📊 Pipeline de Dados e Análise com SQL no BigQuery

Objetivo.

Este projeto demonstra a construção de um pipeline de dados analítico no Google BigQuery, transformando dados transacionais brutos de uma livraria online na Livraria DevSaber em um mini data warehouse. O objetivo foi resolver o desafio de gerenciar dados dispersos em planilhas, garantindo a integridade das informações e permitindo a extração de insights valiosos para o negócio.
O pipeline foi estruturado em três etapas principais, seguindo as melhores práticas da engenharia de dados:
Modelagem e Definição do Schema DDL: Foram criadas tabelas normalizadas Clientes, Produtos, Vendas no BigQuery, estabelecendo uma estrutura de dados lógica e otimizada para análises futuras.
Ingestão de Dados DML: Os dados brutos foram migrados e inseridos nas tabelas, garantindo a consistência e a limpeza das informações.
Análise e Automação de Relatórios: Consultas analíticas foram desenvolvidas para responder a perguntas de negócio. Uma VIEW foi criada para encapsular a lógica de relatório, simplificando o acesso a dados consolidados para a equipe de gestão.
Tecnologias e Ferramentas
Google BigQuery: Banco de dados analítico em nuvem, serverless e de alta escalabilidade.
SQL Standard SQL: Linguagem padrão para definição, manipulação e consulta dos dados.
Google Cloud Console: Interface para gerenciamento dos recursos e execução das consultas.
Detalhamento Técnico do Projeto

1. Estruturando o Armazenamento:

**create_tables _ big query.sql**
O primeiro passo foi criar a estrutura de tabelas, o que foi feito com atenção às particularidades do BigQuery. A modelagem normalizada garante que a informação não seja duplicada. O script a seguir cria as tabelas Clientes, Produtos e Vendas, utilizando a nomenclatura completa (projeto.dataset.tabela) e os tipos de dados nativos do BigQuery.
Diferenciais do BigQuery: O projeto demonstra a compreensão de conceitos importantes, como a ausência de restrições de chaves **PRIMARY KEY, FOREIGN KEY**, o que é um paradigma fundamental em bancos de dados analíticos.

• Tabela de Clientes
• Armazena informações únicas de cada cliente.
• No BigQuery, chaves primárias não são impostas, mas ID_Cliente serve como identificador lógico.

```
CREATE OR REPLACE TABLE `t1engenharinadados.livraria_devsaber.Clientes` (
   ID_Cliente INT64,
   Nome_Cliente STRING,
   Email_Cliente STRING,
   Estado_Cliente STRING
);
```

**1) Tabela Clientes**

| ID Nome Cliente | Email Cliente | Estado Cliente |
|---|---|---|
| 1. Ana Silva | ana.s@email.com | SP |
| 2. Bruno Costa | b.costa@email.com | RJ |
| 3. Carla Dias | carla.d@email.com | SP |
| 4. Daniel Souza | daniel.s@email.com | MG |

• Tabela de Produtos
• Armazena informações únicas de cada produto.

```
CREATE OR REPLACE TABLE `t1engenharinadados.livraria_devsaber.Produtos` (
   ID_Produto INT64,
   Nome_Produto STRING,
   Categoria_Produto STRING,
   Preco_Produto NUMERIC
);
```

**2) Tabela Produto**

| ID Nome Produto | Categoria Produto | Preco Produto |
|---|---|---|
| 101. Fundamentos de SQL | Dados | 60,00 |
| 102. Duna | Ficção científica | 80,50 |
| 103. Python para Dados | Programação | 75,00 |
| 104. O Guia do Mochileiro | Ficção científica | 42,00 |

• Tabela de Vendas
• Tabela de fatos que relaciona clientes e produtos, registrando cada transação.
• As relações com Clientes e Produtos são lógicas, mantidas pelos campos de ID.

```
CREATE OR REPLACE TABLE `t1engenharinadados.livraria_devsaber.Vendas` (
   ID_Venda INT64,
   ID_Cliente INT64,
   ID_Produto INT64,
   Data_Venda DATE,
   Quantidade INT64
);
```

**3) Tabela Vendas**

| VENDA | CLIENTE | PRODUTO | DATA VENDA | QUANTIDADE |
|---|---|---|---|---|
| 1 | 1 | 101 | 2024-01-15 | 1 |
| 2 | 2 | 102 | 2024-01-18 | 1 |
| 3 | 3 | 103 | 2024-02-02 | 2 |
| 4 | 1 | 104 | 2024-02-10 | 1 |

**2. Ingestão de Dados**

insert_data_bigquery.sql
Com o schema definido, a próxima etapa foi a ingestão dos dados de origem, que foram normalizados para evitar redundâncias. O script abaixo insere os dados de clientes e produtos de forma única em suas respectivas tabelas antes de popular a tabela de vendas, garantindo a consistência do modelo de dados.
•Atenção à Escalabilidade: O uso de INSERT INTO é ideal para este projeto de pequena escala. Para cenários de Big Data com milhões de registros, seria mais eficiente utilizar métodos como o carregamento em lote via Google Cloud Storage.

**3. Análise e Reuso**

analysis_queries_bigquery.sql
Esta etapa é a de extração de valor. O script a seguir contém consultas de exemplo que respondem a perguntas de negócio e, mais importante, a criação de uma VIEW para reuso e abstração da lógica de JOINs complexos.
•Otimização de custos: A consulta na VIEW é otimizada, pois o BigQuery cobra pela quantidade de dados processados. Ao usá-la, a equipe da livraria pode executar a mesma lógica de forma consistente e com menos linhas de código.

**Análise e Respostas.**
**Perguntas sobre a Estrutura.**
Com base nos dados brutos, quais outras duas tabelas precisamos criar? Que colunas e tipos de dados elas teriam?
**Resposta**: Precisamos de uma tabela Produtos (colunas: ID_Produto INT64, Nome_Produto STRING, Categoria_Produto STRING, Preco_Produto NUMERIC) e uma tabela Vendas (colunas: ID_Venda INT64, ID_Cliente INT64, ID_Produto INT64, Data_Venda DATE, Quantidade INT64)

2. Se o BigQuery não tem chaves estrangeiras, como garantimos que um ID_Cliente na tabela de vendas realmente existe na tabela de clientes?
**Resposta**: A responsabilidade por essa integridade é transferida para o processo de ETL e para as consultas. Garantimos a consistência no momento da análise, utilizando a cláusula JOIN com a condição Vendas.ID_Cliente = Clientes.ID_Cliente. Se um ID não existir na tabela Clientes, a linha correspondente simplesmente não será retornada com um INNER JOIN.

**Perguntas sobre a Ingestão**

Por que é uma boa prática inserir os clientes e produtos em suas próprias tabelas antes de inserir os dados de vendas?
**Resposta**: Esta é uma prática de normalização. Ela evita a duplicação de informações de clientes e produtos, o que melhora a integridade, reduz o espaço de armazenamento e simplifica atualizações futuras.

Em um cenário com milhões de vendas por dia, o INSERT INTO seria a melhor abordagem?
**Resposta**: Não. O INSERT INTO é eficiente apenas para pequenos volumes. Para milhões de registros, a abordagem recomendada seria o carregamento em lote de arquivos via Google Cloud Storage ou a ingestão por streaming.

**Perguntas sobre a VIEW**

Qual é a principal vantagem de usar uma VIEW em vez de simplesmente salvar o código em um arquivo de texto?
**Resposta**: A VIEW é um objeto do banco de dados que pode ser consultado como uma tabela. Ela simplifica o acesso a lógicas complexas, promove o reuso de código e garante que todos os analistas usem a mesma definição para o relatório.
Se o preço de um produto mudar na tabela Produtos, o Valor_Total na VIEW será atualizado automaticamente na próxima vez que a consultarmos?

**Resposta**: Sim. Como a VIEW é lógica, ela executa a consulta subjacente a cada vez que é acessada. Isso significa que o Valor_Total será recalculado com os preços mais recentes da tabela Produtos.