Documentação

- Dúvidas, suposições e comentários gerais:

- Referente ao requerimento: “Aplicar Delta de atualização nos últimos 2 meses.”, penso que após a ingestão total da tabela, apenas os dois últimos meses seriam atualizados em cada execução da pipeline e que o mês atual não era de importância para a tabela, apenas quando completo.
- Comecei a desenvolver uma solução utilizando DLT, que considero ser viável para o desafio, mas falhei em estabilizar a lógica de atualização incremental dentro do tempo disponível. A pipeline seguia sobrescrevendo a tabela, e não consegui controlar esse comportamento. Por isso, optei por não usar esses notebooks como solução principal neste teste, mas eles estão disponíveis para consulta.

- Referente ao ponto “Criar uma tabela Gold com a quantidade de caixas vendidas por ano, mes, sigla_uf”, dentro do meu entendimento, normalmente essa tabela também receberia o princípio ativo, mas como ele não foi mencionado no enunciado do teste, segui o escopo proposto.

- Arquitetura utilizada:

Bucket do Google Cloud como origem dos dados. Caso tivesse adição de orquestração neste teste, os notebooks do Databricks seriam a definição da lógica dos Jobs. Armazenamento utilizado foi o “Serverless Starter Warehouse”, disponível na versão Community do Databricks. Processamento do teste foi efetuado pelo Apache Spark, utilizando o “micro-cluster” disponível na versão Community. Não há etapa de consumo neste teste.

- Localização dos dados:

O link enviado me redirecionou para uma tabela de dados da Anvisa no BigQuery. Como a versão Community do Databricks não permite a instalação de conectores, optei por transferir os dados do BigQuery para um bucket público do Google Cloud, viabilizando a ingestão via download direto para a camada Raw.

- Papel de cada Notebook:

    - Raw_notebook.dbc:

        - Através desse notebook eu realizo o download de cada Parquet que está armazenado no bucket público do Google Cloud. Usei um range(87) fixo, considerando que o número de arquivos Parquet na fonte já era conhecido. Em uma situação de produção real teria sido utilizado uma variável que contasse a quantidade de arquivos para iterar, ou a leitura do bucket inteiro através de conectores, sem necessidade de iteração.
        - Emulei uma situação de produção ao definir uma função para a tarefa de ingestão do Google Cloud, garantindo reutilização do código quando houver a possibilidade.

    - Bronze_notebook.dbc:

        - O Notebook da Bronze é responsável pela transformação dos Parquets em Delta, com transformações mínimas. Já que foi definido que a janela de atualização dos dados é dos dois últimos meses, os notebooks vão trabalhar com flags que verificam se já existe a pasta _delta_log no diretório da tabela. Caso exista, isso significa que a Delta Table já existe, portanto vai realizar todo o processo apenas com os dois últimos meses. Caso não exista, irá realizar uma ingestão inteira.
        - As únicas duas transformações realizadas na Bronze são extremamente básicas, remoção de linhas completamente nulas (sem valores) e remoção de linhas totalmente idênticas com .distinct(). No geral os passos transformativos serão acompanhados da criação de um log para auditabilidade do impacto dessa transformação.

- Quando estamos trabalhando dentro do período de dois meses, a lógica de atualização da tabela é de delete e insert para os dados destes meses. Eu também pensei nas possibilidades de utilizar Merge, sendo o impedimento a falta de uma chave única, ou particionar por ano/mês a partir da Bronze e deletar as últimas duas partições.

- Silver_notebook.dbc:

- Segue a mesma lógica de verificar a existência da Delta Table para decidir se vai trabalhar com a Bronze inteira ou somente os últimos dois meses dela.
- Através da variável “colunas_criticas” eu defino que a linha do dado não teria valor para uma análise caso as colunas principio_ativo, ano, mes, unidade_medida ou quantidade_vendida estejam com valores nulos, portanto estas linhas são removidas. Esse processo é acompanhado da geração de um log para avaliação do impacto.
- Realizo uma limpeza básica ao remover os espaços presentes em início ou final de strings.
- Percebi a existência de princípios ativos e descrições iguais (com as mesmas letras), porém com a acentuação inconsistente, portanto optei por remover qualquer acentuação desses dois campos para não prejudicar a análise.
- Utilizei ZORDER nas colunas de alta cardinalidade, conforme indicado no enunciado. Apesar disso, vale ressaltar que definir se isso é desejável de fato depende de um contexto mais completo de uso e volume, que vai além do escopo do teste.

- Gold_notebook.dbc:

    - Utiliza lógica de overwrite por ser uma tabela agregada em poucas linhas.
    - Realize a agregação desejada, primeiramente filtrando os dados para conterem somente vendas de caixas, para depois realizar a agregação.
    - Realizo a criação da Delta Table com o particionamento por ano e mês.
    - Devido ao baixo volume de dados da tabela optei por não realizar o ZORDER. A tabela já tem particionamento na data e não tem cardinalidade alta nas suas colunas. Ter utilizado lógica de overwrite torna o ZORDER uma operação custosa, também.