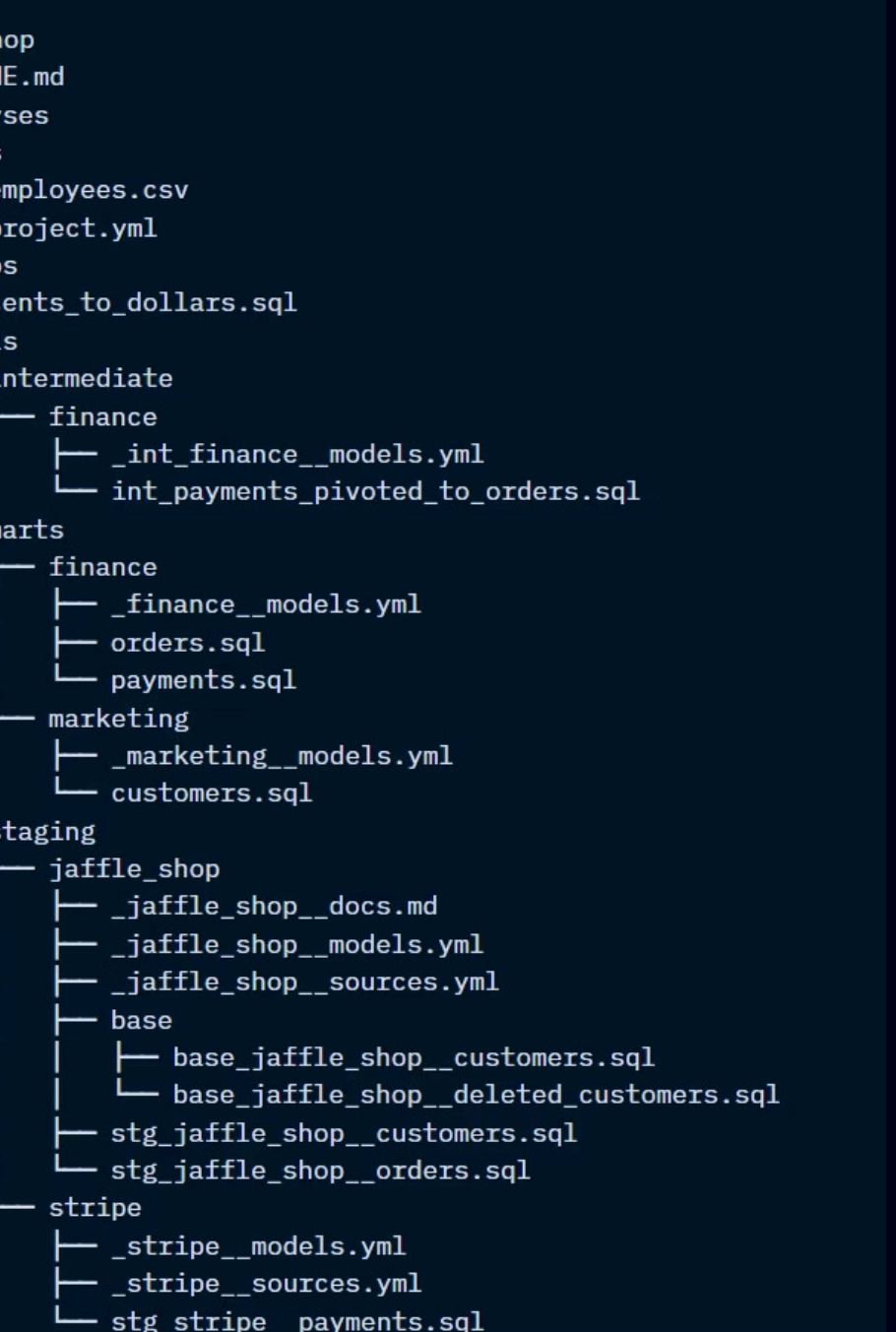


Dateneo®

Estructura de proyectos en dbt

Una guía completa basada en las mejores prácticas de dbt Labs para organizar proyectos de ingeniería analítica de manera escalable y colaborativa.

¿Por qué importa la estructura?

La importancia de los patrones consistentes

La ingeniería analítica busca que los equipos trabajen juntos para tomar mejores decisiones. Como solo tenemos un tiempo y energía limitados para decidir, necesitamos sistemas y patrones claros para colaborar de forma eficaz.

Por eso, en proyectos donde se colabora, es clave tener reglas coherentes y fáciles de entender. Así, el equipo puede usar su tiempo limitado en resolver problemas complejos, en lugar de discutir dónde guardar archivos o cómo nombrarlos.

Crear un buen proyecto dbt es un trabajo de equipo. Reúne el conocimiento de cada área para entender los objetivos de toda la empresa. Por ello, es muy importante establecer patrones claros y completos que permitan a muchas personas usar su experiencia de la mejor manera posible.

El principio fundamental

1

Source-Conformed

Datos moldeados por sistemas externos fuera de nuestro control

2

Business-Conformed

Datos moldeados por las necesidades, conceptos y definiciones que creamos

Un principio fundamental que se aplica a todos los proyectos dbt es la necesidad de establecer un arco cohesivo que mueva los datos de **source-conformed** a **business-conformed**. Sin importar qué patrones o convenciones definas dentro de tu proyecto, este proceso sigue siendo el propósito esencial de la capa de transformación.

Las tres capas principales

Staging

Creando nuestros átomos, nuestros bloques de construcción modulares iniciales, a partir de datos fuente

Intermediate

Apilando capas de lógica con propósitos claros y específicos para preparar nuestros modelos staging

Marts

Reuniendo nuestras piezas modulares en una visión amplia y rica de las entidades que le importan a nuestra organización

Capa Staging

Preparando bloques atómicos

Esta es la base de nuestro proyecto, donde traemos todos los componentes individuales que vamos a usar para construir nuestros modelos más complejos y útiles.

Staging: estructura de archivos y carpetas

Organización por sistema fuente

Los subdirectorios deben basarse en el sistema fuente.

Por ejemplo una base de datos transaccional interna, los datos que obtenemos de una API o los eventos de telemetría que llegan de una herramienta.

Los sistemas fuente tienden a compartir métodos de carga y propiedades similares entre tablas, y esto nos permite operar en esos conjuntos similares fácilmente.

⊗ **Evita:** subdirectorios basados en el cargador (dltHub, Fivetran, syncs personalizados) - esto es demasiado amplio para ser útil en un proyecto de cualquier tamaño real.

Staging: convenciones de nomenclatura

✅ Patrón recomendado

`stg_[source]__[entity]s.sql`

El doble guión bajo entre sistema fuente y entidad ayuda a distinguir visualmente las partes separadas en caso de que un nombre de fuente tenga múltiples palabras.

❌ Evitar

`stg_[entity].sql`

Podría ser suficientemente específico al principio, pero se romperá con el tiempo. Agregar el sistema fuente al nombre del archivo ayuda en la capacidad de descubrimiento.

✅ Usar plural

SQL, y particularmente SQL en dbt, debe leerse tanto como prosa como podamos lograr. A menos que haya una sola orden en tu tabla de órdenes, plural es la forma correcta de describir lo que hay en una tabla con múltiples filas.

Staging: criterios de los modelos

✅ Transformaciones permitidas

- Renombrado de columnas
- Casting de tipos
- Cálculos básicos (ej. centavos a dólares)
- Categorización (usando lógica condicional para agrupar valores)

✅ Materialización

Como vistas. No están destinados a ser artefactos finales en sí mismos, sino bloques de construcción para modelos posteriores.

❌ Transformaciones prohibidas

- **Joins:** El objetivo es limpiar y preparar conceptos individuales source-conformed
- **Agregaciones:** Las agregaciones implican agrupación, y no estamos haciendo eso en esta etapa

ⓘ **Principio DRY:** Los modelos staging nos ayudan a mantener nuestro código DRY. Cualquier transformación que siempre queramos usar debe ir lo más upstream posible.

Staging: modelos base

A veces, para mantener una capa staging limpia y DRY, necesitamos implementar algunos joins para crear un concepto sólido para nuestros bloques de construcción. En estos casos, recomendamos crear un subdirectorio en el directorio staging para el sistema fuente en cuestión y construir modelos base.

Unir tablas de eliminación separadas

A veces un sistema fuente podría almacenar eliminaciones en una tabla separada. Típicamente querremos asegurarnos de poder marcar o filtrar registros eliminados.

Unir fuentes dispares pero simétricas

Un ejemplo típico sería si operas múltiples plataformas de ecommerce en varios territorios. Tendrías esquemas perfectamente idénticos, pero todos cargados por separado.

Capa Intermediate

Pasos de transformación con propósito específico

Una vez que tenemos nuestros átomos listos para trabajar, nos disponemos a reunirlos en formas moleculares más intrincadas y conectadas.

Intermediate: estructura y nomenclatura

Organización por agrupaciones de negocio

A diferencia de la capa staging, aquí nos movemos hacia ser business-conformed, dividiendo nuestros modelos en subdirectorios no por su sistema fuente, sino por su área de preocupación empresarial.

Convención de nombres

`int_[entity]s_[verb]s.sql`

La variedad de transformaciones que pueden ocurrir dentro de la capa intermediate hace más difícil dictar estrictamente cómo nombrarlas. El mejor principio guía es pensar en **verbos** (ej. pivoted, aggregated_to_user, joined, fanned_out_by_quantity, etc.).

⚠ **¡No optimices demasiado temprano!** Si tienes menos de 10 modelos marts y no estás teniendo problemas desarrollándolos y usándolos, siéntete libre de prescindir de subdirectorios completamente hasta que el proyecto haya crecido para realmente necesitarlos.

Intermediate: criterios de los modelos

❌ No expuestos a usuarios finales

Los modelos intermediate generalmente no deben exponerse en el esquema principal de producción. No están destinados para salida a objetivos finales como dashboards o aplicaciones.

✅ Materializados como efímeros

Una opción popular es que los modelos intermediate se materialicen de forma efímera por defecto. Esto mantendrá modelos innecesarios fuera de tu warehouse con configuración mínima.

✅ Vistas en esquema personalizado

Una opción más robusta es materializar tus modelos intermediate como vistas en un esquema personalizado específico, fuera de tu esquema principal de producción.

Intermediate: casos de uso comunes

Simplificación estructural

Reunir un número razonable (típicamente 4 a 6) de entidades o conceptos que se unirán con otro modelo intermediate de propósito similar para generar un mart, en lugar de tener 10 joins en nuestro mart.

Re-granularidad

Los modelos intermediate se usan a menudo para expandir o colapsar modelos al grano compuesto correcto. Si estamos construyendo un mart para order_items que requiere expandir nuestras órdenes basadas en la columna cantidad.

Aislar operaciones complejas

Es útil mover cualquier pieza de lógica particularmente compleja o difícil de entender a sus propios modelos intermediate. Esto no solo los hace más fáciles de refinar y solucionar problemas.

Principio: simplifica el DAG

Hasta la capa "marts" (nuestros resultados finales), el diagrama de tu proyecto (DAG) debe parecer una flecha apuntando a la derecha. A medida que pasamos de datos crudos (source-conformed) a datos útiles para el negocio (business-conformed), también pasamos de muchos conceptos pequeños y separados a menos conceptos, pero más grandes y conectados.

Estamos uniendo nuestras piezas de datos en conceptos más amplios y completos. Esto crea esa forma de flecha en nuestro DAG. Así, cuando lleguemos a la capa "marts", tendremos un conjunto sólido de componentes que se pueden usar de forma rápida y fácil para responder muchas preguntas.

> **Consejo práctico:** Un modelo puede recibir datos de varias fuentes (muchas flechas entrando), pero **no** debe enviar datos a múltiples lugares (muchas flechas saliendo es una señal de alerta).

Capa Marts

Entidades definidas por el negocio

Esta es la capa donde todo se une y comenzamos a organizar todos nuestros átomos y moléculas en células completas que tienen identidad y propósito.

Marts: estructura y nomenclatura

✅ Agrupar por departamento

Si tienes menos de 10 marts aproximadamente, puede que no tengas mucha necesidad de subcarpetas. Si te encuentras necesitando insertar más estructura y agrupación, usa conceptos de negocio útiles aquí. En nuestra capa marts, ya no nos preocupamos por datos source-conformed, así que agrupar por departamentos (marketing, finanzas, etc.) es la estructura más común en esta etapa.

✅ Nombrar por entidad

Usa español simple para nombrar el archivo basado en el concepto que forma el grano del mart: clientes, ordenes.

❌ **Anti-patrón:** Construir el mismo concepto de manera diferente para diferentes equipos. `finance_orders` y `marketing_orders` típicamente se considera un anti-patrón.

Excepción: Hay casos donde finanzas puede tener necesidades específicas, por ejemplo reportar ingresos al gobierno de una manera que diverge de cómo la empresa en su conjunto mide los ingresos día a día.

Marts: criterios de los modelos

✅ Materializados como tablas

Una vez que llegamos a la capa marts, es hora de comenzar a construir no solo nuestra lógica en el warehouse, sino los datos mismos. Esto da a los usuarios finales un rendimiento mucho más rápido para estos modelos posteriores que están realmente diseñados para su uso.

✅ Amplios y desnormalizados

A diferencia del warehousing de la vieja escuela, en el stack de datos moderno el almacenamiento es barato y es el cómputo lo que es caro. Empaqueta estos en conceptos desnormalizados muy amplios que pueden proporcionar todo lo que alguien necesita sobre un concepto como objetivo.

❌ Demasiados Joins

Una buena regla práctica al construir transformaciones dbt es evitar reunir demasiados conceptos en un solo mart. Si estás reuniendo más de 4 o 5 conceptos para crear tu mart, puedes beneficiarte agregando algunos modelos intermediate para mayor claridad.

Marts: consideraciones adicionales

Grano de entidad

El aspecto más importante de los marts es que contienen todos los datos útiles sobre una entidad particular a un nivel granular. Eso no significa que no traigamos muchas otras entidades y conceptos, ¡sí lo hacemos! Solo significa que las órdenes individuales siguen siendo el grano central de nuestra tabla.

Construir sobre Marts separados cuidadosamente

Aunque nos esforzamos por preservar un DAG que se estrecha hasta la capa marts, una vez aquí las cosas pueden volverse un poco menos estrictas. Un ejemplo común es pasar información entre marts en diferentes granos. Ahora que realmente estamos 'gastando' cómputo y almacenamiento, es sensato aprovechar recursos previamente construidos.

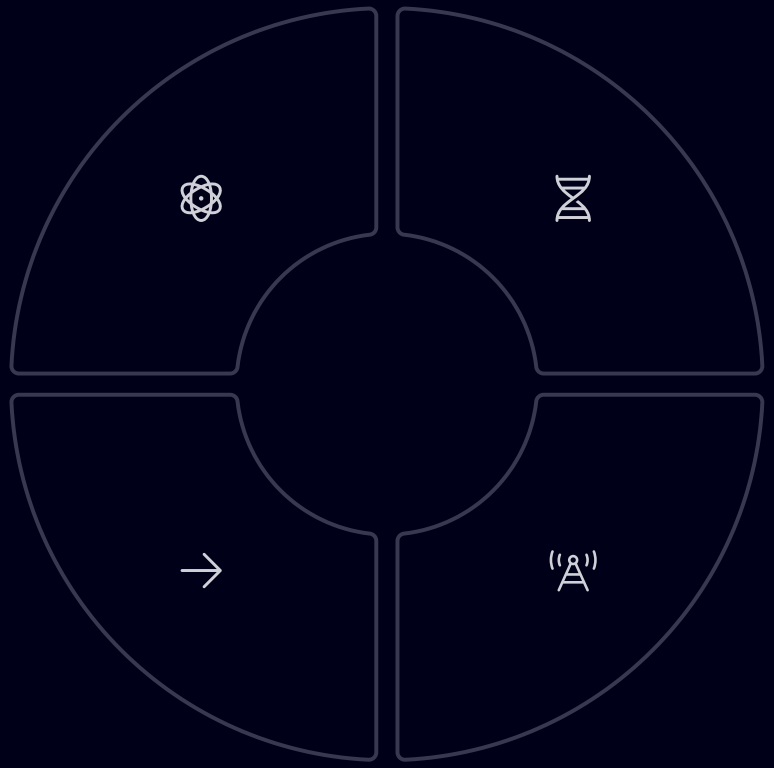
Solución de problemas vía tablas

Mientras que apilar vistas y modelos efímeros hasta nuestros marts es ideal en producción, puede presentar algunas dificultades en desarrollo. Si tienes problemas identificando dónde o qué te está diciendo un error de base de datos, puede ser útil construir temporalmente una cadena específica de modelos como tablas.

Resumen: principios clave de dbt Labs

Staging como átomos

Modelos 1-a-1 con fuentes, transformaciones básicas, materializados como vistas. Organizados por sistema fuente.



Intermediate como moléculas

Transformaciones con propósito específico, no expuestos a usuarios finales. Organizados por área de negocio.

Flujo Source-to-Business

Movimiento cohesivo de datos source-conformed a business-conformed a través de las capas de transformación.

Marts como células

Entidades business-conformed, amplias y desnormalizadas, materializadas como tablas. Grano de entidad específica.

Esta guía está basada en las recomendaciones y mejores prácticas desarrolladas por dbt Labs. Recuerda que estos son puntos de partida - puedes decidir que prefieres un enfoque diferente para tu proyecto, pero lo importante es pensar a través del razonamiento para esos cambios y mantenerte consistente.