BAB III

PELAKSANAAN KERJA

3.1 Kedudukan dan Koordinasi

Bagian ini berisi keterangan/informasi mengenai posisi penulis dan alur koordinasi penulis dengan pembimbing lapangan pada saat pengerjaan suatu proyek/pengerjaan.

3.1.1 Kedudukan

Penulis ditempatkan di Biro Data Management B di bawah Divisi Data Management, yang merupakan bagian dari *Group Enterprise Architecture, Data, and Service Quality* (ADQ) PT Bank Central Asia, Tbk (BCA). Pada lingkup biro tersebut, penulis bergabung dengan tim Data Standard, yaitu tim yang bertanggung jawab atas proses standarisasi, validasi, serta otomatisasi integrasi data ke sistem *Data Warehouse* (DW) dan *Big Data* (BD).

Sebagai mahasiswa magang dengan posisi Data Engineer Intern, penulis berperan dalam membantu tim dalam proses pengembangan skrip otomatisasi (*Shell Script*) yang digunakan untuk konversi file bisnis ke format standar Avro, serta mendukung kegiatan operasional harian seperti proses *consume data* ke *Datalake* untuk kebutuhan *User Acceptance Test* (UAT). Selama masa magang, seluruh kegiatan dilakukan di bawah supervisi langsung dari mentor tim *Data Standard*, yang memberikan bimbingan teknis dan arahan terkait pengembangan proyek.

3.1.2 Koordinasi

Selama pelaksanaan program magang di PT Bank Central Asia, Tbk, penulis melaksanakan tugas dengan mengikuti alur koordinasi yang bersifat hierarkis sesuai dengan struktur organisasi pada Biro Data Management B di bawah *Group Enterprise Architecture Data, and Service Quality* (ADQ). Proses koordinasi dan pelaporan tugas dilakukan secara berjenjang mulai dari

mentor hingga pimpinan unit kerja, sehingga setiap kegiatan magang dapat terpantau dan terarah sesuai dengan kebijakan serta standar yang berlaku di lingkungan perusahaan.

Secara umum, koordinasi dimulai dari mentor yang memberikan arahan teknis harian kepada penulis terkait pelaksanaan tugas proyek, seperti pengembangan skrip otomatisasi, validasi data, dan kegiatan support untuk *User Acceptance Test* (UAT). Mentor bertanggung jawab untuk mengevaluasi hasil kerja penulis sebelum dilaporkan kepada *Sub-Team Leader Data Standardization*, yang kemudian melakukan pengecekan kesesuaian hasil pekerjaan dengan standar tim. Apabila hasil kerja telah memenuhi kriteria, proses koordinasi dilanjutkan ke *Team Leader Historical Data Development*, yang berperan dalam memverifikasi kelayakan pekerjaan untuk diteruskan ke tahap implementasi proyek.

Selanjutnya, hasil pekerjaan yang telah dinilai layak akan diteruskan secara berjenjang ke Kepala Biro Data Management B dan Kepala Divisi Data Management untuk proses validasi strategis dan pelaporan kepada Kepala *Group Enterprise Architecture Data, and Service Quality* (ADQ). Alur koordinasi ini memastikan bahwa setiap kegiatan, baik bersifat teknis maupun administratif, dilakukan secara sistematis dan terdokumentasi dengan baik. Dengan demikian, penulis tidak hanya belajar menjalankan tugas teknis, tetapi juga memahami mekanisme komunikasi dan koordinasi lintas jenjang dalam lingkungan kerja profesional.

Gambar 3.1 Bagan kedudukan dan Alur Koordinasi

3.2 Tugas yang Dilakukan

Selama pelaksanaan program magang di PT Bank Central Asia Tbk, penempatan dilakukan pada tim Data Standard di bawah Biro Data Management B. Tim ini berperan dalam memastikan standarisasi dan otomatisasi proses pengolahan data yang mengalir ke sistem *Data Warehouse* (DW) dan *Big Data* (BD). Berdasarkan Table 3.1, kegiatan magang dilaksanakan secara bertahap, mulai dari pengenalan jobdesk kerja, pengembangan script otomatisasi berbasis *Shell Script* (SH) yang mampu menampung file bisnis, hingga pengujian dan validasi untuk memastikan fungsionalitas dan keandalan *script*. Selain itu, dilakukan pula enhancement terhadap script generator guna meningkatkan efisiensi proses otomatisasi serta implementasi script ke sistem produksi untuk mendukung aktivitas operasional harian, termasuk konsumsi data dari *Datalake* dalam proses

User Acceptance Test (UAT) pada tahapan ELT. Adapun rincian kegiatan dan tugas yang telah diselesaikan selama periode magang adalah sebagai berikut:

Tabel 3.1 Detail Pekerjaan yang Dilakukan

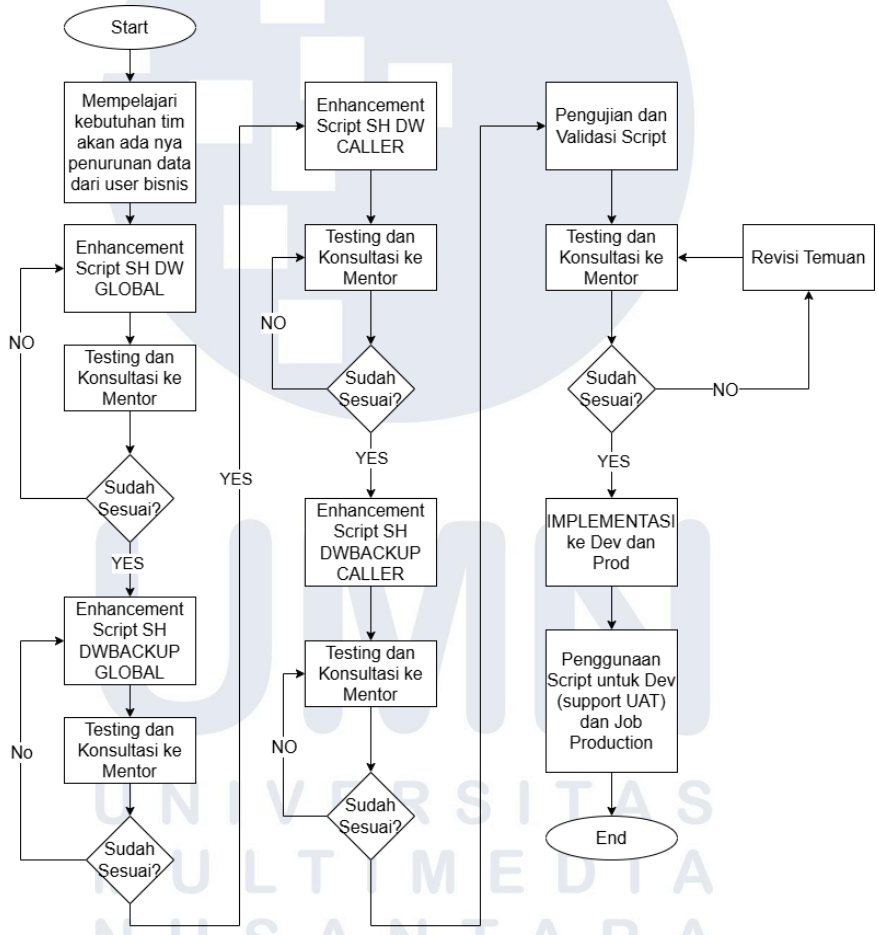
No.	Minggu	Proyek	Keterangan
1	Minggu 1 (2–6 Juni 2025)	Pengenalan jobdesk kerja	Orientasi awal terhadap lingkungan kerja, sistem internal, serta pemahaman dasar proyek di tim Data Standard.
2	Minggu 2–12 (9 Juni–29 Agustus 2025)	Improvement SH untuk menampung file bisnis	Pengembangan skrip Shell (SH) dengan penambahan fungsi parameterisasi dan penyesuaian terhadap kebutuhan integrasi file bisnis.
3	Minggu 5–17 (2 Juli–25 September 2025)	Pengembangan Script Generator	Penyempurnaan (enhancement) sistem generator agar mampu membuat skrip pemanggil otomatis berdasarkan konfigurasi proyek.
4	Minggu 6–16 (5 Juli–20 September 2025)	Pengujian dan validasi SH Global	Pengujian fungsional dan error handling terhadap skrip global untuk memastikan stabilitas proses konversi dan distribusi file.
5	Minggu 9–17 (1 Agustus–29 September 2025)	Pengujian hasil Script Generator	Pengujian generator pada berbagai skenario proyek untuk memastikan parameter dan keluaran sesuai dengan standar otomasi.
6	Minggu 13–14 (29 Agustus–5 September 2025)	Implementasi ke Production	Implementasi skrip hasil pengembangan ke lingkungan produksi setelah melalui tahap validasi dan review mentor.
7	Minggu 15–17 (2 Juni–25 September 2025)	Consume data ke Datalake (Support UAT)	Pelaksanaan proses harian untuk pemindahan data hasil penurunan aplikasi ke Datalake sebagai dukungan terhadap pengujian UAT.

Seluruh kegiatan magang dilaksanakan secara bertahap, dimulai dari tahap pengenalan jobdesk kerja hingga implementasi script ke dalam sistem produksi.

Script berbasis *Shell Script* (SH) dikembangkan untuk memproses dan mengelola file bisnis yang mengalir ke sistem *Data Warehouse* (DW), *Big Data* (BD), serta *Datalake*. Sebelum diimplementasikan, script diuji terlebih dahulu menggunakan data dummy guna memastikan integritas dan fungsionalitasnya. Selain itu, kegiatan monitoring juga dilakukan sebagai bagian dari tahap *Load* dalam proses ELT (*Extract, Load, Transform*).

3.3 Uraian Pelaksanaan Kerja

3.3.1 Proses Pelaksanaan



Gambar 3.2 Proses Alur Kerja

Gambar 3.2 di atas menampilkan alur kerja yang dilakukan penulis selama pelaksanaan proyek pengembangan *Shell Script* (SH) Global di lingkungan PT Bank Central Asia, Tbk. Proses kerja dimulai dengan tahap pemahaman terhadap kebutuhan tim *Data Standard*, khususnya terkait

penurunan data yang dikirim oleh tim bisnis. Tahapan awal ini menjadi dasar dalam menentukan arah pengembangan skrip agar selaras dengan kebutuhan operasional perusahaan. Setelah pemetaan kebutuhan dilakukan, tahap berikutnya adalah pengembangan skrip inti, yaitu SH DW Global dan SH DW Caller, yang dirancang untuk memproses, memvalidasi, serta mendistribusikan data ke sistem *Data Warehouse* secara otomatis.

Setiap tahapan pengembangan skrip selalu melalui proses testing dan konsultasi bersama mentor. Apabila hasil uji menunjukkan ketidaksesuaian, penulis melakukan revisi sesuai dengan arahan yang diberikan hingga skrip memenuhi standar teknis yang berlaku. Mekanisme ini memastikan bahwa setiap komponen yang dikembangkan telah diuji secara menyeluruh dan dapat berfungsi stabil dalam lingkungan kerja yang kompleks. Pendekatan iterative testing tersebut diterapkan secara konsisten pada seluruh skrip, termasuk *DWBackup Global* dan *DWBackup Caller*, untuk menjamin kesesuaian logika, integrasi antar komponen, serta keandalan sistem sebelum diimplementasikan ke tahap berikutnya.

Setelah seluruh proses pengujian dan validasi selesai, hasil pengembangan diimplementasikan secara bertahap pada lingkungan development dan production. Implementasi dilakukan dengan koordinasi antara penulis, mentor, dan tim terkait guna memastikan tidak terjadi gangguan pada sistem yang sedang berjalan. Tahapan akhir dari alur ini adalah pemanfaatan skrip untuk mendukung kebutuhan pengujian (*User Acceptance Test*/UAT) serta pelaksanaan otomasi job production harian. Dengan demikian, alur kerja ini menggambarkan proses pengembangan yang sistematis dan terstruktur, mencakup fase analisis kebutuhan, pengembangan, pengujian, revisi, hingga penerapan dan penggunaan dalam operasional aktual perusahaan. Seluruh proses pengembangan dan implementasi ini didukung oleh berbagai tools, di antaranya:

A. DBeaver

Gambar 3.3 Logo DBeaver

Sumber : [16]

DBeaver berfungsi sebagai klien grafis lintas basis data yang menghubungkan berbagai sistem manajemen, termasuk *Hive* dan *Impala*, untuk mendukung penelusuran data, inspeksi skema tabel, dan verifikasi keluaran kueri melalui panel visual interaktif. Perangkat ini menyediakan pelengkapan otomatis dan kemampuan eksekusi SQL lintas platform, sehingga aktivitas uji serta analisis pada fase pengembangan dan penelusuran kesalahan dapat dilakukan lebih tertib dan efisien [17].

B. MobaXterm

Gambar 3.4 Logo MobaXterm

Sumber : [18]

MobaXterm berperan sebagai *emulator terminal* dan *klien SSH* untuk akses jarak jauh ke server, yang dalam praktik pengembangan memungkinkan pengelolaan berkas, eksekusi skrip, dan pemantauan log langsung pada lingkungan target; dukungan terhadap *multiple session/host* dalam satu antarmuka terpadu mengonsolidasikan alur kerja dan menjaga efisiensi operasional [19].

C. Shell scripting

Gambar 3.5 Logo Shell Scripts

Sumber : [20]

Shell scripting berbasis *Bash* dijadikan fondasi otomasi selama masa magang, mencakup pembangunan komponen generator, skrip pemanggil, hingga mekanisme validasi berkas untuk menjamin portabilitas, fleksibilitas, dan kemudahan integrasi pada ekosistem Linux di perusahaan; pendekatan ini sekaligus menyediakan jejak pelacakan yang lebih sistematis dan memungkinkan otomasi menyeluruh tanpa ketergantungan pada intervensi manual [20], [21].

Seluruh perangkat dan sistem yang telah dijelaskan digunakan secara aktif selama pelaksanaan kegiatan magang untuk mendukung penyelesaian tugas-tugas yang berkaitan dengan pengembangan dan optimalisasi proses otomasi data. *Tools* dan sumber daya tersebut dimanfaatkan dalam konteks kolaborasi tim, pengujian teknis, serta implementasi solusi berbasis *Shell Script* (SH) yang terintegrasi dengan lingkungan *Data Warehouse* (DW), *DW Backup*, dan *Datalake*. Fokus utama kegiatan magang diarahkan pada *enhancement Script SH Global*, serta implementasinya ke dalam lingkungan development dan production.

Proses konversi file bisnis ke Avro memberikan sejumlah keunggulan yang signifikan dalam mendukung efisiensi dan keandalan pengelolaan data di lingkungan perusahaan. Melalui mekanisme otomatisasi yang terintegrasi, sistem tidak hanya mempercepat proses pemrosesan data, tetapi juga memastikan konsistensi, integritas, serta kesiapan data untuk digunakan dalam sistem analitik yang lebih luas. Tabel berikut merangkum beberapa

aspek utama yang menjadi keunggulan dari penerapan proses konversi file bisnis ke Avro beserta penjelasan fungsionalnya.

Tabel 3.2 Perbandingan sebelum dan sesudah refaktorisasi

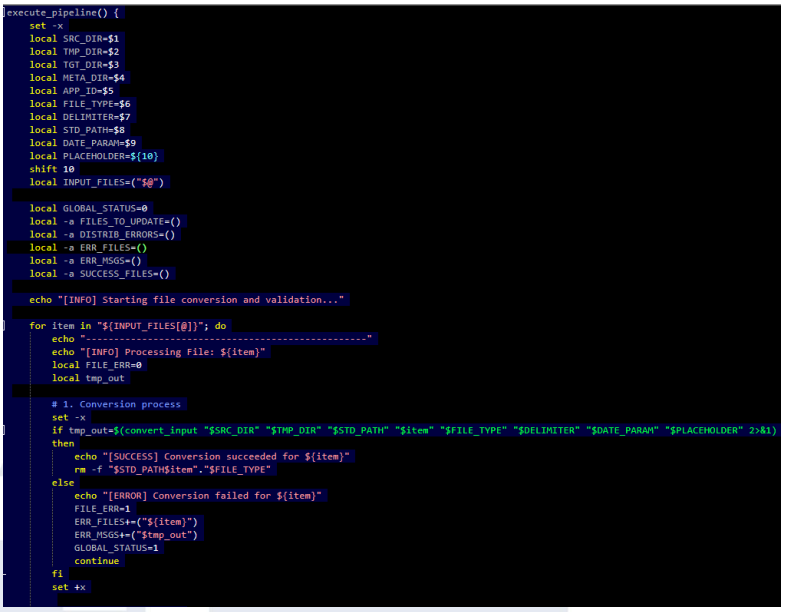
Aspek	Keunggulan	Penjelasan
Efisiensi Pemrosesan	Proses konversi berjalan otomatis tanpa intervensi manual	Otomatisasi mengurangi waktu pemrosesan dan kesalahan manusia dalam pengelolaan file bisnis.
Konsistensi Format Data	Data dikonversi ke format Avro yang terstandarisasi	Format Avro menjamin struktur data seragam dan kompatibel dengan sistem *Data Warehouse* serta *Big Data*.
Integritas Data Terjaga	Validasi dilakukan di setiap tahap pemrosesan	Proses konversi dilengkapi dengan fungsi pengecekan dan pembandingan *schema* untuk memastikan data tidak rusak atau hilang.
Skalabilitas Tinggi	Dapat menangani banyak file dalam satu alur proses	Struktur pipeline memungkinkan pemrosesan paralel untuk sejumlah besar file bisnis secara konsisten.
Kualitas Data Terjamin	File melalui tahap validasi, perbaikan, dan distribusi otomatis	Sistem memastikan hanya data valid yang diteruskan ke lingkungan produksi.
Efisiensi Penyimpanan	File hasil konversi dikompresi tanpa kehilangan data	Penggunaan kompresi dalam format Avro menghemat ruang penyimpanan tanpa menurunkan kualitas data.

Dengan adanya keunggulan-keunggulan tersebut, proses konversi file bisnis ke Avro mampu meningkatkan efisiensi operasional sekaligus menjaga kualitas data yang dihasilkan. Seluruh tahapan, mulai dari validasi, konversi, hingga distribusi, dijalankan secara otomatis dan terstandarisasi, sehingga meminimalkan potensi kesalahan serta mempercepat integrasi data ke dalam sistem *Data Warehouse* maupun *Big Data*. Penerapan proses ini juga memberikan fleksibilitas bagi perusahaan dalam mengelola volume data yang besar secara konsisten dan dapat diandalkan untuk mendukung kebutuhan analisis serta pengambilan keputusan berbasis data Penjelasan lebih rinci mengenai proses tersebut disajikan pada bagian berikut:

3.3.1.1 Enhancement DW GLOBAL

Enhancement pada *DW Global SH* dilakukan sebagai upaya untuk meningkatkan kemampuan sistem dalam mengelola proses otomasi data yang terintegrasi ke dalam lingkungan *Data Warehouse* (DW). Sebelumnya, skrip *DW Global* hanya menjalankan fungsi dasar seperti validasi dan distribusi file tanpa dukungan mekanisme konversi otomatis. Namun, dengan meningkatnya kompleksitas kebutuhan bisnis dan variasi sumber data, sistem memerlukan pembaruan yang memungkinkan pengolahan file dalam berbagai format, termasuk konversi file bisnis ke format Avro. Melalui proses pengembangan ini, alur kerja *DW Global* diperluas agar mampu menangani tahap-tahap otomatis seperti validasi struktur, konversi format, hingga pendistribusian hasil akhir ke direktori tujuan secara efisien dan terstandarisasi.

Selain itu, pembaruan pada *DW Global SH* juga berfungsi untuk meningkatkan efisiensi pemeliharaan serta fleksibilitas sistem dalam jangka panjang. Dengan menerapkan struktur modular, setiap fungsi di dalam skrip dapat diperbarui atau diperluas tanpa mengubah keseluruhan logika utama. Hal ini memungkinkan *DW Global* untuk menjadi pusat kendali utama dalam pipeline otomasi data, sekaligus memastikan seluruh proses integrasi ke sistem Data Warehouse berlangsung konsisten, aman, dan mudah dikembangkan di masa mendatang. Pembaruan ini menjadikan DW Global SH tidak hanya sebagai komponen eksekusi, tetapi juga sebagai fondasi utama dalam implementasi sistem otomasi konversi dan validasi data di lingkungan PT BCA.



```
execute_pipeline() {
    set -x
    local SRC_DIR=$1
    local TMP_DIR=$2
    local TGT_DIR=$3
    local META_DIR=$4
    local APP_ID=$5
    local FILE_TYPE=$6
    local DELIMITER=$7
    local STD_PATH=$8
    local DATE_PARAM=$9
    local PLACEHOLDER=${10}
    shift 10
    local INPUT_FILES=("$@")

    local GLOBAL_STATUS=0
    local -a FILES_TO_UPDATE=()
    local -a DISTRIB_ERRORS=()
    local -a ERR_FILES=()
    local -a ERR_MSGS=()
    local -a SUCCESS_FILES=()

    echo "[INFO] Starting file conversion and validation..."

    for item in "${INPUT_FILES[@]}"; do
        echo "------------------------------------------------"
        echo "[INFO] Processing file: ${item}"
        local FILE_ERR=0
        local tmp_out

        # 1. Conversion process
        set -x
        if tmp_out=$(convert_input "$SRC_DIR" "$TMP_DIR" "$STD_PATH" "$item" "$FILE_TYPE" "$DELIMITER" "$DATE_PARAM" "$PLACEHOLDER" 2>&1)
        then
            echo "[SUCCESS] Conversion succeeded for ${item}"
            rm -f "$STD_PATH$item"."$FILE_TYPE"
        else
            echo "[ERROR] Conversion failed for ${item}"
            FILE_ERR=1
            ERR_FILES+=("${item}")
            ERR_MSGS+=("$tmp_out")
            GLOBAL_STATUS=1
            continue
        fi
        set +x
```

Gambar 3.6 Script SH Global DW

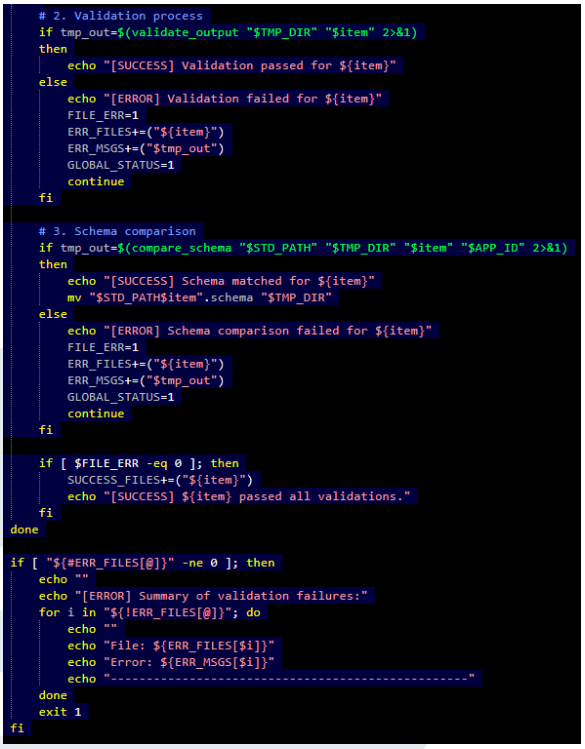
Selain untuk memperluas fungsi, enhancement ini juga ditujukan untuk meningkatkan efisiensi dan kemudahan pemeliharaan sistem dalam jangka panjang. Dengan struktur baru yang lebih modular, setiap pembaruan logika atau penambahan fitur dapat dilakukan secara terpusat pada *Global SH*, tanpa perlu melakukan perubahan di masing-masing script proyek. Pendekatan ini memperkuat keseragaman standar pengolahan data, mempercepat proses pengembangan, serta mendukung skalabilitas sistem otomasi yang digunakan perusahaan. Dengan demikian, Global SH tidak hanya berfungsi sebagai inti proses pemrosesan data, tetapi juga sebagai kerangka kerja utama yang adaptif terhadap kebutuhan bisnis dan perkembangan teknologi data yang terus berkembang.

Gambar 3.6 menampilkan bagian awal dari fungsi *execute_pipeline()* yang berfungsi sebagai pengendali utama dalam proses otomatisasi konversi dan validasi file. Pada bagian ini, dilakukan inisialisasi variabel-variabel lokal yang digunakan untuk mendefinisikan direktori sumber, direktori sementara, direktori tujuan, serta lokasi *metadata*. Selain itu, parameter penting seperti tipe file, pembatas kolom, dan tanggal

pemrosesan juga ditetapkan agar sistem dapat menyesuaikan proses konversi sesuai dengan konfigurasi yang berlaku. Inisialisasi ini menjadi tahap krusial karena memastikan seluruh variabel dan parameter yang digunakan pada fungsi berikutnya memiliki nilai yang konsisten dan terdefinisi dengan baik.

Setelah tahap inisialisasi, fungsi menampilkan informasi status proses untuk menandakan dimulainya tahap konversi dan validasi data. Daftar file yang akan diproses disusun dalam bentuk *array* agar sistem dapat mengeksekusi setiap file secara berurutan dalam satu alur kerja yang terotomatisasi. Melalui mekanisme ini, fungsi dapat memastikan bahwa setiap file yang masuk ke sistem akan melewati tahapan pemrosesan yang sama, mulai dari pembacaan file, pengonversian format, hingga validasi hasil. Pendekatan berurutan ini juga membantu menjaga konsistensi hasil konversi, terutama ketika sistem menangani sejumlah besar file dengan format dan struktur yang seragam.

Bagian akhir dari cuplikan ini menunjukkan tahap awal proses konversi yang dilakukan menggunakan fungsi konversi khusus. Pada tahap ini, setiap file diuji keberhasilannya untuk dikonversi sesuai format standar yang digunakan oleh sistem penyimpanan data. Jika proses konversi berhasil, file sementara yang digunakan akan dihapus untuk menjaga efisiensi ruang penyimpanan. Sebaliknya, apabila proses gagal, sistem akan mencatat nama file serta pesan kesalahan untuk ditinjau pada tahap validasi akhir. Dengan demikian, bagian awal fungsi ini memiliki peran penting dalam mengatur struktur kerja *pipeline* dan memastikan proses konversi berlangsung secara sistematis serta terdokumentasi dengan baik.



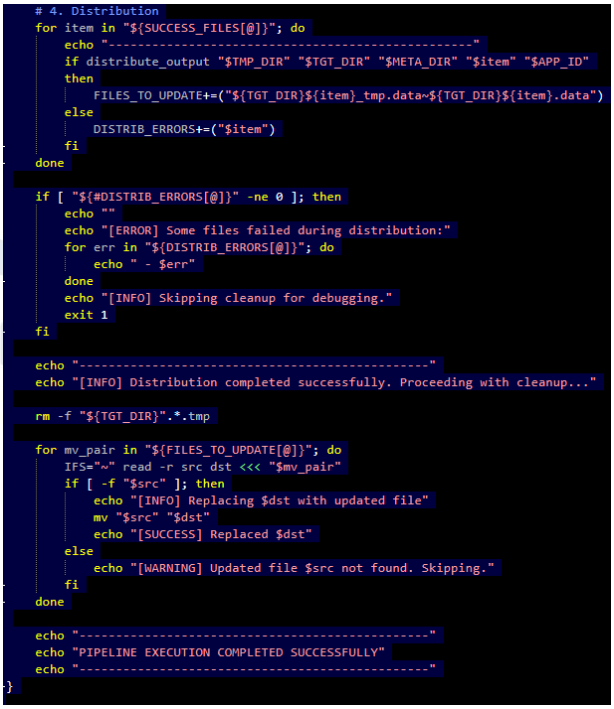
Gambar 3.7 Script Execute Pipeline

Gambar 3.7 menampilkan bagian tengah dari fungsi *execute_pipeline()* yang berfokus pada proses validasi hasil konversi dan perbandingan struktur data. Pada tahap ini, sistem melakukan pemeriksaan terhadap file yang telah dikonversi untuk memastikan bahwa hasil konversi sesuai dengan ketentuan yang berlaku. Fungsi validasi dijalankan secara otomatis untuk memverifikasi integritas file serta memastikan bahwa data yang dihasilkan tidak mengalami kerusakan atau ketidaksesuaian format. Apabila proses validasi berhasil, sistem menampilkan pesan keberhasilan; namun, jika terjadi kegagalan, sistem mencatat file yang bermasalah beserta pesan kesalahannya untuk ditinjau pada tahap evaluasi.

Setelah validasi selesai, tahap selanjutnya adalah perbandingan struktur atau schema comparison antara file hasil konversi dengan file acuan. Tahapan ini bertujuan memastikan bahwa format dan struktur data pada file baru sesuai dengan standar yang ditetapkan oleh sistem. Jika

ditemukan ketidaksesuaian, sistem secara otomatis menandai file tersebut sebagai gagal dan melanjutkan pemrosesan pada file berikutnya tanpa menghentikan keseluruhan alur kerja. Dengan demikian, proses ini tidak hanya menjamin kesesuaian format data, tetapi juga meningkatkan ketahanan sistem terhadap kesalahan file individual tanpa mengganggu proses file lainnya.

Bagian akhir dari cuplikan ini menunjukkan mekanisme pelaporan kesalahan yang merangkum seluruh file yang gagal divalidasi beserta pesan error yang terkait. Ringkasan ini ditampilkan secara otomatis setelah seluruh file selesai diproses untuk memudahkan proses identifikasi dan perbaikan. Apabila seluruh file berhasil melewati tahapan validasi dan perbandingan struktur tanpa kendala, sistem akan menandai file tersebut sebagai berhasil dan siap didistribusikan. Dengan adanya tahap validasi dan pelaporan yang terstruktur ini, keandalan proses otomasi data dapat terjamin, sekaligus meminimalkan risiko kesalahan dalam integrasi data pada sistem yang lebih besar.



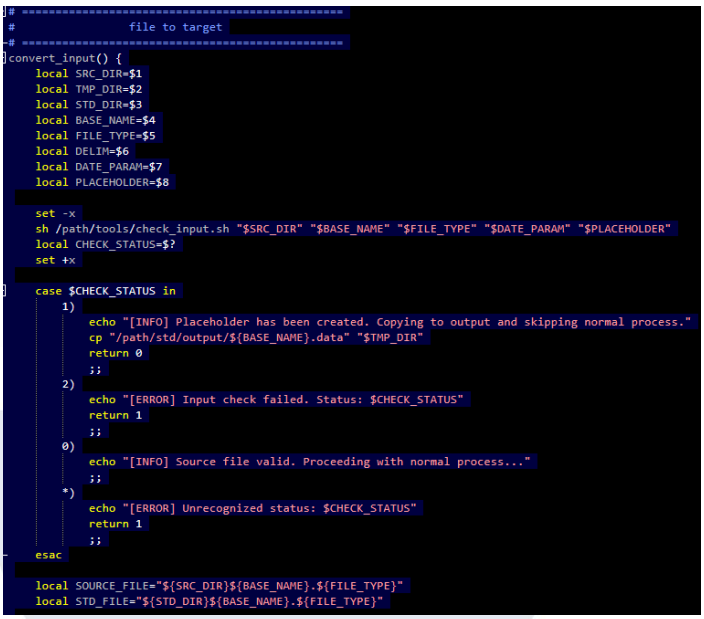
Gambar 3.8 Proses akhir execute pipeline

Gambar 3.8 memperlihatkan bagian akhir dari fungsi *execute_pipeline()* yang berfokus pada proses distribusi hasil konversi dan pembersihan file setelah seluruh tahapan utama selesai dijalankan. Pada bagian ini, sistem menyalin file yang telah melalui tahap validasi ke direktori tujuan sesuai struktur yang sudah ditetapkan. File yang berhasil dikonversi dan diverifikasi kemudian dikirim ke lokasi penyimpanan akhir yang terhubung dengan sistem pengelolaan data. Jika terjadi kesalahan dalam proses distribusi, sistem akan mencatat file yang bermasalah dan menampilkan pesan kesalahan agar dapat dilakukan pengecekan lebih lanjut.

Setelah distribusi selesai, sistem melakukan pemeriksaan tambahan untuk memastikan seluruh file berhasil dipindahkan dengan benar. Jika terdapat file yang gagal dipindahkan, proses pembersihan akan ditunda dan file tersebut akan ditandai untuk kebutuhan pemeriksaan. Namun, apabila semua file berhasil didistribusikan, sistem akan melanjutkan ke tahap pembersihan direktori sementara dengan menghapus file yang tidak lagi diperlukan. Langkah ini dilakukan untuk menjaga efisiensi ruang penyimpanan serta memastikan direktori kerja tetap rapi dan siap digunakan kembali pada proses berikutnya.

Bagian akhir dari fungsi ini menunjukkan tahap penutup, di mana sistem mengganti file lama dengan versi terbaru yang telah melewati proses konversi dan validasi. File yang telah diperbarui menjadi hasil akhir dari keseluruhan proses. Fungsi ini diakhiri dengan pesan keberhasilan yang menandakan bahwa seluruh tahapan telah berjalan dengan baik. Secara keseluruhan, *execute_pipeline()* berfungsi untuk mengotomatisasi proses pengolahan data sekaligus menjaga keakuratan dan kelancaran alur kerja. Setelah fungsi utama ini selesai dijalankan, sistem menggunakan beberapa fungsi tambahan yang memiliki tugas spesifik pada tiap tahapan, seperti konversi, validasi, distribusi, dan

pembersihan data, yang bekerja secara terpisah namun tetap saling terhubung dalam satu sistem.



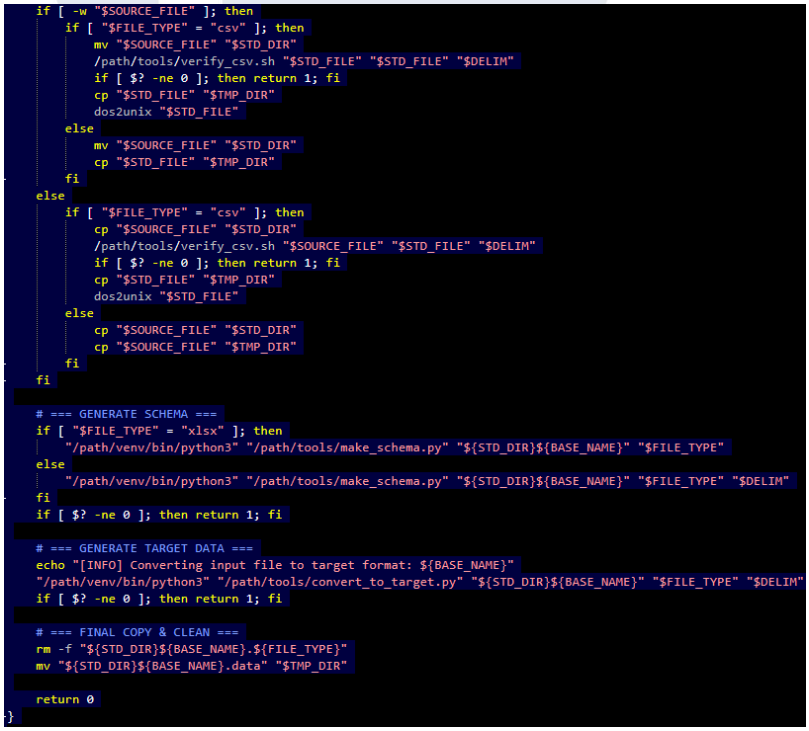
Gambar 3.9 Fungsi Convert

Gambar 3.9 memperlihatkan bagian awal dari fungsi *convert_input()* yang digunakan untuk memulai proses konversi file dari format sumber ke format standar sistem. Pada bagian ini, dilakukan inisialisasi sejumlah variabel lokal yang berfungsi untuk menentukan lokasi direktori sumber, direktori sementara, direktori standar, serta parameter tambahan seperti jenis file, pembatas kolom, tanggal pemrosesan, dan status *placeholder*. Selanjutnya, sistem menjalankan fungsi pemeriksaan awal terhadap file yang akan diproses menggunakan skrip terpisah untuk memastikan bahwa file tersebut valid dan dapat diolah.

Hasil pemeriksaan disimpan dalam variabel status, yang kemudian dievaluasi melalui struktur case untuk menentukan langkah berikutnya. Apabila ditemukan file *placeholder*, sistem akan langsung menyalinnya ke direktori sementara dan melewati proses konversi normal. Sebaliknya, jika terjadi kesalahan atau file tidak valid, proses akan dihentikan dan pesan kesalahan ditampilkan. Jika file sumber dinyatakan valid, sistem

melanjutkan ke tahapan konversi berikutnya dengan menandai status file sebagai siap diproses. Mekanisme ini memungkinkan sistem untuk mendeteksi kondisi file sejak awal sehingga dapat meminimalkan kesalahan pada tahap pemrosesan selanjutnya.

Bagian akhir pada cuplikan ini menunjukkan penentuan jalur file sumber dan file standar yang akan digunakan dalam proses konversi. Kedua variabel tersebut berperan penting dalam memastikan sistem dapat mengakses lokasi file yang benar selama tahap transformasi berlangsung. Dengan struktur yang sistematis, bagian awal fungsi ini memastikan bahwa proses konversi berjalan secara terkontrol, hanya mengeksekusi file yang telah memenuhi kriteria, dan menjaga integritas data sejak tahap awal pemrosesan.



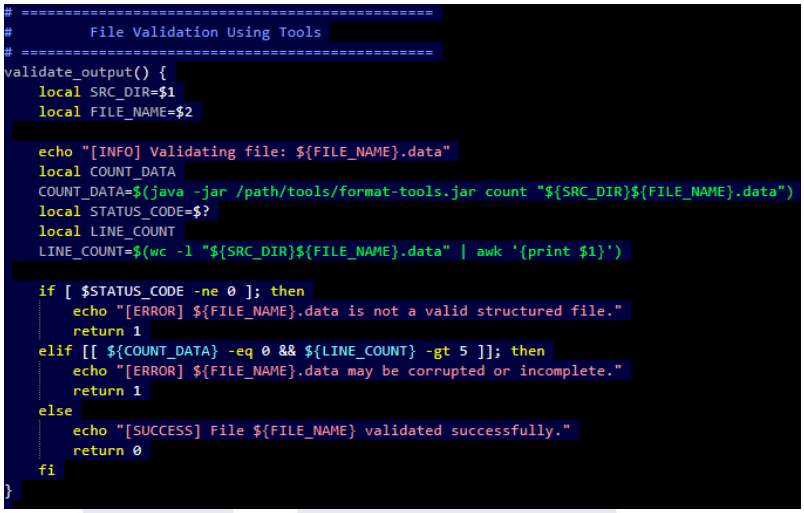
Gambar 3.10 Fungsi Convert untuk distribusi

Gambar 3.10 menampilkan bagian akhir dari fungsi *convert_input()* yang berfokus pada proses pemindahan file, pembuatan schema, konversi data, serta tahap akhir pembersihan file sementara. Pada bagian

ini, sistem memeriksa jenis file yang akan diproses, kemudian menentukan langkah pemindahan dan verifikasi yang sesuai. Untuk file dengan tipe CSV, dilakukan proses verifikasi struktur menggunakan skrip validasi eksternal guna memastikan kesesuaian format sebelum dilanjutkan ke tahap konversi. Apabila hasil verifikasi tidak memenuhi ketentuan, sistem akan menghentikan proses guna mencegah kesalahan data.

Setelah file dinyatakan valid, sistem melanjutkan ke tahap pembuatan schema yang berfungsi untuk mendefinisikan struktur data sesuai standar sistem. Proses ini dijalankan secara otomatis menggunakan skrip berbasis Python, baik untuk file bertipe CSV maupun XLSX. Hasil schema yang dihasilkan akan menjadi acuan dalam pembentukan file akhir agar seluruh atribut data memiliki definisi dan tipe yang seragam. Mekanisme ini memastikan konsistensi antara data sumber dan hasil konversi, sekaligus memudahkan proses integrasi pada tahap selanjutnya.

Tahap terakhir pada bagian ini adalah proses konversi file ke format target yang diikuti dengan pembersihan file sementara. Setelah konversi selesai dilakukan, sistem menghapus file asli yang telah diproses untuk menjaga efisiensi ruang penyimpanan, kemudian memindahkan hasil akhir ke direktori keluaran. Proses ini menandai selesainya fungsi *convert_input()* dengan memastikan seluruh tahapan, mulai dari verifikasi, pembuatan schema, hingga konversi, berjalan secara terstruktur dan sesuai dengan standar integrasi data yang diterapkan.

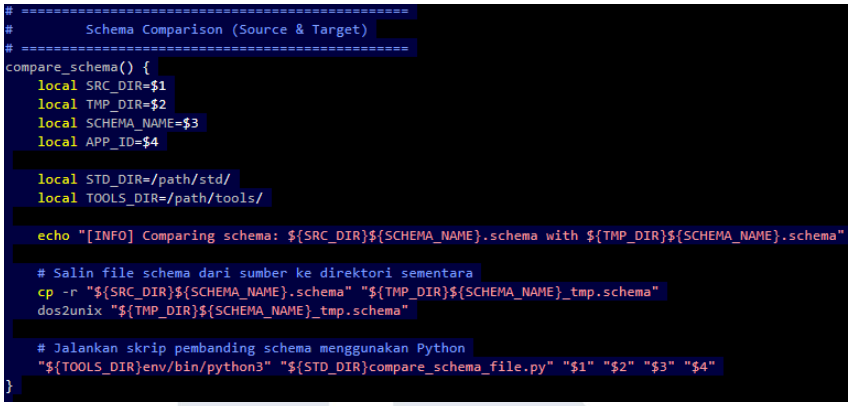


```
# ============================================
#         File Validation Using Tools
# ============================================
validate_output() {
    local SRC_DIR=$1
    local FILE_NAME=$2

    echo "[INFO] Validating file: ${FILE_NAME}.data"
    local COUNT_DATA
    COUNT_DATA=$(java -jar /path/tools/format-tools.jar count "${SRC_DIR}${FILE_NAME}.data")
    local STATUS_CODE=$?
    local LINE_COUNT
    LINE_COUNT=$(wc -l "${SRC_DIR}${FILE_NAME}.data" | awk '{print $1}')

    if [ $STATUS_CODE -ne 0 ]; then
        echo "[ERROR] ${FILE_NAME}.data is not a valid structured file."
        return 1
    elif [[ ${COUNT_DATA} -eq 0 && ${LINE_COUNT} -gt 5 ]]; then
        echo "[ERROR] ${FILE_NAME}.data may be corrupted or incomplete."
        return 1
    else
        echo "[SUCCESS] File ${FILE_NAME} validated successfully."
        return 0
    fi
}
```

Gambar 3.11 Fungsi Validate

Gambar 3.11 menampilkan fungsi *validate_output()* yang digunakan untuk melakukan pemeriksaan terhadap hasil file yang telah dikonversi pada tahap sebelumnya. Fungsi ini memastikan bahwa file hasil proses konversi memiliki struktur dan isi data yang sesuai dengan standar sistem sebelum dilanjutkan ke tahap distribusi. Pada awal fungsi, dilakukan inisialisasi variabel untuk menyimpan lokasi direktori sumber serta nama file yang akan divalidasi. Setelah itu, sistem menjalankan perintah berbasis Java melalui tools eksternal guna menghitung jumlah data dalam file dan menilai status keberhasilan proses validasi.

Hasil dari pemeriksaan ini kemudian dibandingkan dengan jumlah baris yang terdapat pada file menggunakan perintah tambahan berbasis Unix. Jika sistem mendeteksi adanya kesalahan format atau file tidak dapat dibaca, maka proses akan dihentikan secara otomatis dan pesan kesalahan akan ditampilkan. Sebaliknya, apabila jumlah data valid dan struktur file sesuai, fungsi akan memberikan notifikasi bahwa file telah melewati tahap validasi dengan baik. Melalui mekanisme ini, fungsi *validate_output()* berperan penting dalam menjaga integritas dan konsistensi data agar hanya file yang valid dan bebas dari kerusakan yang dapat dilanjutkan ke tahap pemrosesan berikutnya.



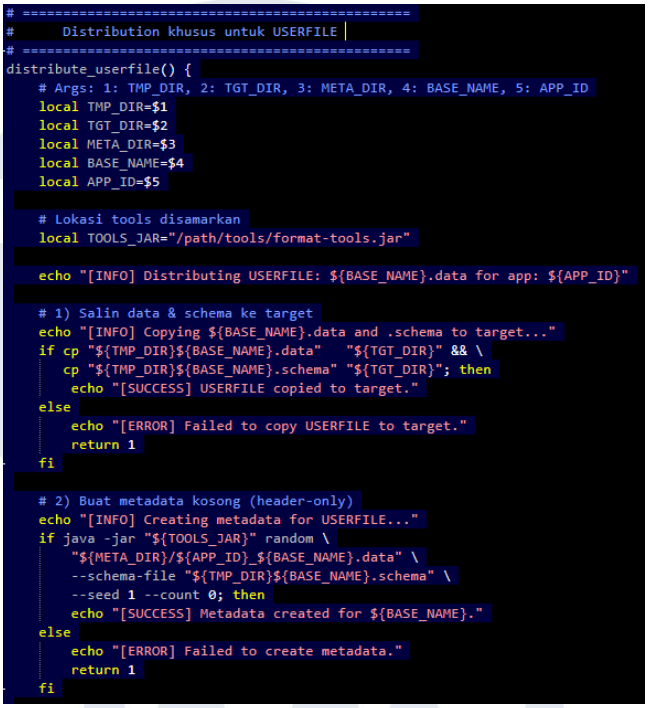
Gambar 3.12 Fungsi komparasi

Seperti yang terlihat pada Gambar 3.12, fungsi *compare_schema()* yang digunakan untuk melakukan proses perbandingan struktur antara dua berkas schema dari direktori sumber dan direktori sementara. Tujuan utama fungsi ini adalah memastikan bahwa format, atribut, serta tipe data yang terdapat dalam file hasil konversi telah sesuai dengan standar struktur data yang berlaku. Pada bagian awal, dilakukan inisialisasi parameter yang digunakan untuk menentukan lokasi file sumber, direktori sementara, serta identitas aplikasi yang sedang diproses. Seluruh parameter tersebut berfungsi untuk mengatur jalur eksekusi dan menjaga agar proses pembandingan berlangsung secara terarah.

Setelah inisialisasi selesai, sistem menyalin file schema dari lokasi sumber ke direktori sementara untuk keperluan verifikasi. Proses penyalinan ini diikuti oleh konversi format baris menggunakan perintah *dos2unix* agar file dapat diproses tanpa gangguan perbedaan format sistem operasi. Tahap ini penting untuk memastikan kompatibilitas antara berkas yang dibandingkan, terutama ketika file berasal dari lingkungan atau sistem yang berbeda.

Selanjutnya, fungsi menjalankan skrip berbasis Python untuk membandingkan dua berkas schema tersebut. Skrip ini bertugas untuk mengidentifikasi adanya perbedaan struktur, elemen, atau definisi kolom yang tidak sesuai dengan standar yang telah ditetapkan. Apabila hasil

perbandingan menunjukkan kesesuaian, maka file dianggap valid dan siap untuk digunakan pada proses distribusi data berikutnya. Dengan adanya fungsi ini, sistem dapat memastikan integritas dan keseragaman struktur data antarfile, sehingga meminimalkan risiko kesalahan integrasi pada tahap pemrosesan lanjutan.



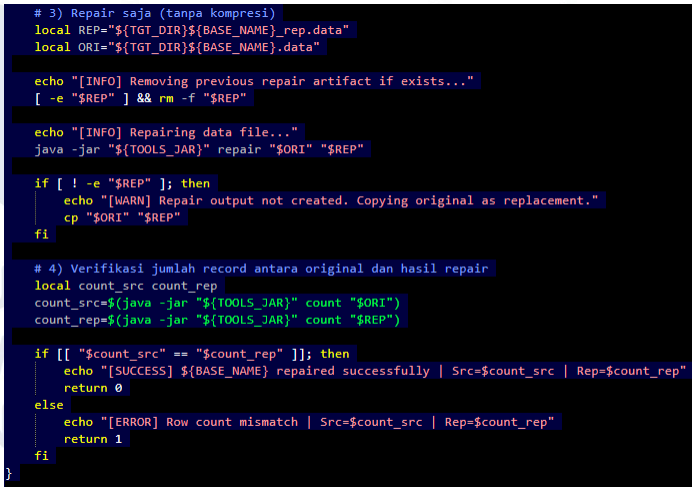
```
# ================================================
#       Distribution khusus untuk USERFILE
# ================================================
distribute_userfile() {
    # Args: 1: TMP_DIR, 2: TGT_DIR, 3: META_DIR, 4: BASE_NAME, 5: APP_ID
    local TMP_DIR=$1
    local TGT_DIR=$2
    local META_DIR=$3
    local BASE_NAME=$4
    local APP_ID=$5

    # Lokasi tools disamarkan
    local TOOLS_JAR="/path/tools/format-tools.jar"

    echo "[INFO] Distributing USERFILE: ${BASE_NAME}.data for app: ${APP_ID}"

    # 1) Salin data & schema ke target
    echo "[INFO] Copying ${BASE_NAME}.data and .schema to target..."
    if cp "${TMP_DIR}${BASE_NAME}.data"   "${TGT_DIR}" && \
       cp "${TMP_DIR}${BASE_NAME}.schema" "${TGT_DIR}"; then
        echo "[SUCCESS] USERFILE copied to target."
    else
        echo "[ERROR] Failed to copy USERFILE to target."
        return 1
    fi

    # 2) Buat metadata kosong (header-only)
    echo "[INFO] Creating metadata for USERFILE..."
    if java -jar "${TOOLS_JAR}" random \
        "${META_DIR}/${APP_ID}_${BASE_NAME}.data" \
        --schema-file "${TMP_DIR}${BASE_NAME}.schema" \
        --seed 1 --count 0; then
        echo "[SUCCESS] Metadata created for ${BASE_NAME}."
    else
        echo "[ERROR] Failed to create metadata."
        return 1
    fi
```

Gambar 3.13 Fungsi Distribusi

Gambar 3.13 menampilkan fungsi *distribute_userfile()* yang digunakan untuk mengatur proses distribusi hasil konversi file ke direktori tujuan serta pembuatan metadata yang menyertainya. Pada bagian awal fungsi, dilakukan inisialisasi variabel untuk menentukan lokasi direktori sementara, direktori target, direktori metadata, serta identitas file dan aplikasi yang sedang diproses. Parameter ini diperlukan agar sistem dapat menjalankan distribusi file dengan arah yang sesuai dan memastikan setiap data hasil konversi tersimpan di lokasi yang tepat. Selain itu, lokasi tools eksternal juga ditetapkan di awal untuk mendukung proses otomatisasi yang dilakukan melalui perintah berbasis Java.

Setelah seluruh variabel diinisialisasi, sistem menyalin file hasil konversi beserta berkas schema-nya dari direktori sementara ke direktori target. Tahapan ini merupakan langkah penting dalam proses integrasi data, karena memastikan bahwa hasil pemrosesan dapat diakses oleh sistem penyimpanan utama dengan struktur file yang lengkap. Apabila proses penyalinan berjalan dengan baik, sistem menampilkan pesan keberhasilan; namun, jika terjadi kegagalan, proses akan dihentikan dan pesan kesalahan akan ditampilkan sebagai indikator perbaikan.

Tahapan berikutnya adalah pembuatan berkas metadata yang berfungsi sebagai deskripsi atau penanda identitas data hasil konversi. Proses ini dilakukan secara otomatis menggunakan perintah *Java* dengan referensi schema yang telah dihasilkan sebelumnya. *Metadata* tersebut berperan penting dalam mendukung integrasi dan pengenalan file di dalam sistem pengelolaan data. Melalui fungsi ini, sistem memastikan bahwa setiap userfile yang telah melalui proses konversi dan validasi akan disertai dengan metadata yang sesuai, sehingga dapat diidentifikasi dan diolah dengan benar pada tahap pemrosesan data berikutnya.



Gambar 3.14 Fungsi Repair

Gambar 3.14 memperlihatkan bagian akhir dari fungsi *distribute_userfile()* yang berfokus pada proses perbaikan file serta

verifikasi kesesuaian hasil perbaikan terhadap file aslinya. Pada tahap ini, sistem menjalankan perintah perbaikan menggunakan tools eksternal berbasis Java untuk memastikan bahwa file hasil konversi bebas dari kerusakan dan dapat dibaca dengan benar oleh sistem. Sebelum proses perbaikan dimulai, sistem terlebih dahulu menghapus file hasil perbaikan sebelumnya apabila masih tersimpan, guna menghindari tumpang tindih data dari eksekusi sebelumnya.

Apabila hasil perbaikan tidak berhasil dihasilkan secara otomatis, sistem akan menyalin kembali file asli sebagai pengganti agar alur distribusi tetap dapat dilanjutkan tanpa mengganggu kestabilan proses. Langkah ini dilakukan sebagai bentuk mekanisme penanganan kesalahan agar data tetap dapat disalurkan dengan aman dan terkendali. Setelah proses perbaikan selesai, dilakukan perhitungan jumlah record antara file asli dan hasil perbaikan untuk memastikan bahwa tidak terjadi kehilangan atau duplikasi data selama proses berlangsung.

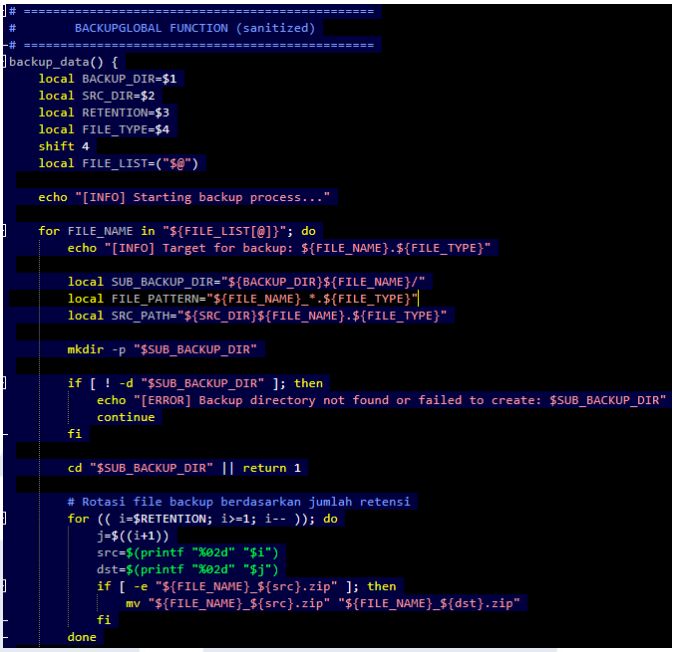
Tahapan verifikasi ini menjadi bagian penting dalam menjaga keandalan dan konsistensi data di seluruh pipeline pemrosesan. Apabila jumlah *record* antara file asli dan hasil perbaikan sama, maka file dinyatakan berhasil diperbaiki dan siap digunakan untuk tahap berikutnya. Namun, apabila ditemukan perbedaan, sistem akan memberikan peringatan dan menghentikan proses agar data yang tidak sesuai tidak dilanjutkan ke sistem utama. Melalui mekanisme ini, fungsi distribute_userfile() memastikan bahwa setiap file yang disalurkan telah melewati proses validasi struktural dan kualitas yang ketat.

3.3.1.2 Enhancement DWBACKUP GLOBAL

Enhancement pada DWBackup Global SH dilakukan untuk memperkuat mekanisme pencadangan data yang menjadi bagian penting dari alur otomasi di sistem Data Warehouse. Sebelumnya, proses backup dilakukan secara manual atau melalui skrip terpisah yang belum sepenuhnya terintegrasi dengan pipeline utama. Melalui pengembangan

ini, DWBackup Global diperbarui agar mampu melakukan proses pencadangan secara otomatis setelah file hasil konversi dari DW Global selesai diproses. Fungsi utama yang ditambahkan mencakup kompresi file Avro dan AVSC ke dalam format arsip (zip), penghapusan salinan sementara, serta rotasi file berdasarkan kebijakan retensi perusahaan. Dengan integrasi penuh ke dalam sistem DW, proses backup kini dapat dijalankan secara berurutan tanpa intervensi manual, sehingga efisiensi dan keamanan data dapat lebih terjamin.

Selain meningkatkan otomatisasi, pembaruan pada DWBackup Global SH juga difokuskan pada aspek keandalan dan auditabilitas proses. Setiap tahapan pencadangan dilengkapi dengan mekanisme pengecekan status keberhasilan dan validasi ukuran arsip untuk memastikan file yang dicadangkan telah terbentuk dengan benar dan tidak korup. Proses ini juga menghasilkan log terperinci sebagai dokumentasi setiap aktivitas pencadangan, yang berfungsi sebagai referensi untuk pemantauan dan audit data di kemudian hari. Dengan penerapan enhancement ini, DWBackup Global SH berperan tidak hanya sebagai skrip pendukung, tetapi juga sebagai komponen integral yang menjaga kontinuitas dan konsistensi data dalam siklus pemrosesan otomatis di lingkungan Data Warehouse PT Bank Central Asia, Tbk.



Gambar 3.15 Fungsi Backup

Gambar 3.15 menampilkan bagian awal dari fungsi *backup_data()* yang merupakan bagian dari modul *BACKUPGLOBAL*. Fungsi ini dirancang untuk mengelola proses pencadangan otomatis terhadap file hasil pemrosesan data. Pada bagian awal, dilakukan inisialisasi variabel lokal yang mendefinisikan lokasi direktori sumber, direktori tujuan pencadangan, jumlah retensi file yang akan disimpan, serta jenis file yang diproses. Parameter tersebut digunakan untuk memastikan bahwa sistem dapat menjalankan proses pencadangan dengan terstruktur dan sesuai kebijakan penyimpanan data yang berlaku.

Setelah proses inisialisasi, fungsi melakukan pemeriksaan terhadap direktori tujuan untuk memastikan bahwa folder pencadangan tersedia dan dapat digunakan. Jika direktori belum ada, sistem akan membuatnya secara otomatis; namun jika pembuatan gagal, proses akan dilewati untuk mencegah gangguan pada file lainnya. Selanjutnya, dilakukan proses rotasi file cadangan berdasarkan jumlah retensi yang telah ditetapkan. File lama akan diganti nama dan digeser sesuai urutan penyimpanan agar

sistem tetap dapat mempertahankan beberapa versi file cadangan tanpa menumpuk secara berlebihan.

Tahapan rotasi ini berfungsi untuk menjaga keteraturan arsip file cadangan dan mencegah penggunaan ruang penyimpanan yang berlebihan. Dengan mekanisme ini, sistem hanya menyimpan versi file dalam jumlah tertentu yang dianggap relevan, sementara versi lama secara otomatis digantikan oleh yang terbaru. Bagian awal dari fungsi ini menunjukkan bagaimana sistem pencadangan dikembangkan secara terstruktur untuk memastikan keandalan, efisiensi, serta ketertelusuran file dalam proses pengelolaan data.

```
        cd "$SUB_BACKUP_DIR" || return 1

        # Rotasi file backup berdasarkan jumlah retensi
        for (( i=$RETENTION; i>=1; i-- )); do
            j=$((i+1))
            src=$(printf "%02d" "$i")
            dst=$(printf "%02d" "$j")
            if [ -e "${FILE_NAME}_${src}.zip" ]; then
                mv "${FILE_NAME}_${src}.zip" "${FILE_NAME}_${dst}.zip"
            fi
        done

        cd "$SRC_DIR" || return 1
        matched_files=($(find . -maxdepth 1 -type f -name "${FILE_PATTERN}" -printf "%P\n"))

        if [ ${#matched_files[@]} -eq 0 ] && [ -f "$SRC_PATH" ]; then
            matched_files=("${FILE_NAME}.${FILE_TYPE}")
        fi

        if [ ${#matched_files[@]} -eq 0 ]; then
            echo "[WARNING] No matching files found for ${FILE_NAME}.${FILE_TYPE}"
            continue
        fi

        echo "[INFO] Compressing files: ${matched_files[*]}"
        zip "${SUB_BACKUP_DIR}${FILE_NAME}_01.zip" "${matched_files[@]}" >/dev/null

        if [ -s "${SUB_BACKUP_DIR}${FILE_NAME}_01.zip" ]; then
            echo "[SUCCESS] Backup created: ${SUB_BACKUP_DIR}${FILE_NAME}_01.zip"
        else
            echo "[ERROR] Failed to create valid backup archive for ${FILE_NAME}"
            continue
        fi

        echo "[INFO] Cleaning up original files..."
        for f in "${matched_files[@]}"; do
            rm -f "$f"
        done
        rm -f "${FILE_NAME}.data"

        echo "[SUCCESS] Backup completed for: ${FILE_NAME}"
    done

    echo "[INFO] All backup operations finished successfully."
    return 0
}
```

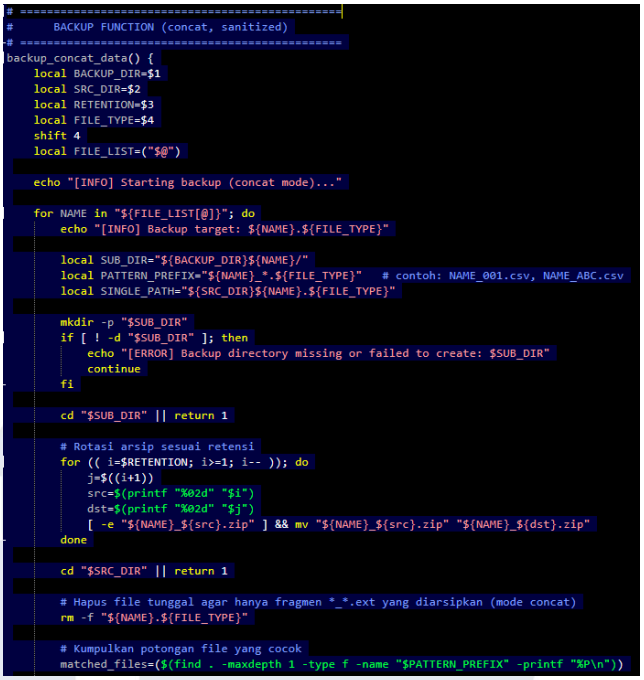
Gambar 3.16 Lanjutan Fungsi Backup

Gambar 3.16 Gambar tersebut menampilkan bagian lanjutan dari fungsi *backup_data()* yang menunjukkan tahapan utama dalam proses pencadangan dan pembersihan file. Setelah proses rotasi selesai, sistem melakukan identifikasi terhadap file yang akan dicadangkan dengan mencari pola nama file tertentu di direktori sumber. Apabila file yang sesuai ditemukan, sistem akan melanjutkan proses kompresi

menggunakan format arsip terstandarisasi. Proses kompresi ini bertujuan untuk menghemat ruang penyimpanan dan menjaga integritas data cadangan, sehingga file tetap dapat digunakan kembali bila diperlukan.

Setelah proses kompresi berhasil dijalankan, sistem melakukan verifikasi terhadap hasil arsip untuk memastikan bahwa berkas cadangan telah terbentuk dengan benar dan dapat dibaca. Apabila proses pembuatan arsip gagal atau menghasilkan file kosong, sistem akan menampilkan pesan kesalahan dan melewati file tersebut untuk mencegah potensi gangguan pada pencadangan file lain. Langkah ini menunjukkan bahwa sistem telah dilengkapi dengan mekanisme deteksi kesalahan otomatis guna menjamin keandalan proses pencadangan secara keseluruhan.

Tahap terakhir dari fungsi ini adalah proses pembersihan (cleanup), di mana file sumber yang telah berhasil dicadangkan akan dihapus untuk mengurangi duplikasi dan menjaga efisiensi ruang penyimpanan. Setelah seluruh file selesai dicadangkan dan diverifikasi, sistem menampilkan pesan keberhasilan yang menandakan bahwa seluruh proses telah diselesaikan tanpa kendala. Secara keseluruhan, bagian ini menggambarkan bagaimana fungsi backup_data() dirancang dengan pendekatan terstruktur dan defensif, memastikan seluruh data yang diproses terlindungi dengan baik serta mendukung pengelolaan arsip data yang efisien dan aman.



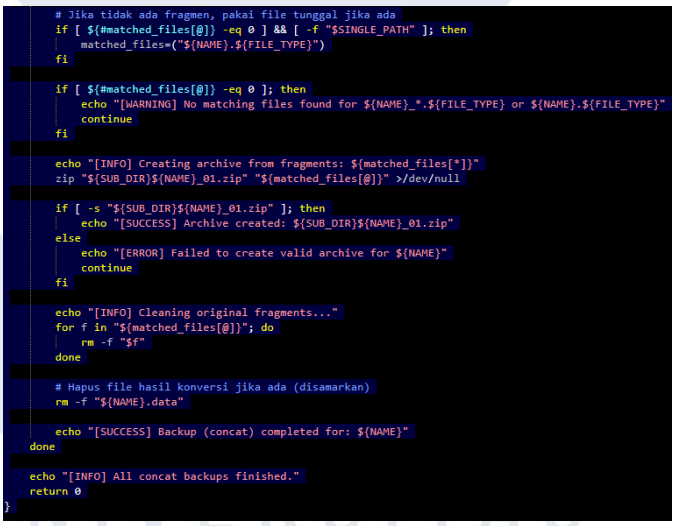
Gambar 3.17 Fungsi Backup Concat

Gambar 3.17 tersebut menampilkan bagian awal dari fungsi *backup_concat_data()* yang merupakan varian dari proses pencadangan otomatis dengan metode *concat*. Fungsi ini dirancang untuk menangani file hasil pemrosesan yang memiliki struktur terpecah menjadi beberapa bagian atau fragmen. Pada tahap awal, sistem melakukan inisialisasi parameter utama seperti direktori sumber, lokasi penyimpanan cadangan, jumlah retensi arsip yang diizinkan, serta tipe file yang akan diproses. Parameter tersebut memastikan bahwa proses pencadangan berjalan sesuai dengan konfigurasi dan kebijakan penyimpanan yang berlaku dalam sistem.

Setelah inisialisasi, fungsi memulai pembuatan direktori pencadangan untuk setiap file target yang terdaftar. Apabila direktori belum tersedia, sistem akan secara otomatis membuatnya, sementara apabila pembuatan gagal, proses pencadangan untuk file tersebut dilewati guna menjaga kestabilan eksekusi keseluruhan. Tahap berikutnya adalah proses rotasi arsip berdasarkan jumlah retensi yang

telah ditentukan. File arsip lama digeser secara berurutan agar versi baru dapat tersimpan tanpa menghapus riwayat versi sebelumnya. Mekanisme ini menjamin bahwa arsip lama tetap tersedia untuk keperluan audit maupun pemulihan data historis.

Selain itu, fungsi ini juga menjalankan proses pembersihan file tunggal agar hanya fragmen data yang relevan yang akan diarsipkan. Hal ini dilakukan untuk menghindari duplikasi dan memastikan bahwa hanya potongan file yang sesuai dengan pola penamaan tertentu yang akan dikompres. Tahapan ini menjadi dasar dari mode *concat*, di mana beberapa potongan data dikonsolidasikan dalam satu proses pencadangan terstruktur. Pendekatan ini memperkuat keandalan sistem dalam mengelola file berukuran besar atau terpecah, sekaligus mempertahankan efisiensi dalam manajemen ruang penyimpanan.



Gambar 3.18 Lanjutan Fungsi Backup Concat

Gambar 3.18 menampilkan bagian akhir dari fungsi *backup_concat_data()* yang menunjukkan tahapan utama dalam proses pembuatan arsip dan pembersihan file sumber. Setelah file fragmen terdeteksi, sistem memverifikasi kembali keberadaannya untuk memastikan bahwa file yang sesuai tersedia sebelum proses kompresi dilakukan. Apabila tidak ditemukan file yang cocok, sistem

menampilkan peringatan dan melewati proses untuk file tersebut. Namun, apabila fragmen ditemukan, sistem akan membuat arsip terkompresi dalam format standar dan menyimpannya pada direktori cadangan yang telah ditentukan. Langkah ini menjamin efisiensi penggunaan ruang penyimpanan sekaligus memastikan setiap file hasil pemrosesan dapat diarsipkan secara sistematis.

Proses verifikasi dilakukan setelah tahap kompresi untuk memastikan bahwa arsip yang dihasilkan valid dan dapat digunakan kembali bila diperlukan. Jika pembuatan arsip berhasil, sistem akan menandai proses pencadangan sebagai berhasil; sebaliknya, jika ditemukan kesalahan, sistem mencatat pesan kesalahan dan melanjutkan ke file berikutnya tanpa menghentikan keseluruhan proses. Pendekatan ini menunjukkan bahwa fungsi telah dirancang dengan toleransi kesalahan yang baik agar proses pencadangan dapat berjalan stabil meskipun terdapat file yang gagal dikompres.

Tahap akhir dari fungsi ini adalah proses pembersihan file sumber, termasuk penghapusan fragmen yang telah berhasil diarsipkan dan file sementara hasil konversi. Langkah ini bertujuan untuk menjaga kebersihan direktori kerja serta menghindari penumpukan data yang tidak lagi diperlukan. Setelah seluruh file selesai diproses, sistem menampilkan pesan konfirmasi bahwa seluruh proses pencadangan dalam mode *concat* telah selesai dengan sukses. Struktur ini menunjukkan bahwa fungsi *backup_concat_data()* dirancang secara efisien dan berlapis untuk menjamin keandalan, keamanan, serta konsistensi dalam proses penyimpanan data hasil otomasi.

3.3.1.3 Enhancement DW CALLER

Enhancement pada DW Caller SH dilakukan untuk meningkatkan efisiensi dan fleksibilitas dalam proses eksekusi fungsi-fungsi utama yang terdapat pada DW Global SH. Sebelumnya, DW Caller hanya berfungsi sebagai skrip pemanggil sederhana yang menjalankan urutan

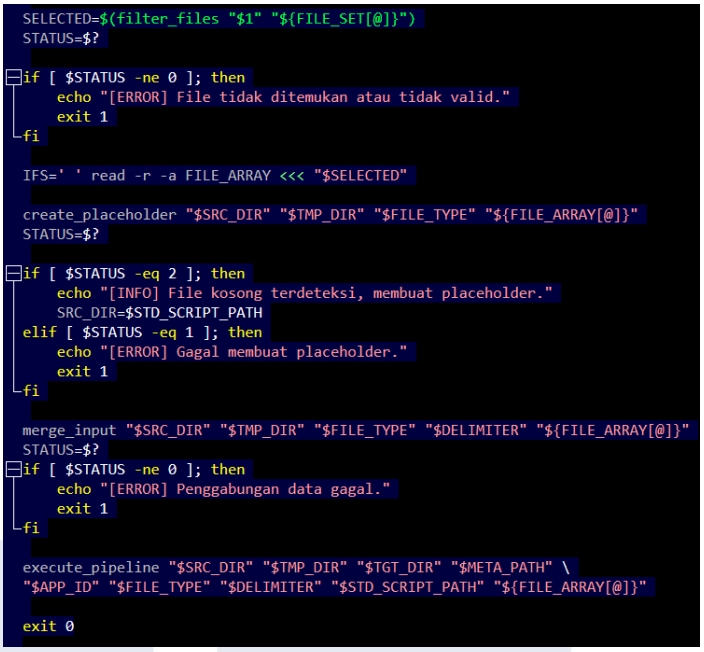
proses secara statis tanpa dukungan parameterisasi dinamis. Melalui pengembangan ini, DW Caller diperbarui agar mampu menginisialisasi variabel seperti nama aplikasi, jenis file, direktori sumber, dan jalur keluaran secara otomatis berdasarkan kebutuhan proyek. Pembaruan ini memungkinkan proses eksekusi berjalan lebih adaptif terhadap berbagai skenario pemrosesan data bisnis yang diterima oleh tim Data Standard. Selain itu, DW Caller kini dilengkapi dengan mekanisme kontrol alur kerja yang lebih baik, termasuk pengecekan file sumber, pembuatan placeholder otomatis, dan penggabungan (concat) file sebelum masuk ke tahap konversi.

Perubahan lain yang signifikan pada DW Caller adalah integrasinya dengan sistem error handling yang lebih informatif dan berlapis. Setiap tahapan eksekusi kini menghasilkan pesan status yang menunjukkan keberhasilan atau kegagalan proses, lengkap dengan indikator untuk identifikasi cepat terhadap penyebab kesalahan. Hal ini memudahkan proses debugging dan memastikan setiap fungsi yang dipanggil dari DW Global dapat berjalan sesuai urutan yang telah ditentukan. Dengan adanya enhancement ini, DW Caller berperan sebagai pengendali utama dalam pipeline otomasi Data Warehouse, yang tidak hanya mengeksekusi fungsi global, tetapi juga menjamin konsistensi, stabilitas, serta keterlacakan proses di setiap tahapan pengolahan data.

Gambar 3.19 Skrip DW Caller

Gambar 3.19 tersebut menampilkan cuplikan konfigurasi *caller* yang menunjukkan pemanggilan fungsi global serta inisialisasi parameter lingkungan yang telah disesuaikan untuk menjaga kerahasiaan sistem. Pada bagian ini, berbagai variabel seperti identitas aplikasi, direktori sumber data, direktori sementara, direktori tujuan, serta jalur metadata diatur untuk memastikan alur pemrosesan data berjalan secara terstruktur. Selain itu, parameter tambahan seperti mode pengecekan, tipe file, dan karakter pembatas kolom juga diinisialisasi guna menyesuaikan proses dengan karakteristik data yang akan diolah.

Daftar berkas input ditetapkan dalam sebuah struktur array yang memungkinkan pemilihan dan penyaringan file dilakukan secara sistematis sesuai kebutuhan proses. Pengaturan tersebut menjadikan caller berfungsi sebagai pengendali urutan eksekusi yang memanggil modul-modul pemrosesan terpusat. Selain itu, caller juga dirancang untuk menangani berbagai kondisi khusus, seperti pembuatan placeholder apabila file sumber tidak ditemukan atau penggabungan beberapa berkas sebelum proses konversi dan distribusi dilakukan. Struktur ini memastikan bahwa setiap tahap pemrosesan data berjalan secara terkoordinasi, efisien, dan sesuai dengan standar otomasi yang diterapkan.



Gambar 3.20 Skrip pemanggilan pada Caller

Gambar 3.20 tersebut menampilkan bagian utama dari script caller yang berfungsi menjalankan proses validasi dan penanganan awal terhadap file bisnis sebelum dilakukan konversi dan distribusi. Pada tahap awal, sistem melakukan pemanggilan fungsi penyaring file untuk memastikan bahwa seluruh berkas yang akan diproses telah terdaftar dan sesuai dengan ketentuan yang berlaku. Mekanisme ini menjadi langkah penting untuk menjamin bahwa hanya file yang memenuhi kriteria validasi yang dapat diproses lebih lanjut dalam sistem otomatisasi.

Setelah proses penyaringan selesai, hasil seleksi file dibaca dan dimasukkan ke dalam struktur *array* agar dapat diproses secara berurutan. Dengan cara ini, setiap file yang teridentifikasi akan diperlakukan secara konsisten sesuai urutan eksekusi yang telah ditentukan. Apabila file yang diharapkan tidak ditemukan atau tidak sesuai dengan daftar yang telah ditetapkan, sistem akan menghentikan proses secara otomatis. Tindakan ini berfungsi sebagai langkah pengamanan untuk mencegah terjadinya kesalahan pada tahap-tahap pemrosesan berikutnya.

Tahapan berikutnya mencakup pembuatan placeholder apabila file sumber tidak tersedia, serta penggabungan data jika terdapat beberapa potongan file yang perlu disatukan sebelum dikonversi. Proses ini dikendalikan secara sistematis dengan mekanisme pemeriksaan status pada setiap tahapan, sehingga jika terjadi kegagalan, sistem akan memberikan pesan kesalahan yang informatif dan menghentikan proses secara aman. Pendekatan ini memastikan bahwa setiap potensi kesalahan dapat diidentifikasi sejak dini, tanpa mengganggu integritas keseluruhan alur kerja.

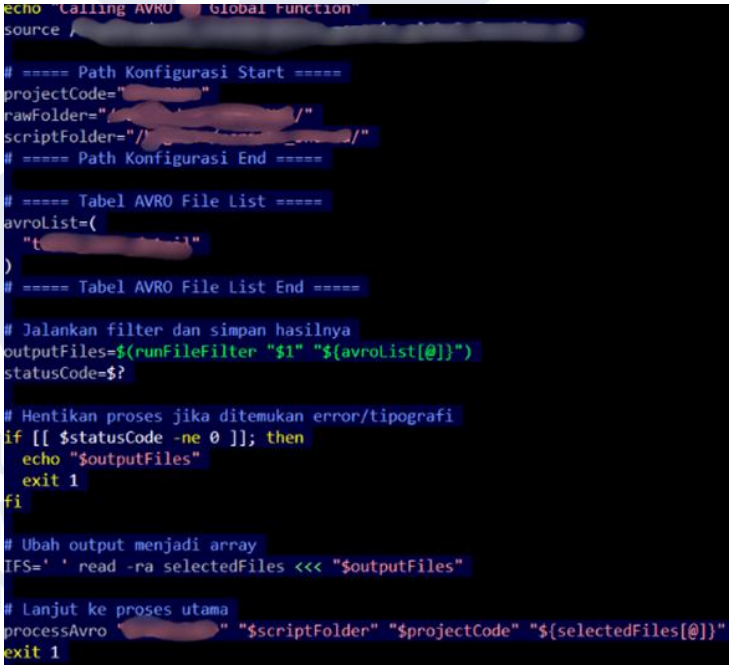
Setelah seluruh file berhasil divalidasi dan dipastikan dalam kondisi siap, fungsi utama dijalankan untuk melakukan pemrosesan dan distribusi hasil konversi ke direktori tujuan yang telah ditentukan. Struktur script caller tersebut menunjukkan bahwa sistem dirancang dengan pendekatan berlapis yang menekankan pada ketelitian, keamanan, dan keandalan proses. Dengan demikian, mekanisme ini tidak hanya memastikan ketepatan dalam pengolahan data, tetapi juga meningkatkan efisiensi serta stabilitas dalam sistem otomasi integrasi data yang diterapkan.

3.3.1.4 Enhancement DW Backup CALLER

Enhancement pada *DWBackup Caller SH* dilakukan untuk menyempurnakan integrasi antara proses utama *DWBackup Global SH* dengan pipeline eksekusi data di sistem *Data Warehouse*. Sebelumnya, *DWBackup Caller* hanya berfungsi sebagai skrip pemicu sederhana yang menjalankan proses pencadangan tanpa mekanisme pengawasan hasil. Melalui pengembangan ini, struktur *DWBackup Caller* diperbarui agar mampu memanggil fungsi pencadangan secara otomatis dengan parameter dinamis, seperti nama file, jenis data, direktori sumber, serta jumlah retensi arsip yang berlaku. Pembaruan ini memastikan bahwa setiap hasil konversi yang telah diproses oleh *DW Global* dapat langsung dicadangkan tanpa perlu intervensi manual. Selain itu, skrip ini juga

dilengkapi dengan sistem pemeriksaan status untuk memverifikasi apakah proses backup berhasil dijalankan dan file cadangan telah terbentuk dengan benar di direktori tujuan.

Peningkatan lainnya terletak pada implementasi logging dan validasi hasil backup yang lebih komprehensif. *DWBackup Caller* kini menghasilkan catatan eksekusi yang mendetail, mencakup waktu proses, jumlah file yang dicadangkan, serta status keberhasilan setiap tahapan. Apabila terjadi kegagalan pada salah satu langkah, sistem akan menampilkan pesan kesalahan yang spesifik dan menghentikan proses secara aman tanpa mengganggu data lainnya. Dengan adanya pembaruan ini, *DWBackup Caller* tidak hanya berfungsi sebagai penghubung antara sistem utama dan proses pencadangan, tetapi juga sebagai komponen kontrol yang menjamin keamanan, integritas, dan keterlacakan setiap aktivitas backup di lingkungan *Data Warehouse*.



Gambar 3.21 Script DW Backup Caller

Gambar 3.21 tersebut menampilkan cuplikan DWBackup Caller SH, yaitu skrip pengendali yang bertugas memanggil dan mengeksekusi

fungsi utama dari *DWBackup Global SH* secara otomatis. Pada bagian awal skrip, dilakukan proses inisialisasi variabel penting seperti project code, raw folder, dan script folder untuk memastikan jalur eksekusi sesuai dengan struktur direktori proyek. Selain itu, terdapat daftar file AVRO yang akan diproses, yang didefinisikan dalam bentuk array agar sistem dapat mengeksekusi beberapa file sekaligus secara terstruktur. Inisialisasi ini menjadi tahap penting karena memastikan setiap file yang akan dicadangkan telah terdaftar dengan benar, serta menghindari risiko kesalahan penulisan atau ketidaksesuaian nama file.

Selanjutnya, skrip menjalankan proses penyaringan (*filtering*) file menggunakan fungsi *runFileFilter* untuk memastikan hanya file yang valid dan sesuai standar yang akan diteruskan ke tahap pencadangan. Apabila fungsi mendeteksi adanya kesalahan atau tipografi pada nama file, sistem secara otomatis menghentikan eksekusi dan menampilkan pesan error untuk mencegah terjadinya kesalahan pada tahap selanjutnya. Mekanisme ini memperkuat ketahanan sistem terhadap input yang tidak sesuai, sekaligus memastikan bahwa hanya file yang benar-benar siap diproses yang akan dikirim ke *DWBackup Global SH*. Pendekatan ini sejalan dengan prinsip otomasi defensif yang diterapkan di lingkungan Data Warehouse PT Bank Central Asia, Tbk, di mana validasi berlapis dilakukan sebelum setiap proses penting dijalankan.

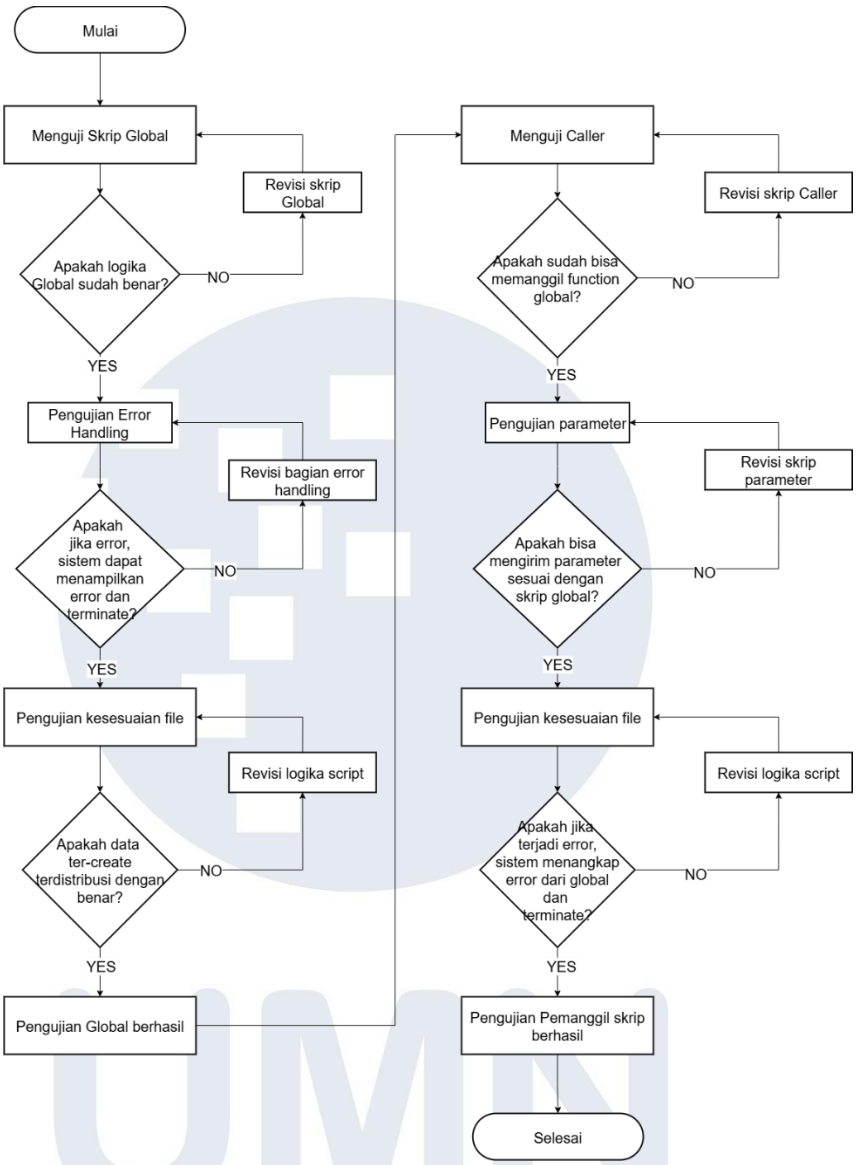
Bagian akhir dari skrip menunjukkan proses konversi hasil penyaringan menjadi bentuk array agar dapat diteruskan secara efisien ke fungsi utama backup. Nilai-nilai ini kemudian digunakan untuk menjalankan proses pencadangan melalui fungsi processAvro, yang memanggil dan mengelola eksekusi backup berdasarkan parameter yang telah diatur. Dengan struktur seperti ini, *DWBackup Caller SH* tidak hanya berfungsi sebagai pemicu proses backup, tetapi juga sebagai pengendali logika yang menjamin setiap tahap dijalankan dalam urutan yang benar dan sesuai kebijakan sistem. Integrasi antara *DWBackup*

Caller dan *DWBackup Global SH* memungkinkan proses pencadangan berjalan otomatis, stabil, dan terdokumentasi dengan baik, memperkuat keandalan sistem penyimpanan data perusahaan.

3.3.1.5 Testing dan Validasi

Pengujian dan validasi skrip merupakan tahapan penting dalam proses implementasi sistem otomatisasi, yang bertujuan untuk memastikan bahwa setiap komponen yang dikembangkan berfungsi sesuai dengan rancangan awal. Tahap ini tidak hanya memverifikasi hasil keluaran dari proses konversi dan distribusi data, tetapi juga menilai konsistensi logika serta ketahanan sistem terhadap kondisi kesalahan (*error handling*). Melalui serangkaian uji fungsional dan logika, keandalan sistem diuji dengan berbagai skenario, baik menggunakan data uji maupun data aktual. Hal ini dilakukan agar setiap perubahan atau peningkatan fungsi pada skrip global maupun skrip pemanggil dapat memberikan hasil yang sesuai dengan kebutuhan operasional tanpa menimbulkan gangguan terhadap sistem yang sudah berjalan.

Dalam pelaksanaannya, pengujian dibagi menjadi dua fokus utama, yaitu pengujian pada *Global Script* dan *Caller Script*. Pengujian pada *Global Script* dilakukan untuk memverifikasi bahwa logika dan fungsi inti bekerja dengan benar serta mampu mengelola data secara otomatis mulai dari tahap validasi hingga distribusi. Sementara itu, pengujian pada *Caller Script* berfokus pada kemampuan skrip dalam memanggil fungsi global dengan parameter yang sesuai, serta memastikan integrasi antar modul berjalan dengan baik. Kedua pengujian ini dilengkapi dengan validasi hasil yang mencakup pengujian kesesuaian file, pemeriksaan pesan kesalahan, dan pengujian terhadap ketahanan sistem saat menghadapi input yang tidak valid.



Gambar 3.22 Diagram proses support pengujian SH Global dan Skrip Pemanggil

Gambar 3.22 tersebut memperlihatkan alur pengujian dan validasi yang diterapkan pada skrip global serta skrip pemanggil, di mana setiap tahap pengujian disertai dengan mekanisme revisi apabila ditemukan ketidaksesuaian. Pengujian dimulai dengan pengecekan terhadap logika skrip global untuk memastikan struktur fungsi dan alur eksekusi sudah berjalan sesuai rancangan. Apabila ditemukan kesalahan, perbaikan dilakukan pada bagian logika atau *error handling* hingga sistem dapat menampilkan pesan kesalahan dan menghentikan proses secara aman.

Setelah logika utama dinyatakan benar, tahap berikutnya adalah pengujian kesesuaian file untuk memastikan bahwa hasil keluaran berupa data terdistribusi dengan benar dan sesuai format yang telah ditentukan.

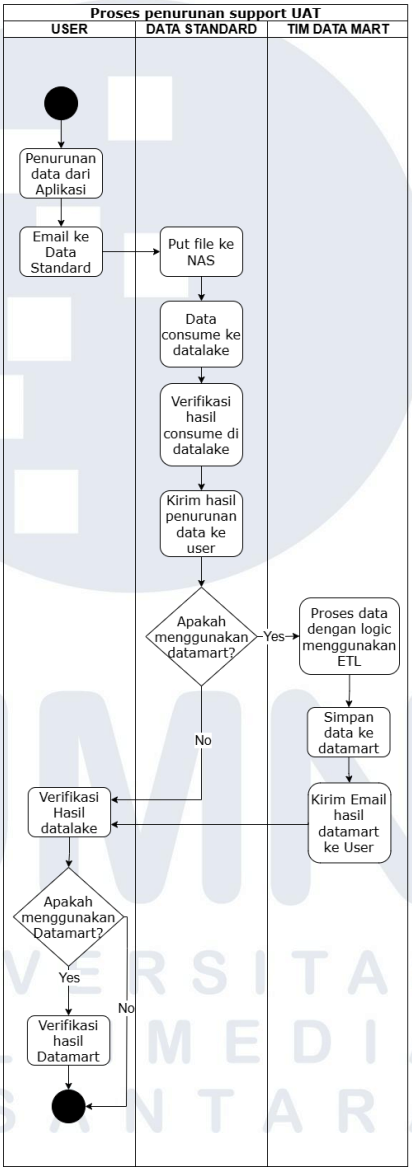
Selanjutnya, pengujian dilakukan terhadap *Caller Script* untuk memastikan fungsi global dapat dipanggil secara tepat dan parameter dapat diteruskan sesuai kebutuhan. Validasi dilakukan untuk menilai apakah sistem mampu menangkap kesalahan dari skrip global serta menghentikan proses eksekusi dengan benar ketika terjadi gangguan. Tahapan ini menegaskan pentingnya pengujian berlapis dalam menjamin stabilitas sistem dan integrasi antar komponen. Hasil dari seluruh rangkaian uji menunjukkan bahwa skrip yang telah dikembangkan telah berfungsi sesuai harapan, dengan logika yang stabil, kemampuan deteksi kesalahan yang baik, serta keluaran data yang sesuai dengan standar perusahaan.

3.3.1.6 Support UAT

Sebagai bagian dari kegiatan operasional harian, proses support terhadap pengujian aplikasi (*User Acceptance Test* atau UAT) menjadi salah satu tanggung jawab utama tim Data Standard. Dalam proses ini, tim bertugas untuk memastikan bahwa data hasil penurunan dari sistem aplikasi dapat dimuat ke *Datalake* secara konsisten dan sesuai standar yang berlaku. Aktivitas ini tidak hanya bersifat rutin, tetapi juga krusial dalam mendukung kelancaran kegiatan pengujian yang dilakukan oleh tim bisnis. Oleh karena itu, seluruh tahapan pelaksanaan dilakukan secara terstruktur dengan mengacu pada alur komunikasi yang jelas antara tim user dan tim Data Standard.

Setiap proses penurunan data dimulai dari pihak user yang mengajukan permintaan atau melakukan penurunan data langsung dari aplikasi, kemudian menginformasikan hal tersebut kepada tim Data Standard. Tim Data Standard bertanggung jawab untuk mengambil data yang telah diturunkan, menempatkannya pada direktori penyimpanan

sementara (NAS), dan menjalankan skrip otomatisasi untuk melakukan proses consume ke *Datalake*. Proses ini memastikan bahwa data yang diterima dari tim user dapat terintegrasi ke sistem analitik perusahaan dan siap digunakan untuk keperluan pengujian maupun validasi hasil oleh tim terkait.



Gambar 3.23 Swimlane diagram proses support UAT

Gambar 3.23 menunjukkan swimlane diagram proses penurunan data yang dilakukan dalam mendukung kegiatan support UAT. Alur proses terbagi menjadi dua bagian utama, yaitu peran tim user dan tim Data

Standard. Proses dimulai dari penurunan data oleh tim user, diikuti dengan pengiriman notifikasi melalui email kepada tim Data Standard. Setelah menerima permintaan, tim Data Standard menempatkan file ke dalam direktori NAS, kemudian menjalankan proses consume data ke Datalake. Setelah data berhasil dimuat, sistem melakukan verifikasi untuk memastikan kesesuaian struktur dan isi data yang telah diunggah.

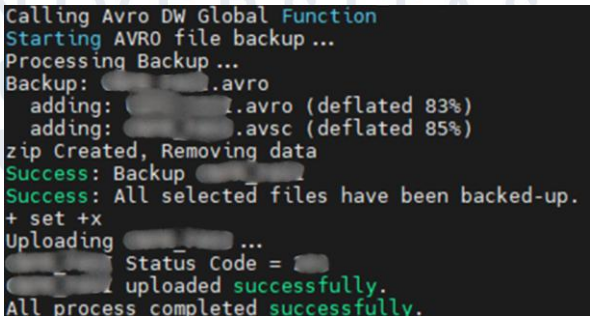
Tahapan selanjutnya adalah pengiriman hasil verifikasi kembali kepada tim user untuk dilakukan pemeriksaan akhir. Jika data dinyatakan sesuai, maka proses penurunan dinyatakan selesai, dan hasil konsumsi data dapat digunakan dalam proses pengujian aplikasi. Alur kerja ini menggambarkan adanya koordinasi yang erat antara tim user dan tim Data Standard untuk memastikan data penurunan tidak hanya dimuat dengan benar, tetapi juga terverifikasi dengan baik. Dengan demikian, proses ini berperan penting dalam menjaga kualitas dan keandalan data selama kegiatan UAT berlangsung di lingkungan pengembangan perusahaan.

Seperti yang telah disebutkan sebelumnya, jalannya skrip dilakukan secara berurutan, dimulai dari proses DW (*Data Warehouse*), dilanjutkan dengan *DWBackup*, dan kemudian proses BD (*Business Domain*) yang berjalan secara paralel. Berikut ini ditampilkan log eksekusi dari masing-masing proses tersebut beserta keterangannya untuk memberikan gambaran lebih lanjut terkait jalannya alur otomatisasi dalam pemrosesan data.

Gambar 3.24 Log Proses DW

Gambar 3.24 menampilkan log hasil eksekusi skrip yang dijalankan dalam rangka proses support UAT, di mana skrip tersebut digunakan untuk melakukan validasi serta distribusi data hasil penurunan dari tim aplikasi ke dalam Datalake. Berdasarkan hasil eksekusi, dapat dilihat bahwa proses dimulai dari tahap pengecekan tanggal modifikasi file, kemudian dilanjutkan dengan validasi struktur dan format Avro yang digunakan dalam sistem. Setiap file yang berhasil melewati tahap validasi akan dibandingkan dengan skema referensi (AVSC) untuk memastikan kesesuaian struktur data. Pesan *success* yang muncul pada tahap ini menandakan bahwa seluruh data yang diuji telah memenuhi kriteria validasi dan siap dipindahkan ke tahap distribusi berikutnya.

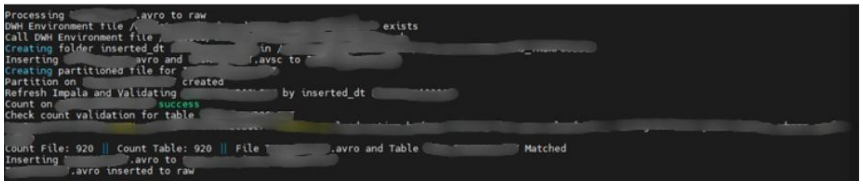
Selanjutnya, sistem menjalankan proses distribusi dengan menyalin file hasil validasi dari NAS ke direktori kerja sementara, kemudian melanjutkannya ke direktori data warehouse yang digunakan sebagai penyimpanan terpusat. Setelah file Avro dan AVSC berhasil disalin, sistem secara otomatis membuat file metadata sebagai referensi bagi proses selanjutnya. Log ini menunjukkan bahwa seluruh proses berjalan lancar tanpa adanya kesalahan atau kegagalan pada tahap pemrosesan. Dengan demikian, hasil eksekusi skrip ini menjadi bukti bahwa proses consume data untuk support UAT berhasil dilakukan, dan seluruh data yang diproses telah terverifikasi serta siap digunakan untuk kebutuhan pengujian oleh tim terkait.



Gambar 3.25 Log Proses DWBackup

Gambar 3.25 memperlihatkan log pelaksanaan proses *DWBackup* yang dijalankan sebagai bagian dari rangkaian pengujian dalam UAT. Hasil log tersebut menunjukkan bahwa proses *backup* pada jalur DW (*Data Warehouse*) telah berhasil dijalankan secara otomatis oleh sistem. Tahapan pertama yang ditampilkan adalah proses kompresi file Avro dan AVSC ke dalam format arsip (zip) untuk menghemat ruang penyimpanan sekaligus menjaga integritas file selama proses pencadangan. Setelah proses kompresi selesai, sistem secara otomatis menghapus salinan file asli dari direktori sumber untuk mencegah duplikasi data. Pesan *success* yang muncul menandakan bahwa seluruh file yang dipilih telah berhasil dicadangkan dan disimpan dalam format arsip yang valid. Tahapan ini memastikan bahwa seluruh data hasil pemrosesan memiliki salinan cadangan yang siap digunakan untuk keperluan pemulihan maupun audit data di kemudian hari.

Tahap berikutnya adalah proses upload hasil backup ke lokasi penyimpanan yang telah ditentukan, yang dilakukan secara otomatis oleh sistem setelah proses pencadangan selesai. Berdasarkan hasil log, setiap file yang diunggah menunjukkan status keberhasilan dengan pesan *uploaded successfully*, menandakan bahwa proses transfer file berjalan dengan lancar tanpa kendala jaringan atau kesalahan sistem. Pernyataan all process completed successfully di bagian akhir log mengindikasikan bahwa seluruh rangkaian kegiatan backup dan unggah data telah diselesaikan dengan baik. Dengan demikian, hasil ini menunjukkan bahwa fungsi DW Backup dalam sistem telah beroperasi secara optimal dan dapat diandalkan dalam menjaga ketersediaan serta keamanan data selama siklus pemrosesan otomatis berlangsung.



Gambar 3.26 Log Proses BD

Gambar 3.25 menunjukkan proses eksekusi jalur BD (*Big Data*) yang berfungsi untuk melakukan pemuatan data hasil konversi ke dalam sistem penyimpanan terdistribusi. Proses diawali dengan pemanggilan file lingkungan (*environment file*) yang berisi konfigurasi variabel sistem, kemudian dilanjutkan dengan pembuatan direktori kerja dan partisi data berdasarkan parameter tanggal pemuatan. Setelah partisi dibuat, sistem memuat file Avro dan skemanya (AVSC) ke dalam direktori tujuan, kemudian melakukan refresh serta validasi terhadap tabel yang telah diupdate di dalam sistem Impala. Seluruh proses ini dijalankan secara otomatis untuk memastikan bahwa struktur tabel dan partisi data di sistem Big Data telah sesuai dengan standar yang berlaku.

Tahapan berikutnya melibatkan proses verifikasi jumlah data antara file sumber dan tabel target di *raw zone*. Berdasarkan log yang ditampilkan, jumlah baris pada file dan tabel menunjukkan hasil yang sama, yang ditandai dengan pesan matched. Hal ini menandakan bahwa seluruh data telah dimuat secara lengkap tanpa kehilangan atau duplikasi. Setelah validasi jumlah data selesai, sistem mengeksekusi proses insert ke dalam tabel tujuan di lingkungan *raw*, menandai keberhasilan proses pemuatan data ke sistem penyimpanan *Big Data*. Dengan demikian, log ini menunjukkan bahwa keseluruhan proses BD telah berjalan dengan benar, di mana data berhasil diintegrasikan ke dalam sistem dengan tingkat akurasi dan konsistensi yang terjaga.

3.3.2 Kendala yang Ditemukan

Proses kerja magang sebagai bagian dari tim Data Management B di PT Bank Central Asia, Tbk telah berjalan dengan baik dan memberikan

pengalaman pembelajaran yang signifikan. Namun, selama pelaksanaan kegiatan magang, terdapat beberapa kendala yang dihadapi dalam penyelesaian tugas dan pengembangan proyek. Kendala tersebut antara lain sebagai berikut:

A. **Ketidakfamiliaran dengan Bahasa Bash**

Pada tahap awal pelaksanaan magang, pemahaman terhadap sintaks dan logika pemrograman bash scripting masih terbatas. Hal ini menyebabkan proses adaptasi terhadap pola pengembangan skrip yang digunakan oleh tim Data Standard memerlukan waktu tambahan, khususnya dalam memahami struktur modular dan fungsi terintegrasi pada sistem otomasi.

B. **Kompleksitas Sistem dan Alur Kerja**

Lingkungan kerja yang kompleks dengan keterlibatan berbagai sistem dan proses, mulai dari penurunan data, validasi, hingga integrasi ke Data Warehouse dan Big Data, menjadi tantangan tersendiri. Banyaknya komponen dan dependensi antarproses menuntut pemahaman menyeluruh terhadap alur teknis serta keterhubungan antar tim dalam siklus pengelolaan data perusahaan.

C. **Kebutuhan untuk Beradaptasi dengan Pola Komunikasi Tim dan User**

Sebagai bagian dari proses operasional harian, koordinasi dengan tim dan user yang melakukan penurunan data ke Data Warehouse perlu dilakukan secara aktif dan berkelanjutan. Pada awal pelaksanaan magang, komunikasi lintas tim memerlukan penyesuaian, baik dari segi pemahaman istilah teknis maupun mekanisme penyampaian informasi yang efektif agar kebutuhan user dapat terpenuhi dengan tepat.

3.3.3 Solusi atas Kendala yang Ditemukan

Menanggapi kendala yang telah diidentifikasi, beberapa upaya telah ditempuh untuk menghilangkan hambatan tersebut. Berikut ini adalah uraian solusi yang diterapkan:

A. **Pembelajaran Otodidak dan Konsultasi dengan Mentor**

Untuk mengatasi keterbatasan pemahaman terhadap bash scripting, dilakukan pembelajaran mandiri melalui studi dokumentasi resmi dan referensi daring. Selain itu, konsultasi rutin dengan mentor juga dilakukan untuk memahami penerapan sintaks, fungsi, dan struktur logika yang digunakan dalam skrip perusahaan.

B. **Pembelajaran Mandiri dan Eksplorasi Sumber Terbuka**

Kompleksitas sistem dan alur kerja diatasi melalui sesi diskusi intensif bersama mentor. Melalui pendekatan ini, pemahaman terhadap arsitektur sistem, hubungan antarproses, serta mekanisme integrasi data dapat diperoleh secara bertahap dan sistematis

C. **Konsultasi dan Review Berkala dengan Mentor**

Untuk menyesuaikan diri dengan kebutuhan operasional tim dan user, dilakukan pelatihan komunikasi interpersonal serta pembiasaan dalam berkoordinasi langsung dengan pihak terkait. Peserta magang juga secara aktif berdiskusi dengan mentor untuk memahami konteks kebutuhan user dan memastikan proses penurunan data ke Data Warehouse dapat berjalan dengan baik sesuai standar operasional.