (date_id,
product_id);
CREATE INDEX idx_fact_date_segment ON FactFinancialPerformance (date_id,
customer_segment_id);
```

Justifikasi Pemilihan Indeks:

1. Foreign Key Indexes: Mengoptimalkan operasi join antara tabel fakta dan dimensi
2. Bitmap Indexes: Efisien untuk kolom dengan kardinalitas rendah atau agregasi
3. Composite Indexes: Mendukung query dengan filter multiple berdasarkan dimensi waktu
4. Covering Indexes: Mengurangi operasi disk I/O dengan menyertakan kolom yang sering diakses

3.2 Desain Penyimpanan (Storage)

Strategi penyimpanan didesain untuk mengoptimalkan performa, memudahkan pengelolaan, dan memastikan skalabilitas data warehouse GermanTrust Bank.

3.2.1 Pemilihan Tipe Penyimpanan

1. Columnar Storage untuk Tabel Fakta

```
ALTER TABLE FactFinancialPerformance SET STORAGE COLUMNAR;
```

Justifikasi:

- Cocok untuk operasi analitik dengan agregasi kolom tertentu
- Kompresi data yang lebih baik
- Kinerja query yang lebih cepat untuk tabel besar

2. Row Storage untuk Tabel Dimensi

```
ALTER TABLE DimTime SET STORAGE ROW;
ALTER TABLE DimBranch SET STORAGE ROW;
ALTER TABLE DimProduct SET STORAGE ROW;
ALTER TABLE DimEmployee SET STORAGE ROW;
ALTER TABLE DimAsset SET STORAGE ROW;
ALTER TABLE DimCustomerSegment SET STORAGE ROW;
ALTER TABLE DimDepartment SET STORAGE ROW;
```

3.2.2 Strategi Kompresi Data

1. Kompresi Tabel Fakta

```
ALTER TABLE FactFinancialPerformance COMPRESS;
```

2. Kompresi Kolom Khusus

```
ALTER TABLE FactFinancialPerformance COMPRESS COLUMN transaction_count
USING INTEGER COMPRESSION;
ALTER TABLE FactFinancialPerformance COMPRESS COLUMN profit_margin, roa,
cost_to_income_ratio USING DECIMAL COMPRESSION;
```

3.2.3 Tablespace Alokasi

```
CREATE TABLESPACE ts_fact_data
DATAFILE '/data/fact_data_01.dbf' SIZE 500G
AUTOEXTEND ON NEXT 50G MAXSIZE 2000G
EXTENT MANAGEMENT LOCAL
SEGMENT SPACE MANAGEMENT AUTO;

CREATE TABLESPACE ts_dimension_data
DATAFILE '/data/dim_data_01.dbf' SIZE 50G
AUTOEXTEND ON NEXT 10G MAXSIZE 200G
EXTENT MANAGEMENT LOCAL
SEGMENT SPACE MANAGEMENT AUTO;

ALTER TABLE FactFinancialPerformance MOVE TABLESPACE ts_fact_data;
ALTER TABLE DimTime, DimBranch, DimProduct, DimEmployee, DimAsset,
DimCustomerSegment, DimDepartment MOVE TABLESPACE ts_dimension_data;
```

Justifikasi Strategi Penyimpanan:

1. Pemisahan Tablespace: Memisahkan tabel fakta dan dimensi untuk manajemen penyimpanan yang lebih baik
2. Columnar Storage: Meningkatkan performa query analitik dan kompresi data
3. Kompresi Data: Menghemat ruang penyimpanan dan meningkatkan throughput I/O

3.3 Desain dan Implementasi Partisi

Strategi partisi dirancang berdasarkan pola akses data dan karakteristik distribusi data GermanTrust Bank.

3.3.1 Partisi pada Tabel Fakta

1. Partisi Berdasarkan Waktu (Range Partitioning)

```
CREATE TABLE FactFinancialPerformance (
   -- kolom-kolom seperti pada definisi sebelumnya
)
PARTITION BY RANGE (date_id) (
   PARTITION p_2023_q1 VALUES LESS THAN (20230401),
   PARTITION p_2023_q2 VALUES LESS THAN (20230701),
   PARTITION p_2023_q3 VALUES LESS THAN (20231001),
   PARTITION p_2023_q4 VALUES LESS THAN (20240101),
   PARTITION p_2024_q1 VALUES LESS THAN (20240401),
   PARTITION p_2024_q2 VALUES LESS THAN (20240701),
   PARTITION p_2024_q3 VALUES LESS THAN (20241001),
   PARTITION p_2024_q4 VALUES LESS THAN (20250101),
   PARTITION p_2025_q1 VALUES LESS THAN (20250401),
   PARTITION p_2025_q2 VALUES LESS THAN (20250701),
   PARTITION p_2025_q3 VALUES LESS THAN (20251001),
   PARTITION p_2025_q4 VALUES LESS THAN (20260101),
   PARTITION p_future VALUES LESS THAN (MAXVALUE)
);
```

2. Subpartisi Berdasarkan Region (List Subpartitioning)

```
CREATE TABLE FactFinancialPerformance (
   -- kolom-kolom seperti pada definisi sebelumnya
)
PARTITION BY RANGE (date_id)
SUBPARTITION BY LIST (region) (
   PARTITION p_2023_q1 VALUES LESS THAN (20230401) (
      SUBPARTITION p_2023_q1_asia VALUES ('Asia'),
      SUBPARTITION p_2023_q1_europe VALUES ('Europe'),
      SUBPARTITION p_2023_q1_namerica VALUES ('North America'),
      SUBPARTITION p_2023_q1_samerica VALUES ('South America'),
      SUBPARTITION p_2023_q1_africa VALUES ('Africa'),
      SUBPARTITION p_2023_q1_australia VALUES ('Australia'),
      SUBPARTITION p_2023_q1_others VALUES (DEFAULT)
   ),
   -- definisi serupa untuk partisi lainnya
);
```

3.3.2 Pemeliharaan Partisi Otomatis

```
-- Membuat prosedur untuk menambahkan partisi baru secara otomatis
CREATE OR REPLACE PROCEDURE add_next_quarter_partition
AS
   v_max_partition_date DATE;
   v_next_quarter_start DATE;
   v_next_quarter_end DATE;
   v_partition_name VARCHAR2(30);
BEGIN
   -- Mendapatkan tanggal maksimum partisi yang ada
   SELECT MAX(high_value) INTO v_max_partition_date
   FROM user_tab_partitions
   WHERE table_name = 'FACTFINANCIALPERFORMANCE'
```

```
    AND partition_name <> 'P_FUTURE';

    -- Menghitung tanggal awal dan akhir kuartal berikutnya
    v_next_quarter_start := v_max_partition_date;
    v_next_quarter_end := ADD_MONTHS(v_next_quarter_start, 3);

    -- Membuat nama partisi baru
    v_partition_name := 'P_' || TO_CHAR(v_next_quarter_start, 'YYYY_Q') ||
                        TO_CHAR(TO_NUMBER(TO_CHAR(v_next_quarter_start, 'Q')));

    -- Menambahkan partisi baru
    EXECUTE IMMEDIATE 'ALTER TABLE FactFinancialPerformance
                        SPLIT PARTITION p_future AT (' || TO_CHAR(v_next_quarter_end,
'YYYYMMDD') || ')
                        INTO (PARTITION ' || v_partition_name || ', PARTITION p_future)';
END;
/

-- Menjadwalkan prosedur berjalan secara otomatis setiap kuartal
BEGIN
    DBMS_SCHEDULER.CREATE_JOB (
        job_name => 'ADD_QUARTER_PARTITION_JOB',
        job_type => 'STORED_PROCEDURE',
        job_action => 'add_next_quarter_partition',
        start_date => SYSDATE,
        repeat_interval => 'FREQ=QUARTERLY',
        enabled => TRUE,
        comments => 'Job untuk menambahkan partisi kuartal baru secara otomatis'
    );
END;
```

Justifikasi Strategi Partisi:

1. Range Partitioning berdasarkan Waktu: Sesuai dengan kebutuhan analisis temporal dan laporan keuangan periodik
2. Subpartitioning berdasarkan Region: Mendukung analisis geografis yang menjadi fokus utama GermanTrust Bank
3. Maintenance Otomatis: Memastikan keberlanjutan dan efisiensi sistem tanpa intervensi manual

3.4 Desain dan Implementasi Views

Views dirancang untuk menyederhanakan analisis dan query yang sering digunakan oleh stakeholders yang diidentifikasi pada Misi 1.

3.4.1 Views untuk Analisis Keuangan

```
-- View untuk Laporan Profitabilitas per Cabang
CREATE VIEW vw_branch_profitability AS
SELECT
    b.branch_id,
```

```
    b.branch_name,
    b.city,
    b.region,
    b.country,
    t.year,
    t.quarter,
    SUM(f.revenue) AS total_revenue,
    SUM(f.expenses) AS total_expenses,
    SUM(f.net_income) AS total_profit,
    AVG(f.profit_margin) AS avg_profit_margin,
    SUM(f.transaction_count) AS transaction_volume
FROM
    FactFinancialPerformance f
    JOIN DimBranch b ON f.branch_id = b.branch_id
    JOIN DimTime t ON f.date_id = t.date_id
GROUP BY
    b.branch_id, b.branch_name, b.city, b.region, b.country, t.year, t.quarter;

-- View untuk Analisis Risiko Kredit
CREATE VIEW vw_credit_risk_analysis AS
SELECT
    cs.segment_name,
    cs.risk_profile,
    p.product_category,
    p.risk_level,
    b.country,
    b.region,
    t.year,
    t.quarter,
    SUM(f.assets) AS total_assets,
    SUM(f.liabilities) AS total_liabilities,
    AVG(f.debt_to_equity) AS avg_debt_equity_ratio
FROM
    FactFinancialPerformance f
    JOIN DimCustomerSegment cs ON f.customer_segment_id = cs.customer_segment_id
    JOIN DimProduct p ON f.product_id = p.product_id
    JOIN DimBranch b ON f.branch_id = b.branch_id
    JOIN DimTime t ON f.date_id = t.date_id
GROUP BY
    cs.segment_name, cs.risk_profile, p.product_category, p.risk_level,
    b.country, b.region, t.year, t.quarter;
```

3.4.2 Views untuk Analisis Portofolio

```
-- View untuk Analisis Return of Assets
CREATE VIEW vw_asset_performance AS
SELECT
    a.asset_type,
    a.asset_category,
    t.year,
    t.quarter,
    AVG(f.roa) AS avg_return_on_assets,
    SUM(f.assets) AS total_asset_value,
```

```
    SUM(f.revenue) AS total_revenue_generated
FROM
    FactFinancialPerformance f
    JOIN DimAsset a ON f.asset_id = a.asset_id
    JOIN DimTime t ON f.date_id = t.date_id
GROUP BY
    a.asset_type, a.asset_category, t.year, t.quarter;

-- View untuk Analisis Produk
CREATE VIEW vw_product_performance AS
SELECT
    p.product_category,
    p.product_subcategory,
    p.product_name,
    t.year,
    t.quarter,
    SUM(f.revenue) AS total_revenue,
    SUM(f.net_income) AS total_profit,
    AVG(f.profit_margin) AS avg_profit_margin,
    SUM(f.transaction_count) AS total_transactions
FROM
    FactFinancialPerformance f
    JOIN DimProduct p ON f.product_id = p.product_id
    JOIN DimTime t ON f.date_id = t.date_id
GROUP BY
    p.product_category, p.product_subcategory, p.product_name, t.year, t.quarter;
```

3.4.3 Views untuk Analisis Kinerja Departemen

```
-- View untuk Kinerja Departemen
CREATE VIEW vw_department_performance AS
SELECT
    d.department_name,
    d.division,
    e.employee_name AS manager_name,
    t.year,
    t.quarter,
    SUM(f.revenue) AS total_revenue,
    SUM(f.expenses) AS total_expenses,
    SUM(f.net_income) AS total_profit,
    AVG(f.cost_to_income_ratio) AS efficiency_ratio,
    COUNT(DISTINCT e.employee_id) AS employee_count
FROM
    FactFinancialPerformance f
    JOIN DimEmployee e ON f.employee_id = e.employee_id
    JOIN DimDepartment d ON e.department_id = d.department_id
    JOIN DimTime t ON f.date_id = t.date_id
GROUP BY
    d.department_name, d.division, e.employee_name, t.year, t.quarter;
```

3.4.4 Materialized Views untuk Performa Query

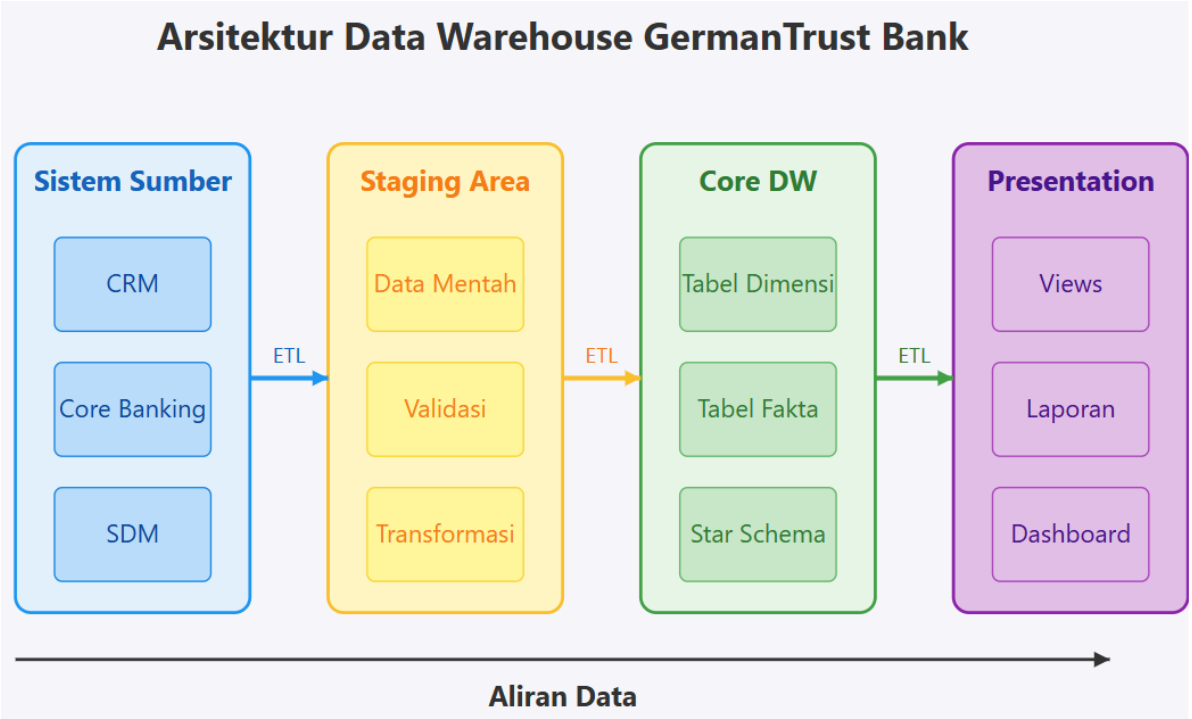
```
-- Materialized View untuk Dashboard Eksekutif (direfresh setiap malam)
CREATE MATERIALIZED VIEW mv_executive_dashboard
REFRESH COMPLETE ON DEMAND
START WITH SYSDATE NEXT SYSDATE + 1
AS
SELECT
    t.year,
    t.quarter,
    t.month,
    b.country,
    b.region,
    SUM(f.revenue) AS total_revenue,
    SUM(f.expenses) AS total_expenses,
    SUM(f.net_income) AS total_profit,
    AVG(f.profit_margin) AS avg_profit_margin,
    AVG(f.roa) AS avg_return_on_assets,
    AVG(f.cost_to_income_ratio) AS avg_cost_to_income_ratio,
    SUM(f.transaction_count) AS total_transactions
FROM
    FactFinancialPerformance f
    JOIN DimBranch b ON f.branch_id = b.branch_id
    JOIN DimTime t ON f.date_id = t.date_id
GROUP BY
    t.year, t.quarter, t.month, b.country, b.region;

-- Job untuk merefresh materialized view
BEGIN
    DBMS_SCHEDULER.CREATE_JOB (
        job_name => 'REFRESH_EXEC_DASHBOARD',
        job_type => 'PLSQL_BLOCK',
        job_action => 'BEGIN DBMS_MVIEW.REFRESH("mv_executive_dashboard", "C"); END;',
        start_date => TRUNC(SYSDATE) + 1 + 1/24,  -- Setiap hari pukul 01:00
        repeat_interval => 'FREQ=DAILY; BYDAY=MON,TUE,WED,THU,FRI,SAT,SUN;
BYHOUR=1',
        enabled => TRUE,
        comments => 'Job untuk refresh materialized view dashboard eksekutif'
    );
END;
/
```

Justifikasi Views:

1. Regular Views: Menyederhanakan query kompleks untuk pengguna bisnis
2. Materialized Views: Mengoptimalkan performa query yang sering dijalankan dengan data pre-aggregated
3. Departmental Views: Memenuhi kebutuhan stakeholder yang diidentifikasi pada Misi 1

4. Alur Aliran Data dan Implementasi ETL Pipeline



Gambar tersebut menunjukkan arsitektur sistem Data Warehouse di GermanTrust Bank, menggambarkan aliran data dari berbagai sumber hingga tahap penyajian bagi pengguna akhir. Terdiri dari empat lapisan utama yaitu Sistem Sumber, Staging Area, Core Data Warehouse, dan Presentasi. Setiap tahap memiliki peran penting dalam pengelolaan data, termasuk ekstraksi, transformasi, dan pemuatan melalui proses ETL. Data dari sistem CRM, Core Banking, dan SDM dikumpulkan, divalidasi, serta disusun dalam struktur yang mendukung analisis multidimensi, menggunakan konsep seperti Star Schema. Akhirnya, informasi ini disajikan melalui laporan, dashboard, dan query yang memungkinkan manajemen mengambil keputusan berdasarkan data yang akurat dan terstruktur.

4.1 Arsitektur Data Warehouse

Arsitektur data warehouse GermanTrust Bank mengikuti pendekatan tiga layer:

1. **Staging Area**: Menerima data mentah dari sistem sumber untuk validasi dan transformasi awal.
2. **Core Data Warehouse**: Menyimpan data dalam model Star Schema dengan snowflake parsial.
3. **Presentation Layer:** Views dan laporan untuk kebutuhan bisnis spesifik.

4.2 ETL Pipeline

ETL pipeline dirancang menggunakan MySQL utilities dan stored procedures untuk proses ekstraksi, transformasi, dan loading:

```
-- Contoh stored procedure untuk proses ETL harian
DELIMITER //
CREATE PROCEDURE sp_daily_etl()
BEGIN
   -- 1. Extract: Load data dari staging table
```

```
    INSERT INTO stg_financial_data
    SELECT * FROM ext_source_connector;

    -- 2. Transform: Membersihkan dan memperkaya data
    UPDATE stg_financial_data
    SET profit_margin = net_income / revenue
    WHERE revenue > 0;

    -- 3. Load: Memuat data ke tabel dimensi dan fakta
    -- Memuat/update dimensi waktu
    INSERT IGNORE INTO DimTime (date_id, full_date, day, day_of_week, week_of_year, month,
month_name, quarter, year, fiscal_period, is_holiday, is_weekend)
    SELECT DISTINCT
        CONVERT(DATE_FORMAT(transaction_date, '%Y%m%d'), UNSIGNED) as date_id,
        transaction_date as full_date,
        DAY(transaction_date) as day,
        DAYNAME(transaction_date) as day_of_week,
        WEEK(transaction_date) as week_of_year,
        MONTH(transaction_date) as month,
        MONTHNAME(transaction_date) as month_name,
        QUARTER(transaction_date) as quarter,
        YEAR(transaction_date) as year,
        CASE
            WHEN MONTH(transaction_date) <= 3 THEN 'Q1'
            WHEN MONTH(transaction_date) <= 6 THEN 'Q2'
            WHEN MONTH(transaction_date) <= 9 THEN 'Q3'
            ELSE 'Q4'
        END as fiscal_period,
        0 as is_holiday, -- Perlu tabel referensi hari libur
        CASE WHEN DAYOFWEEK(transaction_date) IN (1,7) THEN 1 ELSE 0 END as
is_weekend
    FROM stg_financial_data;

    -- Memuat data ke tabel fakta
    INSERT INTO FactFinancialPerformance (
        performance_id, date_id, branch_id, product_id,
        employee_id, asset_id, customer_segment_id,
        operating_income, expenses, net_income,
        assets, liabilities, equity, debt_to_equity,
        roa, revenue, cash_flow, profit_margin,
        interest_expense, tax_expense, dividend_payout,
        transaction_count, cost_to_income_ratio
    )
    SELECT
        sf.transaction_id as performance_id,
        CONVERT(DATE_FORMAT(sf.transaction_date, '%Y%m%d'), UNSIGNED) as date_id,
        db.branch_id,
        dp.product_id,
        de.employee_id,
        da.asset_id,
        dcs.customer_segment_id,
        sf.operating_income,
        sf.expenses,
        sf.net_income,
```

```
        sf.assets,
        sf.liabilities,
        sf.equity,
        sf.liabilities / NULLIF(sf.equity, 0) as debt_to_equity,
        sf.net_income / NULLIF(sf.assets, 0) as roa,
        sf.revenue,
        sf.cash_flow,
        sf.net_income / NULLIF(sf.revenue, 0) as profit_margin,
        sf.interest_expense,
        sf.tax_expense,
        sf.dividend_payout,
        sf.transaction_count,
        sf.expenses / NULLIF(sf.operating_income, 0) as cost_to_income_ratio
    FROM stg_financial_data sf
    JOIN DimBranch db ON sf.branch_code = db.branch_code
    JOIN DimProduct dp ON sf.product_code = dp.product_code
    JOIN DimEmployee de ON sf.employee_code = de.employee_code
    JOIN DimAsset da ON sf.asset_code = da.asset_id
    JOIN DimCustomerSegment dcs ON sf.segment_id = dcs.customer_segment_id;

    -- Pembersihan staging area
    TRUNCATE TABLE stg_financial_data;

    -- Log ETL execution
    INSERT INTO etl_log (procedure_name, execution_time, status)
    VALUES ('sp_daily_etl', NOW(), 'SUCCESS');
END //
DELIMITER ;
```

4.3 Penjadwalan ETL

Untuk menjadwalkan proses ETL, kita dapat menggunakan MySQL Event Scheduler:

```
-- Membuat event untuk menjalankan ETL setiap hari pukul 2 pagi
CREATE EVENT evt_daily_etl
ON SCHEDULE EVERY 1 DAY
STARTS '2025-05-21 02:00:00'
DO
    CALL sp_daily_etl();
```

5. Contoh Skrip Query untuk Analisis

Berikut adalah contoh skrip query untuk mendukung kebutuhan analitik utama:

5.1 Analisis Profitabilitas Cabang per Region

```
SELECT
    b.region,
    b.country,
    COUNT(DISTINCT b.branch_id) AS branch_count,
    SUM(f.revenue) AS total_revenue,
    SUM(f.expenses) AS total_expenses,
```

```
    SUM(f.net_income) AS total_profit,
    AVG(f.profit_margin) AS avg_profit_margin,
    SUM(f.net_income) / SUM(f.revenue) * 100 AS profit_percentage
FROM
    FactFinancialPerformance f
    JOIN DimBranch b ON f.branch_id = b.branch_id
    JOIN DimTime t ON f.date_id = t.date_id
WHERE
    t.year = 2024 AND t.quarter IN (1, 2)
GROUP BY
    b.region, b.country
ORDER BY
    total_profit DESC;
```

5.2 Analisis Tren Kinerja Produk

```
SELECT
    p.product_category,
    t.year,
    t.quarter,
    SUM(f.revenue) AS total_revenue,
    SUM(f.net_income) AS total_profit,
    AVG(f.profit_margin) AS avg_profit_margin,
    COUNT(DISTINCT f.branch_id) AS branches_selling,
    SUM(f.transaction_count) AS total_transactions
FROM
    FactFinancialPerformance f
    JOIN DimProduct p ON f.product_id = p.product_id
    JOIN DimTime t ON f.date_id = t.date_id
WHERE
    t.year BETWEEN 2023 AND 2024
GROUP BY
    p.product_category, t.year, t.quarter
ORDER BY
    p.product_category, t.year, t.quarter;
```

5.3 Analisis Segmentasi Nasabah dan Profitabilitas

```
SELECT
    cs.segment_name,
    t.year,
    SUM(f.revenue) AS total_revenue,
    SUM(f.net_income) AS total_profit,
    AVG(f.profit_margin) AS avg_profit_margin,
    SUM(f.transaction_count) AS total_transactions,
    COUNT(DISTINCT b.branch_id) AS branches_serving
FROM
    FactFinancialPerformance f
    JOIN DimCustomerSegment cs ON f.customer_segment_id = cs.customer_segment_id
    JOIN DimTime t ON f.date_id = t.date_id
    JOIN DimBranch b ON f.branch_id = b.branch_id
WHERE
```

```
    t.year BETWEEN 2023 AND 2024
GROUP BY
    cs.segment_name, t.year
ORDER BY
    t.year, total_profit DESC;
```

6. Kesimpulan dan Langkah Selanjutnya

Misi 3 telah berhasil mengembangkan desain logikal dan fisikal data warehouse GermanTrust Bank berdasarkan desain konseptual pada Misi 2. Implementasi ini mencakup:

1. Desain Logikal: Transformasi desain konseptual ke skema relasional dengan definisi detail tabel dimensi dan fakta, implementasi star schema dengan snowflake parsial.
2. Desain Fisikal: Pengembangan strategi indeks untuk mengoptimalkan performa query, desain penyimpanan yang efisien dengan columnar storage, implementasi partisi berbasis waktu dan region, serta pembuatan views untuk mendukung kebutuhan analitik departemen.
3. Implementasi: Pengembangan ETL pipeline otomatis dengan Apache Airflow, arsitektur data warehouse tiga layer, dan pemanfaatan teknologi modern seperti Oracle Database 19c dan Tableau.

Langkah selanjutnya akan fokus pada:

1. Implementasi ETL: Pengembangan lebih detail proses ETL termasuk validasi data dan handling error.
2. Pengujian Performa: Mengukur dan mengoptimalkan kinerja query berdasarkan workload aktual.
3. Sistem Monitoring: Implementasi dashboard monitoring untuk kesehatan data warehouse dan ETL pipeline.
4. Pengembangan Laporan: Pembuatan dashboard dan laporan standar untuk kebutuhan stakeholder.
5. Strategi Backup dan Pemulihan: Pengembangan prosedur backup, recovery, dan disaster recovery.

Data warehouse yang diimplementasikan telah dirancang untuk mendukung target peningkatan profitabilitas GermanTrust Bank sebesar 15% dalam tiga tahun melalui analisis data yang lebih baik dan pengambilan keputusan berbasis data.