BAB 3
PELAKSANAAN KERJA MAGANG

3.1 Kedudukan dan Koordinasi

Pekerjaan Freelance di PT. Devoteam memiliki peran dan tanggung jawab utama dalam pengembangan data pipeline dan pengelolaan data untuk mendukung transformasi digital klien.

3.1.1 Kedudukan

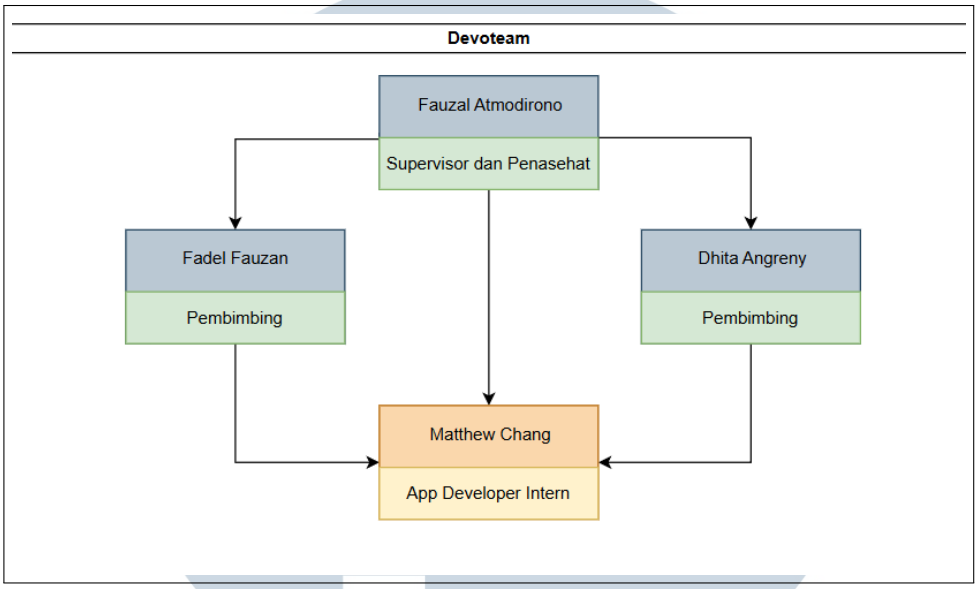
Posisi yang ditempati adalah *Data Engineer Freelance* yang tergabung dalam Tribe *Data Engineering* di PT. Devoteam. Fokus utama penempatan ini adalah pada pengembangan dan implementasi solusi data berbasis cloud. Dalam proyek migrasi *Enterprise Data Platform* (EDP) klien CIMB, tanggung jawab utama yang dimiliki tim ini mencakup perancangan dan pengembangan data pipeline, migrasi dan transformasi data dari sistem on-premises ke platform cloud, serta penerapan tata kelola data, termasuk data masking menggunakan *Policy Tags*. Teknologi utama yang digunakan dalam posisi ini adalah *Google BigQuery* sebagai data *warehouse* tujuan dan *Dataform* sebagai tool untuk pengembangan dan orkestrasi pipeline data.

3.1.2 Koordinasi

Pelaksanaan pekerjaan Freelance dijalankan di bawah struktur pengawasan dan pembimbingan yang terstruktur. Pengawasan dan penasehatan lapangan dilakukan oleh Bapak Fauzal Atmodirono sebagai *Tribe Lead Data* dan *AI/ML*, sementara arahan teknis harian dalam pengembangan data pipeline menjadi tanggung jawab Pembimbing Teknis, yaitu Bapak Fadel Fauzan dan Ibu Dhita Angreny.

Koordinasi selama pekerjaan Freelance di PT. Devoteam dilakukan secara fleksibel karena penerapan sistem kerja *hybrid*. Saat bekerja di kantor, koordinasi langsung dengan tim dan pertemuan mingguan diadakan untuk membahas progres proyek. Ketika bekerja secara *remote*, komunikasi dilakukan melalui *google chat*. Evaluasi hasil pekerjaan dilakukan secara rutin oleh pembimbing dan tim, yang kemudian memberikan masukan untuk perbaikan. Selain itu, keaktifan

dalam diskusi teknis juga dilakukan untuk memastikan bahwa pengembangan data pipeline tetap sesuai dengan standar dan kebutuhan perusahaan.



Gambar 3.1. Struktur Supervisi dan Bimbingan Magang

3.2 Tugas yang Dilakukan

Keterlibatan dalam proyek migrasi *Enterprise Data Platform* (EDP) klien CIMB menuju *Google BigQuery* dilakukan sebagai bagian dari pelaksanaan pekerjaan Freelance di PT. Devoteam. Adapun tugas yang dilakukan dalam proyek strategis ini adalah sebagai berikut:

- **Pengembangan dan Refactoring Data Pipeline:** Berfokus pada pemrosesan data dari sistem on-premises menuju lingkungan cloud-native *Google BigQuery*. Ini melibatkan perancangan dan refactoring skrip transformasi data pada layer Silver dan Gold Collection.
- **Pemetaan Skema Data (*Schema Mapping*):** Melakukan analisis mendalam terhadap skema data sumber (on-premises), menentukan pemetaan kolom, dan menetapkan tipe data *BigQuery* yang optimal (seperti penggunaan tipe data timestamp dan integer) untuk skema target di layer Silver dan Gold.
- **Penggunaan *Dataform*:** Digunakan untuk merancang dan merefaktor skrip transformasi data, termasuk konversi berkas SQL, penggunaan *Common Table Expression* (CTE) untuk modularitas kode.

- **Validasi table:** Fokus pada validasi data yang dihasilkan antara hasil transformasi di *BigQuery* dan data sumber (MIS). Validasi dilakukan dengan memastikan kesamaan 100% jumlah baris antara hasil transformasi dan data sumber.
- **Validasi Kolom:** Dilakukan berdasarkan dua kriteria, yaitu data numerik harus 100% sama antara data sumber dan hasil transformasi, sementara data bertipe string diperbolehkan memiliki perbedaan hingga batas toleransi 5% sebelum dianggap mismatch.
- **Penerapan Tata Kelola Data:** Implementasi *Policy Tags* pada kolom sensitif di tabel Gold untuk memastikan kepatuhan terhadap kebijakan privasi dan keamanan data.

3.3 Uraian Pelaksanaan Magang

Berikut adalah uraian mengenai pelaksanaan kerja magang yang telah dilakukan. Pelaksanaan kerja magang diuraikan seperti pada Tabel 3.1.

Tabel 3.1. Pekerjaan yang dilakukan tiap minggu selama pelaksanaan kerja magang

Minggu Ke -	Pekerjaan yang dilakukan
1	Onboarding, setup environment, Merapihkan script lewat github, memastikan di semua table Gold sudah terimplemen *Policy Tags* yg sesuai, *update* filter pada silver dan gold table, dan mengubah penulisan format *dataform*.
2	*update* script gold yang memiliki source silver banyak dijadikan CTE, dan *update* mappingan di sheets *UAT Progress*.
3	*update* table gold, menentukan fixing category untuk table gold pada sheets *SIT UAT Progress*, mengubah nama table di *dataform* sesuai dengan config name pada gold, dan mengcapture ssis.
4	Cek count rows pada gold collection dan NDB, Fixing table gold dan collection yang memiliki rows berbeda dengan MIS.
5	Membuat SCD untuk table silver di NDB dan fixing gold NDB.
6	Fixing gold NDB yang memiliki rows tidak match dengan MIS.
7	Melanjutkan Fixing gold NDB dan Collection yang tidak match dengan MIS.

Minggu Ke -	Pekerjaan yang dilakukan
8	Melanjutkan Fixing gold Collection yang tidak match dengan MIS.
9	Melanjutkan Fixing gold Collection yang tidak match dengan MIS.
10	Melanjutkan Fixing gold Collection yang tidak match dengan MIS.
11	Melanjutkan Fixing gold Collection yang tidak match dengan MIS dan fixing SCD yang valuenya tidak muncul di table.
12	Fixing SCD yang valuenya tidak muncul di table.
13	Fixing SCD dan fixing table NDB yang belum match.

3.4 User Requirements

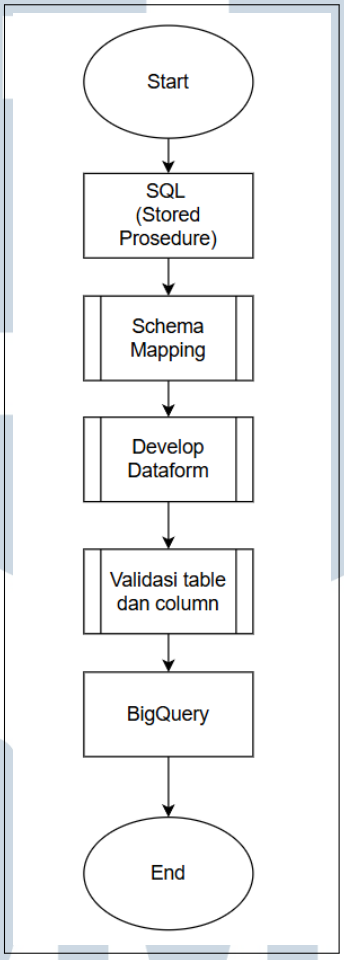
Setiap aktivitas yang dijalankan dalam proses migrasi didasarkan pada kebutuhan dan persyaratan fungsional dari pengguna akhir (*User Requirements*) dan kebutuhan bisnis klien (CIMB). Secara garis besar, kebutuhan utama yang harus dipenuhi oleh *Enterprise Data Platform* (EDP) yang baru ini mencakup aspek fungsional dan non-fungsional, yang dirinci sebagai berikut:

- **Akurasi dan Konsistensi Data:** Data yang dihasilkan oleh sistem EDP harus memiliki akurasi dan konsistensi 100% dengan data sumber (MIS), yang diuji melalui validasi untuk layer Silver dan Gold.
- **Kemudahan Akses Data:** Penyediaan data untuk konsumsi *Business Intelligence* (BI) melalui query SQL yang efisien dan dapat diakses oleh pengguna bisnis.
- **Skalabilitas Platform:** EDP harus dirancang untuk menangani volume data yang besar dan dapat diskalakan seiring pertumbuhan data di masa depan.
- **Keamanan Data:** Penerapan *Policy Tags* untuk data masking pada kolom sensitif, guna memastikan bahwa data yang sensitif terlindungi dengan baik dan memenuhi kebijakan keamanan data.
- **Kemudahan Pemeliharaan (Maintainability):** Platform harus mudah untuk dipelihara dan diperbarui, yang dicapai melalui pengembangan data pipeline yang terstruktur menggunakan *Dataform*.

Semua persyaratan ini menjadi dasar utama dalam perancangan skema dan logika transformasi di *BigQuery*.

3.5 Proses Migrasi

Alur kerja implementasi dimulai dari analisis skrip SQL sumber hingga integrasi akhir ke *Dataform*. Berikut adalah alur proses migrasi yang dilakukan, sesuai dengan alur pada Gambar 3.2:



Gambar 3.2. Alur Migrasi Data dari SQL ke *BigQuery*

Alur migrasi data ini merupakan rangkaian langkah-langkah yang sistematis, dimulai dari tahapan pemahaman logika sumber hingga deployment akhir ke data warehouse. Proses diawali dengan analisis SQL (*Stored Procedure*) yang merupakan logika ETL sumber utama. Tahap ini krusial untuk memetakan semua logika bisnis, join, dan filter yang digunakan pada sistem legacy. Selanjutnya, dilakukan *Schema Mapping*, yaitu proses penentuan skema data yang baru, termasuk pemilihan tipe data *BigQuery* yang tepat dan penamaan kolom yang sudah terstandarisasi untuk *Gold Layer*. Setelah skema disepakati, proses berlanjut ke *Develop Dataform*, di mana semua query migrasi dan transformasi

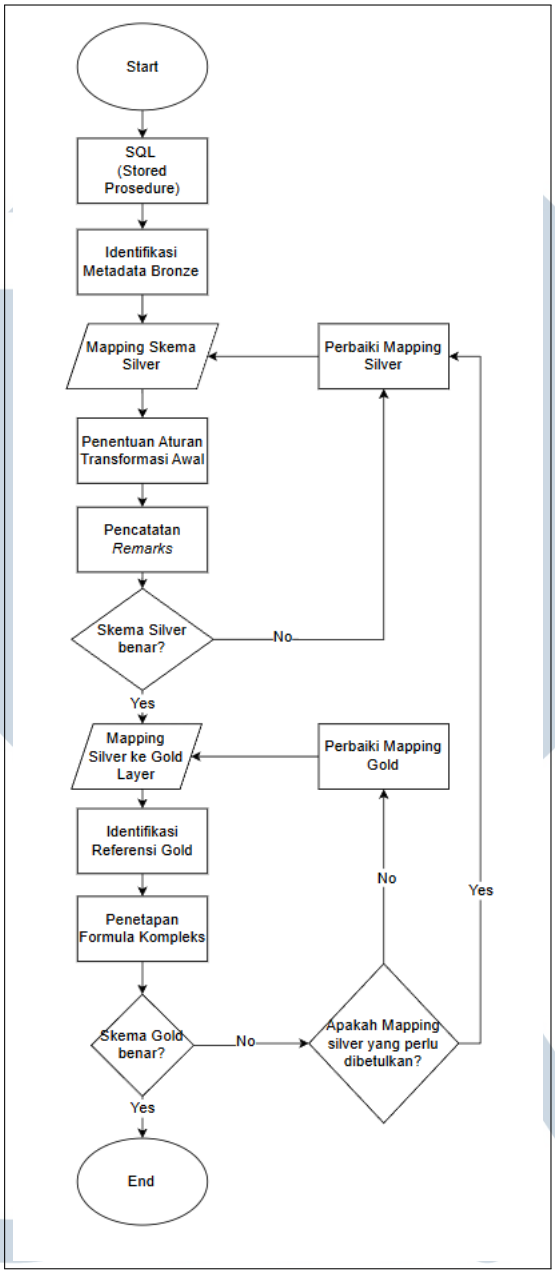
diimplementasikan menggunakan sintaks SQLX *Dataform*, termasuk penentuan materialisasi *View* dan *Incremental*. Tahap kritis berikutnya adalah Validasi Table dan Column, di mana data reconciliation dua langkah (pengecekan Row Count dan Data Quality Check) dilakukan untuk memastikan integritas dan akurasi data. Setelah validasi berhasil, data dimuat secara permanen ke *BigQuery*, mengakhiri siklus migrasi data.

A *Schema Mapping*

Tahap *Schema Mapping* merupakan aktivitas fundamental dan kritikal dalam proyek migrasi data. Aktivitas ini berfokus pada penerjemahan dan penentuan struktur data yang optimal dari sistem sumber on-premises ke lingkungan *Google BigQuery*. Proses ini memiliki tujuan ganda, yaitu memastikan bahwa data tidak hanya tersimpan, tetapi juga terstruktur secara efisien untuk analisis dan memenuhi seluruh persyaratan tata kelola data. Secara umum, proses ini mencakup dua fase transformasi besar dari Bronze Layer ke *Silver Layer*, dan dilanjutkan ke *Gold Layer*.

Alur kerja *Schema Mapping* yang dijalankan bersifat terstruktur dan iteratif. Alur ini dimulai dengan analisis mendalam terhadap SQL (*Stored Procedure*) sumber dan identifikasi Metadata Bronze. Proses kemudian dibagi menjadi dua siklus validasi, di mana Validasi Skema Silver dan Validasi Skema Gold harus berhasil sebelum proses dilanjutkan.

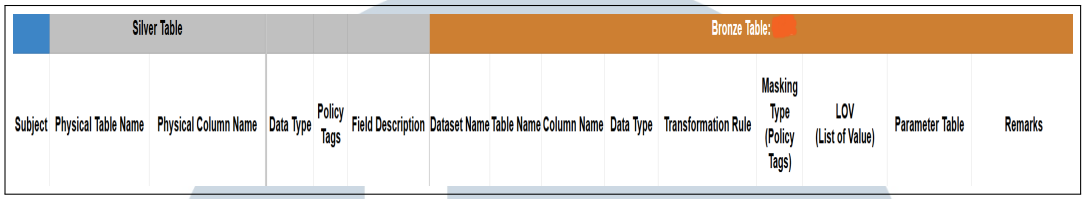
Alur lengkapnya disajikan pada Gambar 3.3. Flowchart tersebut menunjukkan bahwa jika terjadi ketidaksesuaian pada tahap validasi, proses harus kembali ke langkah perbaikan untuk koreksi. Langkah fixing ini memastikan setiap skema layer tervalidasi sebelum melanjutkan ke tahap Pengembangan *Dataform*.



Gambar 3.3. Flowchart *Schema Mapping*

Pemetaan dari Bronze ke *Silver Layer* bertujuan menciptakan shared layer yang bersih dan terstandarisasi. Struktur di *Silver Layer* sendiri dibagi berdasarkan klasifikasi tabel NDB dan COLLECTION. Gambar 3.4 menunjukkan sheets rows untuk pemetaan tersebut. Setiap kolom di *Silver Layer* ditentukan secara hati-hati, termasuk penyesuaian Data Type untuk standarisasi di lingkungan *BigQuery*. Aturan transformasi (*Transformation Rule*) diterapkan untuk case casting atau konversi format data yang diperlukan. *Policy Tags* didefinisikan untuk kolom-kolom

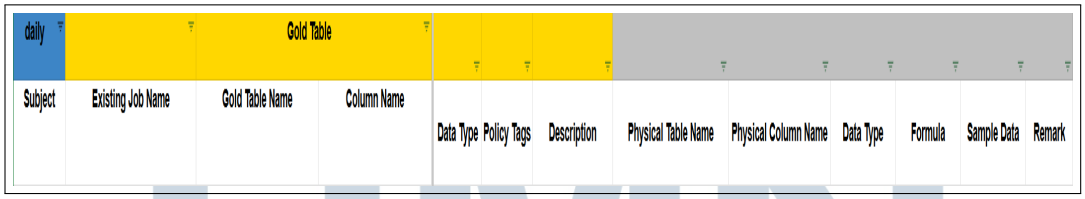
yang memerlukan penyamaran data (*Masking Type*) sejak dini, meskipun sebagian besar hanya dicatat sebagai No Masked.



Gambar 3.4. Mapping bronze ke silver

Dalam proses pemetaan 3.4, diidentifikasi pula kebutuhan akan kolom-kolom yang memerlukan nilai referensi, seperti adanya ketergantungan pada *Parameter Table* untuk mencari nilai deskriptif dari kode, dan disiapkannya kolom *List of Value* (*LOV*) untuk tujuan data profiling nilai-nilai unik. Kolom *Remarks* juga digunakan untuk mencatat informasi penting terkait sumber data yang hilang atau perubahan nama kolom.

Setelah *Silver Layer* selesai dibuat dan divalidasi, proses dilanjutkan dengan pemetaan dari Silver ke *Gold Layer*. Gambar 3.5 menunjukkan contoh mapping sheet untuk transformasi ini.



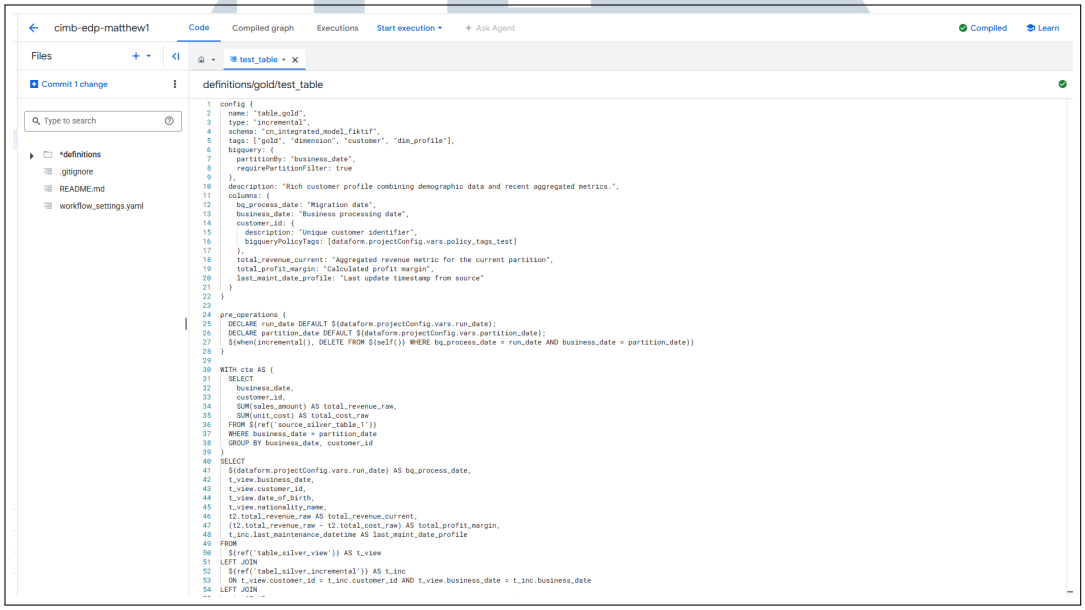
Gambar 3.5. Mapping silver ke gold

Untuk setiap tabel Gold (*Gold Table Name*), skema mencatat kolom sumber dari *Silver Layer* dan mengidentifikasi *Stored Procedure* sumber (*Existing Job Name*) sebagai referensi logika bisnis. Di *Gold Layer*, penetapan *Policy Tags* dan *Data Type* menjadi lebih ketat, menjamin bahwa data siap dikonsumsi, memiliki akurasi tinggi, dan mematuhi regulasi keamanan data.

B *Develop Dataform*

Setelah proses *Schema Mapping* selesai, pengembangan data pipeline dilanjutkan dengan implementasi kode di *Dataform*. Proses ini berfokus pada dua jenis materialisasi di *Silver Layer* dan integrasi kompleks di *Gold Layer*, yang

masing-masing memiliki peran unik dalam menjaga kualitas data dan mendukung analisis bisnis lanjutan. Gambar 3.6 menunjukkan implementasi kode di *Dataform* yang digunakan untuk mengelola pipeline data. Kode ini mencakup proses materialisasi incremental dan pengelolaan partition yang memastikan data yang terproses selalu konsisten dan siap digunakan untuk analisis lebih lanjut.



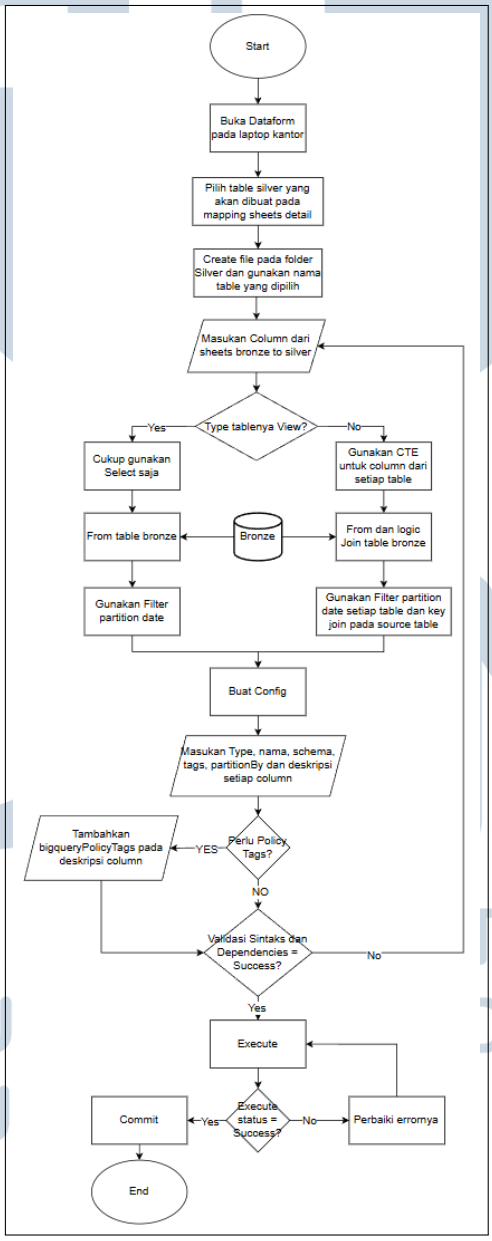
Gambar 3.6. *Dataform* pada *BigQuery*

B.1 Develop Table Silver

Pengembangan tabel di *Silver Layer* merupakan fase krusial dalam arsitektur *Enterprise Data Platform* (EDP) ini. Tahap ini bertujuan utama untuk menciptakan data shared layer yang terstandarisasi, bersih, dan siap konsumsi oleh *Gold Layer* downstream. Di dalam *Silver Layer*, tabel dimaterialisasikan dalam dua tipe yaitu View dan Incremental.

Tabel dengan tipe materialisasi View digunakan pada *Silver Layer* yang fungsinya hanya mereplikasi data dari tabel Bronze secara apa adanya (*as-is*) atau dengan transformasi minimal. Penggunaan View di tahap ini bertujuan untuk mengoptimalkan efisiensi biaya penyimpanan (*storage cost*), karena View tidak menyimpan data secara fisik di *BigQuery* melainkan hanya menyimpan kueri logis. Hal ini memungkinkan sistem untuk menarik data dalam jendela waktu yang lebih luas, misalnya hingga 90 hari terakhir dari sumber Bronze, tanpa menambah beban biaya penyimpanan ganda.

Di sisi lain, tipe materialisasi Incremental digunakan untuk mengelola data berukuran besar dan melakukan semi-aggregation yang kompleks secara efisien. Berbeda dengan View, tipe Incremental menyimpan data secara fisik dalam bentuk tabel. Saat skrip SQLX dijalankan berdasarkan jadwal (*schedule*), data untuk tanggal baru akan langsung ditambahkan (*append*) ke tabel yang sudah ada (*existing table*) tanpa perlu membuat tabel baru dari awal. Hal ini secara signifikan mengurangi beban kerja komputasi dan biaya kueri di *BigQuery*. Alur dari develop silver pada *Dataform* dapat dilihat pada gambar 3.7



Gambar 3.7. Flow *Dataform* Silver

Tabel `tabel_silver_view` mewakili salah satu entitas di *Silver Layer* dengan tipe materialisasi View. Tabel ini dikembangkan untuk menyediakan data yang telah distandardisasi dan siap konsumsi bagi *Gold Layer*. Contoh query tabel view ini dapat dilihat pada kode 3.1.

```
config {
  name: "table_silver_view",
  type: "view",
  schema: "cn_integrated_model",
  tags: ["silver", "ndb", "customer", "silver_ndb"],
  description: "Standardized demographic attributes for NDB customers.",
  columns: {
    customer_id: "Unique identifier for the customer",
    business_date: "Business processing date (partition key)",
    date_of_birth: "Customer's date of birth (standardized format)",
    marital_status_code: "Code representing marital status",
    nationality_name: "Standardized nationality description",
    last_maintenance_date: "Date when the record was last updated"
  }
}

SELECT
  ${dataform.projectConfig.vars.run_date} AS bq_process_date,
  CAST(bussdate_bq AS DATE) AS business_date,
  TRIM(cust_id) AS customer_id,
  SAFE.PARSE_DATE('%Y%m%d', CAST(cust_dob AS STRING)) AS date_of_birth,
  TRIM(cust_marital_sts) AS marital_status_code,
  CASE
    WHEN TRIM(cust_nat_code) = 'ID' THEN 'Indonesia'
    WHEN TRIM(cust_nat_code) = 'SG' THEN 'Singapore'
    ELSE 'Other'
  END AS nationality_name,
  prludt AS last_maintenance_date
FROM
  ${ref('bronze_customer_source')}
WHERE
  bussdate_bq BETWEEN ${dataform.projectConfig.vars.partition_date} - INTERVAL '30' DAY
  AND ${dataform.projectConfig.vars.partition_date}
```

Kode 3.1: Query Create Silver table Tipe View

Implementasi tabel `table_silver_view` pada kode 3.1 menunjukkan fokus utama *Silver Layer* pada standardisasi, data cleansing, dan efisiensi komputasi. Dengan menggunakan type: "view", perusahaan dapat menghemat biaya penyimpanan secara signifikan terutama untuk dataset yang hanya memerlukan standardisasi format tanpa perubahan logika bisnis yang masif. Penggunaan `type: "view"` memastikan bahwa tabel ini tidak menyimpan data fisik sendiri di *BigQuery*, melainkan berfungsi sebagai query standar yang dieksekusi secara on-the-fly setiap kali dipanggil oleh *Gold Layer* atau aplikasi downstream. Hal ini sangat efisien dari sisi biaya penyimpanan (*storage*) dan menjamin data yang ditarik selalu merupakan representasi paling mutakhir dari Bronze Layer.

Dari sisi kualitas data, bagian `SELECT` menerapkan praktik data cleansing awal melalui fungsi `TRIM(cust_id)` dan `TRIM(cust_marital_sts)` untuk menghilangkan *whitespace* yang tidak perlu, yang krusial untuk menjaga integritas kunci saat `join` di tahap *Gold Layer*. Selain itu, fungsi `SAFE.PARSE_DATE()` dan `CAST(... AS DATE)` digunakan untuk mengonversi kolom tanggal mentah (`bussdate_bq`) dari format string atau integer menjadi tipe data DATE yang konsisten dan dapat dipartisi.

Sementara itu, Filter Windowing diterapkan pada klausul `WHERE` untuk membatasi *scan* data sumber (Bronze) hanya pada `partition_date` yang diproses dan jendela *lookback* 30 hari, sehingga menjaga efisiensi komputasi query *BigQuery*.

Tabel `tabel_silver_incremental` mewakili entitas di *Silver Layer* dengan tipe materialisasi incremental. Tabel ini menggunakan tipe incremental untuk mengelola data berukuran besar secara efisien dan menyediakan output yang teragregasi untuk *Gold Layer*. Query dapat dilihat pada kode 3.2.

```
config {
  name: "tabel_silver_incremental",
  type: "incremental",
  schema: "cn_integrated_model_fiktif",
  tags: ["silver", "collection", "sales", "incremental"],
  bigquery: {
    partitionBy: "business_date",
    requirePartitionFilter: true
  },
  description: "Stores daily standardized and semi-aggregated
   sales data.",
  columns: {
    bq_process_date: "Migration date to BigQuery",
```

```
    business_date: "Business processing date",
    store_id: {
      description: "Store identifier",
      bigqueryPolicyTags: [dataform.projectConfig.vars.policy_tags_test]
    },
    total_revenue: {
      description: "Calculated revenue",
      bigqueryPolicyTags: [dataform.projectConfig.vars.policy_tags_test]
    },
    total_cost: {
      description: "Total cost of goods sold",
      bigqueryPolicyTags: [dataform.projectConfig.vars.policy_tags_test]
    },
    shipping_cost: {
      description: "Total shipping cost",
      bigqueryPolicyTags: [dataform.projectConfig.vars.policy_tags_test]
    },
    promo_discount: {
      description: "Total promotional discount applied",
      bigqueryPolicyTags: [dataform.projectConfig.vars.policy_tags_test]
    },
    last_maintenance_datetime: "Last maintenance datetime"
  }
}

pre_operations {
  DECLARE run_date DEFAULT ${dataform.projectConfig.vars.run_date};
  DECLARE partition_date DEFAULT ${dataform.projectConfig.vars.partition_date};
  ${when(incremental(), DELETE FROM ${self()} WHERE bq_process_date = run_date AND business_date = partition_date)}
}

WITH cte_1 AS (
  SELECT
    store_id,
    SUM(quantity) AS total_quantity,
```

```
48      SUM(sales_amount) AS daily_revenue
49    FROM ${ref('source_bronze_table_1')}
50    WHERE bussdate_bq = partition_date
51    GROUP BY store_id
52  ),
53  main_source AS (
54    SELECT
55      store_id,
56      shipping_cost,
57      promo_discount,
58      cust_date_maint,
59      cust_time_maint
60    FROM ${ref('source_bronze_table_3')}
61    WHERE bussdate_bq = partition_date
62  )
63  SELECT
64    run_date AS bq_process_date,
65    CAST(t_main.bussdate_bq AS DATE) AS business_date,
66    t_main.store_id,
67    t1.daily_revenue AS total_revenue,
68    t_main.shipping_cost,
69    t_main.promo_discount,
70    t_main.cust_date_maint AS last_maintenance_datetime
71  FROM main_source t_main
72  LEFT JOIN cte_1 t1
73    ON t_main.store_id = t1.store_id
```

Kode 3.2: Query create silver tabel tipe incremental

Implementasi tabel `tabel_silver_incremental` ini menunjukkan bahwa di *Silver Layer*, dilakukan agregasi tingkat menengah (*semi-aggregation*) dan `join` kompleks langsung dari Bronze Layer. Tipe ini dipilih karena data disimpan secara fisik, sehingga mempercepat akses kueri untuk tahap berikutnya dibandingkan dengan View yang harus menghitung ulang logika setiap kali dipanggil. Strategi ini digunakan untuk mempercepat *time-to-market* data, di mana pemrosesan *heavy-duty* dilakukan pada satu langkah transformasi.

Fitur paling krusial adalah blok `pre_operations` yang menerapkan prinsip *Upsert*. Logika `DELETE FROM $self() WHERE...` memastikan bahwa data lama untuk hari proses (`partition_date`) dihapus sebelum data baru dimasukkan, menjamin integritas data dan memungkinkan pipeline diulang.

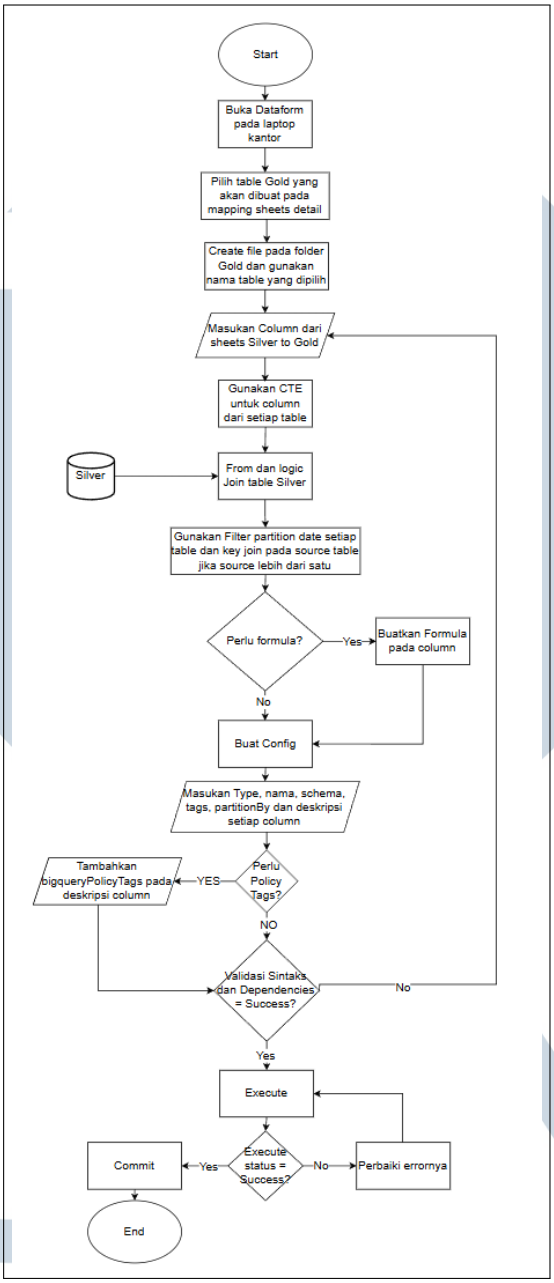
Bagian utama `SELECT` menggunakan *Common Table Expression* (*CTE*) yaitu `cte_1` untuk memisahkan logika agregasi (seperti `SUM(quantity)` dan

`SUM(sales_amount)`) dan perhitungan awal. Logika ini kemudian di LEFT JOIN dengan tabel detail lainnya, memastikan data teragregasi dan terstandarisasi. Logika konversi `last_maintenance_datetime` yang kompleks juga diterapkan, menggabungkan tanggal dan waktu mentah menjadi satu kolom `TIMESTAMP` yang akurat.

Penerapan *Policy Tags* pada kolom sensitif (misalnya, `total_revenue`) menggunakan variabel `policy_tags_test` di `config` menjamin keamanan data.

B.2 Develop Table Gold

Seluruh pengembangan tabel di *Gold Layer* diwajibkan menggunakan tipe materialisasi Incremental. Hal ini dikarenakan *Gold Layer* merupakan tahap akhir yang dikonsumsi langsung oleh laporan MIS dan alat *Business Intelligence* (BI), sehingga performa kueri harus sangat cepat. Dengan menyimpan data secara fisik melalui metode append harian, *BigQuery* tidak perlu memproses ulang data historis, yang secara langsung mengoptimalkan biaya operasional (*OPEX*) dan memastikan ketersediaan data historis yang stabil. Alur dari develop table gold dapat dilihat pada flowchart Gambar 3.8.



Gambar 3.8. Flow *Dataform* Gold

Berikut adalah contoh kode 3.3 query SQL untuk memproses dan menggabungkan data dari berbagai sumber menjadi tabel `table_gold`.

```
config {
  name: "table_gold",
  type: "incremental",
  schema: "cn_integrated_model_fiktif",
  tags: ["gold", "dimension", "customer", "dim_profile"],
  bigquery: {
```

```
    partitionBy: "business_date",
    requirePartitionFilter: true
  },
  description: "Rich customer profile combining demographic data and recent aggregated metrics.",
  columns: {
    bq_process_date: "Migration date",
    business_date: "Business processing date",
    customer_id: {
      description: "Unique customer identifier",
      bigqueryPolicyTags: [dataform.projectConfig.vars.policy_tags_test]
    },
    total_revenue_current: "Aggregated revenue metric for the current partition",
    total_profit_margin: "Calculated profit margin",
    last_maint_date_profile: "Last update timestamp from source"
  }
}

pre_operations {
  DECLARE run_date DEFAULT ${dataform.projectConfig.vars.run_date};
  DECLARE partition_date DEFAULT ${dataform.projectConfig.vars.partition_date};
  ${when(incremental(), DELETE FROM ${self()} WHERE bq_process_date = run_date AND business_date = partition_date)}
}

WITH cte AS (
  SELECT
    business_date,
    customer_id,
    SUM(sales_amount) AS total_revenue_raw,
    SUM(unit_cost) AS total_cost_raw
  FROM ${ref('source_silver_table_1')}
  WHERE business_date = partition_date
  GROUP BY business_date, customer_id
)
SELECT
  ${dataform.projectConfig.vars.run_date} AS bq_process_date,
  t_view.business_date,
  t_view.customer_id,
```

```
  t_view.date_of_birth,
  t_view.nationality_name,
  t2.total_revenue_raw AS total_revenue_current,
  (t2.total_revenue_raw - t2.total_cost_raw) AS
   total_profit_margin,
  t_inc.last_maintenance_datetime AS last_maint_date_profile
FROM
  ${ref('table_silver_view')} AS t_view
LEFT JOIN
  ${ref('tabel_silver_incremental')} AS t_inc
  ON t_view.customer_id = t_inc.customer_id AND t_view.
   business_date = t_inc.business_date
LEFT JOIN
  cte AS t2
  ON t_view.customer_id = t2.customer_id AND t_view.business_date
   = t2.business_date
WHERE
  t_view.business_date = partition_date
```

Kode 3.3: Query create gold table

Implementasi tabel `table_gold` pada kode 3.3 menunjukkan inti dari data warehousing yaitu penggabungan data dari berbagai pipeline yang berbeda, dengan fokus pada performa dan tata kelola data. Tabel ini menggunakan tipe materialisasi `incremental` dan mengandalkan `partitioning` pada `business_date` untuk optimasi, di mana blok `pre_operations` menjamin proses *Upsert* (*penghapusan data lama sebelum data baru dimasukkan*) berjalan lancar, mencegah duplikasi.

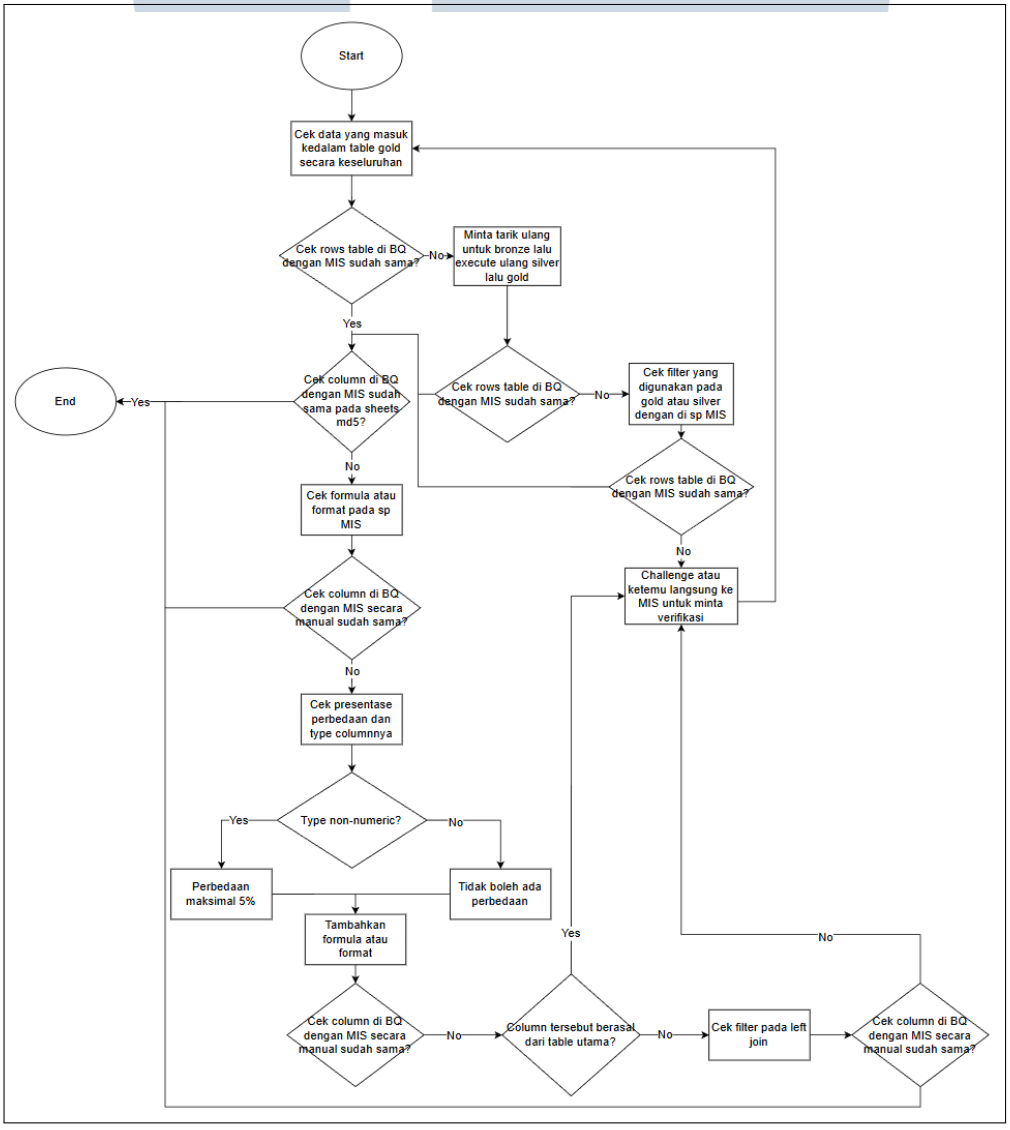
Dari sisi transformasi, *Common Table Expression* (*CTE*) digunakan secara modular (misalnya, `cte_2`) untuk menerapkan Formula bisnis lanjutan, seperti perhitungan `profit_margin`, yang merupakan selisih antara `revenue` dan `cost`. Setelah metrik dihitung, query utama melakukan penggabungan multi-sumber dengan `join` antara data demografi terstandarisasi (`table_silver_view`) dengan data agregasi yang baru dihitung.

Terakhir, aspek tata kelola data dijamin melalui penerapan *Policy Tags* pada kolom sensitif (misalnya, `total_revenue`, `total_profit_margin`) di blok `config`, memastikan kebijakan keamanan (seperti *data masking*) secara otomatis diterapkan pada level *BigQuery*.

C *Validate Table*

Proses validasi tabel (data reconciliation) merupakan tahapan krusial yang dilakukan segera setelah data berhasil dimuat ke *Gold Layer* dari pipeline *Dataform*. Validasi ini bertujuan untuk memastikan integritas, akurasi, dan konsistensi data antara sumber dengan data hasil transformasi di *BigQuery*. Proses ini dilakukan secara berlapis, dimulai dari pengecekan kuantitas hingga pengecekan kualitas kolom berdasarkan toleransi yang telah ditetapkan.

Proses validasi ini mengikuti alur bertingkat, yang memastikan data yang masuk tidak hanya lengkap tetapi juga memiliki nilai yang benar sesuai standar bisnis, seperti yang diilustrasikan pada Gambar 3.9.



Gambar 3.9. Flow Validasi Table

Alur kerja validasi ini dimulai dengan Pengecekan Kuantitas (*Row Count Check*) yang membandingkan jumlah baris antara Tabel Tujuan (Gold) dengan Tabel Sumber (MIS/Bronze). Pengecekan ini diimplementasikan menggunakan query dasar COUNT(*) untuk memastikan tidak ada kehilangan data. Jika kuantitas cocok, alur berlanjut ke Validasi Kualitas Data (*DQC*) yang menerapkan aturan ketat berdasarkan tipe data.

DQC memisahkan kolom yang dapat dicek dengan cara otomatis menggunakan sheets MD5 untuk menunggu update hasil rows column terbaru. Cara lain bisa dilakukan secara manual dengan menggunakan query pada *BigQuery*. Ada dua tipe kolom, Numeric dan Non-Numeric. Untuk kolom Tipe Numeric toleransi perbedaan ditetapkan pada 0% untuk menjamin integritas finansial. Pengecekan dilakukan dengan menghitung total agregasi di kedua tabel:

```
SELECT SUM((nama_column))
FROM tabel_tujuan WHERE business_date = partition_date
```

Kode 3.4: Query untuk cek value column tipe numeric

Jika hasil agregasi tidak sama, terjadi kegagalan kritis. Sebaliknya, untuk kolom Tipe Non-Numeric (misalnya, status code atau string), toleransi perbedaan maksimum diizinkan 5% untuk mengakomodasi inkonsistensi minor. Secara teknis, kolom Non-Numeric diuji menggunakan fungsi string atau checksum pada playground pengujian, seperti:

```
SELECT SUM(playground.test_string(nama_column))
FROM tabel_tujuan WHERE business_date = partition_date
```

Kode 3.5: Query untuk cek value column tipe non-numeric

D Komparatif *before and after implementation*

Pada bagian ini, dilakukan analisis perbandingan teknis untuk mengevaluasi efektivitas migrasi dari sistem legacy berbasis on-premises (menggunakan *Stored Procedure MySQL*) ke sistem modern cloud-native (menggunakan *Dataform* di *Google BigQuery*). Perbandingan ini bertujuan untuk menunjukkan peningkatan efisiensi kueri, ketahanan sistem, serta pengelolaan transparansi data.

1. **Representasi Sistem *Before* (Legacy *Stored Procedure*)**
   Sistem lama menggunakan *Stored Procedure* yang dijalankan pada server fisik. Pendekatan ini memiliki keterbatasan dalam menangani lonjakan data

karena metode pemrosesan yang cenderung statis dan menggunakan prinsip *Full Refresh* (menghapus dan memuat ulang seluruh data), yang memakan waktu lama serta sumber daya komputasi yang besar. Struktur logika kueri pada sistem lama dapat dilihat seperti pada gambar 3.10.

```
-- Contoh Dummy Stored Procedure MySQL (Legacy)
DELIMITER //

CREATE PROCEDURE proc_legacy_migration(IN p_date INT)
BEGIN
    DELETE FROM legacy_schema.tabel_final;

    INSERT INTO legacy_schema.tabel_final (
        col_1,
        col_2,
        col_3,
        col_4,
        col_5
    )
    SELECT
        TRIM(src.col_a),
        src.col_b,
        src.col_c,
        CASE
            WHEN src.col_d = 'X' THEN 'Tipe A'
            ELSE 'Tipe B'
        END,
        STR_TO_DATE(src.col_e, '%Y%m%d')
    FROM
        bronze_db.source_table src
    WHERE
        src.process_date = p_date;

END //

DELIMITER ;
```

Gambar 3.10. Contoh *Store Procedure Mysql*

2. **Representasi Sistem *After* (*Dataform SQLX*)**
   Sistem baru memanfaatkan *Dataform* dengan materialisasi *incremental* yang jauh lebih efisien. Logika pemrosesan dipecah secara modular, di mana data cleansing dilakukan di *Silver Layer* yang dapat dilihat pada gambar 3.11 dan logika bisnis di *Gold Layer* yang dapat dilihat pada gambar 3.12. Penggunaan fitur *Safe-parsing* dan *Lineage* otomatis memastikan pipeline lebih tangguh dan mudah dipantau.

```
config {
  type: "incremental",
  schema: "cn_integrated_model_silver",
  bigquery: {
    partitionBy: "col_date"
  }
}

SELECT
  CAST(col_e AS DATE) AS col_date,
  TRIM(col_a) AS col_1,
  col_b AS col_2,
  col_c AS col_3,
  col_d AS col_status_code,
  SAFE.PARSE_DATE('%Y%m%d', CAST(col_f AS STRING)) AS col_timestamp
FROM
  ${ref("bronze_source")}
WHERE
  col_e = ${dataform.projectConfig.vars.partition_date}
```

Gambar 3.11. Contoh *Bigquery Silver*

```
config {
  type: "incremental",
  schema: "cn_integrated_model_gold",
  columns: {
    col_2: {
      description: "Sensitive Data Column",
      -- Solusi: Masking otomatis menggunakan Policy Tags (Data Governance)
      bigqueryPolicyTags: [dataform.projectConfig.vars.policy_tags_masked]
    }
  }
}

pre_operations {
  ${when(incremental(), `DELETE FROM ${self()} WHERE col_date = ${dataform.projectConfig.vars.partition_date}`)}
}

SELECT
  s.col_date,
  s.col_1,
  s.col_2,
  CASE
    WHEN s.col_status_code = 'X' THEN 'Tipe A'
    ELSE 'Tipe B'
  END AS col_4,
  s.col_timestamp AS col_5
FROM
  ${ref("silver_table")} AS s
WHERE
  s.col_date = ${dataform.projectConfig.vars.partition_date}
```

Gambar 3.12. Contoh *Bigquery Gold*

Berdasarkan perbandingan, sistem baru memberikan keunggulan signifikan dalam aspek operasional. Penggunaan *incremental loading* mengoptimalkan biaya kueri di *BigQuery* karena hanya memproses data harian, bukan keseluruhan dataset. Selain itu, transparansi data melalui *lineage* otomatis mempermudah pelacakan error yang sebelumnya sulit dilakukan pada *Stored Procedure* lama.

3.6 Kendala dan Solusi yang Ditemukan

Pengembangan data pipeline di lingkungan *Enterprise Data Platform* (EDP) menggunakan *Dataform* dan *BigQuery* sering menghadapi tantangan terkait kualitas data, performa komputasi, dan kompleksitas logika bisnis. Berikut adalah kendala dan solusi yang diterapkan selama implementasi.

3.6.1 Kendala yang Ditemukan

- Inkonsistensi Kualitas Data (*Data Quality Challenge*): Data mentah yang berasal dari Bronze Layer sering kali memiliki format yang tidak standar, adanya *whitespace* tersembunyi pada kolom *string*, dan ketidaksesuaian tipe data. Hal ini menyebabkan kegagalan dalam proses `JOIN` di *Gold Layer* atau *Silver Layer* karena *primary keys* tidak terbaca secara identik, sehingga menghambat aliran data yang bersih dan konsisten.
- Data Lineage dan Debugging Kesalahan: Kesulitan dalam melacak asal-usul kesalahan atau data drift yang terjadi di *Gold Layer* dan *Silver Layer*, karena beberapa *Gold Layer* dan *Silver Layer* bergantung pada `join` dan agregasi dari banyak tabel *Silver Layer* atau Bronze Layer yang berbeda.

3.6.2 Solusi atas Kendala yang Ditemukan

- Standardisasi dan data cleansing di *Gold Layer* atau *Silver Layer* dengan menggunakan fungsi `TRIM()` untuk menghilangkan whitespace yang tidak terlihat. Konversi tipe data dilakukan menggunakan `SAFE.PARSE_DATE('%Y%m%d', CAST(... AS STRING))` dalam fungsi untuk memastikan data tanggal dikonversi secara benar dan aman dari error, sehingga kolom `date` menjadi konsisten di seluruh layer dan mendapatkan *value* yang sesuai.
- *Dataform* dimanfaatkan untuk mendokumentasikan dan memetakan aliran data secara eksplisit. Pemanfaatan fungsi `$ref('nama_tabel')` digunakan untuk mendefinisikan dependency antar-tabel, yang memungkinkan visualisasi dan pemantauan aliran data secara otomatis. Penggunaan `Tags` (seperti `gold`, `silver`, `customer`) di blok `config` membantu pengembang mengelompokkan dan mengisolasi eksekusi query saat debugging. Jika terjadi kesalahan di *Gold Layer* atau *Silver Layer*, developer dapat dengan

cepat merujuk kembali ke query *Silver Layer* atau cek Bronze Layer di *BigQuery* yang menjadi dependency dengan melacak fungsi `ref`.