Topic

End-to-End Data Architecture for Leather Shoe Company

Background and Motivations

- The company currently relies on a third-party database, which without having the ability to analyze requiring data and access business operation indicators
- A dedicated website is needed to streamline financial record-keeping processes within the company
- There is a lack of actionable business insights, such as operational and sales performance metrics
- Managers require seamless and concise insight assessments, rather than being overwhelmed by raw numbers without context or meaning

Project Objectives

- Develop a robust data warehouse tailored to the company's business needs
- Build a web application using Flask to feature:
    - A dashboard for data visualization and performance monitoring
    - Financial record-keeping functionality
    - Leverage OpenAI API to extract insights and provide explanatory analysis directly from the data through the dashboard
- Implement a scalable ETL pipeline using AWS Glue to efficiently manage data flow from AWS RDS to Amazon Redshift

Stakeholders

- Company Management: Require actionable insights to make data-driven decisions
- Finance Team: Demand a streamlined process for financial record-keeping
- Operations Team: Rely on performance insights for strategy planning
- IT Team: Oversee database maintenance and integration with existing infrastructure

Project Deliverables

- **Stored procedures (T-SQL)** to quickly extract data from 3rd party OLTP databases in **SQL Server** for optimizing the query process
- A **scalable Amazon Redshift data warehouse (OLAP)** for advanced analytics and business intelligence
- A **Flask-based web application** hosted on AWS EC2, featuring:
    - A **dashboard** to visualize operational and sales performance metrics
    - **Financial record-keeping functionality** for streamlined tracking

- **Integration with OpenAI API** to provide natural language insights and analysis
- **AWS Glue ETL pipelines** to:
  - Extract and transform accounting data acquired from the website in **AWS RDS** to **Redshift**
  - For unstructured files from the accountants in the company, stage data in **Amazon S3** then transform and load it into **Amazon Redshift**
- **Comprehensive documentation and ERD** that covering data architecture, workflows, and application usage

Project Highlights	**Measurement Methods/Metrics**
1. Transition from a third-party database to an independent, company-owned system 2. Implementation of machine learning-powered insights using OpenAI API 3. Scalable, cloud-based data warehouse to support future growth 4. Real-time data visualization and analysis tools for decision-making 5. Automated and efficient ETL pipelines using AWS Glue	1. System Performance: Query execution time, dashboard response speed 2. Data Accuracy: Validation of insights and financial records against existing data 3. User Adoption: Number of active users within the company 4. Business Impact: Reduction in time spent on manual data analysis, improvement in decision-making speed

Data Architecture/Data Flow

- Data Sources:
  - Internal Systems: Data generated from the company's third-party ERP system, stored in SQL Server
  - Self-Built Webpage: Website-related accounting and operational data stored in AWS RDS, connected to the EC2-hosted application
- Data Storage:
  - Primary Storage: SQL Server for transactional data, serving as the system of record
  - Website Data Storage: AWS RDS for storing structured website data, linked to the EC2 web application
  - Analytical Storage: Amazon Redshift for advanced querying and insights, receiving transformed data from SQL Server and RDS via AWS Glue
- Data Processing:
  - AWS Glue: Handles data transformation and loading into Amazon Redshift

    - Amazon S3: Serves as an intermediary storage layer before Redshift ingestion
    - Data Transformation & Loading:
        - Store raw data in Amazon S3
        - Transform data in AWS Glue before loading it into Redshift
        - Use Redshift's COPY command for efficient data ingestion
- Data Management:
    - Data Table Relationships: Well-defined schema design for structured data integration across SQL Server, AWS RDS, and Redshift
    - Data Update Methods: Batch updates for historical data using ETL pipelines
- Data Application:
    - Interactive dashboards for real-time management insights
    - LLM-generated insights using OpenAI API
    - Financial tracking and reporting tools to enhance decision-making

Technology Stack

- AWS Glue: Data Pipeline for Cloud Service ETL
- AWS EC2: Cloud-based virtual server hosting the Flask web application and handling backend processing
- AWS RDS: Cloud database for structured website data
- SQL Server: Primary database for structured data and serves as its backup location
- Amazon Redshift: Data warehouse for analytics
- Amazon S3: Staging storage before data ingestion into Redshift
- Flask: Web application framework
- OpenAI API: Natural language processing
- Python: ETL scripts and application backend

MVP Scope

- Week 1:
    - Set up SQL Server database structure
    - Identify data sources and begin ETL pipeline development
- Week 2:
    - Configure Amazon Redshift and test data ingestion
    - Develop basic Flask web application structure
- Week 3:
    - Implement a dashboard with visualizations
    - Integrate OpenAI API for data insights
- Week 4:
    - Conduct user testing and refine features
    - Complete project documentation
- Future:

- Apply **agile project management method and CI/CD** to continuous refine the system

Issue Log

Program	Incident	Solution	Date
EC2	Cannot connect to instance	Checked security group settings, ensured port 22 was open for SSH, and verified correct key pair usage	
Git	Merge conflict in branch	Used `git merge --abort`, manually resolved conflicts, and committed the changes	
REST API	API request returning 500 error	Debugged logs, identified a missing required field in the request, and updated validation rules	
Flask	Form submission not saving data	Verified CSRF token setup and ensured the correct database session commit	
PostgreSQL	`IF NOT EXISTS` does not work functionally	`pg_tables` converts table name into lower case, thus requiring to use lower case in the checking query	2/25/2025
SQL Server	`Option`		
Redshift	`SERIAL` shows error in Redshift	Use `Identity(1,1)` instead	
Redshift	Demand to debug the errors	`sys_load_error_detail`	
Redshift		`DATEFORMAT 'MM/DD/YYYY'`	
Redshift	`Copy`	`iam_role`	
Redshift	`Copy`	FILLRECORD	
	csv files in S3 cannot copy to Redshift correctly, which all the data in a row will	Eliminate `Identity(1,1)` and use: `delimiter ','` `IGNOREHEADER 1`	

	load into one single column	`removequotes;`	
Docker		When using python with version larger than 3 and executing python commands, you have to specify python3 in the RUN function	
Redshift API	Unable to connect to Redshift	Alter admin password and not using *`md5hash`*	

Optimization Direction

- Add a monitoring and notification system to alert when product or branches have some issues
- Add more diagrams to enhance the decision-making process

References

- Date Populate: https://elliotchance.medium.com/building-a-date-dimension-table-in-redshift-6474a7130658