Dia 05

DBT - Materializações

DBT - Materializações

- São as diferentes formas que o DBT pode materializar uma tabela
- Enquanto o usuário foca em escrever declarações de SELECT (DQL), o DBT se encarrega de criar todo o DML/DDL. Materializações permitem ao desenvolvedor ter mais controle sobre essa segunda parte.

DBT - Materialização - Tipos

- Tabela
- Views
- Efêmeros
- Incremental
- Snapshots

DBT - Materialização - Tipos - Tabela

- Tabela
  - São materializadas fisicamente no data warehouse (DDL)
  - Armazenam dados em disco
  - São recriadas do zero durante a fase de execução
- Views
- Efêmeros
- Incremental
- Snapshots

DBT - Materialização - Tipos - Views

- Tabela
- Views
  - São comandos de SELECT armazenados (declarações de DQL)
  - Não armazenam dados em disco
  - São recalculadas toda vez que chamadas. (geralmente lento)
- Efêmeros
- Incremental
- Snapshots

DBT - Materialização - Tipos - Efêmeros

- Tabela
- Views
- Efêmeros(tabela em memória)
  - São tabelas construídas apenas em tempo de execução
  - Armazenam dados em memória (não são construídas fisicamente)
  - São adicionadas como CTE nos modelos subsequentes
  - Não é possível realizar comandos select nesse tipo de tabela (não são armazenadas no DW)
  - Ajuda a manter o DW limpo
  - Código reutilizável entre modelos subsequentes
  - DBT não vai gerar log para essas tabelas, já que elas vão ser utilizadas como CTE nas subsequentes
  - Útil para reusabilidade de CTEs que não precisam ser armazenadas como intermediárias
- Incremental
- Snapshots

Demo

...

Materialização Efêmera

DBT - Materialização - Tipos - Incremental

- Tabela
- Views
- Efêmeros (tabela em memória)
- Incremental (insert only/upsert)
  - São tabelas materializadas fisicamente no DW (DDL)
  - Dados são armazenados no disco
  - Não recria a tabela na fase de execução, apenas realiza insert das novas informações.
  - Bom para tabelas que não recebem operações de delete, apenas insert e update
  - Necessita de um campo chave e uma coluna de última atualização do registro
  - Utiliza menos recursos e geralmente roda mais rápido que uma carga full
  - Cria uma tabela temporária com os novos dados e realiza um merge com a tabela principal.
  - O comando utilizado e o suporte a essa materialização vai depender do DW
- Snapshots

DBT - Materialização - Tipos - Incremental

- Tabela
- Views
- Efêmeros (tabela em memória)
- Incremental (insert only/upsert)
  - O usuário define a regra que separa dados antigos de dados novos
  - Espera-se que os dados históricos não serão alterados
  - É possível realizar uma carga full da tabela utilizando o comando `dbt run --full-refresh`
  - Realizar carga full custa tempo e dinheiro, então utilize o comando acima com parcimonia
  - Particularmente bom para dados de evento (que podem chegar com atraso mas não mudam o histórico)
  - É possível setar um período de tolerância (para eventos atrasados)
- Snapshots

DBT - Materialização - Incremental

- Quando utilizar:
  - Tabela possui uma chave e um campo de última atualização confiáveis
  - Streaming/Eventos imutáveis (sem updates, append-only, tabelas grandes)
  - Quando estar aproximadamente correto é o suficiente
  - Necessidade de aumentar o desempenho de carga de tabelas grandes
- Quando Não utilizar:
  - Tabela com poucos dados
  - Tabelas que mudam com constância (novas colunas, renomeamentos, etc.)
  - Dados atualizados de maneira imprevisível (muitos dados atrasados)
  - Tabelas com cálculos que dependem de outras tabelas (window functions, joins)
  - Quando estar 100% correto é essencial

DBT - Materialização - Incremental

- Como informar para o dbt para adicionar apenas os dados novos em vez de recriar a tabela?
  - Configure a materialização para incremental com `{{ config(materialized='incremental') }}`
- Como informar para o dbt o que são dados novos?
  - Utilize uma cláusula where para seleção de novos dados, e um filtro jinja para carga full da tabela caso seja a primeira carga.
- Como configurar uma janela de tolerância?
  - Utilize jinja para adicionar uma janela de segurança (e.g. puxar dados dos últimos 3 dias)
- E se os dados escaparem da janela de tolerância?
  - Recomenda-se a execução de um full refresh em intervalos maiores de tempo
- Como remover dados duplicados?
  - Adicione uma configuração de chave primária.

DBT - Materialização - Incremental

- Tem uma função Jinja no DBT específica para verificar se um modelos é incremental.
- is_incremental() - verifica quatro condições:
  - O modelo existe no data warehouse?
  - O objeto em questão é uma tabela?
  - A tabela é configurada como incremental?
  - O comando `dbt run` foi passado sem o parâmetro `--full-refresh`?
  - Se “sim” for a resposta para todas as perguntas, então executa uma carga incremental, caso contrário não faz nada

Demo

...

Materialização Incremental

DBT - Materialização - Tipos - Snapshots

- Tabela
- Views
- Efêmeros (Tabela em memória)
- Incremental (insert only/upsert)
- Snapshots (CDC)
  - São tabelas armazenadas fisicamente no DW
  - Dados são armazenados no disco
  - São atualizadas com o comando `dbt snapshot`
  - Ficam em uma pasta de Snapshot separada (Snapshots/*.sql)
  - Mantém o histórico de todas as modificações em uma tabela (CDC)
  - Particularmente bom para tabelas que mudam com frequência (SCD - Tipo 2)
  - Particularmente bom para manter histórico de métricas
  - Preserva o histórico de maneira eficiente, sem duplicar dados que não sofreram alteração
  - Armazena apenas a diferença entre a tabela antes e depois da alteração, e cria automaticamente colunas de validade para cada registro

DBT - Materialização - Tipos - Snapshots

- Cláusula snapshot {% snapshot <nome> %}..{% endsnapshot %}
- Adicionar estratégia de snapshot no arquivo dbt_project.yml
- Configure a estratégia que será utilizada
  - Timestamp (updated_at): rastreia mudanças com base em uma coluna de data/hora
  - Check (check_cols): rastreia mudanças com base em colunas específicas
- Recomenda-se arquivar/descartar dados antigos com o tempo para otimizar o desempenho e reduzir os custos de armazenamento

Snapshots

Estrutura

Snapshots

Configuração

Demo

...

Materialização Snapshot

Materialização	Descrição	Prós	Contras	Configuração
Tabelas	Construído como tabelas no DW	Dados são armazenados no disco. Rápido para consultar.	Mais lento para construir.	{{ config(materialized='table') }}
Views	Construído como visões no DW	A consulta é armazenada no disco. Mais rápido para construir.	Mais lento para consultar.	{{ config(materialized='view') }}
Efêmero	Não existe no DW	Importado como CTE em modelos subsequentes.	Não pode ser consultado direto.	{{config(materialized='ephemeral')}}
Incremental	Construído como tabelas no DW	Adiciona apenas adiciona registros novos (Upsert).	Não captura 100% das alterações.	Configuração avançada
Snapshots	Construído como tabelas no DW	Permite capturar dados históricos (CDC).	Ocupa muito espaço	Configuração avançada

DBT - Comandos

- dbt run --full-refresh
- dbt snapshot:

DBT - Testes Avançados

DBT - Testes

- DBT vai procurar no projeto por testes especificados em arquivos YAML.
- DBT auxilia desenvolvedores a escalar testes entre projetos de forma rápida e fácil.
- DBT permite a criação de testes singulares.
- DBT disponibiliza 4 testes já desenvolvidos (testes genéricos)
  - Unique
  - Not null
  - Values accepted
  - Relationship
- É possível adicionar mais testes ao DBT utilizando Packages.

DBT - Testes - Tipos

- **Singular (importados ou personalizados):** arquivos sql na pasta de teste (Test/*.sql)
- **Genéricos:** são testes parametrizáveis (genéricos) definidos em um bloco de {% test nome_funcao(modelo, coluna) %}...{% endtest %}. Testes genéricos podem ser definidos como propriedades nos arquivos .yml.

DBT - Testes Avançados - Source Freshness

- Testes em DBT esperam que sejam retornadas 0 linhas para serem considerados sucesso.
- É possível configurar alerta baseado no dado mais recente de um campo. O comando dbt source freshness é utilizado para avaliar a atualização das fontes e alertar/gerar erro caso não estejam conforme esperado.
- Também é possível armazenar as linhas resultantes de falhas dos testes utilizando o comando `dbt test --store-failures`.

DBT - Testes - Dicas

- Crie pelo menos um teste para cada bug. “Se quebrou uma vez, vai quebrar de novo”.
- Ajuste a severidade para “Warn” se os testes retornarem muitos falsos positivos.
- Adicione comentários em testes singulares sobre como debugá-los.
- Durante o desenvolvimento de modelos escreva testes para confirmar suas expectativas em relação ao mesmo.
- Também é possível escrever o teste primeiro e desenvolver o modelo depois (TDD)
- Rode todos os testes antes de realizar uma implantação em produção para aumentar a confiança (CI).

Demo

...

Testes Avançados

DBT - Implementação

DBT - Implementação - Objetivos

- Colocar um projeto DBT em produção
- Rodar uma série de comandos DBT de maneira automática/agendada

DBT - Implementação - Ambientes

- **Desenvolvimento:** é o processo de construir, refatorar, e organizar diferentes arquivos de um projeto. Após realizar algumas mudanças, o branch de desenvolvimento é mesclado com o branch de produção.
- **Produção:** é o processo de rodar comandos por agendamento ou reação a eventos, para disponibilizar código confiável às aplicações.

DBT - Implementação - Dicas

- Desenvolvimento
  - Tenha um usuário de banco de dados para cada desenvolvedor (e.g. <username>). Isso ajuda com debug, auditoria, e segurança.
  - Tenha um esquema para cada desenvolvedor (e.g. dbt_<username>). Assim os usuários não constroem seus objetos em cima dos criados por terceiros.
  - Tenha uma branch por feature/bug/card (e.g. jira231)
  - Garanta que os usuários tenham os privilégios necessários para criação, leitura e manipulação de seus esquemas e tabelas.
- Produção
  - Tenha um usuário de banco de dados de produção separado (e.g. dbt_service)
  - Tenha um esquema de produção separado (e.g. analytics)
  - Tenha um branch de produção separado (e.g. main)

DBT Build

- É melhor rodar `dbt build` em vez de `dbt run + dbt test`, por que se houver algum erro de teste na em algum modelo, o primero comando irá para a execução em vez de materializar todos os modelos e só depois rodar os testes (como no segundo)

Demo

...

Projeto pt.2

Hooks

Certificação

Revisão

DBT Conceitos

- Modularidade
- Modelos
- Projeto
- Adaptadores
- Target
- Threads
- Tags
- Testes
- Documentação
- Linhagem
- CI/CD

DBT Conceitos

- Macros
- Packages
- Materializações
- Modelos Efêmeros
- Modelos Incrementais
- Snapshots
- Refatoração de SQL
- Seeds
- Sources
- Exposures
- Analyses

DBT - Arquivos

- profiles.yml
- dbt_project.yml
- models/*.sql
- models/schema.yml
- models/*.yml

DBT - Arquivos

- **profiles.yml:** arquivo utilizado para configuração das credenciais de conexão aos data warehouses.
- **dbt_project.yml:** arquivo principal do projeto utilizado para configurações gerais do projeto (e.g. caminho de pastas, metadados, configurações padrão).
- **models/*.sql:** cada arquivo sql dentro da pasta models representa um modelo (tabela ou view) do data warehouse
- **models/schema.yml:** arquivo utilizado para definição de testes e metadado dos modelos
- **models/*.yml:** geralmente os demais arquivos YAML representam os metadados das fontes de dados.

DBT - Comandos

- dbt run
- dbt test
- dbt build
- dbt docs generate
- dbt docs serve
- dbt source freshness
- dbt deps
- dbt seed
- dbt run-operation

DBT - Comandos

- **dbt run:** compila e executa os modelos no diretório models, resultando em tabelas ou visões no data warehouse.
- **dbt test:** executa testes predefinidos ou personalizados garantindo a qualidade e integridade dos dados.
- **dbt build:** combina os comandos run e test, compilando, executando e depois testando os modelos em uma única etapa para simplificar o fluxo de trabalho.
- **dbt docs generate:** gera automaticamente documentação com base em metadados e comentários.
- **dbt docs serve:** fornece uma interface para visualizar a documentação gerada localmente.
- **dbt source freshness:** avalia a atualidade dos dados em fontes externas, alertando sobre dados desatualizados.
- **dbt deps:** gerencia dependências do projeto, utilizando informações dos arquivos dbt_project.yml e packages.yml. Assegura que todas as macros e pacotes sejam resolvidos e estejam disponíveis.
- **dbt seed:** transforma os CSVs do diretório seed em tabelas, ideal para conjuntos de dados de referência.

DBT Cheat sheet

•••

https://datacoves.com/post/dbt-cheatsheet

DBT Cheat sheet

•••

https://github.com/bruno-szdl/cheatsheets/blob/main/dbt_cheat_sheet.pdf

DBT Jinja - Funções

- env_var
- ref
- source
- doc
- config

DBT Jinja - Funções

- env_var(‘variavel_ambiente’, ‘<valor_padrao>’):
- ref(‘<nome_modelo>’): cria dependências entre os modelos
- source(‘<database>’, ‘<tabela>’): cria dependência entre modelos e sources (é necessário ter um arquivo yaml configurado)
- doc(“<bloco_documentação>”): inserir documentação adicional diretamente dentro do código SQL.
- config(<configuracao>=<valor>, ...): sobrescreve uma configuração padrão para aquele modelo

DBT Jinja Cheat sheet

...

https://datacoves.com/post/dbt-jinja-cheat-sheet

Checklist de conteúdo

Dia 01 - Base

- Business Intelligence
  - ETL
  - Traditional DW
  - Data Viz
  - Q&A: Business intelligence
- DW
  - Snowflake, PostgreSQL, Big Query
  - Modelagem dimensional
  - Star Schema, Snowflake, Data Vault
  - Slowly Changing Dimensions (SCD)
  - Demo: PostgreSQL and Dbeaver Instalation
- DBT Core
  - Contexto
  - Timeline
  - Concepts
  - Project Structure
  - Demo: DBT Core Installation
- Just Enough Python for BI
  - Virtual env
  - Pandas
  - SQL Alchemy
  - Data Viz
  - Demo: BI with Python, DBT Core, and PostgreSQL

Dia 02 - Fundamentos

- Big Data and Cloud
  - Big Data and Cloud
  - Cloud Data Warehouse (CDW)
  - Data lake vs Data Warehouse vs lakehouse
  - External tables
- Modern Data Stack
  - Modern Data Stack
  - Analytics Engineer
  - BI vs MDS
  - ELT vs ETL
  - CDW vs TDW
  - Q&A: BI vs MDS
- Git Overview
  - DevOps
  - Branchs (development, main, etc.)
  - Commands
  - Gitflow
  - Pull Request
  - Demo: Git installation and workflow
  - Demo: Github setup
- DBT Cloud
  - DBT Cloud
  - DBT Core vs Cloud
  - Demo: Airbyte Account Creation
  - Demo: Big Query Account Creation
  - Demo: DBT Cloud Account Creation
  - Demo: MDS with Airbyte, DBT Cloud and Big Query

Dia 03 - Cloud

- Engineering Best Practices
  - DevOps/DataOps
  - Tests and Documentation
  - Data Quality
  - Q&A: Best Practices
- SQL Advanced
  - Joins and unions
  - Aggregations and functions
  - CTEs and subselects
  - Window Functions
  - Query Execution Plan
  - Explain and Optimization
  - Demo: Reviewing SQL Queries
  - Demo: Query optimization
- DBT Features
  - Sources
  - Tags
  - Tests and Advanced Tests
  - Documentation
  - CI/CD
  - Logging and Alerting
  - Demo: Snowflake Account Creation
  - Demo: DBT best practices

Dia 04 - Intermediário

- Analyses
- Seeds
- Advanced Materialization (Incremental & Ephemeral)
- Snapshots
- Singular and Custom Tests
- Exposures

Dia 05 - Avançado

- Refactoring SQL
- Jinja
- Macros
- Packages
- Hooks
- Certification Tips

Recomedações

Recomedações

- Live: Primeiros passos com DBT
- Fast-Track: Git para times de dados
- Fast-Track: Python para engenharia de dados
- Workshop 08: Implementando pipelines SQL com DBT

Obrigado!