Documento Técnico – Projeto MovieStream Analytics

1. Arquitetura da Solução

A arquitetura proposta utiliza Apache Airflow como orquestrador de pipelines de dados, com dois bancos de dados PostgreSQL: um como base de origem (dbinterview) e outro como destino para a área de landing (landing_db). As DAGs do Airflow executam scripts Python que realizam a extração dos dados, executando cargas completas para a maioria das tabelas e Change Data Capture (CDC) para tabelas com maior volume de dados. A comunicação entre os containers e serviços ocorre via Docker Compose.

2. Estratégia de CDC Utilizada

Foi implementada uma abordagem de CDC baseada em coluna de controle temporal (`last_update` e `payment_date`).

As tabelas `rental` e `payment` são monitoradas para identificar alterações realizadas desde a última execução. O script Python armazena o timestamp processado em arquivos `.txt` e utiliza essas informações para aplicar inserções incrementais na base de destino (`landing_db`).

3. Cronograma de Execução das DAGs

A DAG `pipeline_ingestao_cdc` foi configurada com agendamento diário (`@daily`) e executa os seguintes passos:

1. Carga completa das tabelas de apoio e dimensões;
2. Execução do CDC nas tabelas `rental` e `payment` com controle incremental por timestamp;
3. Escrita incremental no banco `landing_db`.

Além disso, foi criada a DAG `pipeline_dbt`, também agendada diariamente, que depende da DAG de ingestão e executa os modelos DBT logo após a atualização dos dados.

4. Tecnologias Utilizadas

- Python: linguagem utilizada para scripts de ingestão de dados.
- Apache Airflow: ferramenta de orquestração de workflows para agendar e monitorar as pipelines.
- PostgreSQL: banco de dados relacional utilizado como origem (dbinterview) e destino (landing_db).
- Docker e Docker Compose: utilizados para criar e gerenciar os containers.
- Pandas e SQLAlchemy: bibliotecas Python para movimentação dos dados.
- DBT (Data Build Tool): ferramenta para modelagem e transformação de dados em camada de Data Warehouse.

5. Instruções de Execução

1. Clonar o repositório e acessar o diretório do projeto.
2. Subir os containers com `docker compose up --build`.
3. Acessar a interface do Airflow via `http://localhost:8080`.
4. Ativar e executar manualmente (ou aguardar agendamento) das DAGs `pipeline_ingestao_cdc` e `pipeline_dbt`.
5. Verificar as transformações DBT diretamente no banco `landing_db`, schema `public`.

6. Desafios Enfrentados

- Conexão entre containers e banco local (resolvido com `host.docker.internal` no Airflow).
- Erros de dependência em tabelas com views no DBT, resolvidos ajustando o modo de escrita no script para `if_exists='append'`.
- Configuração da execução do DBT dentro do container do Airflow, com adição de `Dockerfile` e ajuste no `docker-compose.yml` para incluir o projeto e o perfil do DBT.
- Erro na localização do arquivo `profiles.yml` resolvido com o parâmetro `--profiles-dir` e mapeamento correto do volume.

7. Modelagem e Transformações

A segunda fase do projeto teve como objetivo construir uma camada de Data Warehouse utilizando o DBT.

Foram aplicadas boas práticas de modularização e versionamento para garantir rastreabilidade e clareza na estrutura dos dados.

7.1 Camadas e Estrutura de Projeto

- `staging`: padronização, renomeação de colunas e tipagens dos dados extraídos do landing.
- `marts`: camada analítica com indicadores de negócio prontos para análise.

7.2 Tabelas Derivadas Criadas

- `mart_customer_lifetime_value`: valor total gasto por cliente, data da primeira locação e tempo de relacionamento.
- `mart_film_popularity`: ranking de filmes mais alugados por mês/ano.
- `mart_store_performance`: performance por loja (número de locações, receita e base de clientes distintos).

7.3 Boas Práticas Aplicadas

- Modularização clara com diretórios `staging` e `marts`.
- Uso de `ref()` para controle de dependência entre modelos.
- Documentação de modelos via `schema.yml`.
- Materialização adequada (`view` em staging, `table` em marts).
- Controle via Git com versionamento do projeto DBT.

8. SQL Analítico

As queries abaixo foram construídas em SQL padrão PostgreSQL para responder às perguntas de negócio propostas:

1. **Top 5 clientes que mais geraram receita no último ano**:
```sql
SELECT c.customer_id, c.first_name || ' ' || c.last_name AS cliente, SUM(p.amount) AS
receita_total
FROM payment p
JOIN customer c ON p.customer_id = c.customer_id
WHERE p.payment_date >= CURRENT_DATE - INTERVAL '1 year'
GROUP BY c.customer_id, cliente
ORDER BY receita_total DESC
LIMIT 5;
```

2. **Média de dias entre aluguel e devolução por categoria de filme**:
```sql
SELECT cat.name AS categoria, ROUND(AVG(r.return_date - r.rental_date), 2) AS media_dias
FROM rental r
JOIN inventory i ON r.inventory_id = i.inventory_id
JOIN film f ON i.film_id = f.film_id
JOIN film_category fc ON f.film_id = fc.film_id
JOIN category cat ON fc.category_id = cat.category_id
WHERE r.return_date IS NOT NULL
GROUP BY cat.name
ORDER BY media_dias DESC;
```

3. **Top 3 cidades com maior volume de locações**:
```sql
SELECT ci.city, COUNT(*) AS total_locacoes
FROM rental r
JOIN customer c ON r.customer_id = c.customer_id
JOIN address a ON c.address_id = a.address_id
JOIN city ci ON a.city_id = ci.city_id
GROUP BY ci.city
ORDER BY total_locacoes DESC
LIMIT 3;
```

4. **Ticket médio por loja**:
```sql
SELECT s.store_id, ROUND(SUM(p.amount)/COUNT(DISTINCT p.rental_id), 2) AS
ticket_medio
FROM payment p
JOIN rental r ON p.rental_id = r.rental_id
JOIN inventory i ON r.inventory_id = i.inventory_id
```

JOIN store s ON i.store_id = s.store_id
GROUP BY s.store_id;
```

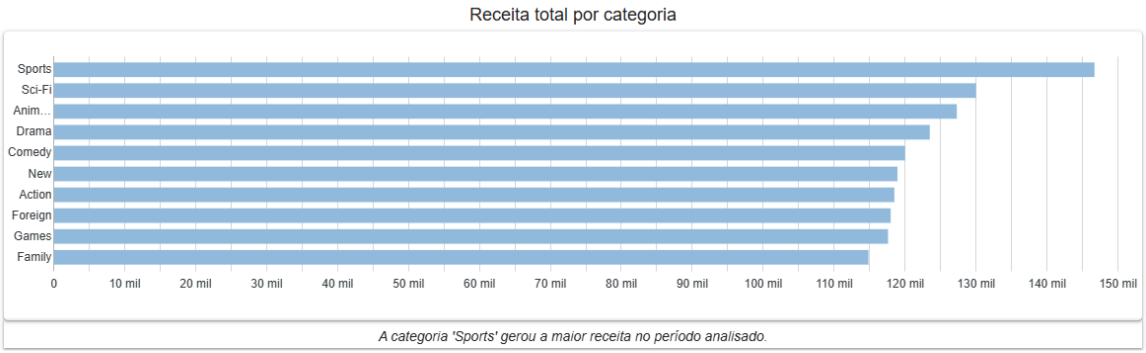
5. **Receita mensal nos últimos 24 meses**:
```sql
SELECT TO_CHAR(p.payment_date, 'YYYY-MM') AS mes_ano, ROUND(SUM(p.amount), 2) AS receita_total
FROM payment p
WHERE p.payment_date >= DATE_TRUNC('month', CURRENT_DATE) - INTERVAL '24 months'
GROUP BY mes_ano
ORDER BY mes_ano;
```

9. Visualização e Storytelling

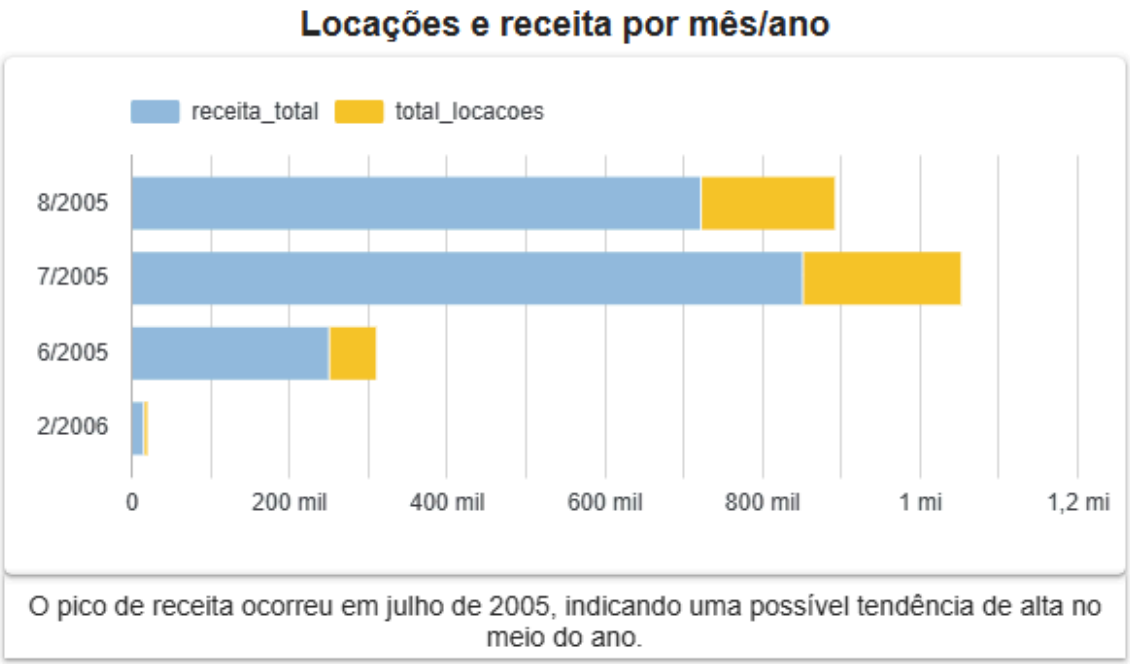
As visualizações foram criadas utilizando o Google Looker Studio com base nos dados exportados do banco `landing_db` e pode ser acessado através do link: https://lookerstudio.google.com/reporting/3d7f94bc-7f79-44c0-b7c4-784983ec04cc

Indicadores apresentados:
1. Receita por Categoria de Filme
2. Locações por Mês (últimos 24 meses)
3. Clientes Ativos por Loja

📌 Insights:
- As categorias que mais geram receita representam preferências claras do público. Utilizar essas informações para campanhas de marketing ou aquisição de novos filmes pode trazer alto retorno.



- Essa visão permite identificar padrões sazonais e avaliar crescimento/declínio da base de clientes.
```



- Ambas as lojas apresentaram a mesma quantidade de clientes ativos, indicando equilíbrio operacional entre unidades.