EMPLOYEE DATA MANAGEMENT SYSTEM

Capstone Project Report

Submitted By

Kanishka Sharma

Date: May 8, 2025

Introduction

In modern organizations, managing employee information efficiently is essential for operational success. This project presents the development of an Employee Data Management System designed to ingest, process, and analyze employee-related data using AWS-based infrastructure and real-time Kafka streaming.

The system integrates multiple datasets including employee records, leave history, quota allocations, and communication logs. It performs daily and monthly analytics to flag excessive leave usage and monitor potential misuse of communication through flagged keywords. Using PySpark and AWS Glue, it ensures scalable, fault-tolerant data pipelines for transforming raw CSV and JSON inputs into structured, deduplicated, and enriched outputs.

The project also includes report generation for HR insights, strike tracking with cooldown logic, and salary adjustment mechanisms. Built with principles of data warehousing, the solution serves as a comprehensive platform for employee lifecycle and behavior monitoring across the organization.

Objectives

The primary objective of this project is to design and implement a scalable Employee Data Management System capable of processing and analyzing employee-related data from multiple sources in real time and batch modes. The system is intended to support organizational decision-making through automation and intelligent reporting.

The specific objectives include:

- Efficiently ingest and store employee data, leave records, and communication logs.
- Track employee leave history, quotas, and identify patterns of excessive leave usage.
- Generate daily and monthly reports for HR analysis and leave abuse detection.
- Monitor employee communication for flagged or inappropriate content using predefined keyword lists.

- Implement strike tracking and salary deductions based on communication violations.
- Ensure data consistency and reliability using scalable ETL pipelines and data warehousing practices.

Key Outcomes

The Employee Data Management System delivers the following key outcomes:

- **Automated Data Pipelines:** Daily ingestion and transformation of employee, leave, and communication data from AWS S3 and Kafka.
- **Accurate Leave Monitoring:** Detection of employees with excessive leave usage through both daily (8% rule) and monthly (80% rule) threshold analysis.
- **Communication Flagging System:** Real-time identification of inappropriate or reserved word usage in employee messages using Kafka and marked word lists.
- **Strike and Salary Adjustment Mechanism:** Salary deductions and strike history tracking with monthly cooldown enforcement for communication violations.
- **Manager-Focused Reporting:** Generation of personalized text reports for managers identifying employees exceeding leave thresholds.
- **Reliable and Scalable Architecture:** A fault-tolerant, cloud-based system built using AWS Glue, PySpark, and Delta Lake following data warehousing principles.

Technology Stack

Component	**Technology Used**
Storage Layers	**Bronze:** Amazon S3 stores raw input files (CSV, JSON) from various sources. **Silver:** Processed, deduplicated, enriched data (e.g., active leave records, employee timelines). **Gold:** Final analytical outputs and reports (e.g., per-manager leave reports in text format).
ETL Processing	AWS Glue using Apache Spark for cleaning, transforming, and enriching raw data.
Database	PostgreSQL on AWS EC2 for maintaining transactional state (e.g., strike history, salary adjustments).
Streaming Ingestion	Apache Kafka for consuming real-time employee messages.
Stream Processing	Spark Structured Streaming to flag reserved words, update strikes, and apply salary deductions.
Visualization	Grafana dashboards for monitoring leave patterns and flagged employee behavior.
Orchestration	Apache Airflow to schedule, monitor, and retry daily and monthly workflows.

System Architecture

1. Bronze Layer – Raw Data Ingestion

This layer holds all raw, unprocessed files ingested from multiple sources:

- **S3 (CSV files)** for employee master data, timeframe records, leave applications, quotas, and holiday calendars.
- **Kafka (JSON messages)** for real-time employee communication events.

Files are stored with minimal validation, serving as an immutable audit trail.

2. Silver Layer – Cleaned and Enriched Data

The Silver layer stores deduplicated and cleaned data for operational use:

- Employee timelines are processed to ensure continuity and flag active designations.
- Leave data is filtered for `ACTIVE` status and corrected for duplicates and cancellations.
- Only a subset of cleaned data is kept in Silver; most outputs are now written directly to PostgreSQL.

3. Gold Layer – Final Business Outputs

This layer contains finalized data and reports for stakeholders:

- Daily designation-wise employee counts in PostgreSQL.
- 8% and 80% leave usage alerts stored in reporting tables.
- Manager-specific text reports written to S3 (Gold) without duplication on retries.
- Final salary and strike history maintained in PostgreSQL for audit and dashboarding.

4. Streaming and Real-Time Processing

A Kafka + Spark-based pipeline is used for live message flagging:

- Kafka producers stream employee messages to the `employee_messages` topic.
- Spark Structured Streaming consumers process these messages, detect reserved words, and write flagged entries to PostgreSQL.
- A cooldown Spark job updates strike counts, reduces salaries, and tracks `INACTIVE` employees based on thresholds.

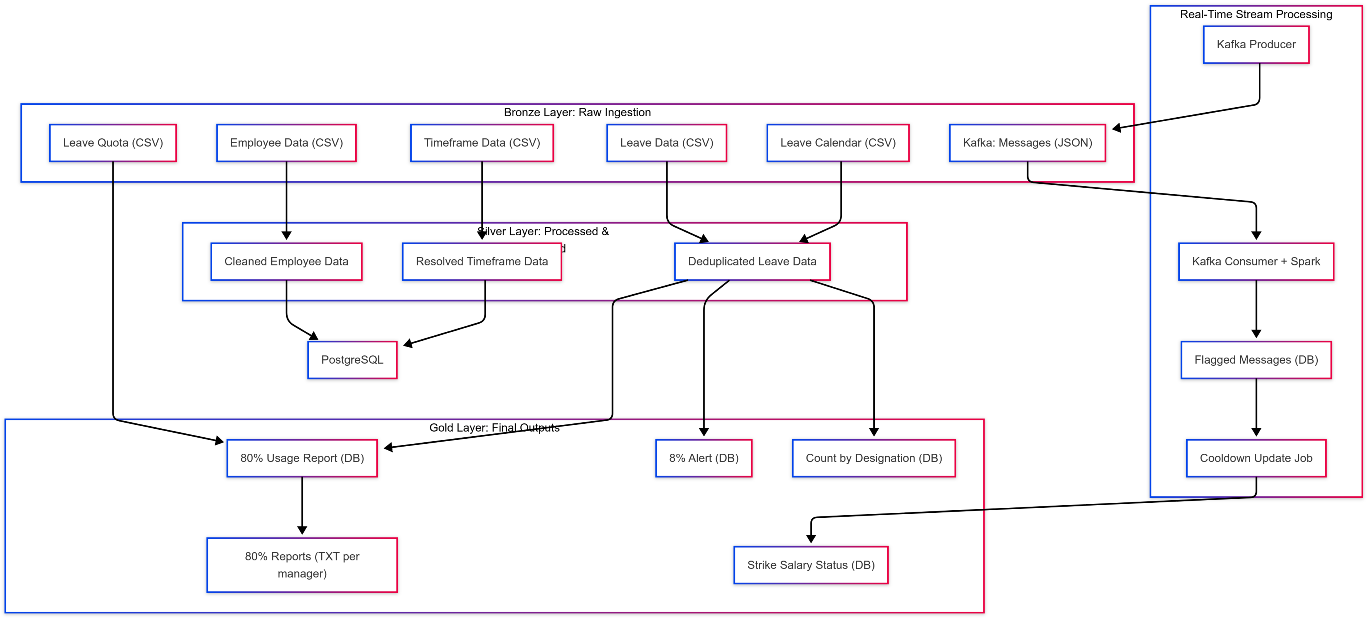


Figure 1: System Architecture

5. Workflow Orchestration

All Glue ETL jobs are orchestrated using Apache Airflow:

- Daily pipelines for leave ingestion, employee data, and active reporting run at 07:00 UTC.
- Monthly jobs (like 80% leave report) run on the 1st day of each month.
- Branch operators and retries ensure job resiliency and task dependencies.

Employee Leave Data Processing

To support leave tracking, quota enforcement, and reporting, the system processes three types of leave-related datasets: leave quota, leave calendar (holidays), and daily leave application data. These datasets are stored in PostgreSQL and form the basis for both real-time dashboards and scheduled reports.

Employee Data and Timeframe Processing

To support workforce analytics and leave tracking, the system ingests and processes employee master data and designation history daily. The processed records are stored in PostgreSQL for downstream reporting and analysis.

1. Employee Data Ingestion and Storage

Objective: Maintain an append-only table with up-to-date employee information (ID, age, and name).

- Source CSVs are uploaded daily to the Bronze layer in S3.

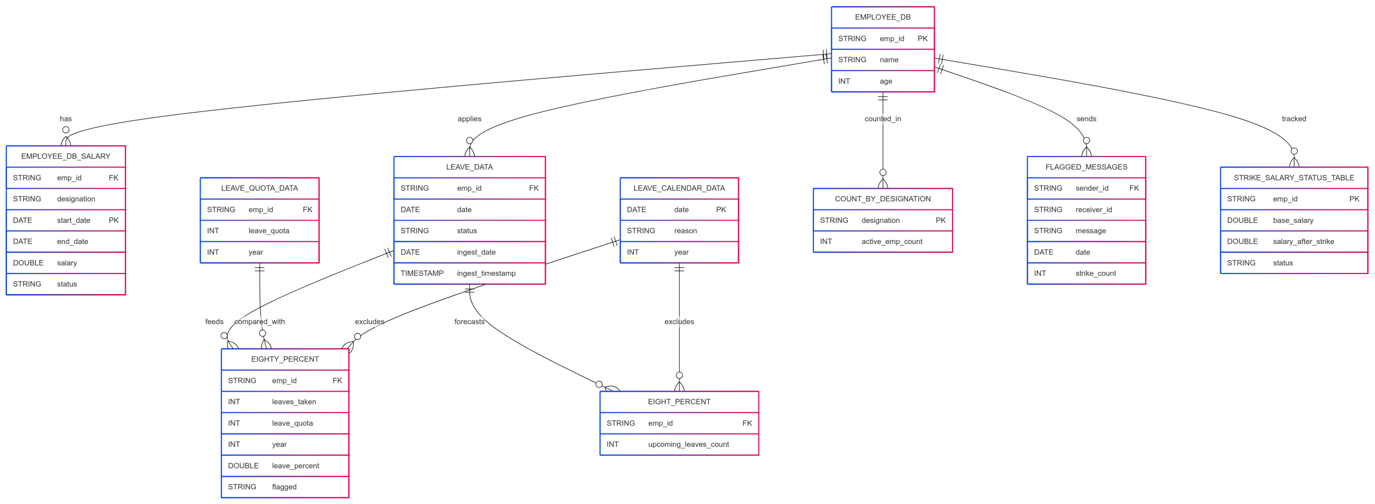


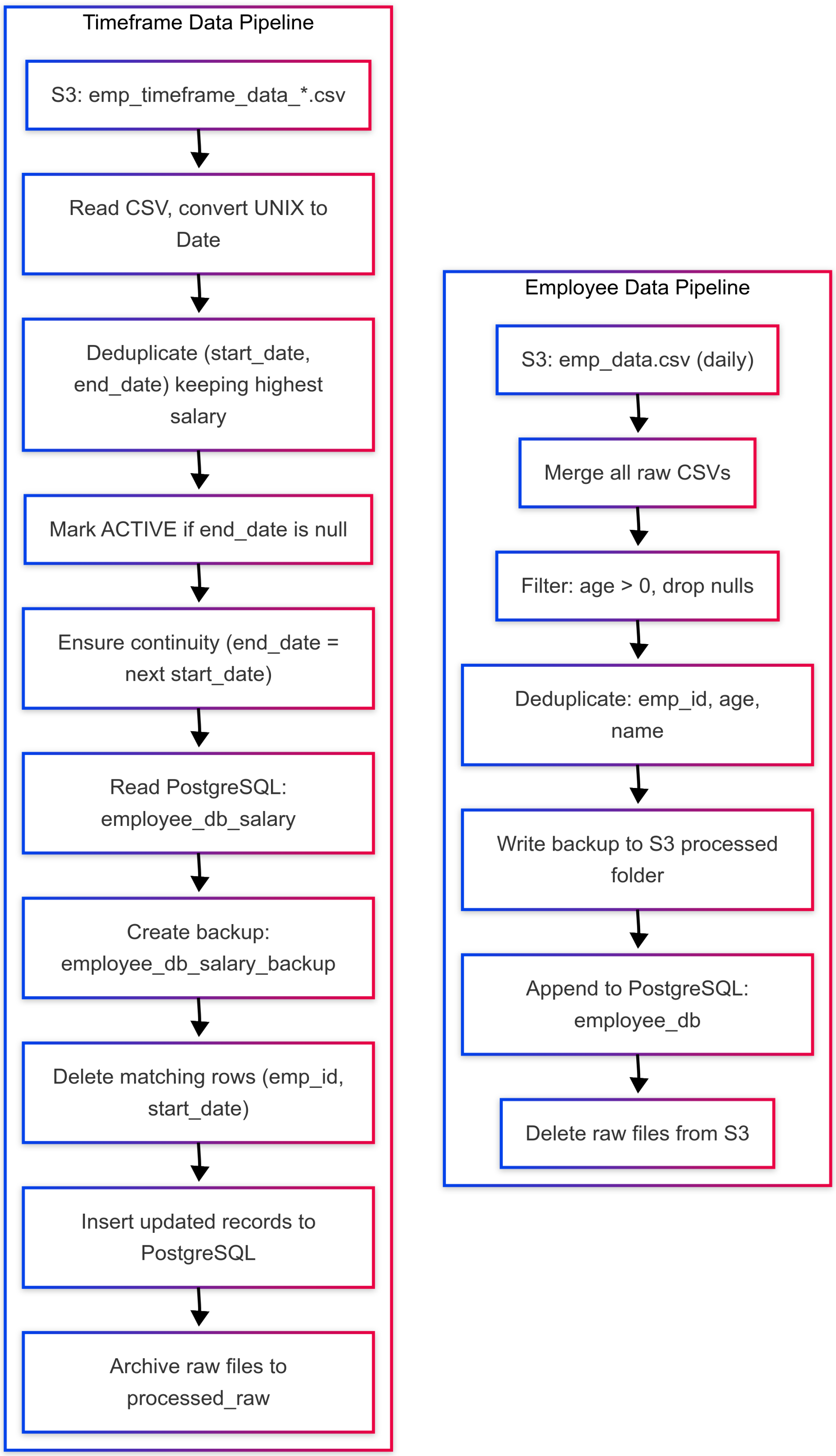
Figure 2: ER Diagram

- A Glue job reads all raw files, merges them, and writes a processed version to a backup folder.
- Duplicates and null entries are removed; records are filtered for positive age values.
- Cleaned records are appended to a PostgreSQL table `employee_db` daily at 07:00 UTC.
- Raw files are deleted after successful backup and ingestion.

2. Employee Timeframe Data Handling

Objective: Track an employee's history of designations and salaries with proper continuity and active status identification.

- Daily incremental files from S3 (Bronze) are read into Spark.
- UNIX timestamps are converted to date format for `start_date` and `end_date`.
- Duplicates across (`emp_id`, `start_date`, `end_date`) are resolved by retaining records with the highest salary.
- Missing `end_date` values are marked as `ACTIVE`; others as `INACTIVE`.
- Continuity is enforced: the `end_date` of the previous record is set to the `start_date` of the next one for each employee.
- Existing PostgreSQL data is read; matching records are deleted before inserting updated data to ensure incremental correctness.
- The final dataset is written to `employee_db_salary`, replacing previous overlapping records.
- A backup of the previous table is created as `employee_db_salary_backup` before overwriting.
- Raw files are archived after successful processing.


Timeframe Data Pipeline
S3: emp_timeframe_data_*.csv
Read CSV, convert UNIX to Date
Deduplicate (start_date, end_date) keeping highest salary
Mark ACTIVE if end_date is null
Ensure continuity (end_date = next start_date)
Read PostgreSQL: employee_db_salary
Create backup: employee_db_salary_backup
Delete matching rows (emp_id, start_date)
Insert updated records to PostgreSQL
Archive raw files to processed_raw
Employee Data Pipeline
S3: emp_data.csv (daily)
Merge all raw CSVs
Filter: age > 0, drop nulls
Deduplicate: emp_id, age, name
Write backup to S3 processed folder
Append to PostgreSQL: employee_db
Delete raw files from S3

Employee Leave Data

1. Leave Quota Data

Objective: Maintain a yearly append-only table of leave quotas for each employee.

- Raw data is ingested from Amazon S3 Bronze layer: `emp_leave_quota/`.
- A predefined schema ensures correct typing of `emp_id`, `leave_quota`, and `year`.
- Records are deduplicated on (`emp_id`, `leave_quota`, `year`).
- Cleaned data is appended to PostgreSQL table `leave_quota_data`.

2. Leave Calendar Data (Holidays)

Objective: Maintain a yearly record of public holidays to exclude them from leave calculations.

- Data is read from the S3 Bronze path: `emp_leave_calender/`.
- Each holiday entry includes a reason and date, cast to proper timestamp.
- A `year` column is derived for partitioning and query filtering.
- Duplicates are removed before writing to PostgreSQL table `leave_calendar_data`.

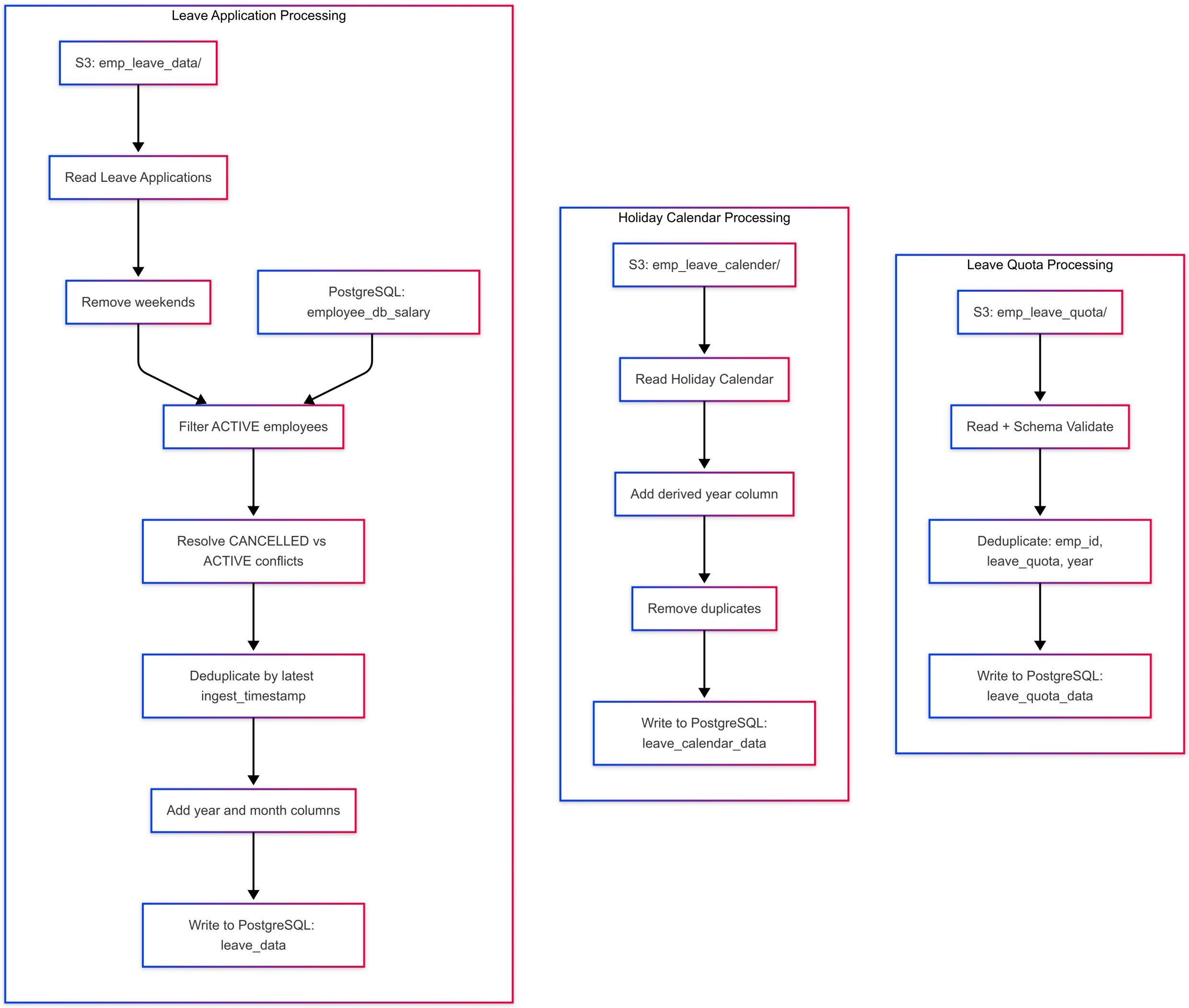
3. Leave Application Data

Objective: Create a daily-updated, deduplicated table of valid employee leaves excluding holidays, weekends, and cancellations.

- Raw leave data is loaded daily from S3 path: `emp_leave_data/`.
- Leave records falling on weekends (Saturday, Sunday) are filtered out.
- Only `ACTIVE` employees from `employee_db_salary` are included.
- `CANCELLED` leave entries are resolved by comparing active vs cancelled counts for the same day.
- Deduplication is applied using latest `ingest_timestamp` for each (`emp_id`, `date`) pair.
- Final output includes derived fields such as `year` and `month` for partitioning.
- Data is written to the PostgreSQL table `leave_data` in overwrite mode, ensuring daily freshness.

Employee Reporting and Alerting

To support HR operations and identify patterns of excessive leave usage, the system produces daily and monthly reports from cleaned and enriched leave data. These reports are stored in PostgreSQL and exported to S3 as needed for manager-level insights.


Leave Application Processing
S3: emp_leave_data/
Read Leave Applications
Remove weekends
PostgreSQL: employee_db_salary
Filter ACTIVE employees
Resolve CANCELLED vs ACTIVE conflicts
Deduplicate by latest ingest_timestamp
Add year and month columns
Write to PostgreSQL: leave_data
Holiday Calendar Processing
S3: emp_leave_calender/
Read Holiday Calendar
Add derived year column
Remove duplicates
Write to PostgreSQL: leave_calendar_data
Leave Quota Processing
S3: emp_leave_quota/
Read + Schema Validate
Deduplicate: emp_id, leave_quota, year
Write to PostgreSQL: leave_quota_data

1. Daily Report – Active Employee Count by Designation

Objective: Track the number of currently active employees per designation on a daily basis.

- Data is sourced from the `employee_db_salary` table in PostgreSQL.
- Only employees marked as `ACTIVE` are included.
- Records are grouped by designation (or `UNKNOWN` if null), and counts are computed.
- The result is written daily to the `count_by_designation` table.

2. Daily Alert – 8% Upcoming Leave Watchlist

Objective: Identify employees whose upcoming leaves exceed 8% of the remaining working days in the current year.

- A Spark job generates a date range from the current date to the end of the year.
- Public holidays and weekends are removed using data from `leave_calendar_data`.
- Valid `ACTIVE` leave entries from the future are filtered.
- If an employee has more than 8% of these remaining working days booked as leave, they are flagged.
- Results are stored in the PostgreSQL table `eight_percent`.

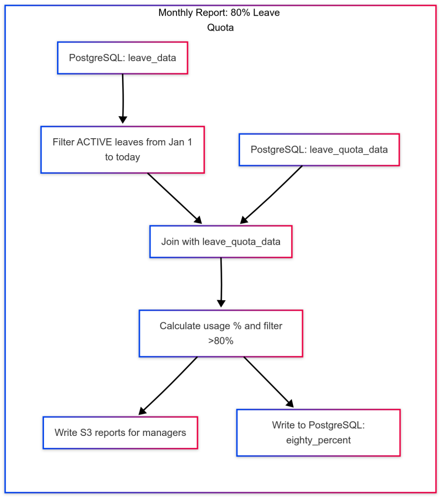
3. Monthly Report – 80% Leave Quota Usage Detection

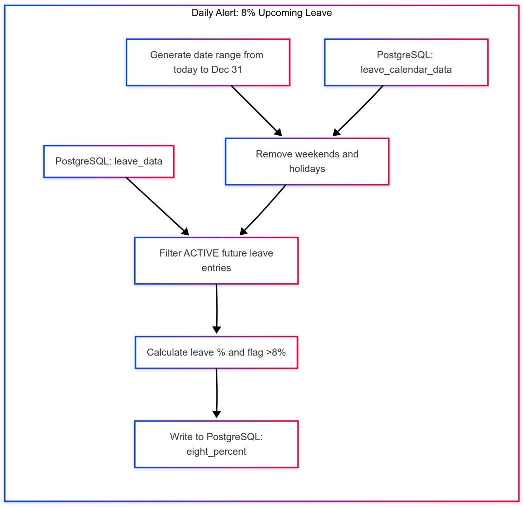
Objective: Flag employees who have used more than 80% of their annual leave quota and notify their managers.

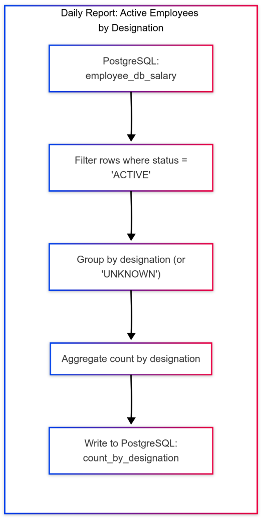
- On the 1st of each month, the system reads the cleaned leave data and leave quota tables.
- Leave records marked as `ACTIVE` are counted per employee from the start of the year to the current date.
- Employees whose usage exceeds 80% are flagged.
- For each flagged employee, a text file report is generated and saved to S3 in the Gold layer.
- Previously generated reports (based on employee ID) are skipped to prevent duplicates if a job fails or reruns.
- A final summary DataFrame is also written to the PostgreSQL table `eighty_percent`.

Real-Time Communication Monitoring and Strike Handling

To monitor employee communication for misuse, a real-time stream processing pipeline is implemented using Kafka, Spark, and PostgreSQL. This system flags inappropriate messages containing reserved words and applies salary deductions and strike penalties.


Monthly Report: 80% Leave Quota
PostgreSQL: leave_data
Filter ACTIVE leaves from Jan 1 to today
PostgreSQL: leave_quota_data
Join with leave_quota_data
Calculate usage % and filter >80%
Write S3 reports for managers
Write to PostgreSQL: eighty_percent


Daily Alert: 8% Upcoming Leave
Generate date range from today to Dec 31
PostgreSQL: leave_calendar_data
PostgreSQL: leave_data
Remove weekends and holidays
Filter ACTIVE future leave entries
Calculate leave % and flag >8%
Write to PostgreSQL: eight_percent


Daily Report: Active Employees by Designation
PostgreSQL: employee_db_salary
Filter rows where status = 'ACTIVE'
Group by designation (or 'UNKNOWN')
Aggregate count by designation
Write to PostgreSQL: count_by_designation

1. Kafka Producer

A Kafka producer reads employee messages from a local JSON file and publishes them to the `employee_messages` topic on a local Kafka broker. Each message simulates real-time communication between employees.

2. Kafka Consumer and Spark Processor

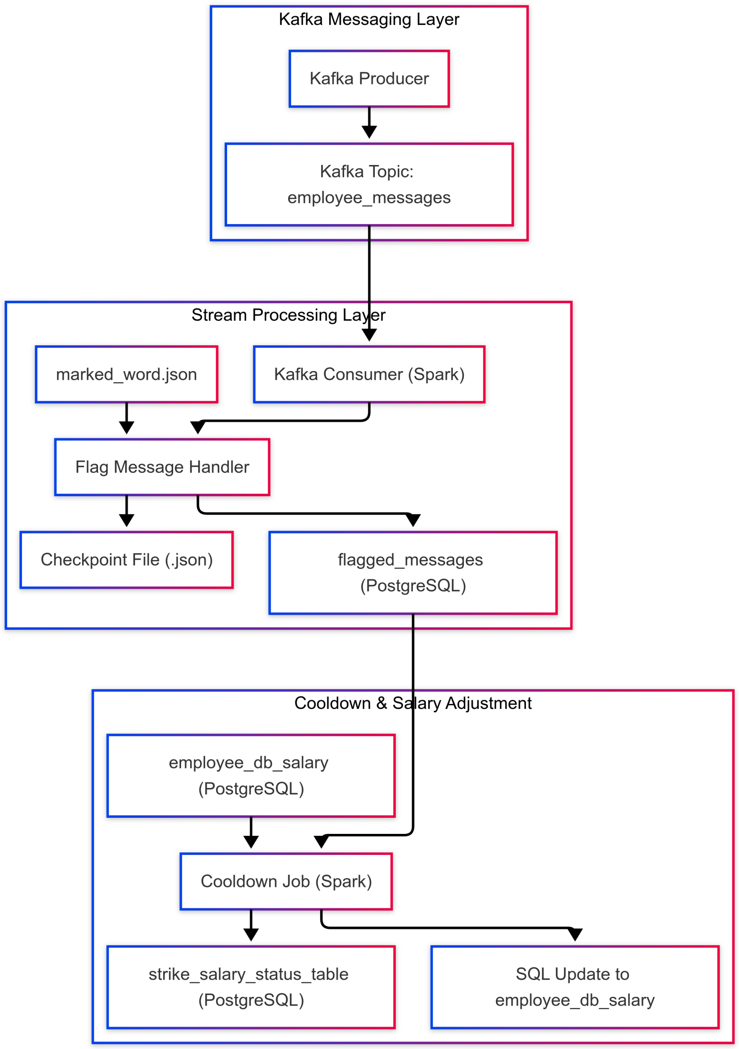
A custom checkpointed Kafka consumer built using Spark Structured Streaming reads the incoming messages and performs the following:

- Deserializes and parses each message.
- Matches message text against a reserved word list from `marked_word.json`.
- Flags any message containing a reserved word and creates a DataFrame row with sender ID, receiver ID, message, date, and strike count.
- Writes flagged messages to a PostgreSQL table named `flagged_messages`.
- Maintains Kafka offset checkpoints in a local JSON file to avoid message duplication on restarts.

3. Cooldown and Strike Logic

A separate long-running Spark application performs periodic cooldown updates:

- Reads flagged messages and salary data from PostgreSQL.
- Filters strikes within the last 30 days and calculates the strike count for each employee.
- Applies 10% salary deduction per strike using exponential decay.
- Marks employees with 10 or more strikes as `INACTIVE`, ensuring their salaries are frozen.
- Preserves previously inactive employees by excluding them from updates.


Kafka Messaging Layer
Kafka Producer
Kafka Topic: employee_messages
Stream Processing Layer
marked_word.json
Kafka Consumer (Spark)
Flag Message Handler
Checkpoint File (.json)
flagged_messages (PostgreSQL)
Cooldown & Salary Adjustment
employee_db_salary (PostgreSQL)
Cooldown Job (Spark)
strike_salary_status_table (PostgreSQL)
SQL Update to employee_db_salary

- Updates a summary table `strike_salary_status_table` and reflects new `INACTIVE` statuses in `employee_db_salary` via direct SQL.

Technical Implementation Overview

Technical Requirements Fulfillment

Data Warehouse Principles:

- Layered architecture (Bronze, Silver, Gold) ensures modular data processing and analytical separation.
- Deduplication, casting, and timestamp management follow normalization and incremental load best practices.

AWS Integration:

- Amazon S3 is used for ingesting and storing raw, processed, and report data.

- AWS Glue runs scheduled ETL jobs for daily ingestion and reporting.
- Apache Kafka is used to ingest real-time employee messages for communication monitoring.

Database Infrastructure:

- PostgreSQL is hosted on an EC2 instance to store cleaned and transactional tables such as `employee_db`, `leave_data`, and `flagged_messages`.

Schema and Relationships:

- Tables are logically related through `emp_id`, supporting joins across employee, leave, and strike data.
- The design aligns with ER modeling conventions and supports auditability.

Success Criteria Fulfillment

Reliable Ingestion and Processing:

- All datasets (employee, leave, timeframes, messages) are ingested daily or in real time.
- Validation, deduplication, and type safety checks ensure high data quality.

Leave and Behavior Reporting:

- Daily reports include active employee designation counts and 8% leave watchlists.
- Monthly reports identify employees exceeding 80% of their annual quota with per-manager text reports.

Streaming Communication Monitoring:

- Kafka consumers detect reserved words in real time.
- Flagged messages are stored and used to update strike counts and adjust salaries.
- A monthly cooldown logic resets eligible strike histories and enforces deactivation at 10 strikes.