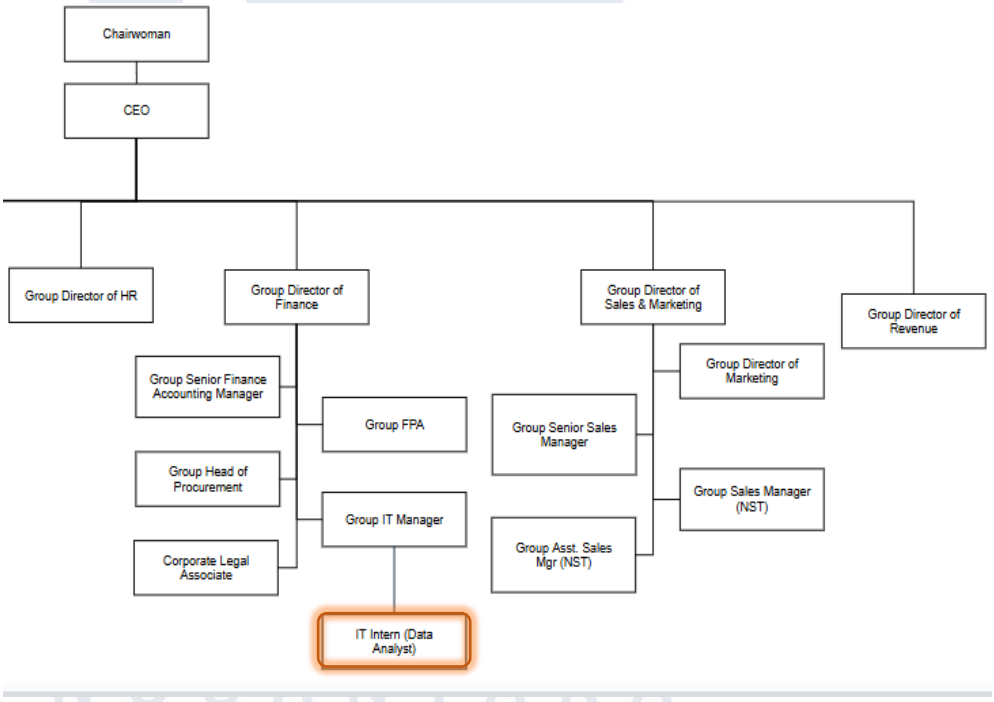
BAB III

PELAKSANAAN KERJA

3.1 Kedudukan dan Koordinasi

Bagian ini membahas peran mahasiswa magang di dalam struktur organisasi perusahaan serta pola koordinasi yang diterapkan selama kegiatan magang berlangsung. Penjelasan ini bertujuan untuk memberikan gambaran mengenai posisi mahasiswa dalam lingkungan kerja serta hubungan kerja yang terjalin dengan pihak terkait. Dengan memahami alur koordinasi tersebut, pelaksanaan tugas magang dapat berjalan terarah dan sesuai dengan bimbingan yang diberikan oleh supervisor pada divisi IT.

3.1.1 Kedudukan



Gambar 3.1 Kedudukan Perusahaan

Gambar 3.1 menunjukkan kedudukan mahasiswa dalam struktur organisasi PT Aryaduta International Management. Pada program magang ini, mahasiswa menempati posisi sebagai IT *Intern* di bawah naungan Group IT *Manager*, yang berada dalam divisi *Finance* & IT dan dikoordinasikan oleh Group *Director of*

Finance. Selama periode magang, mahasiswa dibimbing secara langsung oleh *Group IT Manager* sebagai *supervisor*. Peran *supervisor* mencakup pemberian arahan kerja, pembagian proyek, serta pemantauan progres dan hasil akhir dari setiap tugas yang dikerjakan. Selain itu, supervisor juga memberikan bimbingan dalam mengatasi permasalahan teknis yang muncul, sehingga setiap pekerjaan dapat diselesaikan sesuai dengan standar perusahaan dan mendukung kebutuhan operasional PT Aryaduta International Management.

Sebagai IT Intern dengan fokus pada bidang *Data Analyst*, mahasiswa berperan aktif dalam berbagai proyek pengolahan data dan pengembangan sistem digital yang mendukung proses bisnis perusahaan. Salah satu tanggung jawab utama adalah melakukan otomatisasi data *segment* dari email dengan memanfaatkan platform N8N. Melalui sistem ini, mahasiswa merancang alur otomasi yang mampu melakukan *preprocessing* data secara otomatis dari email harian, kemudian melakukan *fetching* data ke dalam *database* PostgreSQL tanpa perlu intervensi manual. Proses ini bertujuan untuk meningkatkan efisiensi dan mengurangi risiko kesalahan manusia dalam proses pengumpulan serta pelaporan data penghasilan segment dari seluruh cabang hotel Aryaduta di Indonesia.

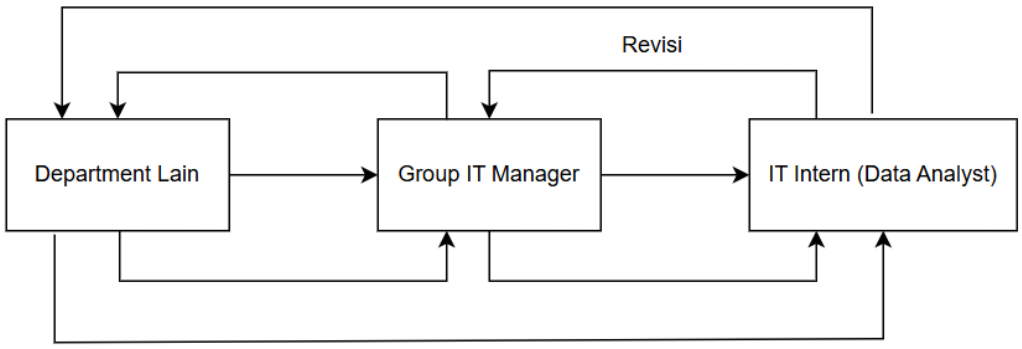
Selain proyek otomasi, mahasiswa juga bertanggung jawab dalam pembuatan dan pengelolaan *database* PostgreSQL yang berfungsi sebagai pusat penyimpanan data terintegrasi untuk berbagai divisi. Database ini dirancang agar mampu menangani data besar (*big data*) yang bersumber dari berbagai sistem internal, serta mendukung kebutuhan analitik dan pelaporan secara *real-time*. Mahasiswa juga melakukan pembuatan *website* berbasis *dashboard* analitik, yang memungkinkan tim keuangan dan manajemen memantau performa harian, tren pendapatan, serta indikator operasional lainnya dengan tampilan visual yang interaktif dan informatif.

Pada proses analisis data, mahasiswa menggunakan beberapa teknologi dan bahasa pemrograman, antara lain Python untuk data *preprocessing*, *segmentation*, dan analisis lanjutan, serta JavaScript untuk integrasi data melalui email dan pengelolaan *workflow* pada platform otomasi. Berdasarkan kombinasi teknologi ini, sistem yang dikembangkan mampu melakukan pemrosesan data dari berbagai

sumber, menyaring dan membersihkan data, hingga menampilkannya dalam bentuk laporan dan visualisasi yang siap digunakan oleh manajemen dalam pengambilan keputusan strategis.

Selama program magang, mahasiswa juga melakukan koordinasi rutin dengan *supervisor* dan tim IT untuk melaporkan hasil pekerjaan, perkembangan proyek, serta kendala teknis yang dihadapi. Proses evaluasi dilakukan melalui pertemuan mingguan baik secara *online* maupun *onsite*, di mana mahasiswa diberikan masukan dan arahan terkait optimalisasi sistem, efisiensi kode, serta keamanan data. Bimbingan langsung dari *supervisor* memberikan pemahaman mendalam mengenai penerapan sistem otomasi, integrasi data lintas departemen, serta pentingnya penerapan *best practice* dalam pengelolaan infrastruktur IT di industri perhotelan berskala nasional.

3.1.2 Koordinasi



Gambar 3.2 Bagan Alur Koordinasi

Gambar 3.2 menunjukkan alur koordinasi kerja mahasiswa selama menjalani program magang sebagai IT Intern (Data Analyst). Dalam pelaksanaan kegiatan magang, mahasiswa berada di bawah koordinasi Group IT Manager yang berperan sebagai supervisor sekaligus pembimbing. Group IT Manager bertanggung jawab dalam menerima permintaan bantuan atau kebutuhan sistem dari berbagai divisi lain di perusahaan, seperti divisi Revenue, Finance, maupun divisi operasional

lainnya, kemudian menerjemahkannya menjadi tugas teknis yang dapat dikerjakan oleh mahasiswa.

Mahasiswa menjalankan tugas berdasarkan arahan dan prioritas yang ditentukan oleh Group IT Manager, mulai dari pengembangan otomasi data dan BI (*Business Intelligence)* untuk kebutuhan analisis revenue, pembuatan RPA (*Robot Process Automation*) untuk mendukung proses keuangan, hingga pengolahan dan visualisasi data sesuai permintaan divisi terkait. Setelah tugas diselesaikan, hasil pekerjaan akan direview oleh *Group IT Manager* untuk memastikan kesesuaian dengan kebutuhan pengguna sebelum disampaikan kembali ke divisi pemohon. Apabila terdapat revisi atau penyesuaian, proses koordinasi dan perbaikan dilakukan secara berulang hingga solusi yang dihasilkan sesuai dengan kebutuhan operasional perusahaan.

3.2 Tugas yang Dilakukan

Selama periode magang yang berlangsung dari tanggal 22 Juli 2025 hingga 31 Desember 2025, mahasiswa diberikan tugas proyek yang harus diselesaikan selama masa magang, yang dapat dilihat lebih rinci di tabel 3.2. Sebagai IT Intern, mahasiswa wajib mematuhi seluruh peraturan yang berlaku di perusahaan. Untuk mendukung pelaksanaan tugas tersebut, mahasiswa menggunakan berbagai alat dan tools, yang akan dijelaskan lebih lanjut pada tabel 3.1.

Tabel 3.1 Alat yang digunakan

Alat	Fungsionalitas
Google Meet	Digunakan untuk rapat *online* setiap pembahasan proyek
Google Spreadsheets	Digunakan untuk melihat kemajuan/hal yang telah dikerjakan oleh mahasiswa setiap harinya dimonitoring oleh supervisor dan lainnya
Visual Studio Code	Digunakan untuk kode RPA menggunakan *robot framework* dengan selenium dan python
DBeaver	Digunakan untuk mengelola *database* Postgre SQL, MariaDB dan berbagai jenis *database*

Alat	Fungsionalitas
Google Meet	Digunakan untuk rapat *online* setiap pembahasan proyek
Google Spreadsheets	Digunakan untuk melihat kemajuan/hal yang telah dikerjakan oleh mahasiswa setiap harinya dimonitoring oleh supervisor dan lainnya
Visual Studio Code	Digunakan untuk kode RPA menggunakan *robot framework* dengan selenium dan python
Draw IO	Digunakan untuk pembuatan *diagram* ERD (Entity Relationship Diagram) dan Struktur Tabel.
Heidi SQL	Digunakan untuk mengelola *database* dengan basis data *MariaDB* tetapi seringkali error
Youtube	Digunakan sebagai *platform* pembelajaran pembuatan *otomasi N8N*
Zoom	Digunakan ketika adanya rapat ataupun pembahasan tugas
N8N	Digunakan sebagai *workflow* yang dapat melakukan otomasi *fetching* data setiap harinya ke dalam database
Postman	Digunakan untuk menguji API dari sunsystem
Jupyter Notebook	Digunakan untuk *preprocessing* data dengan cepat
Appscript	Digunakan untuk *preprocessing* data dari spreadsheet
Excel	Digunakan untuk VBA Macro data pendapatan dan segmentasi
Figma	Digunakan untuk menggambarkan *flowchart*

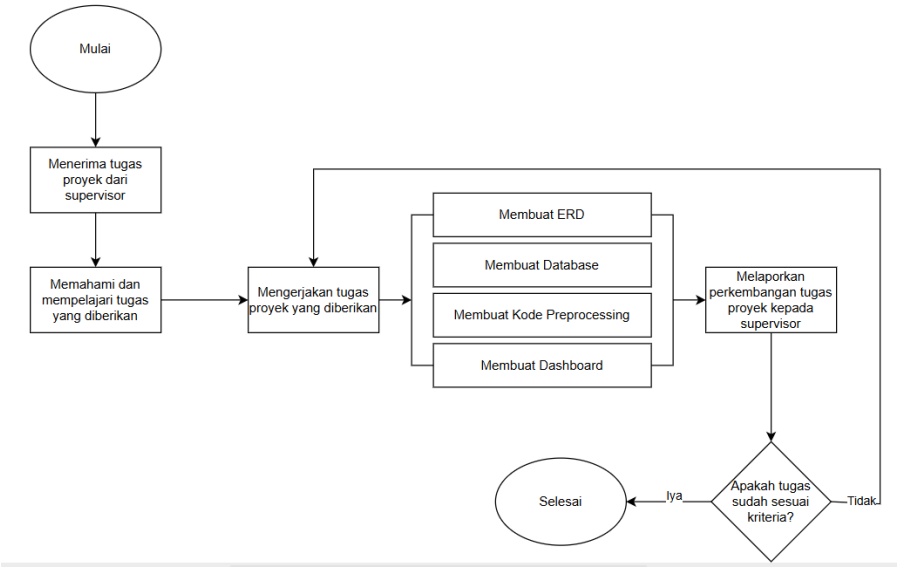
Tabel 3.2 Uraian Kerja Magang

No	Jenis Pekerjaan	Mulai	Selesai
A.	***IT Intern***		
1	**Pengenalan terhadap perusahaan dan tim**		
	Perkenalan mengenai sejarah perusahaan dan struktur organisasi perusahaan	22/07/2025	23/07/2025
	Perkenalan lingkungan kerja dan system yang digunakan dalam perusahaan	24/07/2025	25/07/2025

No	Jenis Pekerjaan	Mulai	Selesai
A.	***IT Intern***		
1	**Pengenalan terhadap perusahaan dan tim**		
	Pembelajaran mengenai system yang digunakan dalam perusahaan dan proyek-proyek yang akan dikerjakan	26/07/2025	27/07/2025
2.	**Pencarian API Endpoint ACA (Account Allocation) Sunsystem menggunakan *Postman***		
	Melakukan pencarian API *Endpoint* dalam sunsystem perusahaan	28/07/2025	29/07/2025
	Menggunakan *postman* untuk mendapatkan *endpoint* dari API yang telah dicari dari sunsystem	30/07/2025	01/08/2025
3.	**Pembuatan Rumus Indirect untuk sheet latest pada excel DRR seluruh unit**		
	Membuat rumus indirect untuk seluruh unit hotel pada data DRR (*daily revenue report)*	04/08/2025	05/08/2025
	Merevisikan rumus indirect yang diminta oleh supervisor	06/08/2025	08/08/2025
4.	**Pembuatan Database untuk data segment yang diolah dari google sheets**		
	Membuat Struktur Tabel Database menggunakan drawio	09/08/2025	10/08/2025
	Membuat Flowchart untuk memastikan langkah untuk insert datanya sudah benar	11/08/2025	12/08/2025
	Membuat Tabel dalam database sesuai struktur tabel yang telah ditentukan	12/08/2025	13/08/2025
	Membuat Kode appscript untuk memasukkan data dari spreadsheet ke dalam database	14/08/2025	16/08/2025
	Membuat Kode data preparation menggunakan python untuk dimasukkan ke dalam database	17/08/2025	22/08/2025
5.	**Pembuatan Workflow dalam sunsystem untuk proses approval printing PO**		
	Membuat workflow dalam sunsystem untuk approval printing po	04/08/2025	05/08/2025
6.	**Pembuatan Website Portal HR menggunakan Wordpress**		
	Membuat seluruh tampilan website dengan menggunakan html untuk tampilan home	06/08/2025	10/09/2025

No	Jenis Pekerjaan	Mulai	Selesai
A.	***IT Intern***		
1	**Pengenalan terhadap perusahaan dan tim**		
	Menambahkan beberapa fungsi website yang terkoneksikan dengan drive dan spreadsheet	11/09/2025	30/09/2025
7.	**Pembuatan RPA untuk PI entry menggunakan robot framework**		
	Menggunakan VS Code dan robot framework untuk membuat otomasi robot menggunakan selenium dan python	22/08/2025	16/09/2025
8.	**Pembuatan Database untuk data DRR yang diolah dari excel, kemudian juga VBA dari DRR yang akan dijadikan Dashboard BI**		
	Membuat struktur tabel database menggunakan draw io	26/09/2025	28/09/2025
	Membuat ERD untuk masing masing relasi antar tabel	29/09/2025	1/10/2025
	Membuat tabel dalam database sesuai dengan struktur dan relasi yang telah dibuat	2/10/2025	3/10/2025
	Membuat kode python untuk preprocessing data untuk dimasukkan ke dalam database	4/10/2025	8/10/2025
	Membuat schema database agar data yang berada di fact table sudah yang bersih	9/10/2025	13/10/2025
	Membuat VBA Macro excel untuk mempermudah orang orang dalam memasukkan data ke dalam database	14/10/2025	18/10/2025
	Membuat visualisasi dashboard dalam Power BI dengan dax calculation	19/10/2025	31/10/2025
9.	**Otomasi Segmentasi data menggunakan N8N dan pembuatan Dashboard BI**		
	Membuat flow dalam N8N untuk otomasi fetching email ke database	24/09/2025	25/06/2025
	Membuat visualisasi dashboard menggunakan Power BI	3/11/2025	14/11/2025

3.3 Uraian Pelaksanaan Kerja



Gambar 3.3 Alur Kerja Magang

Berdasarkan alur kerja yang terlihat pada Gambar 3.3, proses dimulai ketika *supervisor*, *Group IT Manager*, memberikan tugas proyek kepada mahasiswa. Setelah menerima proyek, mahasiswa terlebih dahulu mempelajari dan memahami ruang lingkup pekerjaan yang diberikan, termasuk tujuan proyek, kebutuhan data, serta sistem yang akan digunakan. Langkah awal ini penting untuk memastikan mahasiswa dapat mengimplementasikan solusi yang tepat dan relevan dengan kebutuhan divisi IT Aryaduta Hotel Group. Selanjutnya, mahasiswa mulai mengerjakan proyek yang berkaitan dengan pengolahan data dan otomasi sistem pelaporan. Proses ini melibatkan beberapa tahapan penting, seperti pembuatan *Entity Relationship Diagram* (ERD) untuk memetakan struktur data yang akan digunakan, perancangan dan pembuatan *database* PostgreSQL sebagai penyimpanan utama, serta pengembangan kode preprocessing data menggunakan Python dan JavaScript. Dalam tahap ini, mahasiswa juga mengimplementasikan sistem otomasi berbasis N8N untuk mengintegrasikan data dari berbagai sumber, salah satunya dari email harian, kemudian melakukan auto-fetching data ke dalam database.

Selain pengelolaan data, mahasiswa juga mengembangkan *dashboard* analitik interaktif yang digunakan untuk menampilkan hasil analisis data seperti *Daily Revenue Report* (DRR) dan metrik performa lainnya secara *real-time*.

Dashboard ini membantu tim keuangan dan manajemen untuk memantau kinerja hotel secara efisien, tanpa harus melakukan input data secara manual. Selama pengerjaan proyek, mahasiswa secara rutin melaporkan perkembangan tugas kepada *supervisor* untuk memastikan setiap tahap berjalan sesuai dengan standar yang telah ditentukan. Laporan ini berisi informasi mengenai progres pekerjaan, hasil sementara, serta kendala teknis yang dihadapi dalam proses pengembangan. Berdasarkan laporan tersebut, *supervisor* akan memberikan umpan balik dan masukan terkait optimalisasi sistem, perbaikan logika automasi, atau penyempurnaan desain *dashboard*.

3.3.1 Proses Pelaksanaan

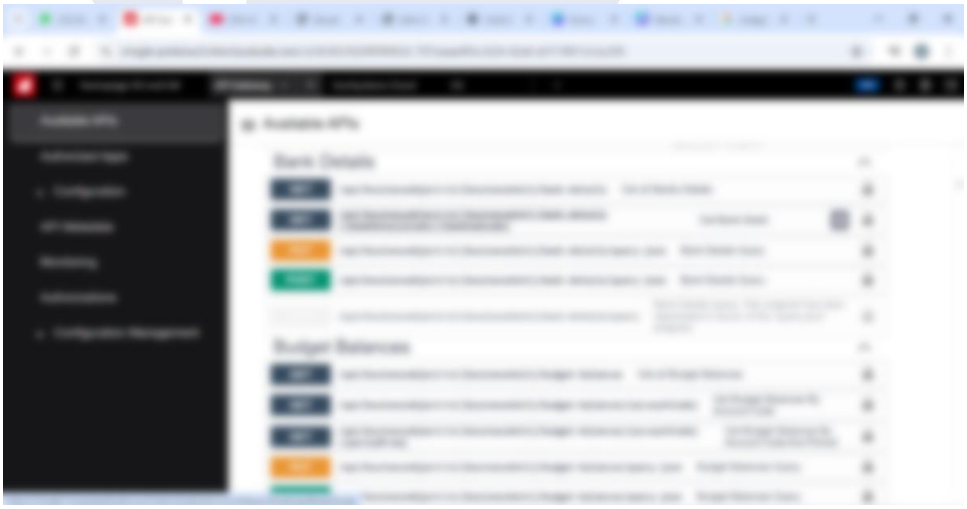
3.3.1.1 Pencarian API Endpoint ACA (Account Allocation) Sunsystem menggunakan Postman

Pada minggu pertama magang, mahasiswa magang memulai kegiatan dengan pembelajaran dan eksplorasi dasar mengenai penggunaan API (*Application Programming Interface*) melalui platform *Postman*. Pembelajaran ini bertujuan untuk memahami proses pengambilan data (*data fetching*) dari sistem eksternal, khususnya Sunsystem (ACA/*Account Allocation*), yang nantinya akan digunakan dalam proses otomasi pelaporan data keuangan perusahaan. Pada tahap ini, mahasiswa mempelajari cara melakukan request dan response API, memahami struktur data dalam format JSON, serta mengidentifikasi endpoint yang relevan untuk kebutuhan integrasi data. Aktivitas ini menjadi langkah awal yang penting karena data hasil fetching API tersebut akan digunakan dalam sistem otomasi dan analisis data berbasis dashboard yang dikembangkan selama periode magang.

Selain melakukan pencarian dan pengujian API, mahasiswa juga mempelajari proses otomasi pengambilan data harian (*Daily Revenue Report* / DRR) menggunakan N8N yang terintegrasi dengan database PostgreSQL. Pada workflow ini, mahasiswa membangun

sistem otomatis untuk mengambil data dari email dan API Sunsystem setiap hari, kemudian melakukan *preprocessing* data menggunakan Python untuk membersihkan, menormalkan, dan mempersiapkan data sebelum disimpan ke *database*. Data yang telah diolah kemudian digunakan untuk pembuatan *dashboard* interaktif sebagai alat monitoring performa keuangan perusahaan secara *real-time*. Melalui kegiatan ini, mahasiswa magang memperoleh pengalaman praktis dalam integrasi sistem berbasis API, otomasi data *pipeline* (RPA), serta pengolahan data menggunakan Python, yang berperan penting dalam mendukung efisiensi operasional dan ketepatan analisis data di lingkungan kerja profesional.

1. Melakukan Pencarian API Endpoint dalam Sunsystem Perusahaan



Gambar 3.4 API Sunsystem

Gambar 3.4 menunjukkan tampilan proses pencarian dan pengujian API *Endpoint Account Allocation* (ACA) pada sistem Infor SunSystems Cloud. Proses ini dilakukan menggunakan Postman untuk memastikan endpoint yang diambil dapat berfungsi dengan baik serta mengembalikan data yang sesuai dengan kebutuhan otomasi proses bisnis di bagian keuangan. Pada tahap awal, mahasiswa melakukan eksplorasi terhadap berbagai endpoint yang tersedia dalam kategori *Bank Details*, salah satunya adalah /api*/businessobject/v1/{businessUnit}/bank-details* dan */budget-balances*, sebelum akhirnya menemukan *endpoint* yang relevan dengan *Account Allocation*. *Endpoint* ini memungkinkan pengambilan data transaksi alokasi akun (seperti kode akun, jumlah debit/kredit, serta status transaksi) yang nantinya akan digunakan dalam proses integrasi dan otomasi laporan keuangan harian *(Daily Revenue Report)*.

Gambar 3.4 menampilkan isi dari modul *Account Allocation* di SunSystems, yang berisi berbagai informasi penting seperti *Account Code, Base Amount*, Debit/Credit, dan *Transaction Reference*. Data ini menjadi dasar bagi mahasiswa magang dalam melakukan proses

fetching API yang kemudian diolah secara otomatis melalui sistem RPA (*Robotic Process Automation*). Tahapan ini menjadi pondasi utama dalam pengembangan sistem otomatisasi pelaporan internal perusahaan, yang bertujuan untuk meningkatkan efisiensi dan akurasi dalam pengelolaan data keuangan.

2. Menggunakan Postman untuk mendapatkan endpoint dari API yang telah dicari dari sunsystem

Gambar 3.5 Postman API

Setelah melakukan pencarian dan identifikasi *endpoint* yang relevan di Infor SunSystems, tahap selanjutnya adalah melakukan proses pengujian dan pengambilan data API *(API fetching)* menggunakan Postman. Gambar 3.5 menunjukkan hasil dari proses tersebut, di mana mahasiswa magang menggunakan metode GET dengan autentikasi berupa *Bearer Token* untuk mengakses data dari endpoint SunSystems. Token ini berfungsi sebagai kunci keamanan agar hanya pengguna yang memiliki otorisasi tertentu yang dapat mengakses data perusahaan. Pada tampilan tersebut terlihat hasil

respon status 200 OK, yang menandakan bahwa permintaan API berhasil dilakukan dan data berhasil diterima dari server.

Data yang diperoleh dari hasil permintaan API berformat JSON (JavaScript Object Notation) dan berisi berbagai informasi penting, seperti kode alamat (*addressCode*), nama hotel, alamat lengkap, status operasi, serta tanggal pembaruan terakhir. Data tersebut kemudian digunakan untuk mendukung proses otomasi sistem di lingkungan Aryaduta Hotel Group, terutama dalam pengelolaan data finansial yang terdapat pada modul *Account Allocation* di SunSystems. Melalui API yang telah berhasil diakses ini, mahasiswa magang mengimplementasikan otomasi berbasis Robot Framework untuk melakukan proses input data secara otomatis ke dalam sistem, seperti pengisian *Account Code* dan alokasi transaksi tanpa perlu melakukan input manual.

3.3.1.2 Pembuatan Rumus Indirect untuk sheet latest pada excel DRR seluruh unit

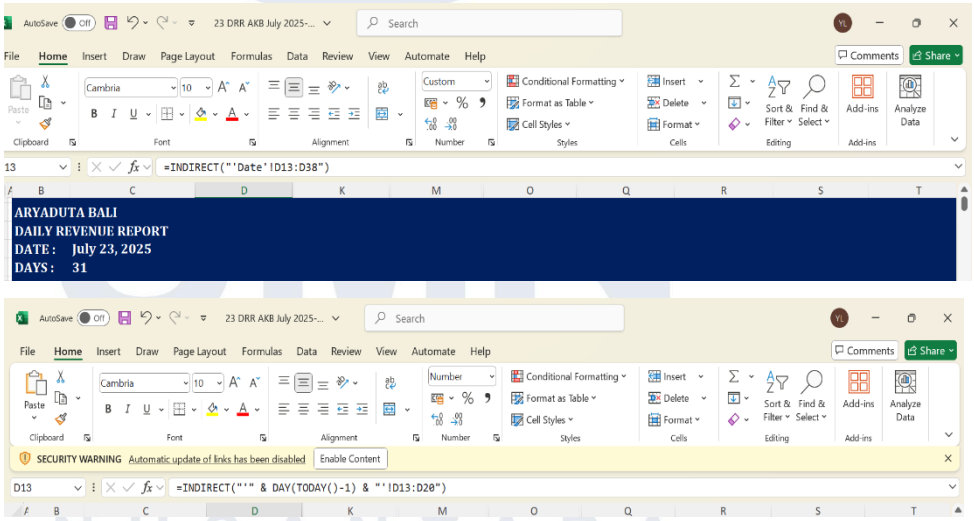
Setelah tahap pengambilan data *API endpoint* selesai dilakukan, langkah selanjutnya dalam proyek otomasi adalah melakukan penyeragaman *template Daily Revenue Report (DRR)* untuk seluruh unit hotel dan *leisure* yang dikelola oleh Aryaduta Hotel Group. Tujuan utama dari tahap ini adalah memastikan bahwa seluruh format laporan dari berbagai unit memiliki struktur data yang seragam sehingga dapat diintegrasikan dengan mudah ke dalam *database* pusat untuk proses otomasi dan analisis lebih lanjut.

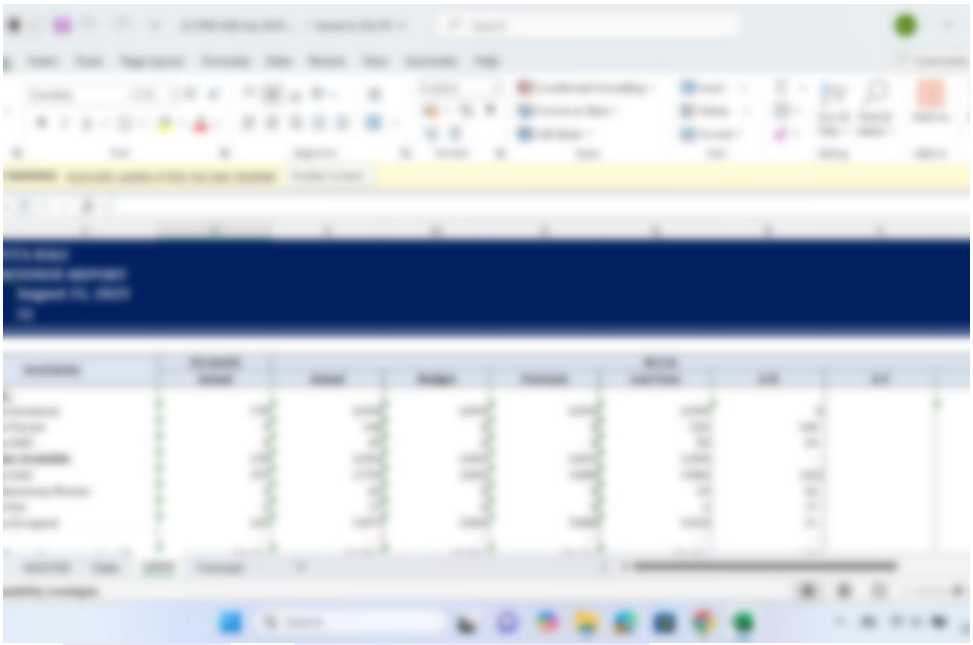
Langkah awal dalam proses ini dimulai dengan pembuatan rumus *Indirect* pada *sheet "Latest"* di *file* Excel masing-masing unit. Rumus ini berfungsi untuk mengambil dan menghubungkan nilai-nilai data DRR secara otomatis dari setiap template unit hotel tanpa perlu melakukan pemindahan data secara manual. Memanfaatkan fungsi Indirect di semua excel unit, sistem dapat membaca referensi sel dari

sheet atau file lain secara dinamis, sehingga ketika nama sheet atau sumber data berubah, rumus tetap dapat menyesuaikan secara otomatis.

Implementasi rumus *Indirect* ini menjadi fondasi penting dalam proses integrasi data karena memastikan setiap perubahan atau pembaruan nilai pada template DRR di masing-masing hotel langsung tercermin pada *sheet "Latest"*. Hasil akhir dari proses ini adalah satu format DRR terpusat dan konsisten, yang nantinya digunakan untuk otomasi pengambilan data harian ke dalam PostgreSQL *database* melalui alur kerja otomatis berbasis N8N. Tahapan ini juga menjadi dasar bagi pembuatan sistem pelaporan otomatis dan *dashboard analytics* yang akan menampilkan data *real-time* dari seluruh unit Aryaduta secara terintegrasi.

1. Membuat rumus indirect untuk seluruh unit hotel pada data daily revenue





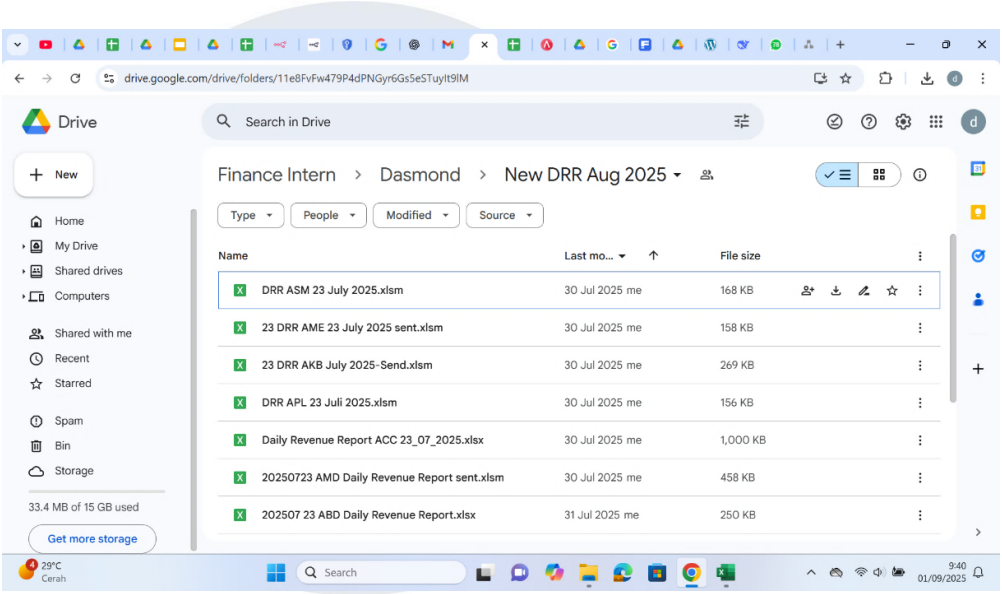
Gambar 3.6 Rumus Indirect

Seperti terlihat pada Gambar 3.6, rumus yang digunakan adalah =INDIRECT("'Date'!D13:D38") dan dikembangkan menjadi =INDIRECT("'" & DAY(TODAY())-1 & "'!D13:D20") untuk menyesuaikan pengambilan data dengan tanggal sebelumnya (H-1). Hal ini dilakukan karena laporan DRR dikirim setiap pagi untuk data hari sebelumnya. Cara ini membuat sistem mampu menampilkan data terkini yang selalu *up-to-date* sesuai jadwal operasional masing-masing unit.

Setiap unit hotel dan *leisure* memiliki format atau template *Daily Revenue Report* (DRR) tersendiri yang berbeda-beda sehingga rumus *INDIRECT* juga harus disesuaikan berdasarkan nama sheet dan struktur data dari masing-masing unit. Contohnya, ada unit yang menggunakan penamaan *sheet* berdasarkan tanggal, sementara unit lain menggunakan nama bulan atau kode unik tertentu, oleh karena itu, penyesuaian rumus diperlukan agar referensi sel tetap akurat saat menarik data dari file sumber. Penyesuaian ini dilakukan dengan cara memodifikasi bagian referensi dalam fungsi *INDIRECT*, baik pada

nama *sheet*, rentang sel, maupun format tanggal yang digunakan, sehingga seluruh file DRR dari berbagai unit dapat diolah menjadi satu format standar untuk integrasi ke tahap otomasi berikutnya.

2. Merevisikan rumus indirect yang diminta oleh supervisor



Gambar 3.7 File *Template* DRR

Setelah seluruh *template Daily Revenue Report* (DRR) dari setiap unit hotel berhasil diseragamkan, tahap berikutnya adalah melakukan revisi terhadap rumus *INDIRECT* berdasarkan arahan dari *supervisor*. Revisi ini dilakukan karena ditemukan beberapa ketidaksesuaian antara nilai yang ditarik menggunakan rumus dengan nilai pada template asli masing-masing unit. Permasalahan tersebut umumnya terjadi akibat perbedaan struktur kolom dan adanya spasi pada nama sheet atau referensi sel, yang menyebabkan rumus *INDIRECT* tidak dapat berjalan dengan semestinya, terutama saat file dipindahkan ke Google Spreadsheet.

Akhirnya mahasiswa magang melakukan penyesuaian pada setiap rumus agar dapat membaca nilai dari kolom yang tepat dan memastikan tidak terdapat spasi berlebih dalam penulisan referensi.

Revisi ini penting agar proses pengambilan data harian berjalan otomatis dan konsisten di seluruh unit. Penyesuaian ini membuat hasil yang diambil melalui rumus *INDIRECT* menjadi akurat, sesuai dengan format *template* aslinya, dan siap digunakan dalam tahap berikutnya yaitu integrasi data ke dalam sistem otomasi berbasis N8N dan PostgreSQL *database*.

3.3.1.3 Pembuatan Database untuk data segment yang diolah dari google sheets

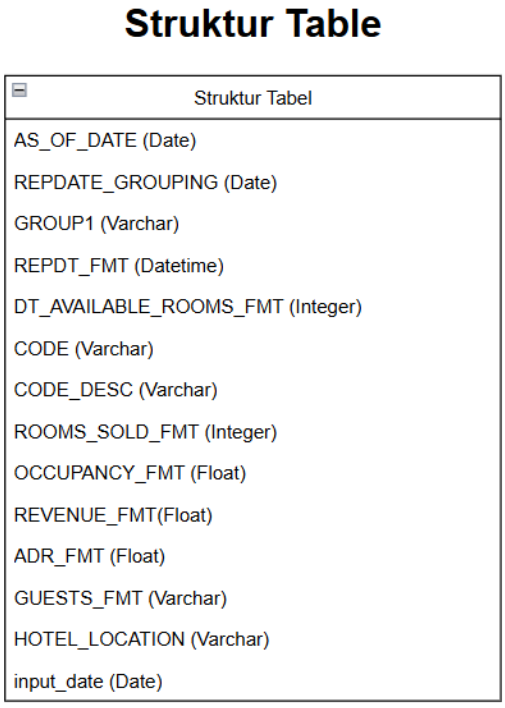
Setelah tahap revisi rumus *INDIRECT* selesai dan seluruh data pada template DRR berhasil distandardisasi, langkah selanjutnya adalah membangun *database* khusus untuk data *segment* yang sebelumnya dikelola melalui Google Sheets. Pembuatan *database* ini dilakukan untuk memastikan bahwa seluruh data segment dapat disimpan secara terstruktur, mudah diakses, serta terintegrasi dengan proses otomasi yang akan diterapkan.

Tahap pertama dimulai dengan merancang struktur tabel *database* menggunakan Draw.io, di mana mahasiswa magang menyusun relasi antar-tabel, menentukan atribut utama, serta memastikan setiap kolom sesuai dengan format data pada Google Sheets. Setelah itu, dibuat flowchart proses insert data untuk memetakan alur kerja mulai dari pengambilan data di spreadsheet hingga data masuk ke dalam *database*. Flowchart ini berfungsi sebagai acuan agar proses otomatisasi berjalan konsisten dan menghindari kesalahan dalam pemindahan data.

Setelah struktur dan alur proses tervalidasi, mahasiswa magang melanjutkan dengan membangun tabel *database* langsung di Mariadb terlebih dahulu tetapi Mariadb terdapat kendala dimana jika data yang tersimpan terlalu banyak, *loadingnya* sangat lama sehingga tidak sanggup dalam menampung data yang jumlahnya sangat besar, terakhir dipindahkan semua data yang telah dibuat di Mariadb ke

PostgreSQL sesuai data yang telah dibuat sebelumnya. Selanjutnya dibuat kode Google Apps Script untuk mengambil data dari Google Sheets dan mengirimkan data tersebut ke *database* melalui API *endpoint* internal. Tahap terakhir adalah mengembangkan kode data preparation menggunakan Python, yang berfungsi untuk membersihkan, memvalidasi, dan memformat ulang data sehingga siap dimasukkan ke dalam *database* tanpa error.

1. Membuat Struktur Tabel Database menggunakan Drawio



Gambar 3.8 *Struktur Tabel Segment*

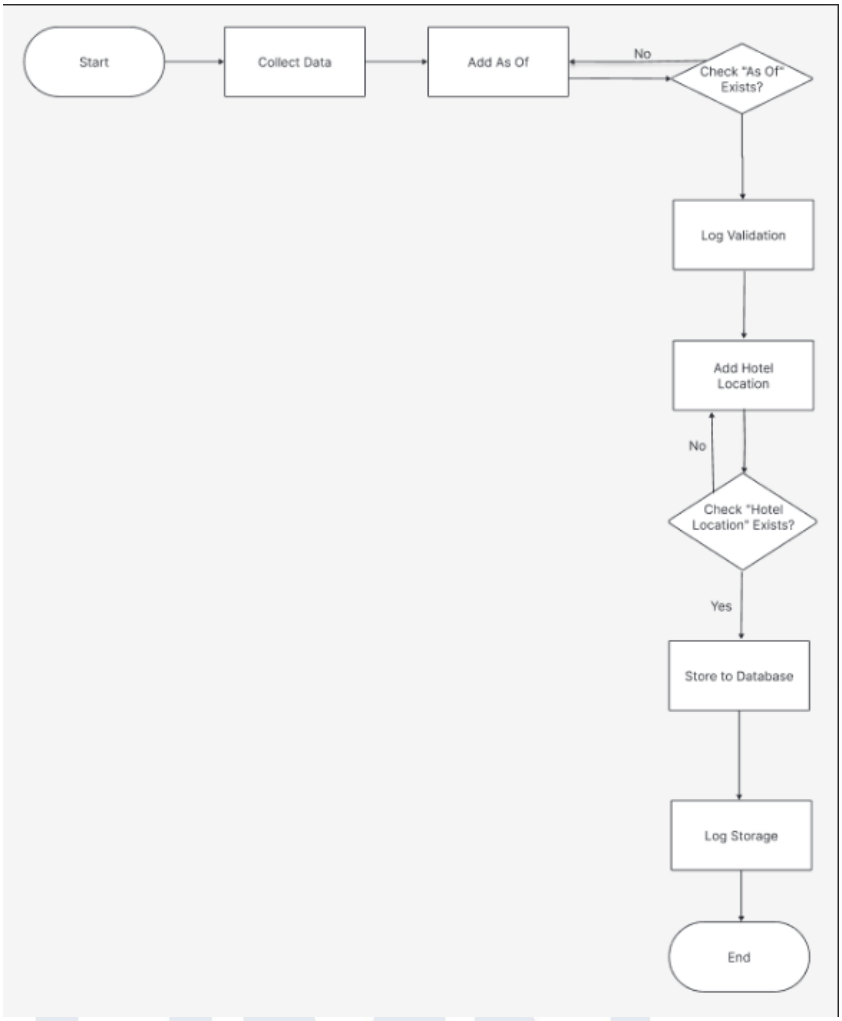
Gambar 3.8 menunjukkan rancangan struktur tabel untuk data segment yang dibuat menggunakan aplikasi Draw.io sebelum diimplementasikan ke dalam *database* PostgreSQL. Pada tahap ini, mahasiswa magang terlebih dahulu menyusun setiap kolom yang diperlukan berdasarkan kebutuhan data dari Google Sheets, kemudian memetakan tipe data yang paling sesuai agar proses penyimpanan dan pemrosesan data dapat berjalan optimal.

Struktur tabel ini terdiri dari beberapa *field* penting, seperti *AS_OF_DATE* dan *REPDATE_GROUPING* yang menggunakan tipe *Date* untuk mencatat tanggal pelaporan dan pengelompokan data. *Field GROUP1* menggunakan tipe *Varchar* karena berisi kategori segmentasi yang bersifat teks. Selanjutnya, terdapat *REPDT_FMT* dengan tipe *Datetime* untuk menyimpan format tanggal lengkap beserta waktu. *Field* numerik seperti *DT_AVAILABLE_ROOMS_FMT, ROOMS_SOLD_FMT, OCCUPANCY_FMT, REVENUE_FMT, ADR_FMT*, dan *GUESTS_FMT* menggunakan tipe *Integer* maupun *Float* agar data perhitungan dapat disimpan dan diolah secara akurat.

Selain itu, terdapat *field CODE* dan *CODE_DESC* yang menggunakan tipe *Varchar*, berfungsi sebagai kode identifikasi *segment* dan deskripsi segment yang nantinya akan berguna untuk proses analisis lebih lanjut. *Field HOTEL_LOCATION* juga menggunakan tipe *Varchar* untuk menampung lokasi unit hotel yang terkait. Terakhir, terdapat kolom *input_date* dengan *tipe Date*, yang digunakan untuk mencatat kapan data tersebut dimasukkan ke dalam *database* sebagai bagian dari proses *logging* dan data *tracking*.

Penyusunan struktur tabel melalui Draw.io ini menjadi langkah penting sebelum implementasi teknis, karena membantu memastikan bahwa setiap elemen data telah terdefinisi dengan baik, memiliki tipe data yang tepat, dan siap digunakan dalam proses otomatisasi pengolahan data *segment* melalui Google Apps Script, Python, maupun *pipeline database* lainnya.

2. Membuat Flowchart untuk memastikan langkah untuk insert datanya sudah benar



Gambar 3.9 *Flowchart Segment*

Gambar 3.9 menampilkan *flowchart* proses data *insertion* yang digunakan untuk memastikan bahwa seluruh langkah pengolahan data *segment* telah berjalan sesuai alur yang benar sebelum data disimpan ke dalam *database*. *Flowchart* ini dibuat sebagai panduan untuk memvalidasi setiap tahapan, mulai dari pengambilan data mentah hingga proses penyimpanan, sehingga meminimalkan risiko kesalahan data seperti duplikasi, kekosongan nilai penting, atau inkonsistensi antar unit.

Proses dimulai dari tahap *Start*, kemudian dilanjutkan dengan langkah *Collect Data*, yaitu proses pengambilan data hasil olahan dari Google Sheets yang sebelumnya telah diproses menggunakan rumus

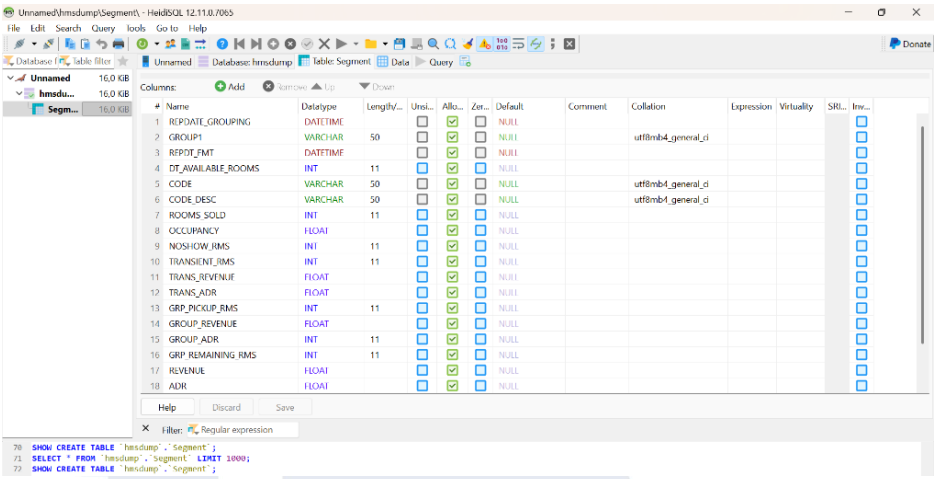
INDIRECT dan formatting tambahan. Setelah data terkumpul, langkah berikutnya adalah *Add As Of*, yaitu proses menambahkan tanggal referensi pelaporan (*as_of_date*) yang berfungsi sebagai penanda dari tanggal laporan *segment* tersebut.

Selanjutnya, sistem melakukan pengecekan pertama melalui proses *Check "As Of" Exists*? untuk memastikan apakah data dengan tanggal yang sama sudah pernah tersimpan sebelumnya di dalam *database*. Jika data dengan *as_of_date* tersebut sudah ada, alur akan diarahkan untuk melakukan *Log Validation*, yaitu pencatatan bahwa data tidak dapat dimasukkan karena berpotensi duplikasi. Jika data belum ada, proses dilanjutkan ke langkah berikutnya.

Tahap berikutnya adalah pengecekan lokasi unit melalui *Check "Hotel Location" Exists*?, yang memvalidasi apakah lokasi hotel sudah sesuai dengan data referensi yang tersedia. Jika lokasi hotel belum ditemukan, sistem akan menambahkan informasi lokasi melalui proses *Add Hotel Location* agar data dapat dipetakan dengan benar. Jika lokasi sudah valid, proses langsung diarahkan menuju langkah penyimpanan.

Setelah seluruh validasi terlewati, data akan masuk ke tahap *Store to Database*, yaitu proses penyimpanan data *segment* ke dalam tabel *database* sesuai struktur yang telah dirancang sebelumnya pada Draw.io. Setelah data berhasil disimpan, sistem akan mencatat aktivitas tersebut melalui *Log Storage* sebagai bagian dari proses pencatatan dan monitoring. Tahapan kemudian diakhiri pada bagian *End*, menandakan bahwa seluruh langkah proses penyimpanan data telah berhasil dilakukan.

3. Membuat Tabel dalam database sesuai struktur tabel yang telah ditentukan



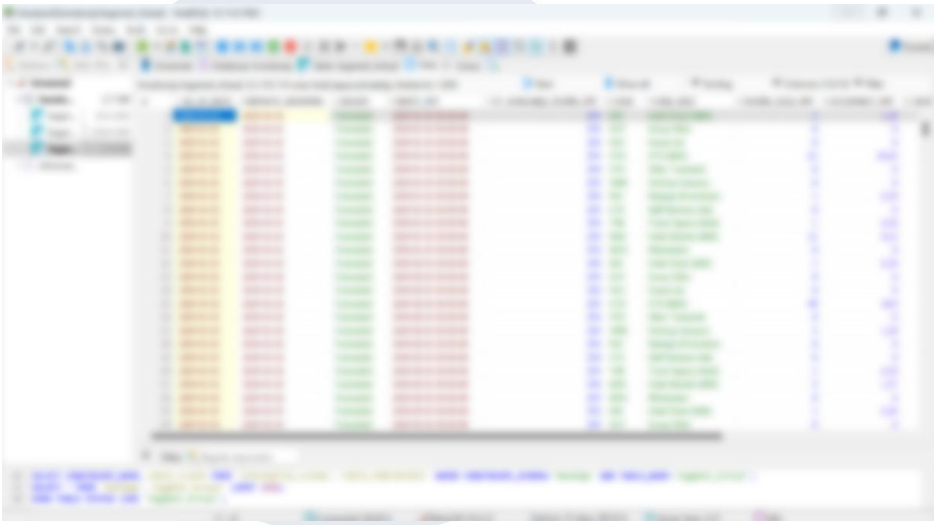
Gambar 3.10 Mariadb *Segment*

Setelah tahap pembuatan *flowchart* yang digunakan untuk memastikan alur proses *insertion* data sudah benar, langkah berikutnya adalah membangun struktur *database* sebagai tempat penyimpanan data *segment* yang diambil dari Google Sheets. *Database* ini dibuat menggunakan sistem manajemen basis data MariaDB, yang dipilih karena mampu menangani dataset dalam jumlah besar serta mendukung proses otomatisasi yang akan dijalankan.

Pada gambar terlihat bahwa tabel *segment* telah dibentuk berdasarkan struktur tabel yang sebelumnya dirancang dalam diagram Draw.io. Setiap kolom dalam tabel ini disesuaikan dengan *field* yang dibutuhkan untuk menyimpan data operasional hotel dari tahun 2021 hingga 2025. Kolom-kolom seperti *AS_OF_DATE, REPDPT, ROOMS_SOLD_FM, REVENUE_FM*, hingga *OCCUPANCY_FM* disusun secara sistematis dengan tipe data yang sesuai, seperti *DATE*, *DATETIME, INT*, dan *FLOAT*, untuk memastikan bahwa proses penyimpanan data berjalan dengan konsisten dan akurat.

Selain itu, penentuan tipe data, panjang karakter, serta aturan *nullability* juga diperhatikan agar struktur *database* tetap optimal ketika digunakan dalam proses otomatisasi. *Database* ini nantinya

menjadi pusat penyimpanan seluruh data segment harian dari unit hotel dan leisure, yang sebelumnya diproses melalui Google Sheets dan disiapkan menggunakan formula *INDIRECT* sesuai *template* masing-masing unit. *Database* ini membantu data historis dari tahun 2019 hingga 2025 dapat tersimpan dengan terstruktur dan siap digunakan untuk proses analitik maupun pembuatan *dashboard*.



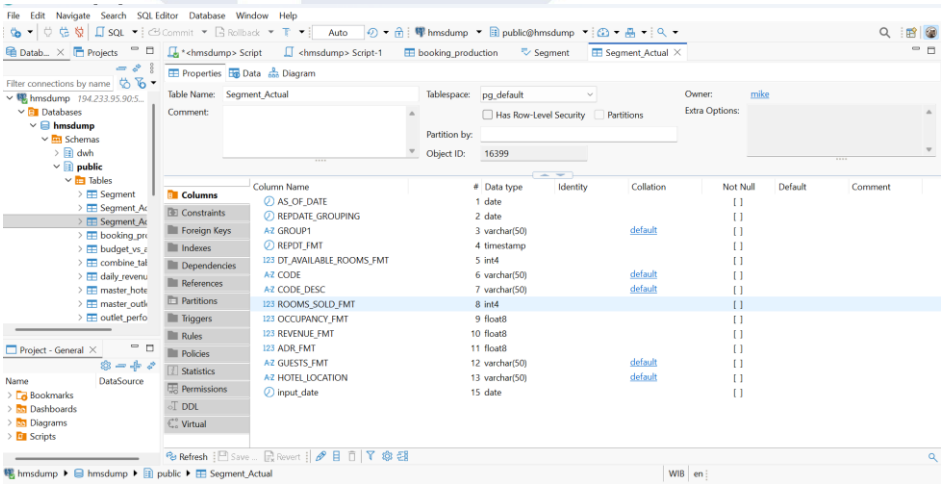
Gambar 3.11 *Segment Database*

Setelah struktur tabel utama berhasil dibuat, proses selanjutnya adalah melakukan pengisian data awal untuk memastikan bahwa tabel dapat menampung seluruh informasi dengan benar. Data yang dimasukkan berasal dari hasil pengolahan pada Google Sheets, yang telah distandarkan melalui penyesuaian formula dan validasi format. Seluruh data dipindahkan secara bertahap untuk menghindari inkonsistensi saat proses import berlangsung. Setiap baris data diuji coba melalui *query* sederhana di HeidiSQL agar tidak terjadi kesalahan tipe data maupun kerusakan struktur tabel.

Volume data yang sangat besar menimbulkan tantangan pada performa *database*. Total data yang tersimpan mencapai puluhan juta baris. Kondisi ini membuat satu tabel tidak lagi efektif menampung seluruh informasi. Tabel tambahan kemudian dibuat untuk

memisahkan data berjalan dan data historis. Tabel utama hanya menyimpan data terbaru yang masih relevan untuk proses operasional harian. Tabel kedua berfungsi sebagai penyimpanan arsip untuk data tahun-tahun sebelumnya. Pendekatan ini menjaga kestabilan performa *query*, terutama untuk proses analitik yang dilakukan secara rutin.

Proses pemisahan data ini juga memberikan kemudahan dalam pengelolaan. Data yang jarang digunakan tidak membebani eksekusi perintah *SELECT*, terutama pada proses otomatisasi yang memerlukan respons cepat. Sistem menjadi lebih efisien karena pemanggilan data dapat berlangsung tanpa hambatan. *Database* yang telah tersusun kemudian menjadi fondasi bagi tahapan berikutnya, yaitu integrasi data ke dalam *pipeline* automatisasi untuk kebutuhan *dashboard* analitik. Seluruh data yang tersimpan sudah siap diolah lebih lanjut tanpa memerlukan penyesuaian struktur tambahan.



Gambar 3.12 Postgre SQL *Table*

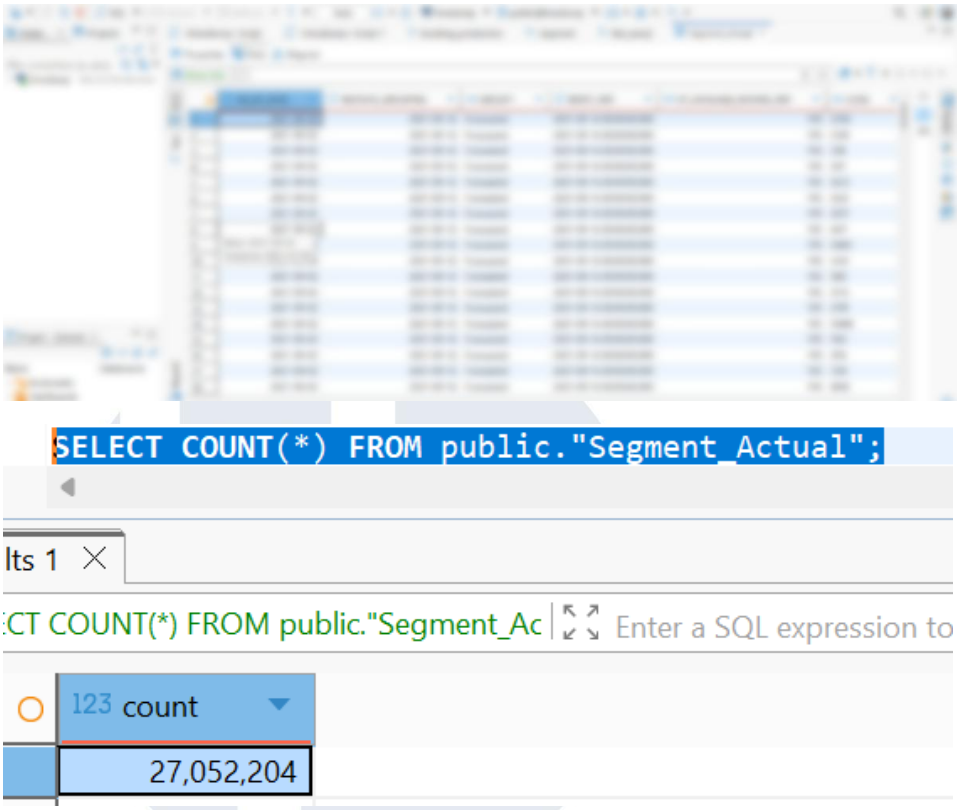
Setelah struktur tabel diimplementasikan dalam MariaDB sesuai rancangan awal, proses pengelolaan data mulai dilakukan. Data yang diambil dari Google Sheets terus bertambah setiap hari. Kondisi ini membuat ukuran tabel berkembang sangat cepat. MariaDB mulai menunjukkan penurunan performa ketika melakukan *query* pada data yang sudah mencapai jutaan baris. Proses pemanggilan data terasa

lambat. Waktu *loading* pada beberapa operasi juga meningkat. Situasi ini menandakan bahwa sistem penyimpanan perlu ditingkatkan agar proses otomatisasi dapat berjalan lebih stabil.

Migrasi *database* dilakukan sebagai solusi. PostgreSQL dipilih karena memiliki kemampuan manajemen data yang lebih efisien untuk dataset besar. PostgreSQL juga menyediakan fitur *indexing* yang lebih fleksibel dan mendukung proses analitik yang lebih berat. Kemampuan dalam menangani operasi komputasi yang kompleks menjadikan PostgreSQL lebih sesuai untuk kebutuhan proyek ini. Proses migrasi dilakukan dengan tetap mempertahankan struktur tabel yang telah dirancang sebelumnya sehingga tidak mengubah alur bisnis data.

Perpindahan aplikasi manajemen *database* juga dilakukan. HeidiSQL yang sebelumnya digunakan bersama MariaDB memiliki keterbatasan dalam pembuatan *stored procedure*. Beberapa fungsi yang dibutuhkan untuk otomatisasi *insertion* data tidak dapat berjalan dengan optimal. DBeaver dipilih sebagai pengganti. Aplikasi ini mendukung PostgreSQL secara penuh. Proses pembuatan *procedure*, *function*, maupun *trigger* menjadi jauh lebih fleksibel. DBeaver juga menyediakan tampilan antarmuka yang lebih lengkap untuk memantau performa *query* dan struktur *database*.

Lingkungan kerja yang baru ini memberikan proses yang lebih cepat dan stabil. Data *segment* yang sebelumnya ditampung sejak tahun 2021 hingga 2025 dapat diakses dengan lebih efisien. Data historis tetap tersimpan tanpa mengganggu performa *query* harian. Alur otomatisasi juga menjadi lebih mudah dikembangkan karena platform baru mendukung seluruh kebutuhan teknis untuk pemrosesan data skala besar.



Gambar 3.13 Data *Segment* PostgreSQL

Tampilan pada gambar 3.13 menunjukkan salah satu tabel hasil ekstraksi data harian dari jaringan hotel Aryaduta. Tabel ini memberi gambaran bahwa proses pengambilan data dari Google Spreadsheet ke *database* sudah berjalan dengan baik. Data yang sebelumnya harus diproses secara manual kini sudah dapat dibaca dan dimasukkan ke *database* melalui proses *fetch* yang dilakukan menggunakan Google Apps Script dan Python. Jumlah data yang tersimpan sudah mencapai 27.052.204 baris. Jumlah ini menunjukkan bahwa proses pengambilan dan pemindahan data dapat berjalan konsisten. Data yang terus bertambah setiap hari tetap masuk dengan rapi. Alur kerja utama juga tetap berjalan tanpa terganggu.

Setiap baris yang terlihat pada editor DBeaver berasal dari proses *fetch* harian yang dijalankan melalui Apps Script dan Python. Kedua proses ini mengambil data dari spreadsheet, membaca pembaruan yang terjadi, lalu mengirimkan hasilnya ke *database* PostgreSQL.

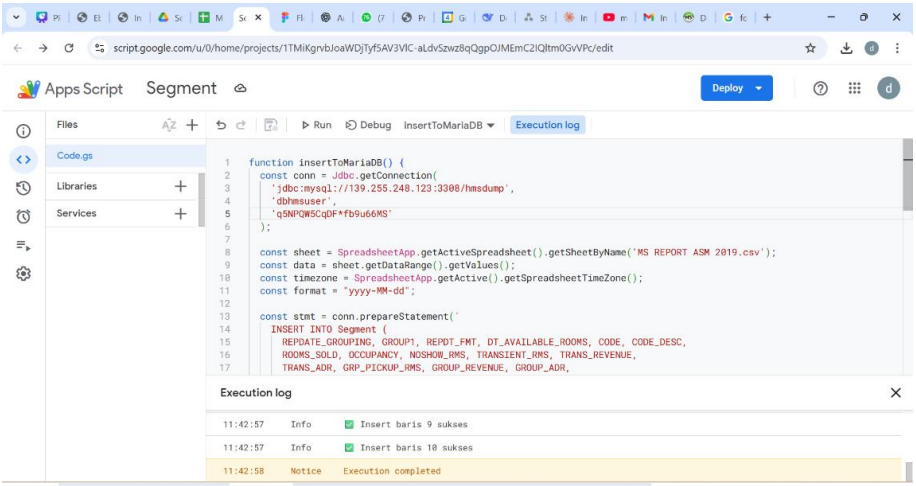
Spreadsheet Google menjadi sumber awal. Sistem membaca nilai baru yang muncul. Sistem mencatat setiap perubahan. Sistem menyiapkan datanya agar sesuai dengan struktur *database*. Sistem kemudian mengirimkannya ke tabel yang sudah disiapkan.

Ukuran dataset yang besar membuat proses pembersihan data menjadi bagian penting. Data harus dinormalisasi sebelum dimasukkan ke *database*. Normalisasi dibutuhkan karena format *timestamp* pada spreadsheet sering berbeda. Beberapa nilai juga memiliki karakter khusus yang bisa menyebabkan error saat *parsing*. Proses *preprocessing* memeriksa setiap nilai. Proses ini memastikan bahwa tipe data sudah sesuai. Nilai yang tidak sesuai akan diabaikan. Tahap pembersihan dilakukan agar data yang masuk tetap konsisten dan siap digunakan untuk kebutuhan *query*.

PostgreSQL digunakan sebagai *database* utama karena mampu menangani data dalam jumlah besar. *Query* yang membaca jutaan baris masih dapat diproses dengan efisien. *Indexing* pada kolom tanggal membuat pencarian data menjadi lebih cepat. *Query* dapat mengambil data tertentu tanpa membaca seluruh isi tabel. Struktur tabel dirancang berdasarkan prinsip *database* relasional. Prinsip ini membantu menjaga performa meskipun jumlah data terus meningkat.

DBeaver digunakan sebagai alat untuk melihat dan memantau data. Aplikasi ini menyediakan fitur yang lengkap. Pembuatan *function* dan *stored procedure* juga menjadi lebih mudah. DBeaver memiliki tampilan statistik yang membantu memantau *runtime query*. Informasi ini membantu dalam menentukan apakah indeks baru perlu ditambahkan. Fitur tersebut memastikan *database* tetap dalam keadaan optimal untuk menerima data harian.

4. Membuat Kode appscript untuk memasukkan data dari spreadsheet ke dalam database



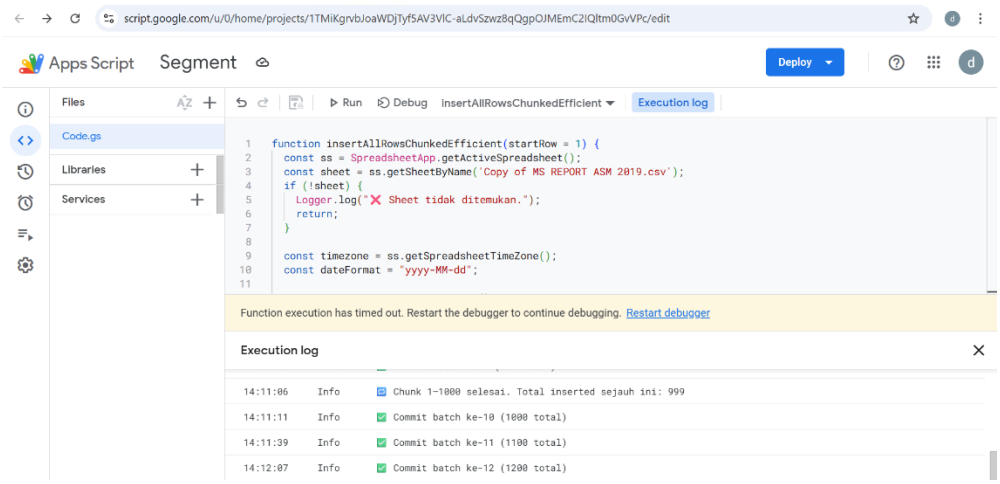
Gambar 3.14 Appscript Spreadsheet

Tahap berikutnya setelah struktur *database* selesai dibuat adalah proses *preprocessing* dan pengiriman data dari Google Spreadsheet ke *database*. Proses ini dilakukan dengan Google Apps Script yang berfungsi sebagai penghubung antara spreadsheet dan *database*. Apps Script mengambil data mentah dari sheet lalu membersihkannya terlebih dahulu. Data yang sudah rapi kemudian dikirim ke *database* melalui koneksi JDBC. Langkah ini dibutuhkan karena data awal di spreadsheet memiliki banyak variasi bentuk dan jumlahnya cukup besar. Setiap kolom harus dipilih dan dicek agar sesuai dengan kebutuhan tabel Segment.

Kode JavaScript pada gambar adalah bagian dari fungsi *insertToMariaDB*(). Fungsi tersebut membaca isi sheet, memilih kolom yang diperlukan, lalu menyesuaikan format tanggal agar seragam. Format tanggal perlu dibenahi karena Google Sheets sering menampilkan tanggal dalam bentuk berbeda untuk tiap tahun atau tiap unit kerja. Proses konversi dilakukan agar nilai tanggal cocok dengan tipe *DATE* pada database. Data numerik juga disesuaikan dulu agar cocok dengan tipe *INT* atau *FLOAT* yang sudah ditentukan.

Proses ini diuji dengan melakukan *trial insertion* menggunakan 10 baris data pertama. Uji coba dilakukan untuk memastikan koneksi JDBC bekerja dengan benar. *Query INSERT INTO* juga diperiksa agar tidak menimbulkan error. Format tanggal, nilai angka, dan nilai kosong ikut diuji. Setiap data yang berhasil dimasukkan akan muncul pada *execution log*, misalnya "Insert baris 9 sukses". *Log* tersebut menunjukkan bahwa alur *preprocessing* dan pengiriman data sudah berjalan seperti yang diharapkan.

Kode *preprocessing* juga dilengkapi pengecekan nilai null. Banyak data lama berisi sel kosong yang dapat menimbulkan error saat dimasukkan ke *database*. Nilai kosong difilter sejak awal agar proses insert tetap aman dan data yang masuk ke tabel *Segment* lebih bersih. Pengecekan ini membuat sistem hanya menyimpan data yang benar-benar valid dan sesuai dengan kebutuhan pengguna.



Gambar 3.15 *Chunk* Appscript

Setelah melalui tahap uji coba awal untuk memastikan bahwa proses *inserter* data dari Google Spreadsheet ke *database* dapat berjalan dengan benar, ditemukan kendala baru terkait keterbatasan eksekusi pada Google Apps Script. Secara default, Apps Script hanya memberikan waktu maksimum 6 menit untuk menjalankan satu

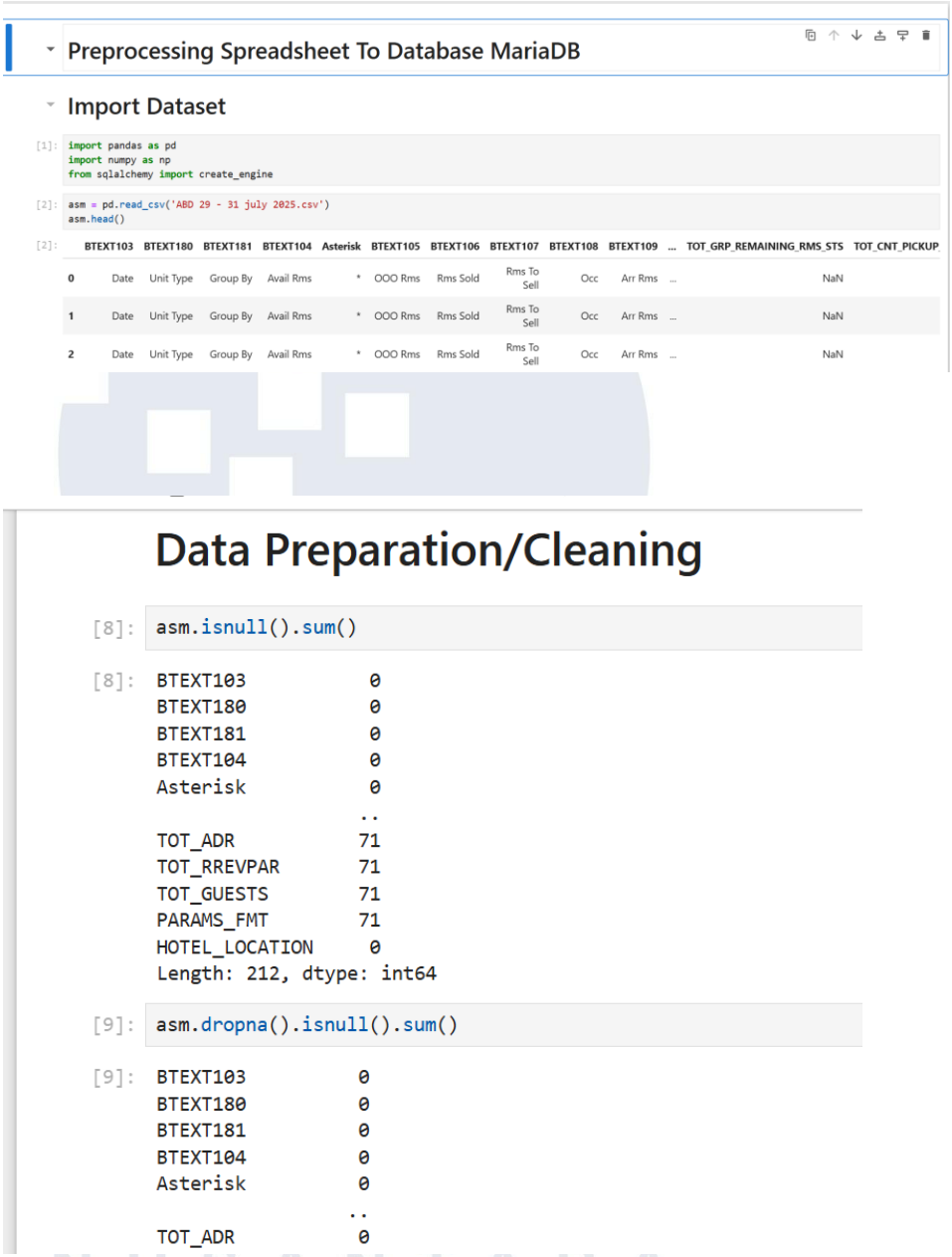
fungsi. Ketika jumlah data yang harus diproses hanya puluhan baris, batasan ini tidak menjadi masalah. Namun, ketika sistem harus menangani data dalam jumlah besar, mulai dari ribuan hingga jutaan baris, proses ini menjadi tidak efisien dan berpotensi gagal karena melampaui batas waktu eksekusi.

Pada pengujian lanjutan, terlihat bahwa memasukkan data dalam jumlah besar secara langsung menyebabkan Apps Script berhenti sebelum seluruh proses selesai, yang ditunjukkan oleh pesan "*Function execution has timed out*." Hal ini dikarenakan mekanisme pemrosesan yang dilakukan secara satu kali jalan (*single run*) membutuhkan waktu lebih panjang daripada yang diizinkan oleh Apps Script.

Untuk mengatasi keterbatasan tersebut, dibuatlah sebuah mekanisme *chunking*, yaitu strategi membagi dataset menjadi beberapa bagian kecil yang diproses secara terpisah. Teknik ini memungkinkan Apps Script hanya memproses sebagian data pada setiap eksekusi, misalnya 500 atau 1.000 baris per *batch*, sehingga waktu pemrosesan tetap berada di bawah limit 6 menit. Dengan demikian, proses ekstraksi dan penyimpanan data tetap dapat berjalan hingga seluruh data selesai diproses tanpa mengalami *timeout*.

Implementasi fungsi *chunk* ini juga dilengkapi dengan sistem *logging*, sehingga setiap *batch* yang berhasil diproses akan tercatat, misalnya "*Commit batch* ke-10 (1000 total)" sampai "*Commit batch* ke-12 (1200 total)." Informasi ini sangat penting untuk memantau progres dan memastikan tidak ada data yang terlewat. Jika terjadi gangguan di tengah proses, sistem dapat memulai kembali dari *batch* yang terakhir berhasil diproses, tanpa harus mengulang dari awal.

5. Membuat Kode data preparation menggunakan python untuk dimasukkan ke dalam database



Preprocessing Spreadsheet To Database MariaDB

Import Dataset

```
[1]: import pandas as pd
     import numpy as np
     from sqlalchemy import create_engine

[2]: asm = pd.read_csv('ABD 29 - 31 july 2025.csv')
     asm.head()
```

[2]:

	BTEXT103	BTEXT180	BTEXT181	BTEXT104	Asterisk	BTEXT105	BTEXT106	BTEXT107	BTEXT108	BTEXT109	...	TOT_GRP_REMAINING_RMS_STS	TOT_CNT_PICKUP
0	Date	Unit Type	Group By	Avail Rms	*	OOO Rms	Rms Sold	Rms To Sell	Occ	Arr Rms	...	NaN	
1	Date	Unit Type	Group By	Avail Rms	*	OOO Rms	Rms Sold	Rms To Sell	Occ	Arr Rms	...	NaN	
2	Date	Unit Type	Group By	Avail Rms	*	OOO Rms	Rms Sold	Rms To Sell	Occ	Arr Rms	...	NaN	

Data Preparation/Cleaning

```
[8]: asm.isnull().sum()
```

```
[8]: BTEXT103          0
     BTEXT180          0
     BTEXT181          0
     BTEXT104          0
     Asterisk          0
                      ..
     TOT_ADR          71
     TOT_RREVPAR      71
     TOT_GUESTS       71
     PARAMS_FMT       71
     HOTEL_LOCATION    0
     Length: 212, dtype: int64
```

```
[9]: asm.dropna().isnull().sum()
```

```
[9]: BTEXT103          0
     BTEXT180          0
     BTEXT181          0
     BTEXT104          0
     Asterisk          0
                      ..
     TOT_ADR           0
```


Gambar 3.16 *Preprocessing* menggunakan Python

Tahap *preprocessing* tidak hanya dilakukan dengan Apps Script. Python juga dipakai. Apps Script memiliki batas waktu eksekusi yang sangat singkat. Waktu maksimalnya hanya enam menit. Batas ini membuat proses untuk data besar menjadi tidak optimal. Python

dipilih untuk membantu menangani data dalam skala yang jauh lebih besar.

Pada gambar terlihat proses awal dalam Python. Tahap pertama adalah mengimpor *file* CSV ke dalam *notebook*. File ini berisi data mentah dari spreadsheet. Setelah data masuk, langkah berikutnya adalah memahami struktur kolom. Proses ini membantu melihat bentuk data. Proses ini juga menunjukkan apakah ada nilai yang perlu diperiksa lebih lanjut.

Python kemudian digunakan untuk membersihkan data. Banyak kolom memiliki nilai kosong. Nilai kosong dapat menimbulkan *error* ketika data dimasukkan ke *database*. Python melakukan pengecekan null menggunakan fungsi *isnull()*. Hasil pengecekan menunjukkan kolom mana saja yang masih memiliki nilai kosong.

Setelah itu dilakukan proses penghapusan nilai null. Proses ini memakai fungsi *dropna()*. Data menjadi lebih rapi. Data menjadi siap untuk proses selanjutnya. Data yang sudah bersih lebih aman untuk dikirim ke *database*. Python mempermudah proses ini karena mampu mengolah data dalam jumlah besar tanpa kendala waktu.

```
dtype: int64

[11]: kolom_dibutuhkan = [
          'REPDATE_GROUPING', 'GROUP1', 'REPDT_FMT', 'DT_AVAILABLE_ROOMS', 'CODE',
          'CODE_DESC', 'ROOMS_SOLD', 'OCCUPANCY', 'NOSHOW_RMS', 'TRANSIENT_RMS',
          'TRANS_REVENUE', 'TRANS_ADR', 'GRP_PICKUP_RMS', 'GROUP_REVENUE',
          'GROUP_ADR', 'GRP_REMAINING_RMS', 'REVENUE', 'ADR', 'RREVPAR', 'GUESTS', 'HOTEL_LOCATION'
      ]

      asm_trimmed = asm[kolom_dibutuhkan].copy()

[12]: asm_trimmed['REPDATE_GROUPING'] = pd.to_datetime(asm_trimmed['REPDATE_GROUPING'], errors='coerce')

      asm_trimmed['REPDT_FMT'] = pd.to_datetime(asm_trimmed['REPDT_FMT'], errors='coerce', dayfirst=True)
      asm_trimmed['REPDT_FMT'] = asm_trimmed['REPDT_FMT'].dt.strftime('%Y-%m-%d')
```

```
[15]: username = 'dbhmsuser'
password = 'q5NPQW5CqDF*fb9u66MS'
host = '172.16.10.6'
port = '3306'
database = 'hmsdump'

connection_string = f'mysql+pymysql://{username}:{password}@{host}:{port}/{database}'

engine = create_engine(connection_string)
```

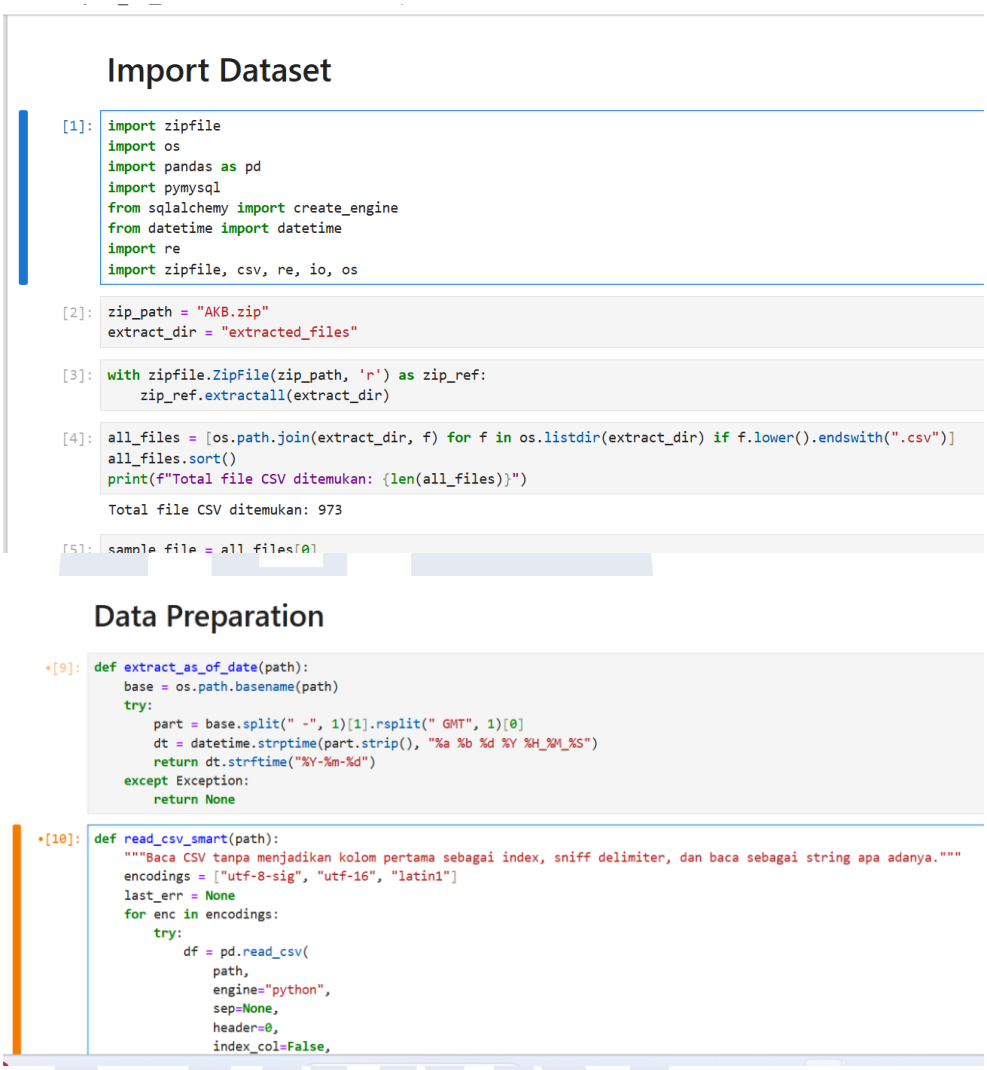
```
[16]: try:
    asm_trimmed.to_sql(
        'Segment',
        con=engine,
        if_exists='append',
        index=False,
        chunksize=1000,
        method='multi'
    )
    print("✅ Data berhasil dimasukkan ke tabel Segment.")
except Exception as e:
    print("❌ Terjadi error saat insert:", e)
```

```
✅ Data berhasil dimasukkan ke tabel Segment.
```

Gambar 3.17 Python data segmentasi

Setelah data selesai *dipreprocessing*. Langkah berikutnya adalah memilih kolom yang akan dimasukkan ke *database*. Setiap kolom harus disesuaikan dengan struktur tabel yang sudah ada. Format tanggal harus diperbaiki lebih dulu agar sesuai dengan format di *database*. Biasanya memakai format YYYY-MM-DD.

Setelah kolom rapi dan format tanggal sudah benar. Data bisa dikirim ke *database*. Proses kirim datanya memakai koneksi Python ke MySQL. Python membuat koneksi memakai *username, password, host, port*, dan nama *database*. Setelah koneksi berhasil dibuat. *Dataframe* tinggal di-*insert* ke tabel yang dituju. Contoh prosesnya seperti di gambar. Python mencoba memasukkan data ke tabel. Jika berhasil. Program menampilkan pesan sukses. Jika gagal. Program memberi tahu bahwa ada *error*.

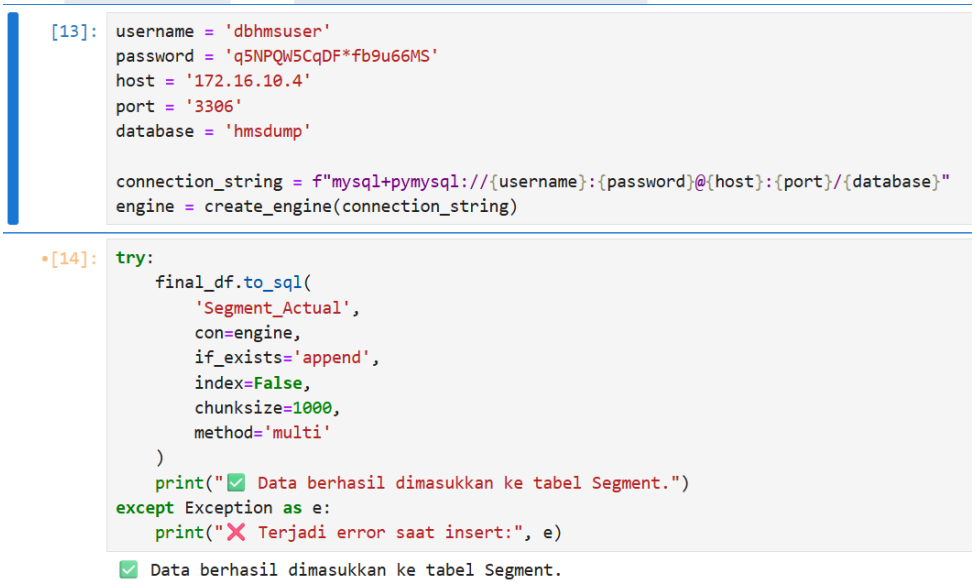


Gambar 3.18 Python *preprocessing* zip data

Gambar 3.18 dimulai dari tahap *importing* data. Dataset awal berbentuk *file* ZIP. Isi *file* ZIP ini adalah ratusan file CSV. Total *file* CSV yang ditemukan pada contoh tersebut adalah 973 *file*. Semua *file* ini diekstrak terlebih dahulu agar bisa dibaca oleh Python. Setelah proses ekstraksi selesai. Setiap file CSV dibaca satu per satu menggunakan fungsi yang sudah disiapkan. Fungsi ini dibuat agar bisa menangani berbagai format CSV. Format CSV sering tidak konsisten sehingga perlu perlakuan khusus. Fungsi yang digunakan memastikan file tetap bisa dibaca meskipun ada perbedaan *encoding* atau *delimiter*.

Tahap berikutnya adalah data *preparation*. Pada tahap ini terdapat proses tambahan yaitu membuat kolom baru bernama *as_of_date*. Nilai kolom ini diambil dari nama *file* CSV. Nama *file* biasanya sudah mengandung informasi tanggal. Tanggal ini digunakan sebagai penanda waktu untuk seluruh data di dalam file tersebut. Dengan begitu data bisa dipakai untuk analisis tren atau prediksi ke depan.

Data yang sudah bersih kemudian dipilih kolom-kolom yang ingin dimasukkan ke dalam *database*. Tidak semua kolom diambil. Hanya kolom yang diperlukan saja. Format tanggal juga diperbaiki agar sesuai dengan format yang digunakan dalam *database* PostgreSQL.



Gambar 3.19 Python *insert* data ke *database*

Tahap terakhir adalah proses memasukkan data ke dalam *database*. Data yang sudah melewati proses *preprocessing* disimpan dalam satu *dataframe* bernama *final_df*. *Dataframe* ini berisi data yang sudah dibersihkan. Kolom yang tidak diperlukan sudah dihapus. Format tanggal sudah disesuaikan. Semua file CSV dari dalam ZIP sudah digabung menjadi satu dataset yang siap dimasukkan ke PostgreSQL.

Python kemudian dihubungkan dengan *database* menggunakan SQLAlchemy. Informasi seperti *username*. *password*. *host*. *port*. dan nama *database* dimasukkan ke dalam *connection string*. *Engine database* dibuat berdasarkan parameter tersebut.

Setelah koneksi siap. Data dimasukkan ke tabel bernama *Segment_Actual*. Proses *insert* dilakukan menggunakan fungsi *to_sql()*. Fungsi ini dilengkapi pengaturan *chunksize*=1000. Data dimasukkan seribu baris per satu batch. Cara ini mempercepat proses *insert*. *Database* juga tidak terlalu terbebani ketika menerima data dalam jumlah besar.

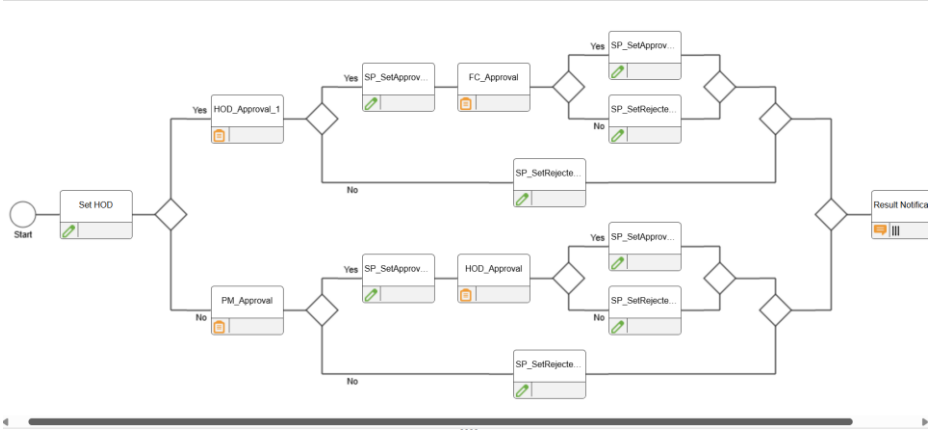
Proses *insert* memakai mode *append*. Data baru langsung ditambahkan ke tabel tanpa menghapus data lama. Setiap proses *insert* dibungkus dengan blok *try–except*. Jika proses berhasil. Python menampilkan pesan bahwa data sudah dimasukkan dengan sukses. Jika terjadi kesalahan. Python menampilkan pesan *error* agar bisa dilakukan pengecekan.

3.3.1.4 Pembuatan Workflow dalam sunsystem untuk proses approval printing PO

Setelah proses pembuatan *database* untuk data segmentasi selesai. Tugas berikutnya adalah membuat *workflow* di dalam SunSystem. *Workflow* ini digunakan untuk mengatur proses persetujuan sebelum dokumen *Purchase Order* atau PO dicetak. *Workflow* dibuat agar proses *approval* berjalan lebih rapi. Alur persetujuan menjadi jelas. Setiap PO yang akan dicetak harus mengikuti urutan yang sudah ditentukan. Sistem mencatat siapa yang menyetujui. Sistem juga mencatat waktu persetujuan dilakukan. Cara ini membantu perusahaan mengurangi kesalahan. Dokumen tidak bisa dicetak tanpa izin dari pihak yang berwenang.

Tahap pertama adalah menentukan siapa saja yang terlibat dalam *approval*. Setiap jabatan dimasukkan ke dalam alur persetujuan. Sistem kemudian diatur agar PO bergerak mengikuti urutan tersebut. Jika satu tahap belum disetujui. PO tidak bisa naik ke tahap berikutnya. Setelah semua tahap selesai. PO dapat dicetak melalui SunSystem.

1. **Membuat workflow dalam sunsystem untuk approval printing PO**



Gambar 3.20 *Workflow* PO dalam Sunsystem

Workflow ini dibuat agar proses pencetakan *Purchase Order* berjalan dengan kontrol yang jelas. Sistem memastikan setiap PO harus melewati persetujuan dari HOD (*Head Of Department*) terlebih dahulu. HOD menerima notifikasi dan mengecek detail PO sebelum memberikan keputusan. Jika setuju, proses bergerak ke tahap berikutnya. Jika tidak setuju, PO langsung kembali ke pembuat untuk direvisi.

Setelah tahap HOD selesai, sistem meneruskan PO ke FC (*Finance Controller)* untuk pemeriksaan akhir. FC mengecek kembali nilai, kebutuhan, dan kesesuaian PO. FC kemudian menentukan apakah PO layak untuk dicetak. Jika disetujui, status PO berubah menjadi siap dicetak. Sistem lalu mengirim notifikasi bahwa PO telah mendapat persetujuan lengkap.

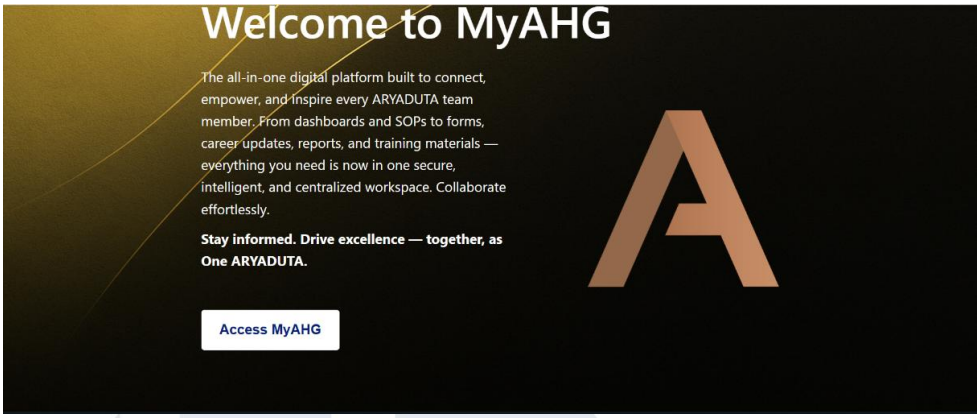
Ketika salah satu pihak menolak, sistem menghentikan alur dan mengirimkan informasi penolakan ke pembuat PO. Pembuat PO dapat meninjau ulang data yang dianggap tidak sesuai. Perbaikan dilakukan sebelum PO dikirim kembali ke alur persetujuan. Proses ini memastikan ketelitian di setiap tahap. Semua keputusan juga terekam secara jelas di SunSystem.

3.3.1.5 Pembuatan Website Portal HR menggunakan Wordpress

Setelah pembahasan mengenai *workflow* PO pada SunSystem, tahap selanjutnya adalah pembuatan *website* portal HR menggunakan WordPress. Portal ini dirancang untuk menjadi pusat informasi bagi seluruh karyawan, seperti pengumuman, dokumen kebijakan, dan akses layanan HR. WordPress dipilih karena fleksibilitasnya serta kemudahan dalam pengelolaan konten. Selain itu, platform ini memungkinkan integrasi plugin yang mendukung kebutuhan HR. Dengan demikian, portal dapat berkembang sesuai kebutuhan organisasi.

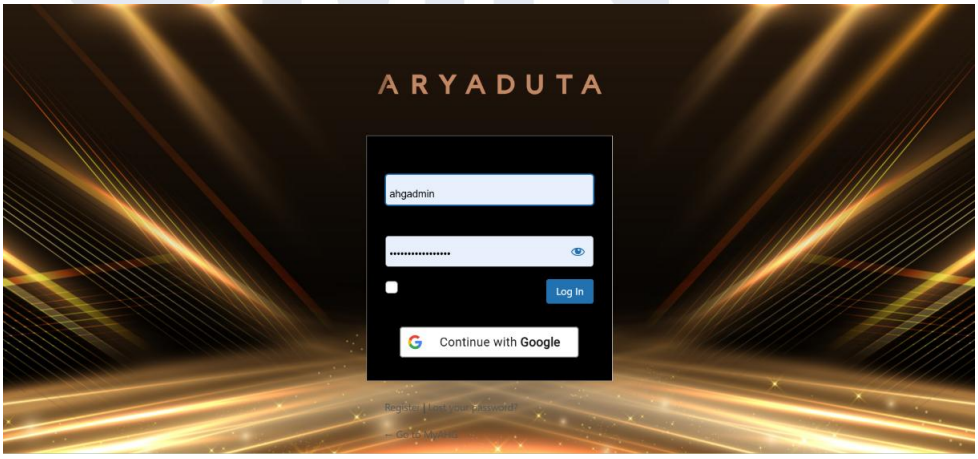
Pada bagian halaman awal, beberapa elemen dibuat secara kustom menggunakan kode HTML dan CSS. Penyesuaian ini dilakukan agar tampilan portal lebih profesional dan konsisten dengan identitas perusahaan. Kode tersebut digunakan untuk membuat *layout* tambahan yang tidak tersedia melalui tema bawaan WordPress. Hal ini termasuk desain *banner*, struktur *grid*, serta komponen visual lainnya. Melalui pendekatan ini, tampilan portal menjadi lebih menarik dan mudah digunakan.

1. **Membuat workflow dalam sunsystem untuk approval printing PO**



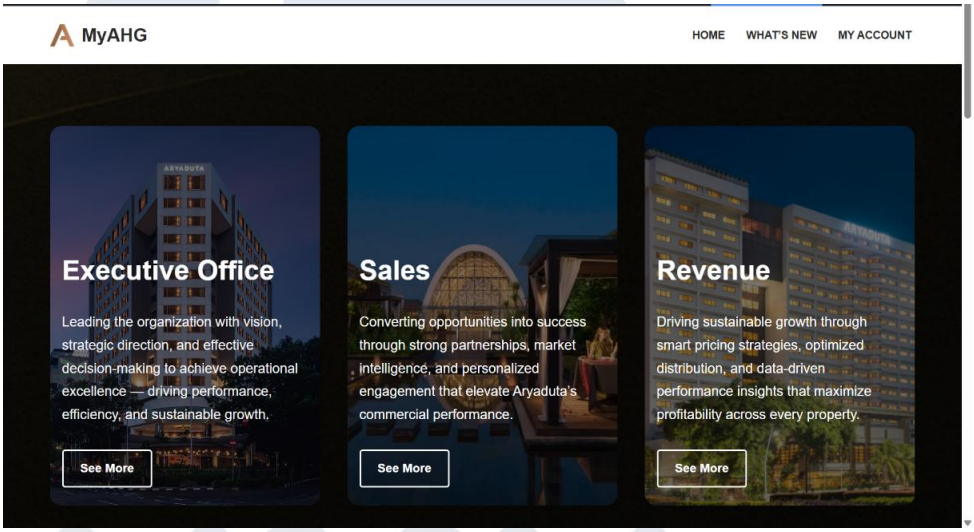
Gambar 3.21 Halaman Awal Website

Gambar 3.21 menunjukkan halaman depan dari *website* portal HR yang berfungsi sebagai pengenalan awal mengenai tujuan dan identitas platform. Pada bagian kiri halaman ditampilkan deskripsi singkat yang menjelaskan bahwa portal ini merupakan pusat informasi dan layanan internal bagi seluruh karyawan. Di sisi kanan, terdapat logo huruf A yang melambangkan Aryaduta dan telah dikustomisasi menggunakan HTML serta CSS sehingga memiliki efek animasi berputar untuk memberikan tampilan yang lebih dinamis. Elemen visual ini membantu memperkuat branding sekaligus membuat halaman terlihat lebih modern. Untuk melanjutkan ke dalam sistem, pengguna cukup menekan tombol *Access* MyAHG yang telah disediakan.



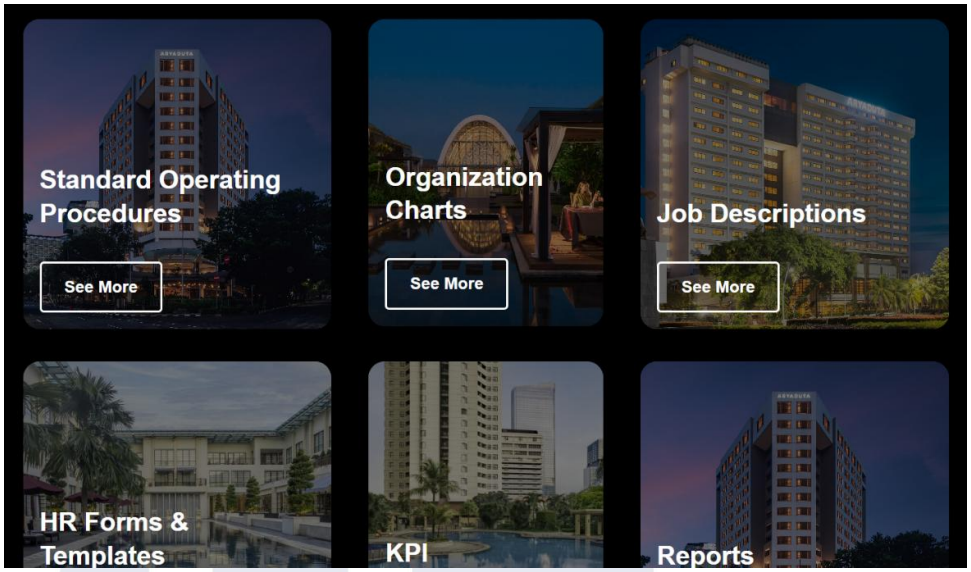
Gambar 3.22 Halaman Login MyAHG

Gambar 3.22 menunjukkan halaman login yang digunakan oleh para pengguna untuk masuk ke dalam *website* portal HR. Tampilan login dibuat sederhana namun tetap konsisten dengan tema visual Aryaduta. Hanya pengguna yang memiliki alamat email resmi Aryaduta yang dapat mengakses portal ini sehingga keamanan data tetap terjaga. Proses login dapat dilakukan dengan dua cara yaitu memasukkan email dan *password* secara manual atau menggunakan opsi Google Login untuk akses yang lebih cepat.



Gambar 3.23 Halaman Departemen

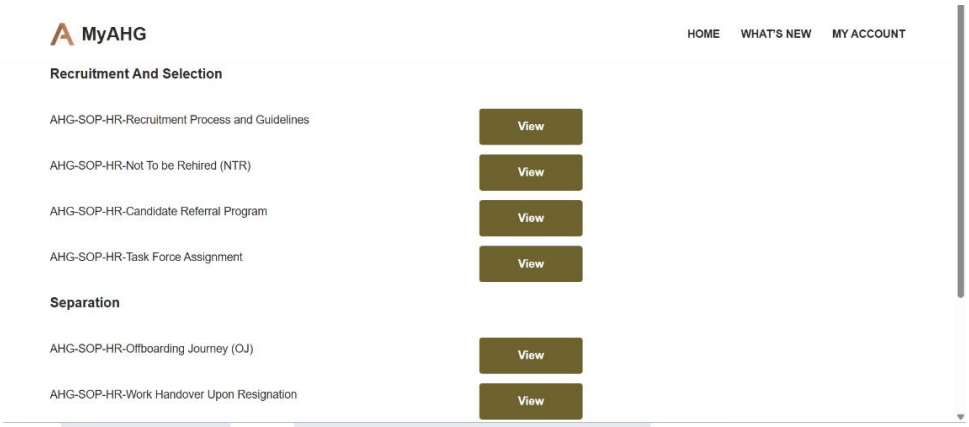
Setelah berhasil login, pengguna akan diarahkan ke halaman utama yang menampilkan daftar departemen dalam bentuk kartu. Setiap kartu mewakili satu departemen dan dilengkapi gambar serta deskripsi singkat agar pengguna mudah mengenalinya. Dari halaman ini, pengguna dapat memilih departemen yang ingin diakses untuk melihat dokumen, informasi, atau materi terkait. Tampilan dibuat rapi dan intuitif supaya pengguna bisa menavigasi portal dengan nyaman. Fitur ini membantu setiap karyawan menemukan kebutuhan mereka tanpa harus mencari secara manual.



Gambar 3.24 Halaman Komponen HR

Setelah pengguna memilih departemen Human Resources, sistem akan menampilkan beberapa kategori dokumen yang tersedia. Setiap kategori dibuat dalam bentuk kartu agar mudah dilihat dan dipilih. Pengguna bisa masuk ke menu apa pun sesuai kebutuhan, seperti SOP, struktur organisasi, job description, formulir, KPI, dan laporan. Tampilan ini membantu pengguna menemukan informasi secara cepat tanpa harus membuka satu per satu. Semua pilihan disusun rapi supaya proses pencarian dokumen jadi lebih nyaman.

2. **Membuat workflow dalam sunsystem untuk approval printing PO**



Gambar 3.25 Halaman *SOP HR*

Setelah pengguna memilih menu SOP pada departemen *Human Resources*, halaman akan menampilkan seluruh SOP yang tersedia. Setiap SOP sudah dikelompokkan berdasarkan kategori sehingga lebih mudah ditemukan. Pengguna bisa langsung memilih dokumen yang ingin dibuka melalui tombol *View* di samping nama SOP. Tombol *view* ini juga terkoneksikan dengan google drive sehingga jika ada perubahan file dalam google drive maka akan otomatis *terupdate*. Tampilan ini dibuat sederhana agar proses pencarian dokumen terasa cepat dan tidak membingungkan. Semua daftar disusun rapi supaya pengguna bisa fokus pada dokumen yang dibutuhkan.

3.3.1.6 Pembuatan RPA untuk Purchase Invoice entry menggunakan robot framework

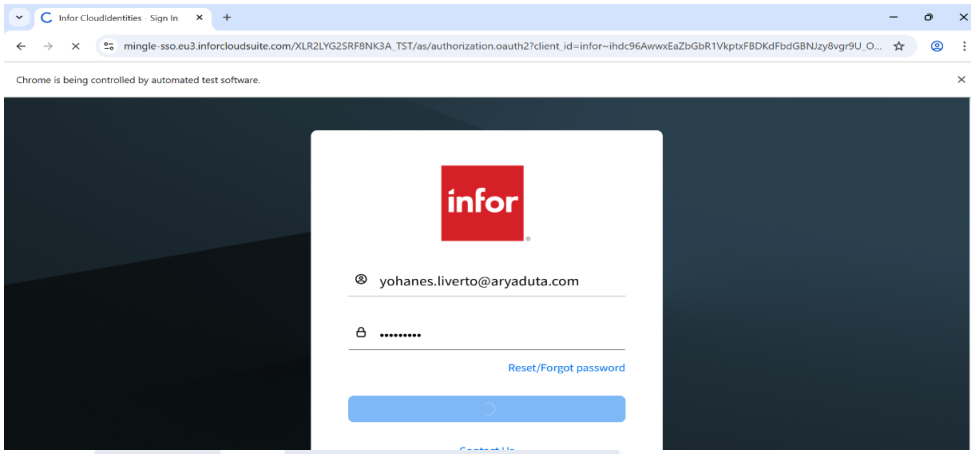
Setelah selesai membangun *website* portal HR sebagai pusat informasi untuk seluruh departemen, tahap berikutnya adalah pengembangan RPA (*Robotic Process Automation*) untuk proses *Purchase Invoice* (PI) Entry. RPA ini dibuat menggunakan Robot Framework dengan tujuan mengotomasi alur kerja input *payment*

journal, sehingga pengguna tidak perlu lagi melakukan pengecekan dan pencocokan *invoice* satu per satu secara manual. Dengan adanya otomasi ini, proses yang sebelumnya memakan waktu lama dapat dilakukan dengan lebih cepat, konsisten, dan minim kesalahan. Sistem RPA akan membaca data *invoice*, melakukan validasi otomatis, lalu memasukkannya ke dalam sistem sesuai format yang ditentukan. Selain mempercepat proses kerja, pengembangan RPA ini juga dirancang untuk menyelesaikan permasalahan nyata yang sering muncul ketika menangani *invoice* berjumlah besar. Misalnya, ketika terdapat satu *invoice* dengan nilai 100 juta yang harus dibagi menjadi 30 *invoice* kecil untuk keperluan proses pembayaran di sistem, pekerjaan tersebut biasanya memakan waktu lama jika dilakukan secara manual. Dengan RPA yang dibangun menggunakan Robot Framework, seluruh proses pemecahan nilai invoice, pembuatan item-item baru, hingga pencocokan terhadap kode rekonsiliasi bank dapat dijalankan secara otomatis dari awal hingga akhir. Pengguna cukup menyiapkan data utama, lalu robot akan bekerja melakukan cross-check, memastikan format sudah sesuai, dan menginput seluruh baris transaksi ke dalam sistem tanpa perlu intervensi berulang. Pendekatan ini tidak hanya meningkatkan efisiensi waktu, tetapi juga mengurangi potensi kesalahan manusia yang sering terjadi saat mengolah data dalam jumlah besar.

1. Menggunakan VS Code dan robot framework untuk membuat otomasi robot menggunakan selenium dan python

Gambar 3.26 *Code* RPA PI *Entry*

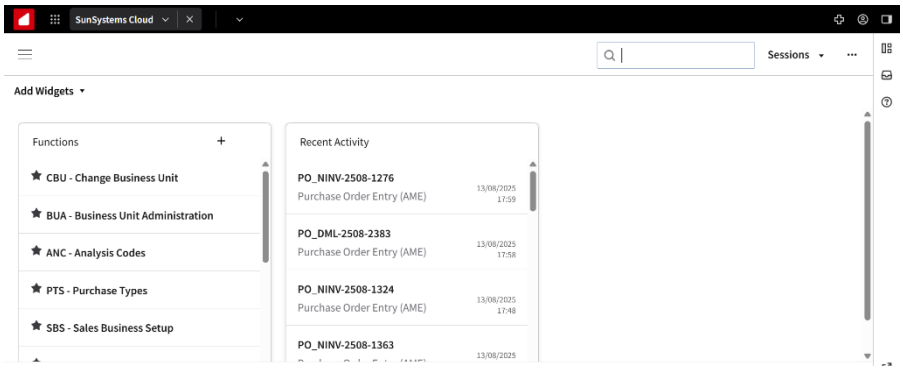
Pada gambar 3.26 terlihat bagian awal dari skrip RPA yang digunakan untuk proses otomatisasi login ke SunSystem. Di dalam file ini terdapat beberapa variabel penting seperti URL, *username*, dan *password* yang menjadi kredensial utama agar robot dapat masuk ke sistem tanpa interaksi manual. Proses login ini menggunakan Selenium sebagai library utama, karena Selenium mampu menirukan perilaku pengguna secara langsung di browser. Selain itu, terdapat juga variabel *Business Unit* seperti AJK (Aryaduta Jakarta Menteng) yang digunakan untuk menentukan unit mana yang akan diproses oleh robot dalam alur berikutnya. Dengan struktur variabel yang tertata seperti ini, robot dapat dijalankan secara fleksibel, cukup dengan mengubah nilai pada parameter yang diperlukan tanpa harus memodifikasi seluruh skrip. Pendekatan ini membuat proses otomatisasi lebih mudah diatur, lebih aman, dan jauh lebih efisien ketika digunakan berulang kali dalam aktivitas operasional sehari-hari.



Gambar 3.27 Tampilan Login *RPA*

Ketika perintah python -m robot test.robot dijalankan melalui terminal, Robot Framework akan langsung memicu Selenium untuk membuka browser Chrome secara otomatis. Pada gambar terlihat bahwa Chrome menampilkan notifikasi "*Chrome is being controlled by automated test software*", yang menandakan bahwa proses ini sepenuhnya dijalankan oleh robot tanpa interaksi manual.

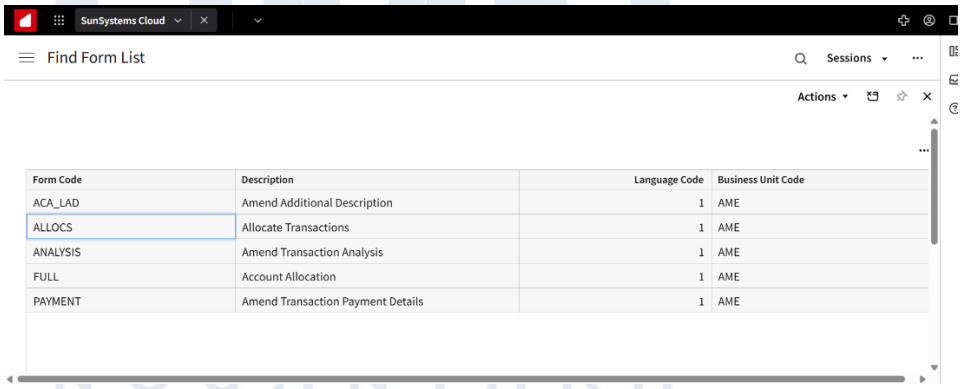
Robot kemudian menavigasi ke URL login Infor SunSystem yang sudah didefinisikan dalam variabel ${URL} di dalam file .robot. Setelah halaman terbuka, skrip otomatis mengisi kolom *username* dan *password* sesuai kredensial yang telah ditentukan. Proses ini dilakukan menggunakan perintah Selenium seperti *Input Text, Wait Until Element Is Visible*, dan *Click Button*.



Gambar 3.28 Tampilan Halaman Sunsystem Cloud

Setelah proses login berhasil, robot akan diarahkan otomatis ke halaman utama SunSystem seperti yang terlihat pada gambar. Di tahap ini, Selenium mulai menjalankan instruksi berikutnya sesuai dengan skenario otomasi. Salah satu langkah awal yang dilakukan robot adalah mengetikkan kode PIE (*Purchase Invoice Entry*) pada kolom pencarian modul di bagian atas halaman. Aksi ini dilakukan untuk membuka menu *Purchase Invoice Entry*, yang menjadi titik awal dalam proses rekonsiliasi invoice yang akan diotomasi.

Begitu modul PIE terbuka, sistem akan meminta pengguna untuk memilih *Business Unit* sebelum bisa melanjutkan. Karena skenario yang dijalankan adalah untuk Aryaduta Medan, robot secara otomatis memilih *Business Unit* AME dari daftar yang tersedia. Pemilihan ini penting karena setiap *business unit* memiliki data dan transaksi yang berbeda, sehingga robot harus memastikan bahwa seluruh proses dilakukan pada unit yang benar. Setelah *business unit* dipilih, barulah robot dapat melanjutkan ke tahap berikutnya seperti pengisian data *invoice*, pengecekan jumlah, hingga pencocokan kode rekonsiliasi secara otomatis.

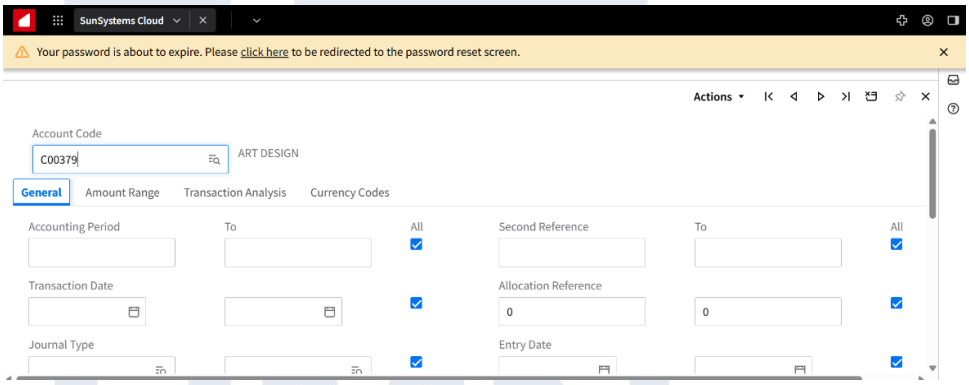


Gambar 3.29 Tampilan Halaman *Form Code*

Setelah robot berhasil memilih *Business Unit* AME, sistem akan menampilkan daftar *form* yang tersedia untuk unit tersebut. Pada tahap ini, otomasi sudah berjalan penuh tanpa interaksi pengguna. Robot kemudian akan mencari form dengan *Form Code*: ALLOCS

(*Allocate Transactions*). Form ini merupakan bagian penting dalam proses alokasi dan pencocokan transaksi, terutama ketika *invoice* yang besar sudah dipecah menjadi beberapa bagian yang lebih kecil. Dengan memilih ALLOCS, robot dapat masuk ke menu yang digunakan untuk mengalokasikan atau mencocokkan transaksi secara otomatis.

Begitu form ALLOCS *terhighlight* sesuai yang ditampilkan pada gambar, robot akan mengeksekusi perintah Enter secara otomatis. Aksi ini membuka form ALLOCS dan membawa robot menuju tahapan berikutnya untuk mulai memproses data invoice, melakukan alokasi, hingga mengeksekusi pencocokan kode rekonsiliasi.



Gambar 3.30 Tampilan Halaman ALLOCS

Setelah robot masuk ke menu ALLOCS, halaman berikutnya menampilkan form alokasi transaksi yang dimulai dari kolom *Account Code*. Pada tahap ini, robot tidak mengisi nilai secara statis, tetapi melakukan pencarian data berdasarkan input yang sudah disiapkan oleh user sebelumnya dalam bentuk file Excel. File tersebut biasanya berisi daftar *invoice* yang harus dialokasikan, lengkap dengan informasi seperti nomor *invoice*, total *invoice* yang sudah dipecah, hingga *account code* yang relevan untuk proses pencocokan.

Robot kemudian membaca file Excel tersebut, mengekstrak nilai *Account Code* yang diperlukan, lalu secara otomatis mengetikkannya ke kolom yang tersedia di layar ALLOCS seperti yang terlihat pada

gambar. Dengan demikian, *user* tidak perlu lagi melakukan input manual ataupun mencari *account code* satu per satu. Setelah *field* ini terisi dengan benar, robot siap melanjutkan tahap berikutnya dalam proses alokasi transaksi sesuai workflow yang sudah ditentukan.

Gambar 3.31 Tampilan Halaman *Account Allocation*

Setelah robot mengisi *Account Code* dan menampilkan seluruh daftar transaksi yang berkaitan, proses berlanjut pada tahap inti yaitu melakukan alokasi otomatis. Pada tampilan *Account Allocation* ini, setiap baris transaksi masih berada pada status *Not Allocated*, yang berarti transaksi tersebut belum dipasangkan atau dicocokkan satu sama lain.

Robot kemudian mulai bekerja dengan membaca nilai-nilai transaksi debit dan kredit, lalu mencocokkannya berdasarkan informasi yang diambil dari file Excel—misalnya total *invoice* yang dipecah menjadi beberapa bagian, kode referensi pembayaran, hingga nilai yang harus diseimbangkan. Seluruh proses pencocokan ini dilakukan secara otomatis tanpa intervensi manual dari user. Ketika nilai debit dan kredit sudah menemukan pasangan yang sesuai, robot akan melakukan aksi alokasi pada SunSystems.

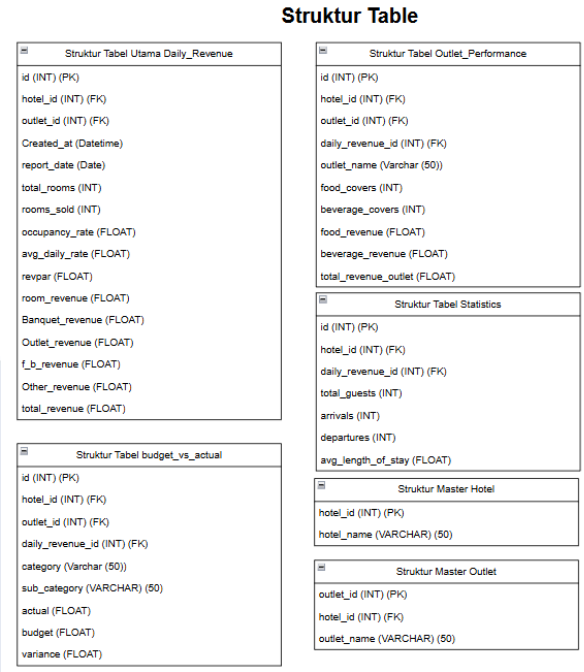
Setelah semua transaksi yang relevan sudah berhasil dicocokkan, kolom *Allocation Marker*, yang awalnya menampilkan status *Not Allocated*, akan berubah menjadi *Allocated*. Perubahan ini

menandakan bahwa transaksi-transaksi tersebut sudah selesai dipasangkan dan sistem telah mengakui alokasinya. Dengan langkah ini, *user* tidak lagi perlu melakukan pencocokan dan alokasi satu per satu secara manual, sehingga menghemat waktu dan meminimalisir risiko *human error* dalam proses rekonsiliasi pembayaran.

3.3.1.7 Pembuatan Database untuk data DRR yang diolah dari excel, kemudian juga VBA dari DRR yang akan dijadikan Dashboard BI

Setelah proses pembuatan RPA selesai, pekerjaan berikutnya berfokus pada pembangunan BI (*Business Inteligence*) DRR (*Daily Revenue Report)*, yang dimulai dari pembuatan *database* sebagai fondasi penyimpanan data utama. Data DRR yang awalnya berada dalam format Excel perlu diolah terlebih dahulu agar strukturnya lebih rapi dan mudah digunakan sehingga proses integrasinya ke dalam *database* dapat berjalan dengan stabil. Selain itu, dibuat juga VBA pada Excel untuk membantu proses penginputan data ke dalam database termasuk dengan pembersihan data, penggabungan file, hingga standarisasi format sebelum diproses lebih lanjut. Tahap berikutnya adalah membangun *dashboard* menggunakan Power BI, di mana seluruh data dari *database* yang merupakan hasil olahan VBA ditampilkan dalam bentuk visual yang informatif dan mudah dianalisis. Dengan kombinasi tiga komponen ini, proses pemantauan DRR menjadi jauh lebih sistematis, cepat, dan akurat bagi seluruh *stakeholder*.

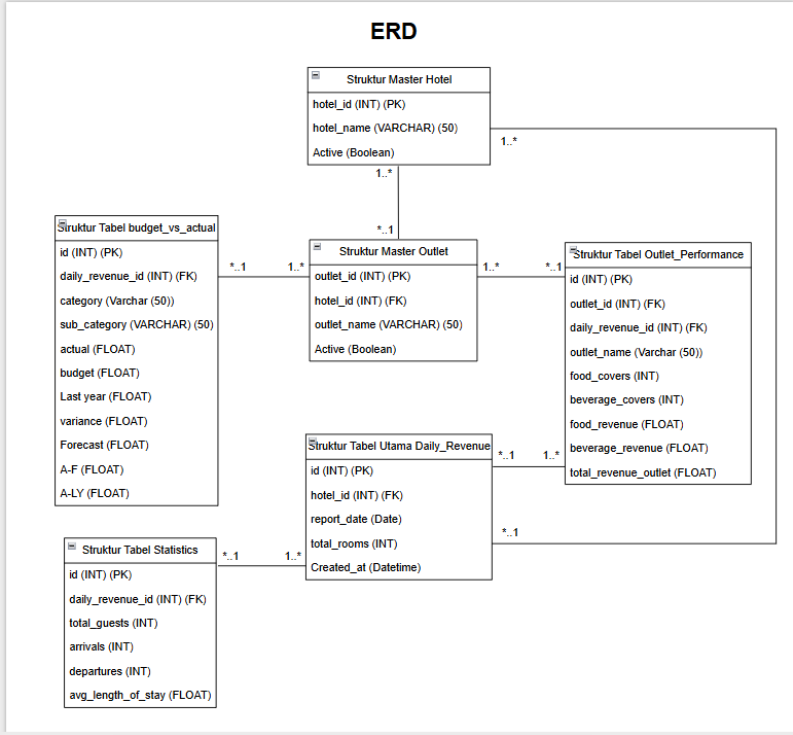
1. Membuat struktur tabel database menggunakan draw io



Gambar 3.32 Struktur Table DRR

Pada gambar 3.32 dapat terlihat bahwa struktur database BI DRR terdiri dari enam tabel utama, di mana masing-masing tabel memiliki fungsi dan jenis data yang berbeda sesuai kebutuhan analisis. Setiap tabel dirancang menggunakan kombinasi tipe data seperti *VARCHAR* untuk teks, *INTEGER* untuk nilai numerik utuh, *FLOAT* untuk angka desimal seperti *revenue* atau *occupancy rate*, serta *DATETIME/DATE* untuk pencatatan waktu transaksi. Perbedaan tipe data ini penting agar proses penyimpanan, pengolahan, dan integrasi data dari Excel ke *database* dapat berjalan dengan akurat dan efisien. Selain itu, adanya relasi antar tabel seperti hotel_id, outlet_id, dan daily_revenue_id memungkinkan proses analitik di Power BI menjadi jauh lebih terstruktur dan mudah ditelusuri. Dengan desain *database* seperti ini, seluruh data DRR dapat dikelola secara konsisten dan siap digunakan dalam pembuatan dashboard BI yang lebih informatif.

2. Membuat ERD untuk masing masing relasi antar tabel



Gambar 3.33 ERD DRR

Gambar 3.33 di atas memperlihatkan bagaimana keenam tabel dalam *database* saling terhubung melalui relasi yang telah dirancang sesuai kebutuhan analisis DRR. Relasi yang paling terlihat adalah antara *Master Hotel* dengan beberapa tabel lain seperti *Daily Revenue*, *Master Outlet*, dan tabel-tabel pendukung lainnya. Hal ini menunjukkan bahwa satu hotel dapat memiliki banyak data *revenue* harian, yang mencakup beragam jenis pendapatan seperti *room revenue, food & beverage revenue*, hingga *other revenue*. Oleh karena itu, relasi yang digunakan adalah *one-to-many*, di mana satu entitas hotel dapat memiliki banyak entri *revenue* di tabel *Daily_Revenue*.

Selain itu, struktur relasi pada *Master Outlet* juga menunjukkan pola yang sama. Satu hotel dapat memiliki banyak outlet, sehingga hubungan antara Hotel dan Outlet kembali menggunakan relasi *one-to-many*, sementara setiap outlet akan terhubung ke tabel Outlet

Performance yang berisi rincian penjualan dan performa outlet tersebut. Begitu juga pada tabel *budget_vs_actual* serta tabel *Statistics*, yang keduanya mengacu ke *Daily_Revenue* dengan relasi *many-to-one*, menandakan bahwa setiap *record* data statistik maupun *budget* terkait dengan satu data *revenue* harian tertentu. Relasi-relasi ini memastikan bahwa seluruh data dapat ditelusuri kembali ke hotel dan tanggal yang sesuai, sehingga laporan DRR dapat disusun dengan akurat, konsisten, dan mudah diolah pada tahap visualisasi di Power BI.

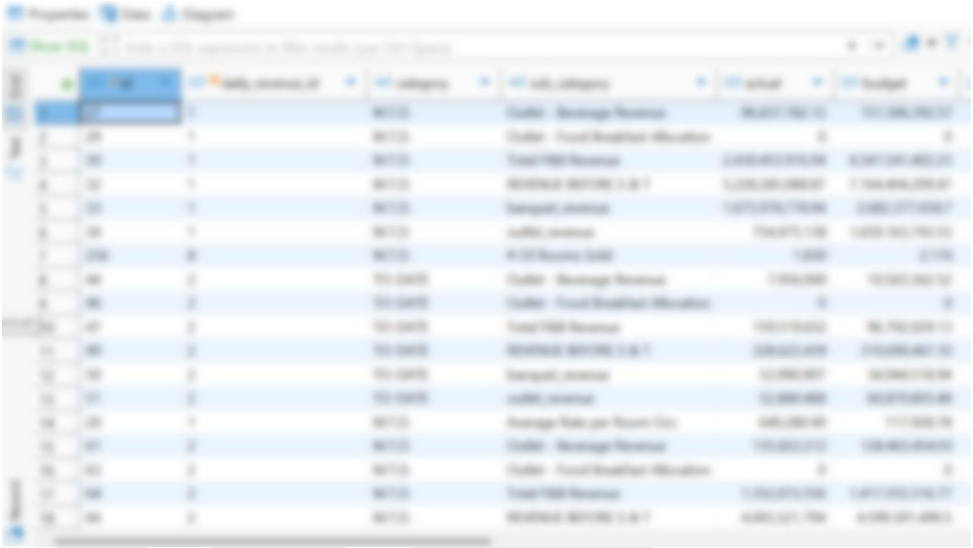
3. Membuat tabel dalam database sesuai dengan struktur dan relasi yang telah dibuat

Properties | Data | Diagram

Show SQL | Enter a SQL expression to filter results (use Ctrl+Space)

Grid | Text

	d	hotel_name	active	hotel_id
1	6	ARYADUTA LIPPO VILLAGE	[v]	1
2	1	ARYADUTA MENTENG JAKARTA	[v]	2
3	3	ARYADUTA MEDAN	[v]	3
4	9	ARYADUTA MANADO	[v]	4
5	8	ARYADUTA PEKANBARU	[v]	5
6	4	ARYADUTA SUITES SEMANGGI	[v]	6
7	7	ARYADUTA PALEMBANG	[v]	7
8	5	ARYADUTA BANDUNG	[v]	8
9	2	ARYADUTA KUTA BALI	[v]	9
10	10	IKG	[v]	11
11	11	PRI	[v]	12
12	12	ACC	[v]	13
13	13	PSBC	[v]	18
14	14	WBO	[v]	17



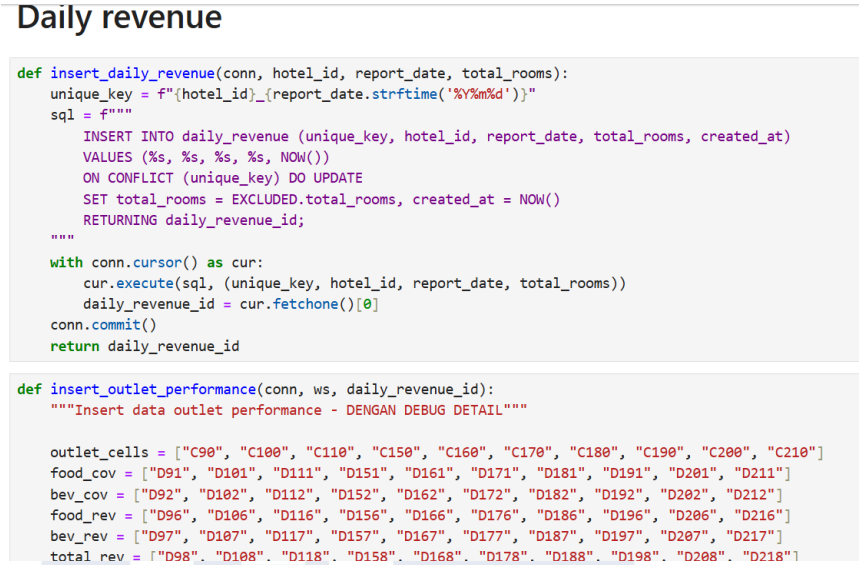
Gambar 3.34 Tabel Sesuai Struktur

Gambar 3.34 menunjukkan tabel-tabel yang telah dibuat sesuai dengan struktur yang sebelumnya dirancang dalam ERD. Tabel pertama berisi data *master hotel*, di mana terlihat terdapat sembilan hotel yang aktif beroperasi beserta lima *business unit* yang tercatat di dalam sistem. Selanjutnya, tabel kedua menampilkan data *budget vs actual* yang merepresentasikan *revenue* harian dari masing-masing hotel, termasuk berbagai kategori pendapatan seperti *room revenue*, *food & beverage revenue*, serta pendapatan khusus *outlet*. Data ini nantinya akan menjadi dasar untuk melakukan analisis performa harian maupun bulanan sehingga proses monitoring keuangan dapat dilakukan dengan lebih cepat dan akurat. Dengan adanya struktur data yang rapi dan terintegrasi, proses pengolahan menuju *dashboard* BI dapat dilakukan secara lebih efisien dan minim kesalahan manual.

4. Membuat kode python untuk preprocessing data untuk dimasukkan ke dalam database

Gambar 3.35 Python Import Zip Data

Pada gambar 3.35 diperlihatkan tahap awal dari proses *preprocessing* menggunakan Python sebelum data dimasukkan ke dalam *database*. Langkah pertama yang dilakukan adalah mengimpor berbagai library yang dibutuhkan, seperti zipfile, pandas, serta konektor PostgreSQL untuk keperluan upload data. Selanjutnya, file ZIP yang berisi kumpulan laporan DRR diekstraksi ke dalam folder khusus agar seluruh file Excel di dalamnya dapat diakses dengan mudah. Setelah proses ekstraksi selesai, sistem melakukan pengecekan otomatis dan berhasil menemukan 30 file Excel yang siap diproses lebih lanjut. Tahap ini memastikan bahwa seluruh data sumber telah teridentifikasi dengan benar sebelum masuk ke proses pembersihan dan transformasi.

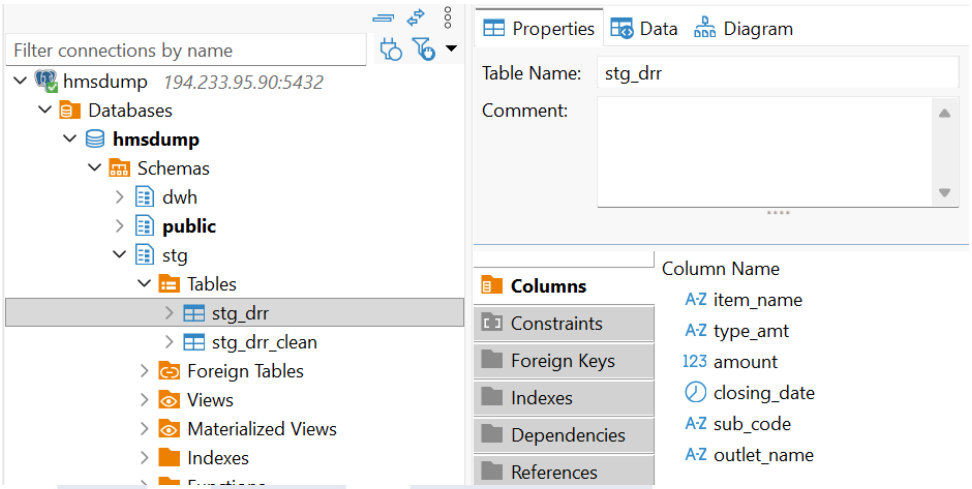


Gambar 3.36 *Code Insert* ke dalam *database*

Setelah proses import dan ekstraksi file selesai, langkah berikutnya adalah menentukan konfigurasi koneksi ke *database*, seperti nama *database, host, port, username*, dan *password*. Koneksi ini digunakan agar Python dapat mengirimkan data ke tabel-tabel yang telah dibuat sebelumnya. Pada tahap ini, setiap kolom di file Excel dipetakan ke kolom yang sesuai di dalam tabel *database*. Fungsi-fungsi Python kemudian membaca sel-sel tertentu di Excel (misalnya kolom *revenue, occupancy, outlet performance*, dan sebagainya) untuk memastikan seluruh nilai yang dibutuhkan dapat diambil dengan benar.

Selanjutnya, setiap data yang berhasil *difetch* akan dieksekusi ke dalam perintah SQL *INSERT* atau *UPDATE* tergantung apakah data tersebut sudah ada atau belum. Proses ini menghasilkan log pada *output* Jupyter Notebook, sehingga pengguna dapat melihat apakah proses *upload* data berhasil atau terdapat *error* yang perlu diperbaiki. Dengan cara ini, setiap baris data dari Excel dapat masuk secara otomatis dan konsisten ke dalam *database*.

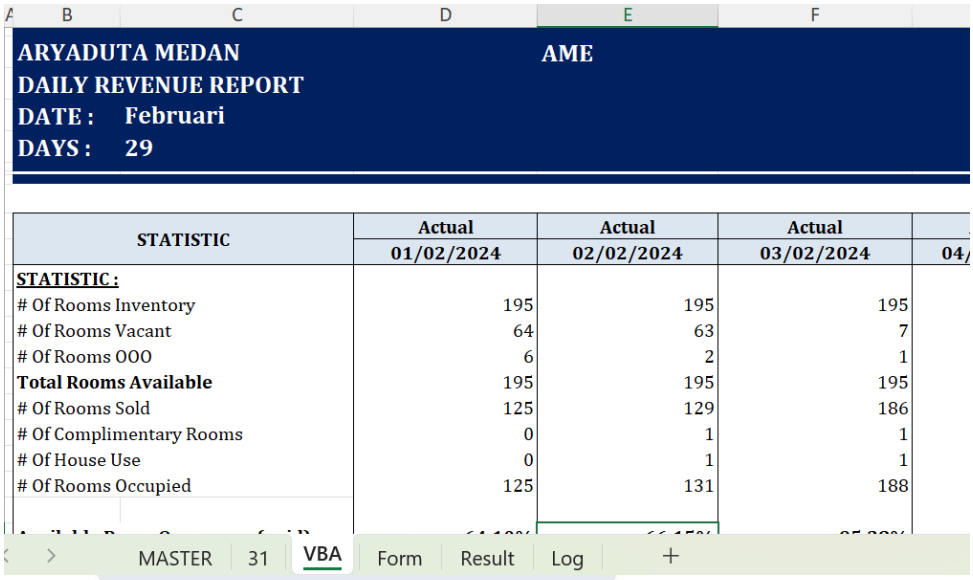
5. Membuat schema database agar data yang berada di fact table sudah yang bersih



Gambar 3.37 *Schema* Dalam *Database*

Setelah data berhasil dimasukkan ke dalam *schema public*, proses selanjutnya adalah memindahkan data tersebut ke *schema staging* (stg) untuk dilakukan pembersihan dan standarisasi. Pada tahap staging ini, data dicek kembali untuk memastikan tidak ada duplikasi, format tanggal sudah konsisten, tipe data sudah sesuai, serta penamaan outlet, item, dan sub-category telah distandardisasi. Tahap ini penting untuk memastikan data benar-benar bersih sebelum naik ke level berikutnya. Setelah itu, data yang sudah melewati proses validasi dan *cleaning* akan dipindahkan ke schema *Data Warehouse* (dwh) atau *fact*, yang merupakan tempat penyimpanan data akhir yang siap digunakan untuk analisis dan visualisasi. Di *schema* ini, data sudah stabil, terstruktur, dan tidak berubah sehingga aman digunakan oleh Power BI dan laporan lainnya.

6. Membuat VBA Macro excel untuk mempermudah orang orang dalam memasukkan data ke dalam database



ARYADUTA MEDAN AME
DAILY REVENUE REPORT
DATE : Februari
DAYS : 29

STATISTIC	Actual 01/02/2024	Actual 02/02/2024	Actual 03/02/2024	04/
STATISTIC :				
# Of Rooms Inventory	195	195	195	
# Of Rooms Vacant	64	63	7	
# Of Rooms OOO	6	2	1	
Total Rooms Available	195	195	195	
# Of Rooms Sold	125	129	186	
# Of Complimentary Rooms	0	1	1	
# Of House Use	0	1	1	
# Of Rooms Occupied	125	131	188	

MASTER | 31 | VBA | Form | Result | Log | +

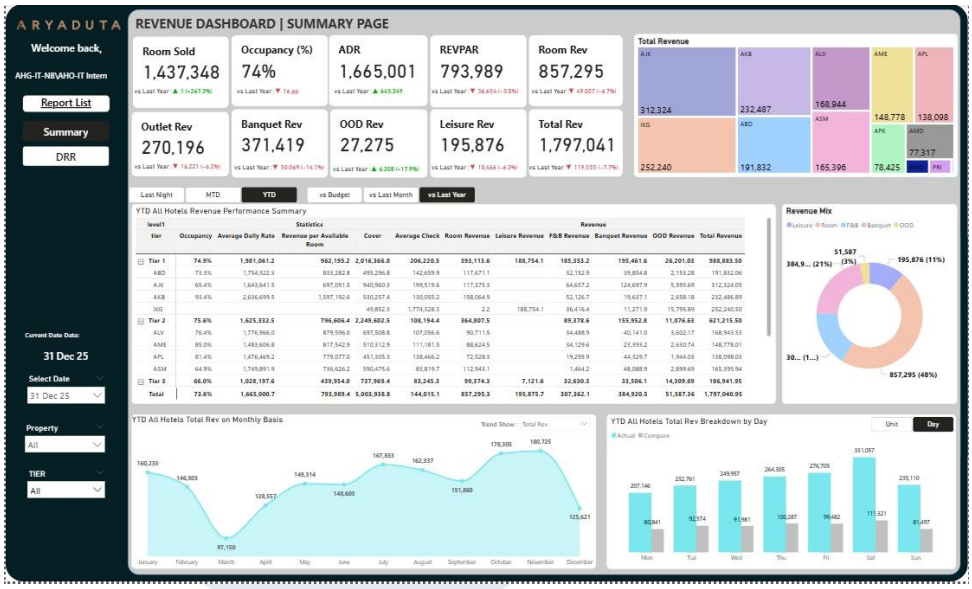
Gambar 3.38 VBA DRR

Gambar 3.38 menunjukkan tampilan template VBA (*Visual Basic for Application*) *Daily Revenue Report* yang digunakan oleh setiap unit hotel Aryaduta untuk melakukan input data pendapatan harian dengan lebih mudah dan konsisten. Melalui tombol macro yang disediakan, *user* tidak perlu lagi memasukkan data secara manual satu per satu ke dalam sistem. Di balik tombol tersebut, terdapat kode VBA yang sudah diatur untuk membaca baris dan kolom tertentu, misalnya data *rooms, occupancy, food & beverage*, hingga statistik operasional lainnya. Semua informasi yang terdeteksi akan dikirimkan langsung ke *database* sesuai alur tiga schema: public → staging → fact.

Selain itu, proses pengiriman data menjadi jauh lebih efisien karena VBA memanggil *stored procedure* yang telah disiapkan sebelumnya. *Stored procedure* ini bertugas membersihkan data, melakukan validasi, dan menaikkan data tersebut sampai ke tabel fact tanpa perlu campur tangan manual. Dengan cara ini, setiap hotel cukup menekan satu tombol untuk memastikan seluruh revenue harian

tercatat rapi, tersimpan di *database* yang benar, dan siap diproses lebih lanjut untuk keperluan analisis melalui Power BI.

7. Membuat visualisasi dashboard dalam Power BI dengan dax calculation



Gambar 3.39 *Dashboard* DRR Aryaduta (*Data Dummy*)

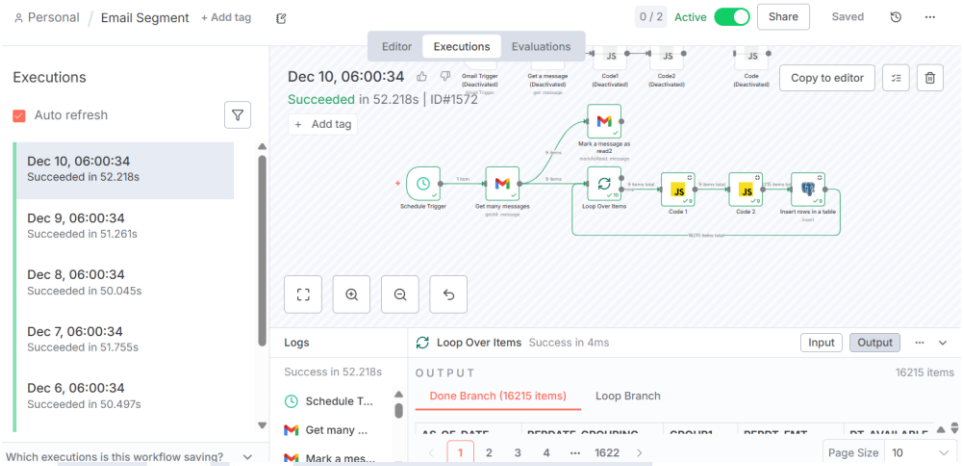
Dashboard Power BI Aryaduta *Revenue Report* ini menampilkan rangkuman lengkap mengenai performa pendapatan seluruh hotel dalam satu tampilan yang informatif. Pada bagian atas *dashboard*, terlihat ringkasan KPI utama seperti *Room Sold, Occupancy, ADR, RevPAR, Room Revenue, F&B Revenue, Banquet Revenue*, dan *Total Revenue*, masing-masing dilengkapi perbandingan terhadap *budget* sehingga memudahkan manajemen mengevaluasi pencapaian aktual. Di bagian berikutnya terdapat ringkasan pendapatan MTD per tier hotel, yang memperlihatkan kontribusi masing-masing hotel dari berbagai sumber revenue seperti *Room*, F&B, *Banquet*, dan lainnya. *Dashboard* ini juga menyertakan statistik operasional penting, seperti *occupancy, rooms sold, rooms available,* ADR, dan RevPAR untuk setiap hotel, sekaligus perbandingannya dengan *budget*. Selain itu, terdapat visualisasi performa *outlet* yang menunjukkan pencapaian

actual versus budget untuk setiap outlet hotel. Bagian bawah *dashboard* menampilkan tren ADR harian yang memperlihatkan fluktuasi kinerja harian seluruh hotel, serta *breakdown* ADR berdasarkan hari untuk melihat pola performa mingguan.

3.3.1.8 *Workflow Automation* Segmentasi data menggunakan N8N dan pembuatan Dashboard BI

Setelah data DRR berhasil diproses dan divisualisasikan dalam bentuk dashboard BI, proyek berikutnya adalah mengotomasi proses segmentasi data menggunakan N8N. Setiap hari, data segmentasi yang berasal dari berbagai sumber diproses secara otomatis oleh *workflow* N8N, kemudian dimasukkan ke dalam *database* sesuai struktur *schema* yang telah ditentukan. Proses otomasi ini memastikan bahwa data selalu terbarui tanpa intervensi manual, mengurangi risiko kesalahan input serta meningkatkan efisiensi pengolahan data. Data segmentasi yang sudah tersimpan di *database* tersebut selanjutnya digunakan sebagai sumber untuk membangun *dashboard Business Intelligence*, sehingga manajemen dapat memantau performa segmentasi pelanggan, tren, serta kontribusi masing-masing *segment* dengan lebih cepat dan akurat.

1. Membuat flow dalam N8N untuk otomasi fetching email ke database



Gambar 3.40 *Workflow* N8N

Gambar 3.40 menunjukkan sebuah *workflow* otomatis yang dibangun menggunakan N8N untuk memproses data segmentasi harian. *Workflow* dimulai dengan *schedule trigger* yang berjalan setiap hari pada pukul 06.00. Setelah aktif, sistem akan mengambil seluruh email yang memiliki label *Segment* dan berstatus *unread*. Untuk mencegah duplikasi data, setiap email yang telah diambil akan langsung di-*mark as read*. Selanjutnya, workflow melakukan proses *looping* secara bertahap untuk membaca setiap email dan mengekstrak *attachment* berupa file CSV yang dikirimkan otomatis oleh sistem setiap hari. Karena format data pada CSV sering berantakan dan mengandung banyak *missing values*, dua tahapan JavaScript coding digunakan untuk membersihkan data: *Code* 1 mengekstrak isi *attachment* CSV, sementara *Code* 2 menyeleksi kolom dan baris yang dibutuhkan sesuai kebutuhan pengguna. Setelah data dibersihkan, hasil akhirnya dimasukkan ke database PostgreSQL ke dalam tabel yang telah ditentukan, sehingga data siap digunakan untuk proses analisis dan pembuatan *dashboard* BI.

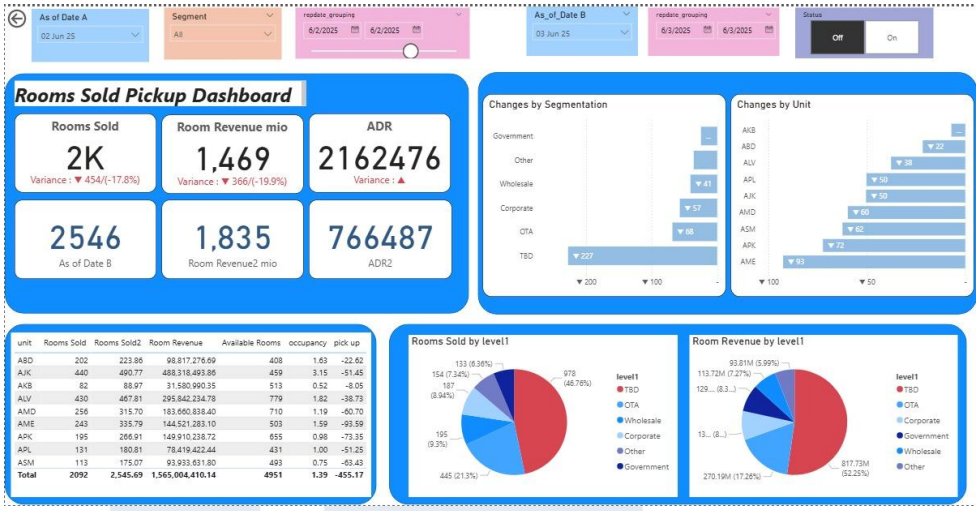
2. Membuat visualisasi dashboard menggunakan Power BI

Gambar 3.41 Visualisasi *Dashboard* Segment

Gambar 3.41 menampilkan visualisasi *dashboard* segmentasi yang dibangun menggunakan Power BI. *Dashboard* ini memungkinkan pengguna untuk membandingkan performa segment baik secara *daily* maupun *monthly*, sehingga analisis dapat dilakukan dengan lebih fleksibel. Pada bagian atas *dashboard*, ditampilkan beberapa indikator kinerja utama (KPI) seperti *Rooms Sold, Room Revenue, Occupancy*, dan *ADR* yang dapat dilihat untuk *Current Year* maupun *Last Year*, sehingga memudahkan dalam melihat pertumbuhan atau penurunan performa dari waktu ke waktu. Di bagian bawah, terdapat tabel detail yang menampilkan kontribusi masing-masing segment, lengkap dengan *Rooms Sold, Room Revenue, Room Night, Occupancy*, dan *ADR*.

Gambar 3.42 Barchart *Daily Performance*

Pada visualisasi gambar 3.42 ditampilkan laporan segmentasi yang memungkinkan pengguna melihat performa harian berdasarkan berbagai indikator seperti *Occupancy, ADR, Room Night, Room Revenue*, dan *Room Sold*. Grafik batang di bagian bawah menunjukkan *landing projection* dari masing-masing *segment* untuk setiap tanggal, sehingga memudahkan identifikasi hari-hari dengan performa terbaik maupun yang perlu mendapat perhatian. Dengan tampilan yang jelas dan interaktif ini, pengguna dapat dengan cepat mengetahui hari mana yang memiliki tingkat *occupancy* paling tinggi, bagaimana fluktuasi performa antar segmen, serta tren pergerakan kinerja dari waktu ke waktu.



Gambar 3.43 *Dashboard Rooms Sold Pickup (Data Dummy)*

Dashboard pada gambar 3.43 ini memberikan gambaran menyeluruh mengenai performa *pickup* harian dari berbagai segmentasi maupun unit hotel. Melalui tampilan KPI seperti *Rooms Sold, Room Revenue*, dan ADR, pengguna dapat langsung melihat perbandingan antara *As of Date* A dan *As of Date* B, lengkap dengan variasinya. Grafik *Changes by Segmentation* dan *Changes by Unit* membantu mengidentifikasi *segment* atau unit mana yang mengalami kenaikan maupun penurunan *pickup* paling signifikan. Selain itu, *dashboard* ini juga memungkinkan perbandingan terhadap performa pada hari yang sama di tahun sebelumnya sehingga dapat diketahui apakah pencapaiannya lebih tinggi atau lebih rendah. Jika performanya lebih rendah, pengguna dapat menelusuri segmen mana yang kurang berkontribusi, apakah itu dari *Corporate*, OTA, *Wholesale*, atau lainnya. Visualisasi berbentuk *pie chart* di bagian bawah turut menunjukkan distribusi *Rooms Sold* dan *Room Revenue* berdasarkan level *segment* sehingga analisis dapat dilakukan lebih mendalam dan berbasis data yang akurat.

3.3.2 Kendala yang Ditemukan

Selama menjalani masa magang sebagai IT *Data Analyst* di PT Aryaduta International Management, ada beberapa kendala yang ditemui, antara lain:

1. Pengolahan data dalam skala besar mengalami hambatan karena keterbatasan infrastruktur yang tersedia. Sistem saat ini berjalan pada *virtual machine* (VM) dengan spesifikasi 8 core CPU, RAM 16 GB, dan *storage* 512 GB, serta sumber daya tersebut digunakan secara bersama (*shared resource*) dengan aplikasi lain. Kondisi ini menyebabkan performa VM menjadi kurang optimal, terutama ketika menjalankan proses transformasi data, *preprocessing* dalam jumlah besar, maupun eksekusi *query* yang kompleks, sehingga waktu pemrosesan data menjadi lebih lama.
2. Perangkat kerja yang digunakan berupa laptop Dell keluaran tahun 2021 dengan prosesor Intel Core i7 generasi ke-11, RAM sebesar 8 GB, dan *storage* 512 GB. Spesifikasi tersebut dirasa kurang memadai untuk menangani pekerjaan yang bersifat intensif, seperti pemrosesan data, pengembangan *workflow* automasi, serta eksekusi *query database* dalam jumlah besar. Akibatnya, laptop sering mengalami penurunan performa, seperti *lag* atau *freeze*, yang berdampak pada produktivitas kerja sehari-hari.
3. Terdapat permasalahan duplikasi data pada *Daily Revenue Report* (DRR) yang disebabkan oleh kesalahan input data, di mana data yang seharusnya dicatat secara harian justru terinput sebagai data bulanan. Kesalahan ini menimbulkan anomali pada data, sehingga memengaruhi keakuratan hasil analisis dan pelaporan, serta memerlukan proses validasi dan data cleansing tambahan sebelum data dapat digunakan lebih lanjut.

3.3.3 Solusi atas Kendala yang Ditemukan

Berdasarkan kendala-kendala yang dihadapi selama magang sebagai IT Data Analyst, berikut adalah solusi yang diambil untuk mengatasi masalah-masalah tersebut:

1. Untuk mengatasi kendala pemrosesan data dalam skala besar akibat keterbatasan infrastruktur, solusi yang diambil adalah melakukan peningkatan spesifikasi *virtual machine* (VM) atau infrastruktur yang digunakan. Upgrade dilakukan dengan menambah kapasitas menjadi 28 *core* CPU, RAM sebesar 128 GB, serta *storage* 2 TB. Peningkatan ini terbukti membantu mempercepat proses komputasi, meningkatkan stabilitas sistem, dan memungkinkan pengolahan data yang lebih kompleks tanpa mengalami penurunan performa.
2. Mengingat spesifikasi perangkat kerja awal tidak mampu menunjang aktivitas pemrosesan data secara optimal, langkah yang dilakukan adalah menginformasikan kondisi tersebut kepada supervisor. Selanjutnya, dilakukan proses pengajuan penggantian perangkat kerja dengan spesifikasi yang lebih tinggi. Perangkat yang digunakan setelah penggantian adalah laptop Dell keluaran tahun 2022 dengan prosesor Intel Core i7 generasi ke-12, RAM 16 GB, dan *storage* 512 GB, sehingga pekerjaan dapat dilakukan dengan lebih efisien dan lancar.
3. Untuk menangani permasalahan duplikasi data yang menyebabkan anomali pada laporan DRR, dilakukan proses pengecekan dan validasi data secara berkala setiap harinya. Selain itu, dibuat *log table* yang divisualisasikan menggunakan Looker Studio sebagai alat monitoring dan *cross-check* untuk memastikan data yang masuk ke dalam *database* sudah akurat. Proses data *cleansing* juga diterapkan pada dataset DRR sebelum data diproses lebih lanjut. Dengan langkah ini, potensi duplikasi dapat diminimalkan dan data yang digunakan pada tahap analisis maupun visualisasi menjadi lebih bersih dan valid.