dbt Labs

Buenos Aires dbt Meetup

Anfitriones: Mutt Data

12 Diciembre 2024 | #local-argentina

Agenda

1. **Llegada y Recepción**
   6:00-6:30 PM

2. **Bienvenida y Presentaciones**
   6:30-6:45 PM

3. **Novedades de dbt Labs**
   6:45-7:00 PM
   Victoria Perez Mola

4. **Optimizando modelos incrementales en dbt-Postgres**
   7:00-7:30 PM
   Gonzalo Villafañe Tapia

5. **Lo que está bajo el capó de dbt: materializaciones**
   7:30-8:00 PM
   Lucas Trubiano

6. **Networking**
   8:00-8:30 PM

Today's host

Alejandro Rusi

Tech Lead

Mutt Data

We are Mutt Data

+10
Years Building Big Data Architectures

+200
Implemented projects of AI and Data Systems

+150
Experts in Machine Learning and Data Engineering

We partner with

aws ASTRONOMER H2O.ai databricks

Meet Mutt Data

Solutions and Companies

Data Stack

- clip
- CONNECTURE DRX
- recargapay
- M2M Capital
- FORUS CONSUMER FANATIC
- NIMBLE GRAVITY
- ClassDojo
- ASTRONOMER
- Addi

Machine Learning

- mercado libre
- mercado pago
- recargapay
- ualá
- Coca-Cola FEMSA
- Rappi
- SIRUM
- Habitat LOGISTICS

Gen AI

- mercado libre
- MODO
- Coca-Cola FEMSA

Why Mutt Data

Expert Tech Guidance

Sail through your Data and ML maturity journey guided by a team of experts that combine industry knowledge with deep, hands-on technical expertise.

Tailor-Made Solutions

Avoid wasting time adapting your processes, reporting and data integrations to inflexible 3rd-party systems. Instead, let us develop solutions that reuse pre-existing tech capabilities to fit your needs.

Modern Data Architecture

We'll enhance your tech capabilities and fast-track business value by helping you build and deploy solid data architectures and ML solutions in the Cloud.

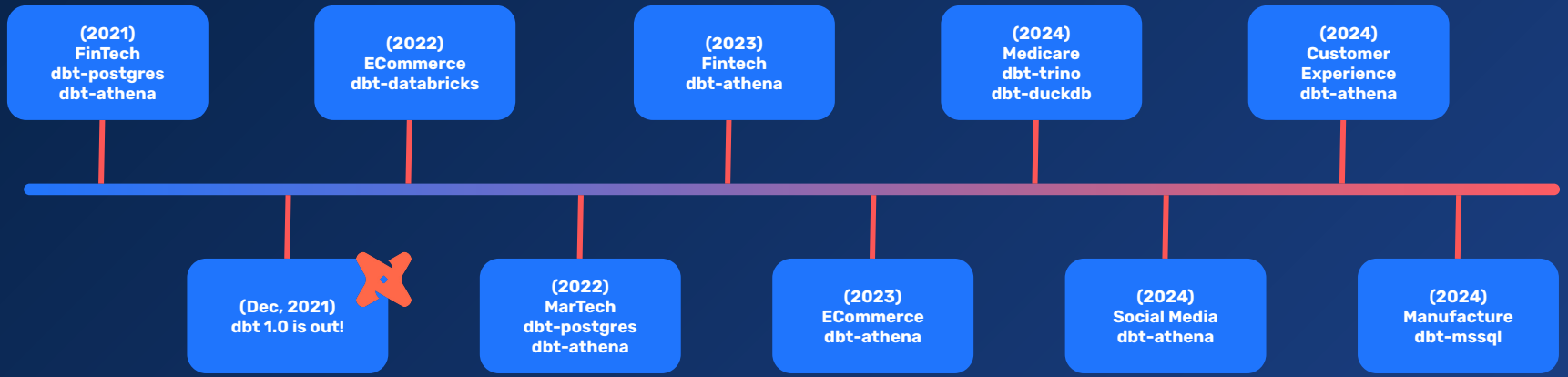
+150
Mutt Data experts

+10 years
working with marketing teams

+200
projects delivered

#1 in South LATAM
AWS Marketing Tech Competency

dbt <> MUTT DATA



dbt helped us transform data to...

- Train **credit scoring** Machine Learning algorithms.
- Build **recommendation systems**.
- Create **awesome dashboards**.
- Monitor efficiency with **SLA metrics**.
- Power our **Digital Marketing Budget Optimizer** (our talk today!)

And much, much more...

We are Hiring!

Data Developer

Data Engineer

Data Engineer Lead

ML Engineer

ML Engineer Lead

All Job Postings

Victoria Perez Mola

Professional Services Regional Manager

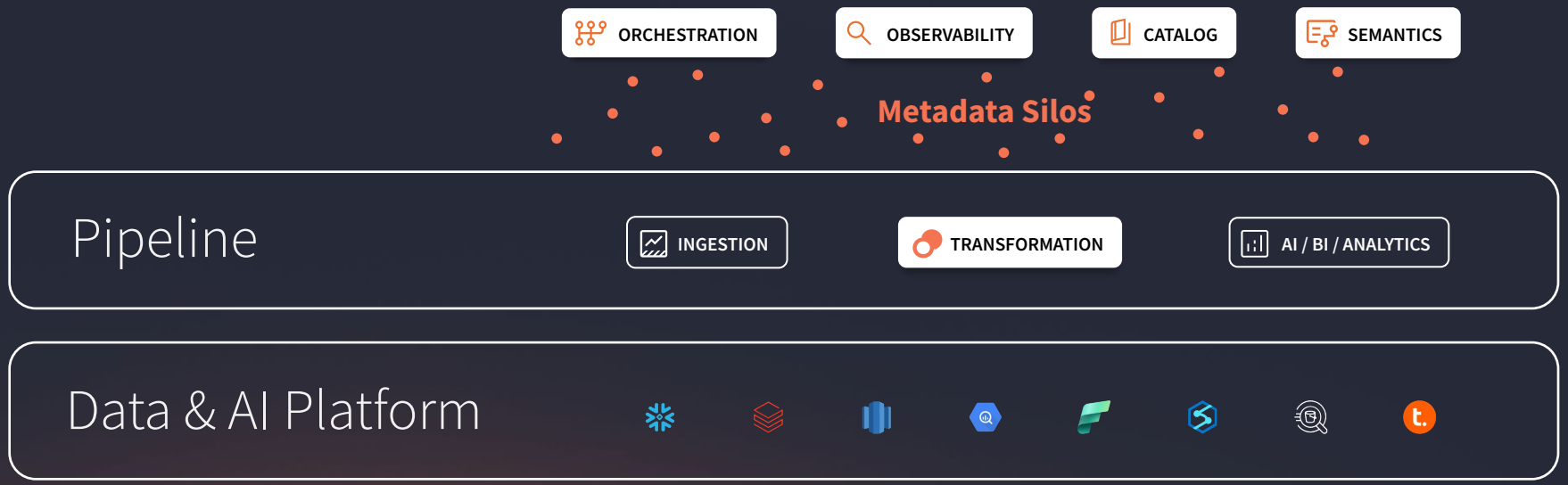
dbt Labs

Novedades de dbt Labs


one

El avance de las plataformas de datos ha creado un ecosistema…

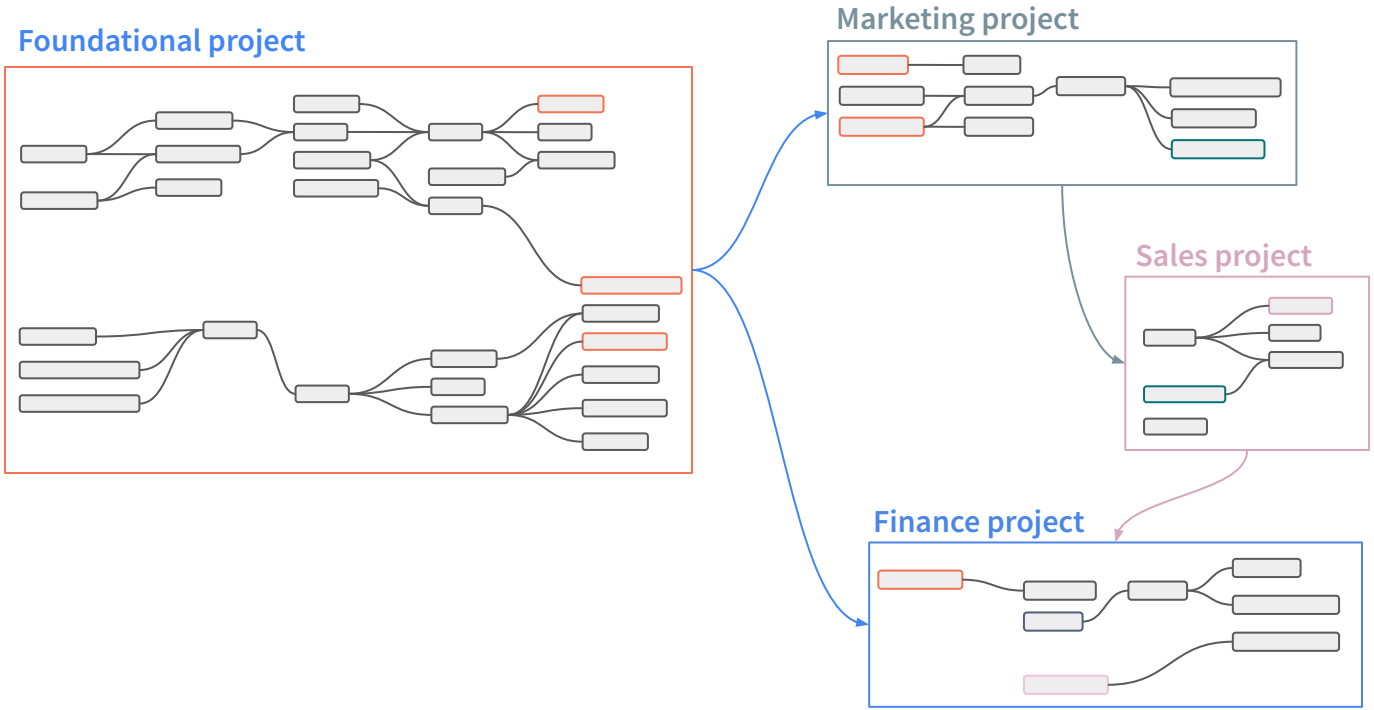
....que ha generado silos de metadatos



dbt Cloud es un plano de control de datos que centraliza estos metadatos y los hace procesables

dbt Mesh (2023)

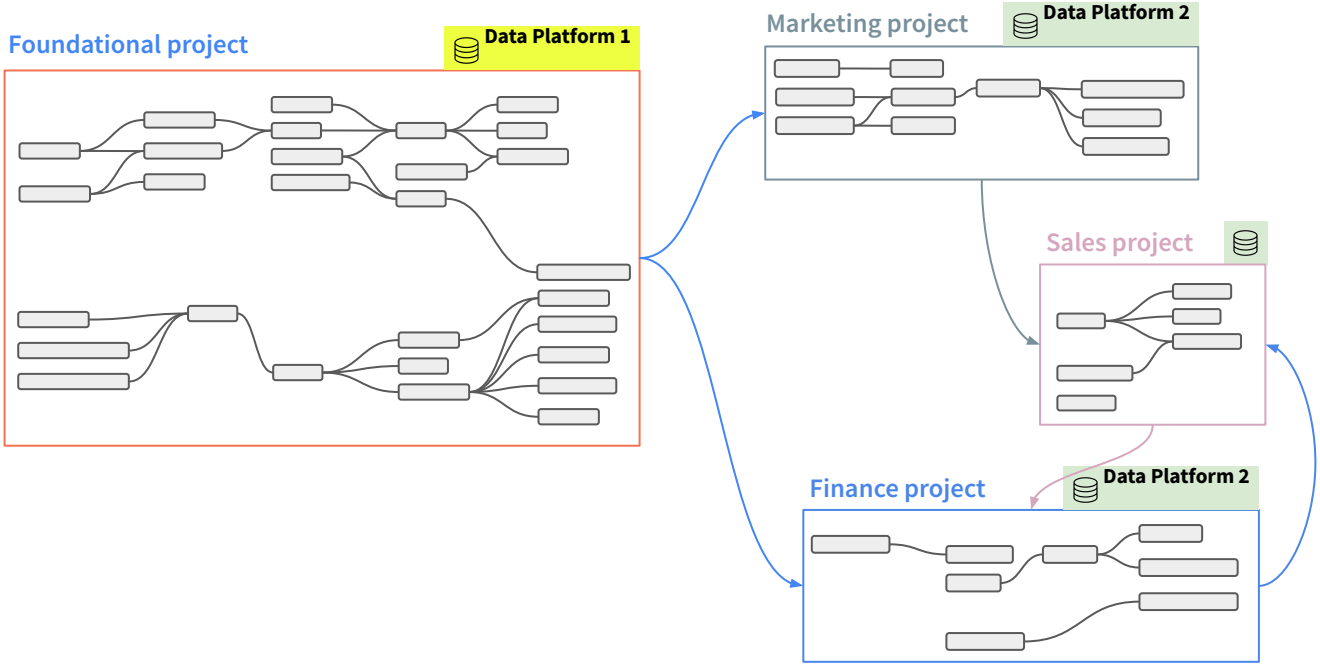
dbt Mesh permite la colaboración gobernada a través de múltiples proyectos dbt conectados



Cross-platform dbt Mesh (2024-25)

dbt Mesh permite la colaboración gobernada a través de la conexión de múltiples:

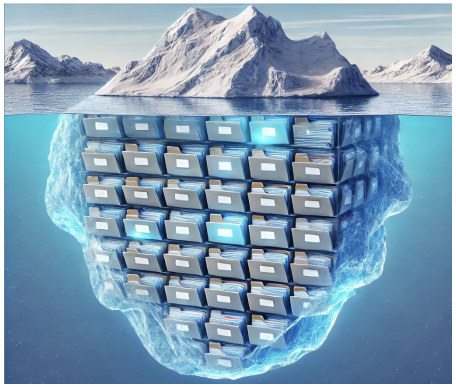
- Proyectos de dbt
- Plataformas de datos



Iceberg

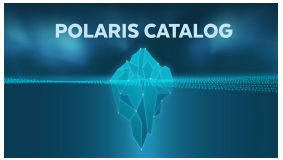
Soporte de calidad para un formato de datos portátil de calidad

Iceberg Table Format



ICEBERG

Iceberg Catalog



AWS Glue

Unity Catalog

Modelos incrementales escalables con microbatch

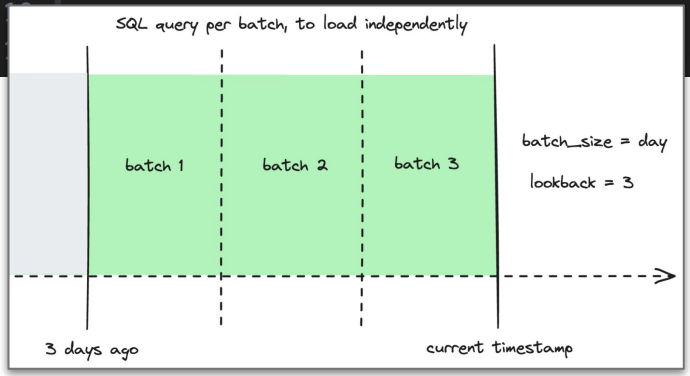
Procesamiento incremental por lotes para los sets de datos más grandes

- Durante las ejecuciones incrementales, dbt procesa nuevos lotes
  - Si la ejecución del modelo se interrumpe, una ejecución posterior continuará construyendo la tabla de destino a partir del lote en el que se interrumpió
  - Para cargar un lote específico, se definen límites superior e inferior en tiempo de ejecución

Beta disponible en dbt Cloud Latest (Versionless) o dbt Core v1.9

models > marts > my_microbatch_model.sql

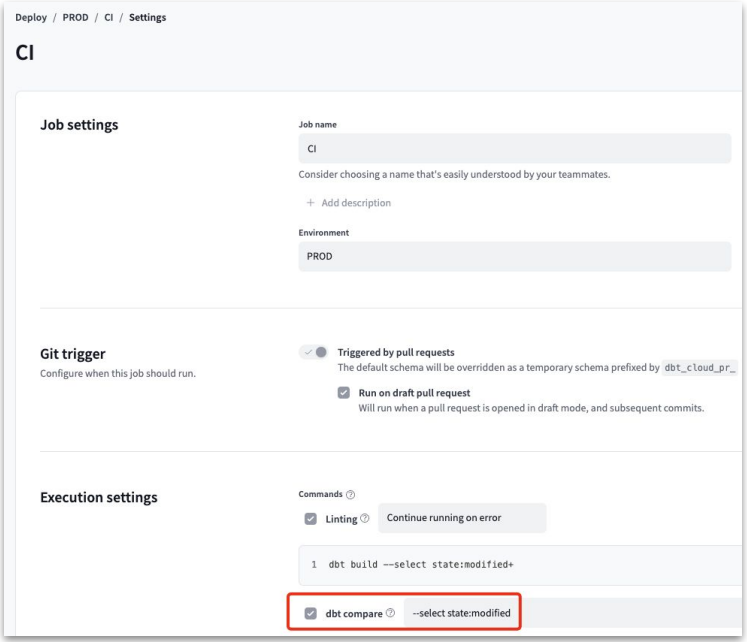
```
{{
    config(
        materialized='incremental',
        incremental_strategy='microbatch',
        event_time='date_day',
        batch_size='day',
        lookback_period='3'
    )
}}
```

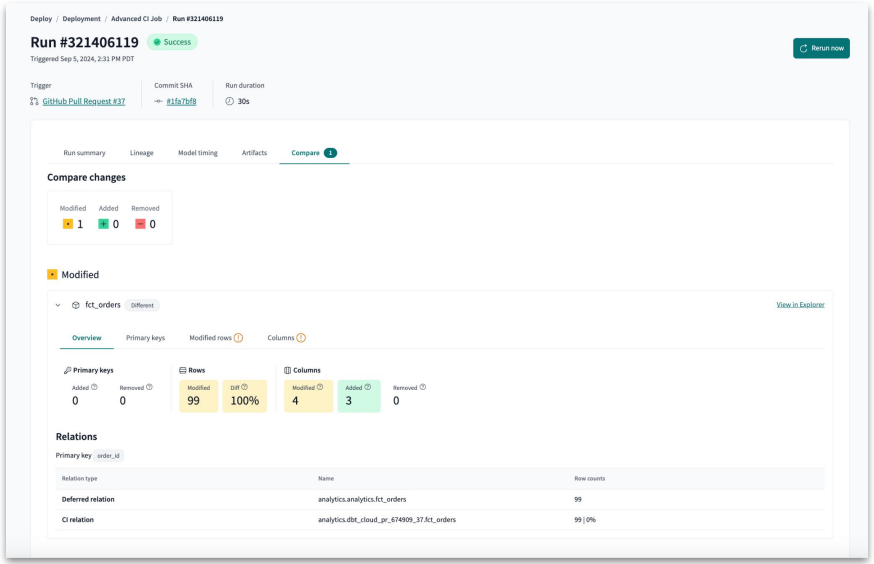


Validate: Testing & CI

Advanced CI checks: diffing

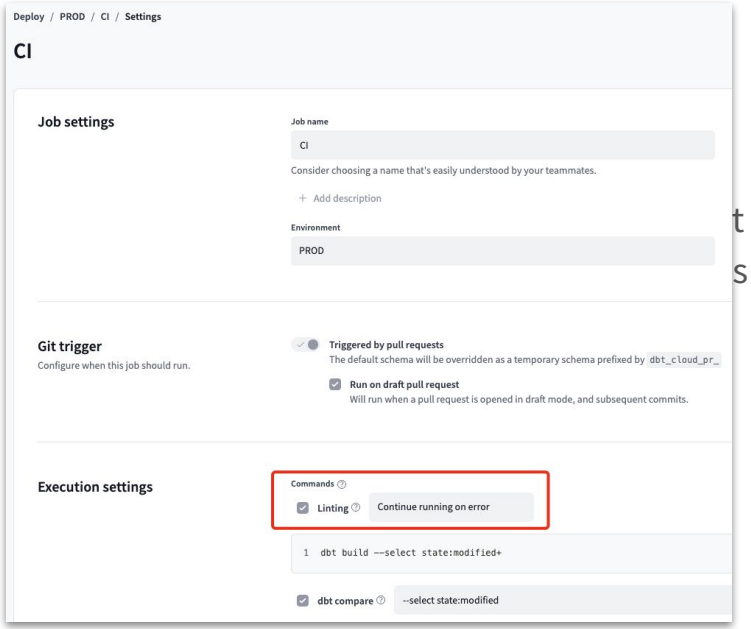
Comprende fácilmente el impacto previsto de un PR

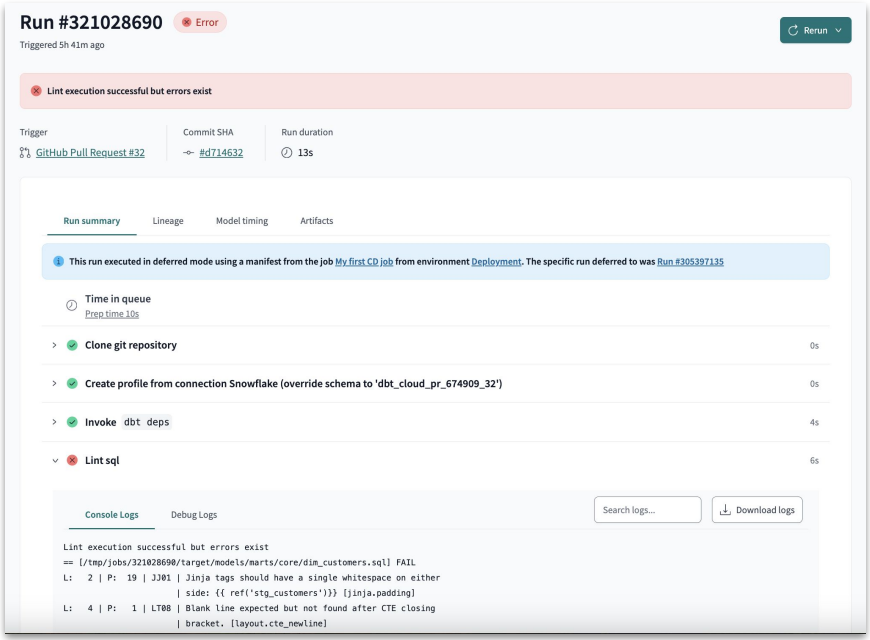




Validate: Testing & CI

Linting in CI





Unite a la comunidad

Slack

Unite en getdbt.com/community

Encontrá nuestros canales:
#local-argentina
#dbt-en-español

Speaker o meetups

Compartí tu interés en este formulario.
Mas información en getdbt.com/community acerca de como contribuir a la comunidad.

Want to learn more about dbt Cloud? dbt Labs will contact you.

Book a demo

You can expect a brief conversation with a dbt expert to assess what dbt Cloud can do to help your team followed by a live product demo with insight into how to get the most benefit from dbt Cloud.

Develop faster

Deliver quality analytics faster with automation, modularity, and repeatability.

Build data quality and trust

Guarantee the reliability of business metrics with automated testing, documentation, and transparency.

Unite your team

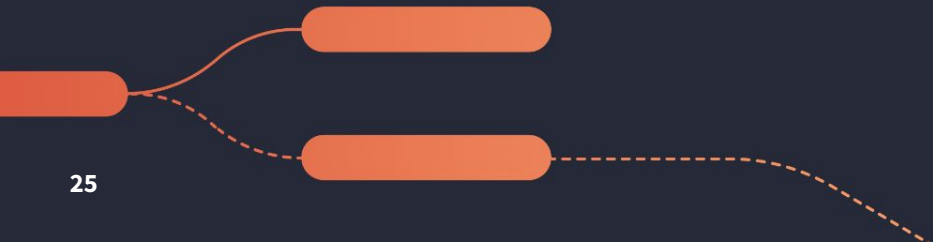
Eliminate bottlenecks and silos with Python or SQL-driven development, so analysts and data engineers can transform data collaboratively.

Today’s presenter #1

Gonzalo Villafañe Tapia
Data Engineer Tech Lead
Mutt Data

Optimizando modelos incrementales en dbt-Postgres

Con tablas de 500GB+



Caso de uso

Modelado de pipeline de datos de marketing digital

Con dbt+postgres obtenemos un stack con una curva de aprendizaje baja y posibilidad de escalar en el futuro

Caso de uso

Modelos incrementales

Al ingestar métricas solo queremos procesar las novedades

- **Macro** para reutilizar lógica
  - Subquery sobre *{{ this }}*
    - índice
  - COALESCE: si la tabla existe pero está vacía no insertaba nada

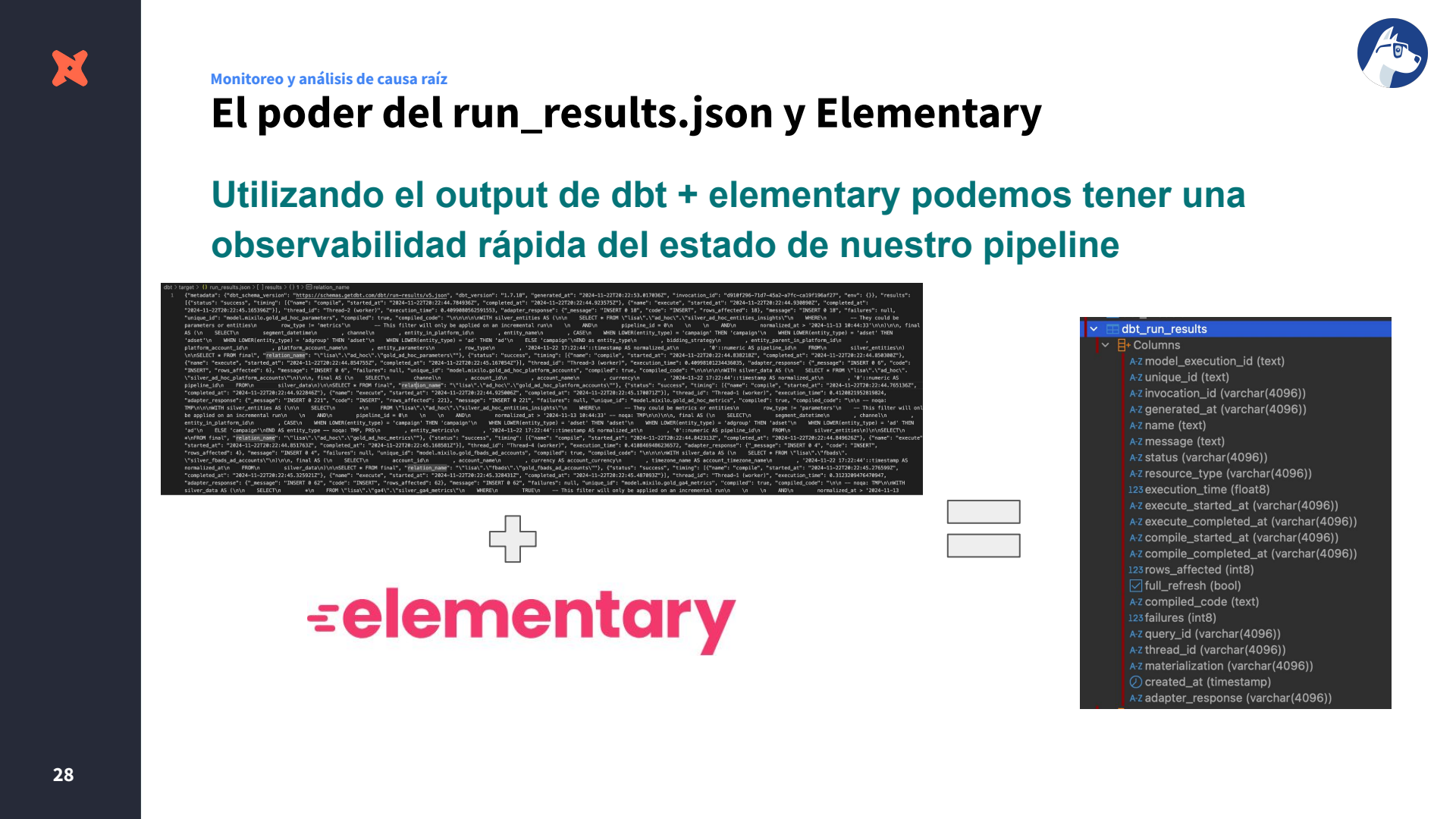
```
{% macro incremental_filter(filter_column, max_column=None, bronze=False, table_alias=None, check_pipeline_
{% if is_incremental() -%}
    -- This filter will only be applied on an incremental run
        {%if table_alias%}{{ table_alias }}.{%endif%}{{ filter_column }} > COALESCE(
            (
                SELECT
                    MAX({{ this }}.
                        {%- if not max_column -%}
                            {{ filter_column }}
                        {%- else -%}
                            {{ max_column }}
                        {%- endif -%}
                    )
                FROM
                    {{ this }}
            )
            , TO_TIMESTAMP({{ var('init_date') }})
        )
{%- endif %}
{%- endmacro %}
```

Filtro

Subquery

- Customizamos *is_incremental* para que funcione con materializaciones propias

```
{% macro is_incremental() %}
    {#- - do not run introspective queries in parsing #}
    {% if not execute %} {{ return(False) }}
    {% else %}
        {% set relation = adapter.get_relation(
            this.database, this.schema, this.table
        ) %}
        {{
            return(
                relation is not none
                and relation.type == "table"
                and model.config.materialized in ["incremental", "upsert_records"]
                and not should_full_refresh()
```



Monitoreo y análisis de causa raíz

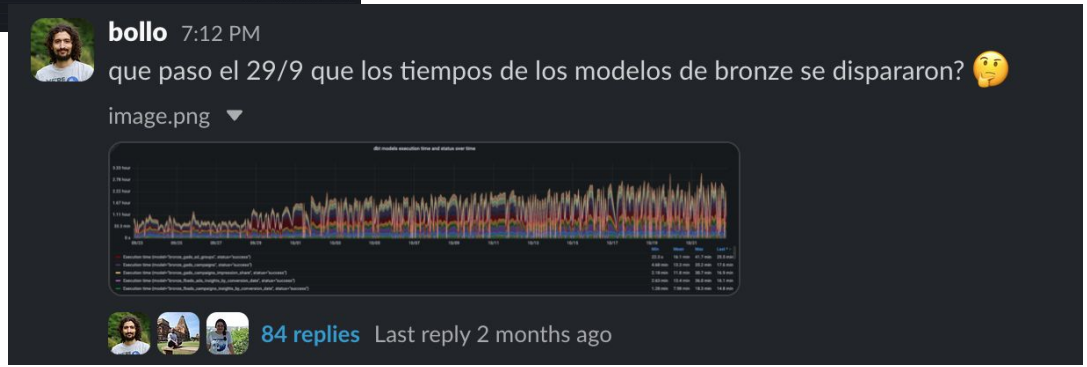
El poder del run_results.json y Elementary

Utilizando el output de dbt + elementary podemos tener una observabilidad rápida del estado de nuestro pipeline

Monitoreo y análisis de causa raíz

Monitoreo con Grafana

- Historia persisted => trazabilidad
- Filtros por fecha, modelo, test
- Anomaly detection + alarms [WIP]



Grafana

Detección del problema - síntomas

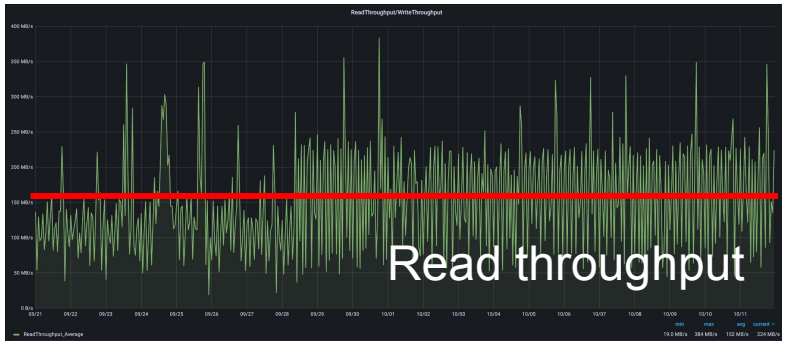
Ingestando una cantidad de rows casi constante e incluso procesando 0 rows el tiempo de ejecución sigue creciendo

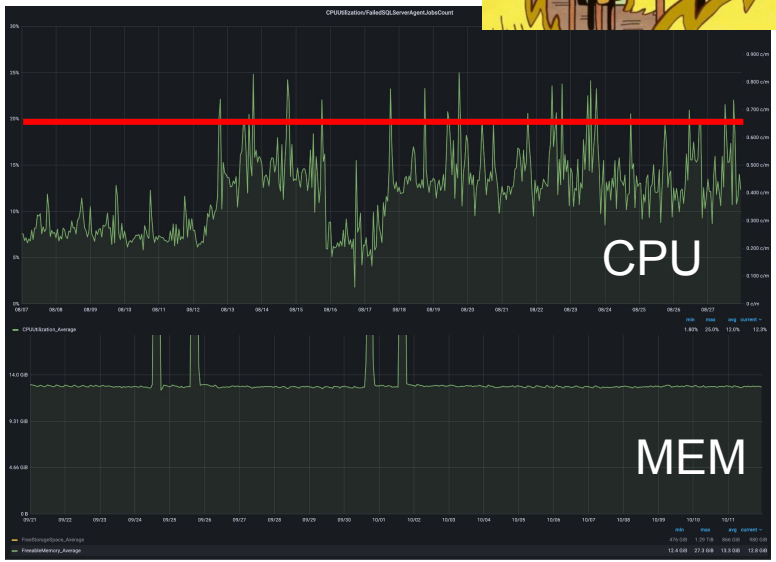
- Pipeline cada vez más lento ~3hs
- Tiempo de ejecución ~x2
- Cantidad de rows procesadas estable
- Que otra variable está cambiando? Volumen de la tabla

Detección del problema - síntomas

Ingestando una cantidad de rows casi constante e incluso procesando 0 rows el tiempo de ejecución sigue creciendo

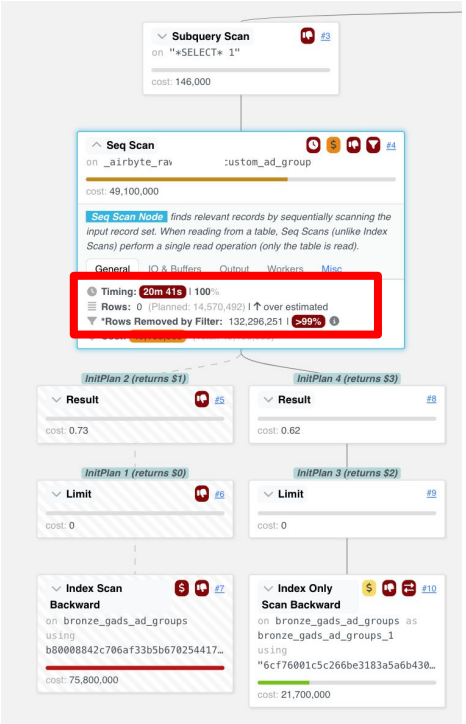
- Consumo de memoria estable
- Consumo de CPU más alto pero lejos de utilización plena
- Escalar verticalmente no soluciona nada
- Lectura a disco con picos altos





Explain + analyze: entendiendo el query planner de Postgres

Podemos visualizar el plan y encontrar cuellos de botella



Node Type	Count	Time	
Seq Scan	5	20m 41s	100%
Gather	1	10.4ms	0%
Bitmap Index Scan	2	5.95ms	0%
Bitmap Heap Scan	1	5.33ms	0%
Index Scan	1	4.79ms	0%
Result	15	4.74ms	0%
Index Scan Backward	7	4.68ms	0%
BitmapAnd	1	0.195ms	0%
Index Only Scan Backward	7	0.037ms	0%
Subquery Scan	7	0.008ms	0%
Append	1	0.007ms	0%
Limit	14	0.005ms	0%

- Tablas cada vez más grandes haciendo seq scan
- Pero si tengo índices, por qué no los usa?

Soluciones

Materializar (o no) CTEs

Los CTEs pueden ser nuestros mejores amigos o peores enemigos

- Tradeoff: query eficiente vs fácil de leer
- **C**ommon **T**able **E**xpressions
  - Simplifican el código
  - Lo hacen más prolijo y mantenible
  - Más fácil de debuggear y de hacer code review
- No siempre se llevan bien con el query planner
  - Push down de filtros
  - Reutilización de data procesada en CTEs
- Forzar materialización

```
with

events as (

    ...

),

{# CTE comments go here #}
filtered_events as (

    ...

)

select * from filtered_events
```

Soluciones

Materializar (o no) CTEs

```
WITH raw AS (
    SELECT * FROM _airbyte_raw  --_airbyte_raw tiene índice en
_airbyte_emitted_at
)
SELECT * FROM raw AS w1
  JOIN raw AS w2 ON w1.key = w2.ref  -- hay 2 referencias al CTE
raw, entonces por default lo materializa
WHERE {{ incremental_filter(...) }};
```

```
WITH raw AS NOT MATERIALIZED ( --forzamos a que no materialice
    SELECT * FROM _airbyte_raw
)
SELECT * FROM raw AS w1
  JOIN raw AS w2 ON w1.key = w2.ref
WHERE {{ incremental_filter(...) }};
```

Materializada por default, postgres no inyecta el filtro en el CTE

=> No filtra

NOT MATERIALIZED

Dos palabras que nos pueden bajar el tiempo de ejecución en 20x

Soluciones

Materializar (o no) CTEs

```
WITH raw AS (
    SELECT * FROM _airbyte_raw
    WHERE {{ incremental_filter(...) }}
)
SELECT * FROM raw AS w1;
```

Inyecto el filtro directamente en la CTE.

El filtro seguro, el índice depende

```
WITH
bookmark AS MATERIALIZED ( --forzamos a que materialice
    SELECT MAX(date) AS max_date FROM {{ this }}
),
raw AS (
    SELECT * FROM _airbyte_raw
    WHERE date > (SELECT max_date FROM bookmark)
)
SELECT * FROM raw AS w1;
```

Saco la subquery del bookmark a un CTE materializado.

Soluciones

Resolver el bookmark de antemano

dbt nos permite realizar queries auxiliares y dbt utils tiene una herramienta particularmente útil: get single value

```
{% macro incremental_filter(filter_column, max_column=None, bronze=False, ta
{% if is_incremental() -%}
    -- This filter will only be applied on an incremental run
        {%if table_alias%}{{ table_alias }}.{%endif%}{{ filter_column }} >
            (
                SELECT
                    MAX({{ this }}.
                        {%- if not max_column -%}
                            {{ filter_column }}
                        {%- else -%}
                            {{ max_column }}
                        {%- endif -%}
                    )
                FROM
                    {{ this }}
```

- **dbt_utils**.get_single_value ejecuta la query antes y nos devuelve el valor
- Al recibir el timestamp hardcodeado Postgres usa el índice (el 99% de las veces)
- Tradeoff: resuelve el bookmark cada vez

```
{% macro incremental_filter(filter_column, bookmark_column="normalized_at", bronze=False, table_alias=None,
{% if is_incremental() -%}
    {% set bookmark_date_statement %}
        SELECT
            COALESCE(MAX({{ this }}.{{ bookmark_column }}), '1987-07-29'::date) AS max_date
        FROM {{ this }}
    {% endset %}

    {%- set bookmark_date = dbt_utils.get_single_value(bookmark_date_statement) -%}

    AND
        {%if table_alias%}{{ table_alias }}.{%endif%}{{ filter_column }} > '{{ bookmark_date }}'
{%- endif %}
{%- endmacro %}
```

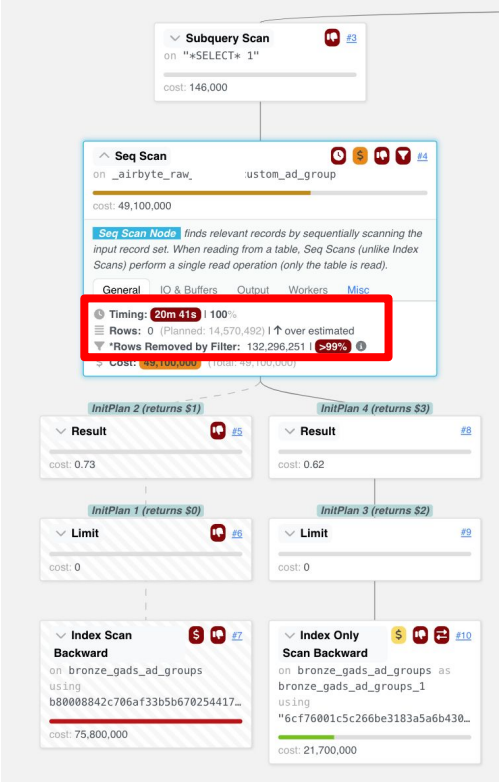
Resuelvo el bookmark

Lo inyecto *hardcodeado*

Soluciones

Explain antes y después

De 0 rows en 1200s segundos a 230k en 7s



Node Type	Count	Time	
Index Scan	5	8s 874ms	54%
Result	1	4s 555ms	28%
Seq Scan	4	3s 6.99ms	18%
Subquery Scan	9	32.9ms	0%
Append	1	0ms	0%
Gather	1	0ms	0%

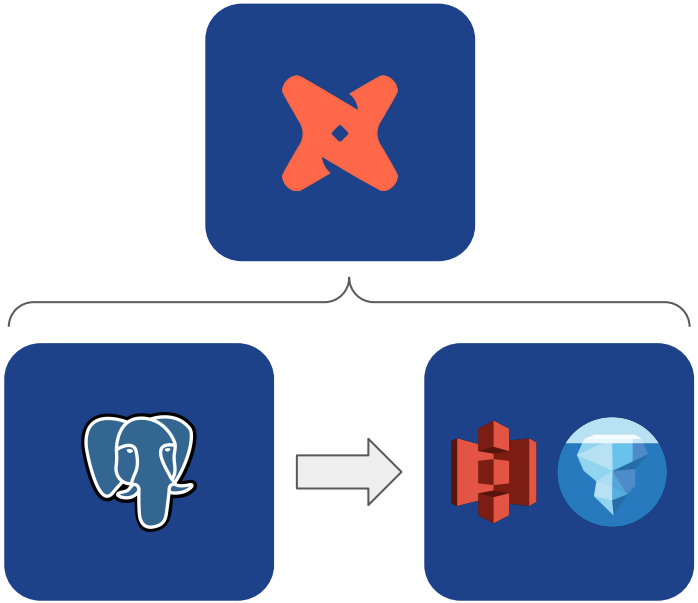
Otros aprendizajes

- Dentro de un CTAS no paraleliza
- Estadísticas actualizadas
  - Autovacuum
  - Planner basa sus decisiones en estimaciones

Siguientes pasos

Migrar a dbt-athena para mayor escalabilidad y costo-eficiencia

1. Migrar capas de mayor volumen a S3
    a. aws_s3 extension + pg_repack
    b. Metadata en Glue
    c. Iceberg
2. Mover ingesta de Postgres a S3
3. Migrar modelado de dbt-postgres a dbt-athena (o dbt-trino?)



Moraleja

El poder de dbt está en su simpleza

- SQL (casi) puro
- Capa fina sobre nuestras queries, complejidad delegada al engine
- Fácil integración con otras herramientas del ecosistema: Airflow, Airbyte, etc.
- Apoyarse en las herramientas a disposición (dbt utils y muchas más de la comunidad)
- El diablo está en los detalles
    - Monitoreo y observabilidad es clave
    - Conocer bien el engine sobre el que trabajamos
    - Tradeoffs, tradeoffs, tradeoffs
- Todo esto era una charla encubierta sobre Postgres? Mas o menos

Gracias

¿Preguntas?

Today’s presenter #2

Lucas Trubiano

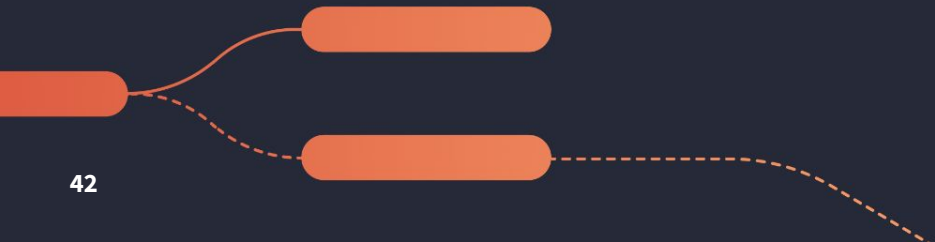
CoE Data Engineering

ITTI

🍇 Mza | 🇦🇷 Arg

Materializaciones

Lo que está bajo el capó de dbt



Cómo superamos las 2 crisis más grandes en la historia de dbt en ITTI

*Esta presentación está basada en **hechos reales.***
*Algunos nombres y eventos han sido **modificados** con fines **~~dramáticos~~ didácticos***

No toques nada,
No toques nada

El CoE saldando la deuda técnica

¿Qué son las materializaciones?

¿Qué son las materializaciones?

Es la forma de persistir la información de un modelo.

Un modelo es una transformación de datos con SQL.

Hay 5 tipos: vista, tabla, incremental, efímeras y vistas materializadas.

ITTI pasaba por un crecimiento enorme. Fusión con otro banco. Crecimiento de 1900%

Montar un Data Lake en 6 meses y crear todos los modelos core de +5 empresas.

Nos movíamos tan rápido que el único estándar era que los modelos tenían que funcionar.


POR FAVOR ALGUIEN QUIERE
PENSAR EN LOS MODELOS !?!
imgflip.com

Crisis 1: Las dim se borraban y recreaban todos los días.

¿Qué causaba esto?
Se borraban:

- Tags de Lake Formation
- Controles de calidad y Metadata de Glue

¿Por qué pasaba?
Usaban materialización tipo tabla.

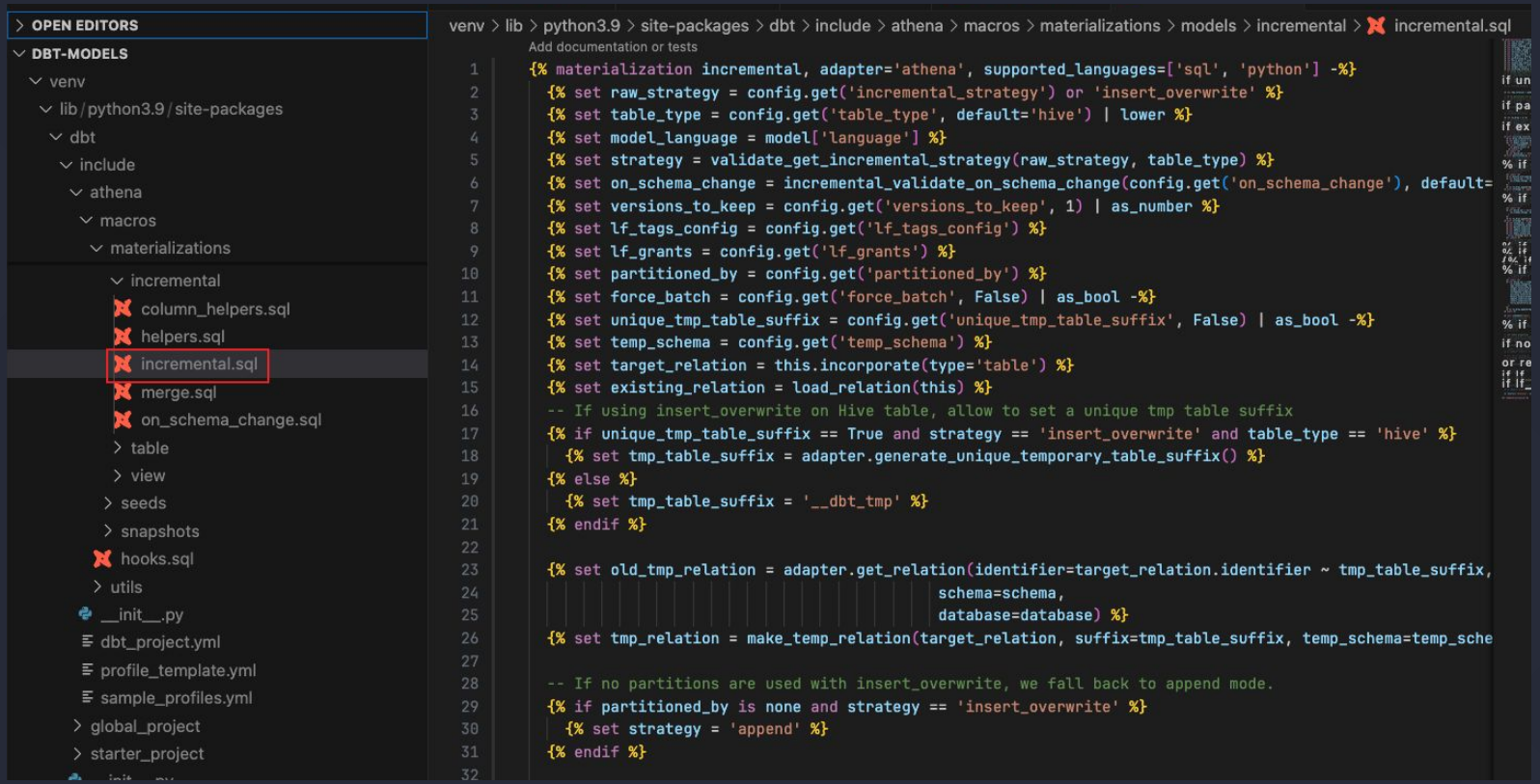
Posible solución:
Armar un script que copiara las propiedades que perdían las tablas cuando se borraban.

Crisis 1: Las dim se borraban y recreaban todos los días.

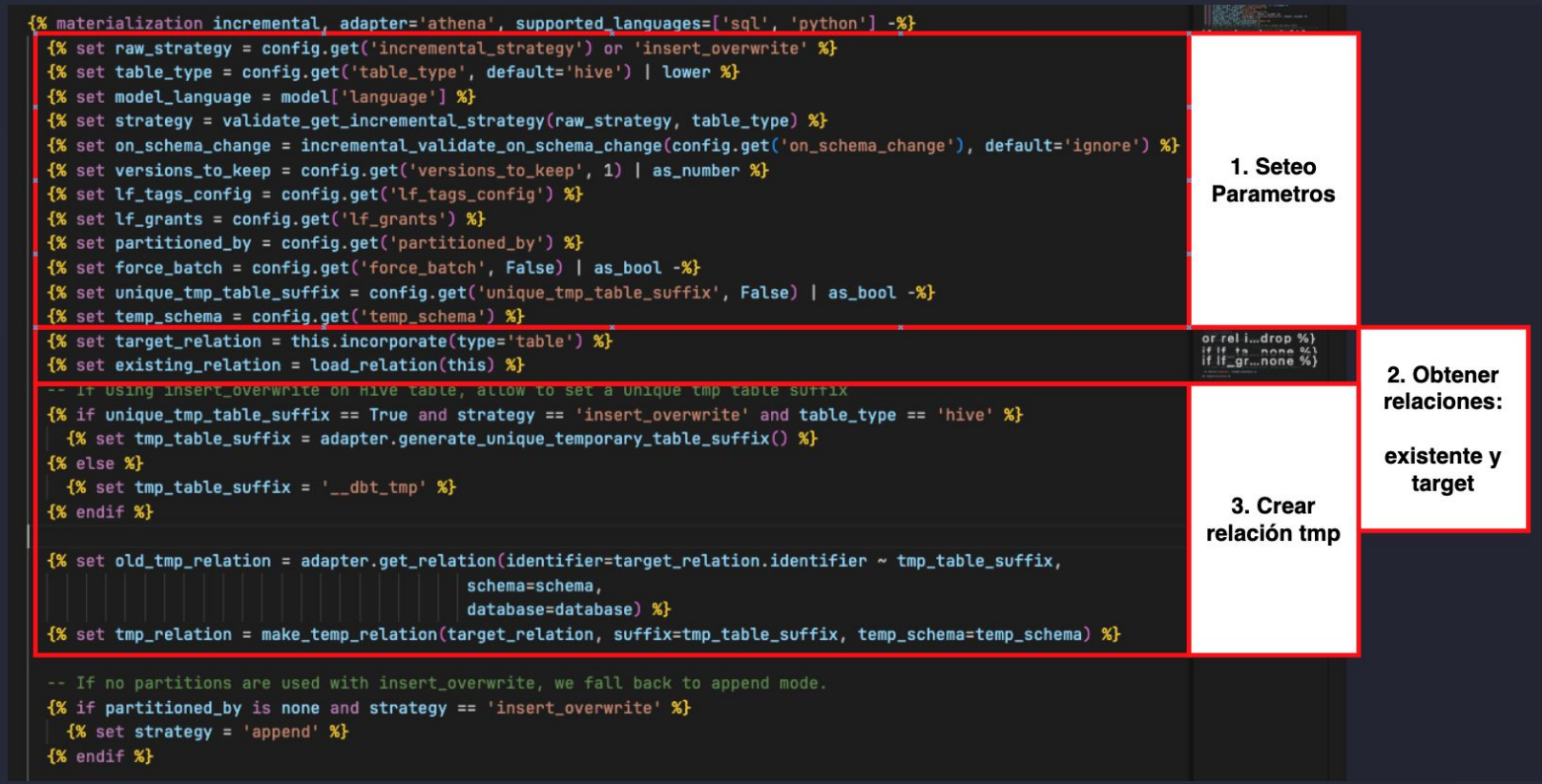
La Posible Solución

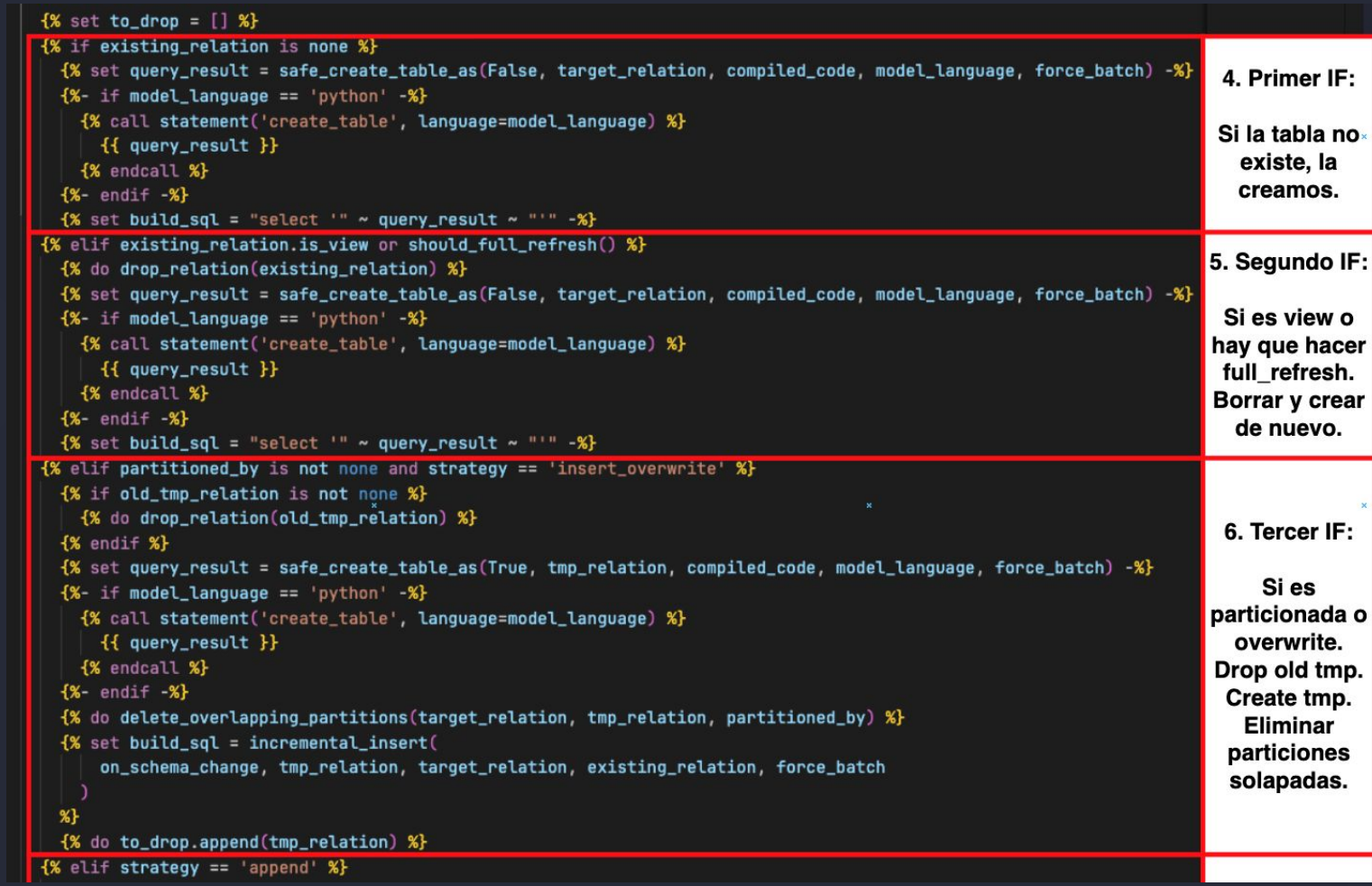
Una solución más CoE:
Cambiar la materialización de TODAS las tablas dim.

¿Cómo se ve una materialización?



¿Qué hace una materialización?




{% set to_drop = [] %}
{% if existing_relation is none %}
{% set query_result = safe_create_table_as(False, target_relation, compiled_code, model_language, force_batch) -%}
{%- if model_language == 'python' -%}
{% call statement('create_table', language=model_language) %}
{{ query_result }}
{% endcall %}
{%- endif -%}
{% set build_sql = "select '" ~ query_result ~ "'" -%}
{% elif existing_relation.is_view or should_full_refresh() %}
{% do drop_relation(existing_relation) %}
{% set query_result = safe_create_table_as(False, target_relation, compiled_code, model_language, force_batch) -%}
{%- if model_language == 'python' -%}
{% call statement('create_table', language=model_language) %}
{{ query_result }}
{% endcall %}
{%- endif -%}
{% set build_sql = "select '" ~ query_result ~ "'" -%}
{% elif partitioned_by is not none and strategy == 'insert_overwrite' %}
{% if old_tmp_relation is not none %}
{% do drop_relation(old_tmp_relation) %}
{% endif %}
{% set query_result = safe_create_table_as(True, tmp_relation, compiled_code, model_language, force_batch) -%}
{%- if model_language == 'python' -%}
{% call statement('create_table', language=model_language) %}
{{ query_result }}
{% endcall %}
{%- endif -%}
{% do delete_overlapping_partitions(target_relation, tmp_relation, partitioned_by) %}
{% set build_sql = incremental_insert(
on_schema_change, tmp_relation, target_relation, existing_relation, force_batch
)
%}
{% do to_drop.append(tmp_relation) %}
{% elif strategy == 'append' %}
4. Primer IF:
Si la tabla no existe, la creamos.
5. Segundo IF:
Si es view o hay que hacer full_refresh. Borrar y crear de nuevo.
6. Tercer IF:
Si es particionada o overwrite. Drop old tmp. Create tmp. Eliminar particiones solapadas.

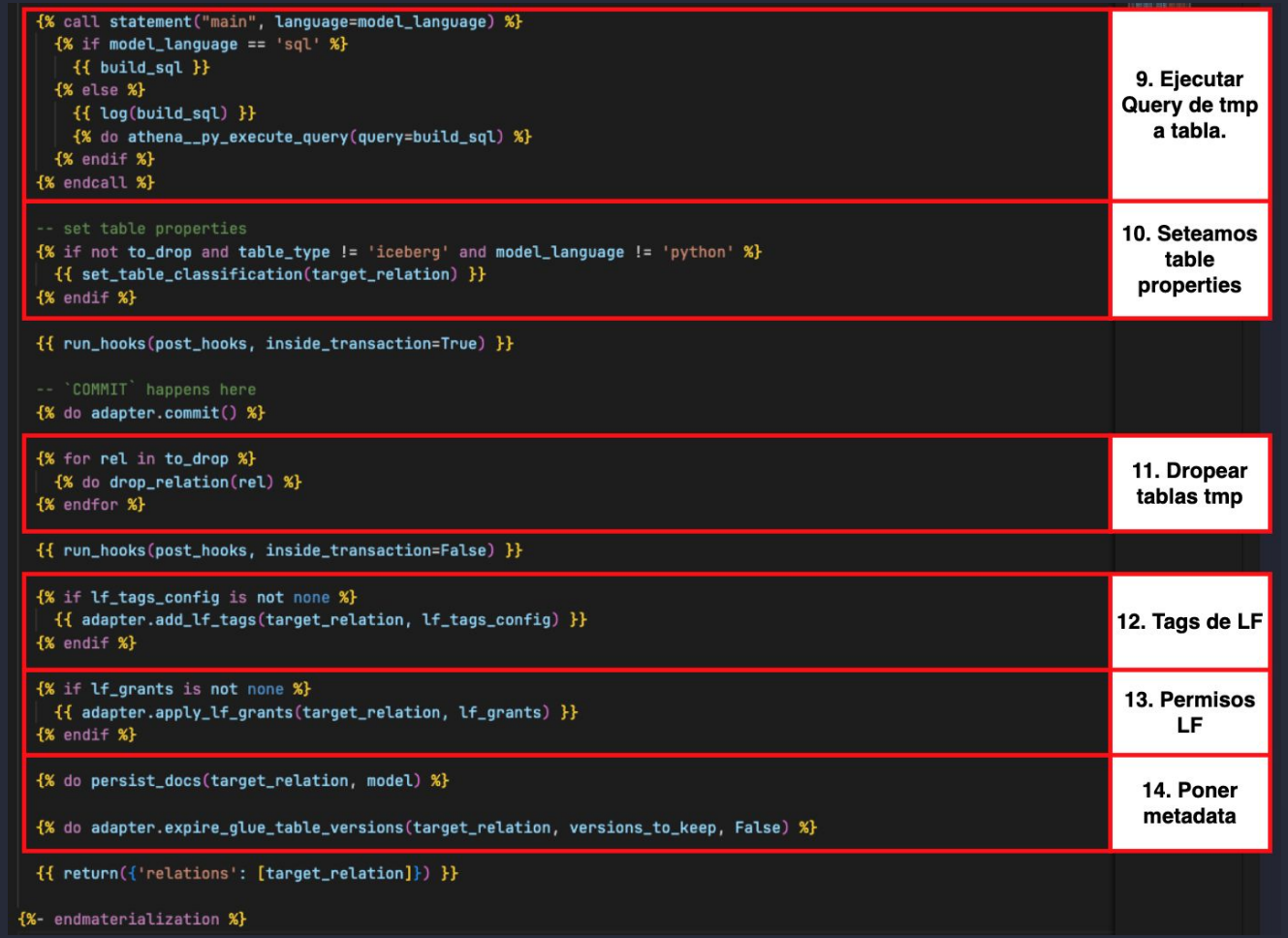
```
  {% do to_drop.append(tmp_relation) %}
{% elif strategy == 'append' %}
  {% if old_tmp_relation is not none %}
    {% do drop_relation(old_tmp_relation) %}
  {% endif %}
  {% set query_result = safe_create_table_as(True, tmp_relation, compiled_code, model_language, force_batch) -%}
  {%- if model_language == 'python' -%}
    {% call statement('create_table', language=model_language) %}
      {{ query_result }}
    {% endcall %}
  {%- endif -%}
  {% set build_sql = incremental_insert(
      on_schema_change, tmp_relation, target_relation, existing_relation, force_batch
    )
  %}
  {% do to_drop.append(tmp_relation) %}
{% elif strategy == 'merge' and table_type == 'iceberg' %}
  {% set unique_key = config.get('unique_key') %}
  {% set incremental_predicates = config.get('incremental_predicates') %}
  {% set delete_condition = config.get('delete_condition') %}
  {% set update_condition = config.get('update_condition') %}
  {% set insert_condition = config.get('insert_condition') %}
  {% set empty_unique_key -%}
    Merge strategy must implement unique_key as a single column or a list of columns.
  {%- endset %}
  {% if unique_key is none %}
    {% do exceptions.raise_compiler_error(empty_unique_key) %}
  {% endif %}
  {% if incremental_predicates is not none %}
    {% set inc_predicates_not_list -%}
      Merge strategy must implement incremental_predicates as a list of predicates.
    {%- endset %}
    {% if not adapter.is_list(incremental_predicates) %}
      {% do exceptions.raise_compiler_error(inc_predicates_not_list) %}
    {% endif %}
  {% endif %}
  {% if old_tmp_relation is not none %}
    {% do drop_relation(old_tmp_relation) %}
  {% endif %}
```

7. Cuarto IF:

Si es append. Drop old tmp. Create tmp. Build incremental query.

8. Quinto IF:

Si es iceberg. Drop old tmp. Create tmp. Build merge query.

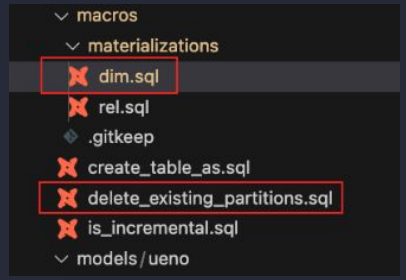

{% call statement("main", language=model_language) %}
{% if model_language == 'sql' %}
{{ build_sql }}
{% else %}
{{ log(build_sql) }}
{% do athena__py_execute_query(query=build_sql) %}
{% endif %}
{% endcall %}
9. Ejecutar Query de tmp a tabla.
-- set table properties
{% if not to_drop and table_type != 'iceberg' and model_language != 'python' %}
{{ set_table_classification(target_relation) }}
{% endif %}
10. Seteamos table properties
{{ run_hooks(post_hooks, inside_transaction=True) }}
-- `COMMIT` happens here
{% do adapter.commit() %}
{% for rel in to_drop %}
{% do drop_relation(rel) %}
{% endfor %}
11. Dropear tablas tmp
{{ run_hooks(post_hooks, inside_transaction=False) }}
{% if lf_tags_config is not none %}
{{ adapter.add_lf_tags(target_relation, lf_tags_config) }}
{% endif %}
12. Tags de LF
{% if lf_grants is not none %}
{{ adapter.apply_lf_grants(target_relation, lf_grants) }}
{% endif %}
13. Permisos LF
{% do persist_docs(target_relation, model) %}
{% do adapter.expire_glue_table_versions(target_relation, versions_to_keep, False) %}
14. Poner metadata
{{ return({'relations': [target_relation]}) }}
{%- endmaterialization %}

Creando materializaciones dim

Table ➡ ✅ **No duplica**
❌ **Se dropean**

Incremental ➡ ✅ **No se dropean**
❌ **Duplica en cada partición**
📔 Si la materialización borra las particiones que se superponen. (Comportamiento insert_overwrite)
💡 No podemos borrar todo y dejar la nueva.

Creando materializaciones dim



dbt_project > dbt_ueno > macros > materializations > dim.sql

```
1-  materialization incremental, adapter='athena', supported_languages=['sql', 'python']
```

```
1+  materialization dim, adapter='athena', supported_languages=['sql', 'python'] -%}
```

```
60 +  {% set query_result = safe_create_table_as(True, tmp_relation, compiled_code, model
61    {%- if model_language == 'python' -%}
62      {% call statement('create_table', language=model_language) %}
63        {{ query_result }}
64      {% endcall %}
65    {%- endif -%}
66 -  {% do delete_overlapping_partitions(target_relation, tmp_relation, partitioned_by)
67    {% set build_sql = incremental_insert(
68        on_schema_change, tmp_relation, target_relation, existing_relation, force_batch
69      )
70    %}
```

```
60    {% set query_result = safe_create_table_as(True, tmp_relation, compiled_code, model_lang
61    {%- if model_language == 'python' -%}
62      {% call statement('create_table', language=model_language) %}
63        {{ query_result }}
64      {% endcall %}
65    {%- endif -%}
66 +  {% do delete_existing_partitions(target_relation, partitioned_by) %}
67    {% set build_sql = incremental_insert(
68        on_schema_change, tmp_relation, target_relation, existing_relation, force_batch
69      )
70    %}
```

Creando materializaciones dim

dbt_project > dbt_ueno > macros > delete_existing_partitions.sql

```
{% macro delete_overlapping_partitions(target_relation, tmp_relation, partitioned_by) %}
  {%- set partitioned_keys = partitioned_by | tojson | replace('\"', '') | replace('[', '') | replace(']', '') -%
  {% call statement('get_partitions', fetch_result=True) %}
    select distinct {{partitioned_keys}} from {{ tmp_relation }};
  {% endcall %}
  {%- set table = load_result('get_partitions').table -%}
  {%- set rows = table.rows -%}
  {%- set partitions = [] -%}
  {%- for row in rows -%}
    {%- set single_partition = [] -%}
    {%- for col in row -%}
      {%- set column_type = adapter.convert_type(table, loop.index0) -%}
      {%- if column_type == 'integer' or column_type is none -%}
        {%- set value = col|string -%}
      {%- elif column_type == 'string' -%}
        {%- set value = "'" + col + "'" -%}
      {%- elif column_type == 'date' -%}
        {%- set value = "'" + col|string + "'" -%}
      {%- elif column_type == 'timestamp' -%}
        {%- set value = "'" + col|string + "'" -%}
      {%- else -%}
        {%- do exceptions.raise_compiler_error('Need to add support for column type ' + column_type) -%}
      {%- endif -%}
      {%- do single_partition.append(partitioned_by[loop.index0] + '=' + value) -%}
    {%- endfor -%}
    {%- set single_partition_expression = single_partition | join(' and ') -%}
    {%- do partitions.append('(' + single_partition_expression + ')') -%}
  {%- endfor -%}
  {%- for i in range(partitions | length) %}

    {%- do adapter.clean_up_partitions(target_relation, partitions[i]) -%}
  {%- endfor -%}
{%- endmacro %}
```

```
{% macro delete_existing_partitions(target_relation, partitioned_by) %}
  {%- set partitioned_keys = partitioned_by | tojson | replace('\"', '') | replace('[', '') | replace(']', '') -%}
  {% call statement('get_partitions', fetch_result=True) %}
    select distinct {{partitioned_keys}} from {{ target_relation }};
  {% endcall %}
  {%- set table = load_result('get_partitions').table -%}
  {%- set rows = table.rows -%}
  {%- set partitions = [] -%}      Rubén Jara, 5 months ago • Agrego files utilizados para materializacion
  {%- for row in rows -%}
    {%- set single_partition = [] -%}
    {%- for col in row -%}
      {%- set column_type = adapter.convert_type(table, loop.index0) -%}
      {%- if column_type == 'integer' -%}
        {%- set value = col|string -%}
      {%- elif column_type == 'string' -%}
        {%- set value = "'" + col + "'" -%}
      {%- elif column_type == 'date' -%}

        {%- set value = "'" + col|string + "'" -%}
      {%- else -%}
        {%- do exceptions.raise_compiler_error('Need to add support for column type ' + column_type) -%}
      {%- endif -%}
      {%- do single_partition.append(partitioned_by[loop.index0] + '=' + value) -%}
    {%- endfor -%}
    {%- set single_partition_expression = single_partition | join(' and ') -%}
    {%- do partitions.append('(' + single_partition_expression + ')') -%}
  {%- endfor -%}
  {%- for i in range(partitions | length) %}
    {{- log("Cleaning up partition: " ~ partitions[i], info=True) -}}
    {%- do adapter.clean_up_partitions(target_relation, partitions[i]) -%}
  {%- endfor -%}
{%- endmacro %}
```

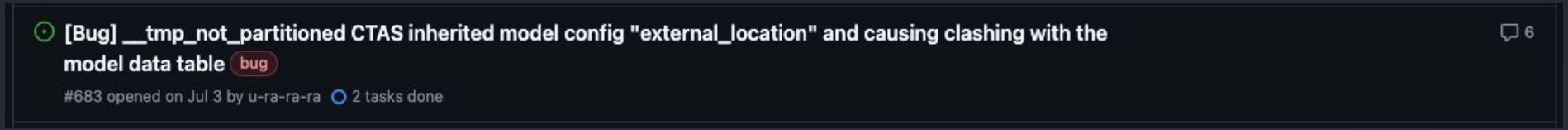
Crisis 2: Las fact no se podían reprocesar de forma histórica.

¿Qué causaba esto?

- Necesidad de reprocesar con un script manual.
- Posibilidad de corte del proceso.
- Tiempos largos para tenerla activa nuevamente.
- Complejidad de las queries.

¿Por qué pasaba?

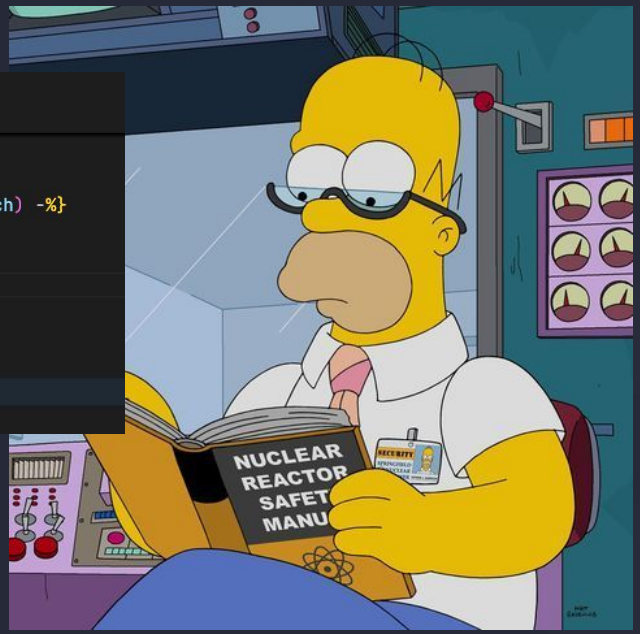
Limitaciones de Athena a 100 particiones y un Bug en las macros de dbt-athena.



Crisis 2: Las fact no se podían reprocesar de forma histórica.

Diagnóstico → Materialización

```
{% materialization incremental, adapter='athena', supported_languages=['sql', 'python'] -%}
  {% if existing_relation is none %}
  {% elif existing_relation.is_view or should_full_refresh() %}
    {% do drop_relation(existing_relation) %}
    {% set query_result = safe_create_table_as(False, target_relation, compiled_code, model_language, force_batch) -%}
    {%- if model_language == 'python' -%}
      {% call statement('create_table', language=model_language) %}
        {{ query_result }}
      {% endcall %}
    {%- endif -%}
    {% set build_sql = "select '" ~ query_result ~ "'" -%}
  {% elif partitioned_by is not none and strategy == 'insert_overwrite' %}…
  {% elif strategy == 'append' %}
```



Crisis 2: Las fact no se podían reprocesar de forma histórica.

Diagnóstico → Materialización → Safe Create Table As

```
{% macro safe_create_table_as(temporary, relation, compiled_code, language='sql', force_batch=False) -%}
    {%- if language != 'sql' -%}
        {{ return(create_table_as(temporary, relation, compiled_code, language)) }}
    {%- elif force_batch -%}
      {%- do create_table_as_with_partitions(temporary, relation, compiled_code, language) -%}
      {%- set query_result = relation ~ ' with many partitions created' -%}
    {%- else -%}
        {%- if temporary -%}
          {%- do run_query(create_table_as(temporary, relation, compiled_code, language, true)) -%}
          {%- set compiled_code_result = relation ~ ' as temporary relation without partitioning created' -%}
        {%- else -%}
          {%- set compiled_code_result = adapter.run_query_with_partitions_limit_catching(create_table_as(temporary, relation, compiled_code)) -%}
          {%- do log('COMPILED CODE RESULT: ' ~ compiled_code_result) -%}
          {%- if compiled_code_result == 'TOO_MANY_OPEN_PARTITIONS' -%}
            {%- do create_table_as_with_partitions(temporary, relation, compiled_code, language) -%}
            {%- set compiled_code_result = relation ~ ' with many partitions created' -%}
          {%- endif -%}
        {%- endif -%}
    {%- endif -%}
    {{ return(compiled_code_result) }}
{%- endmacro %}
```

Diagnóstico → Materialización → Safe Create Table As → Create Table As With Partitions → Create Table As → Athena Create Table As → Generate S3 Location

Add documentation or tests

```
{% macro create_table_as(temporary, relation, compiled_code, language='sql', skip_partitioning=false) -%}
  {{ adapter.dispatch('create_table_as', 'athena')(temporary, relation, compiled_code, language, skip_partitioning) }}
{%- endmacro %}


{% macro athena__create_table_as(temporary, relation, compiled_code, language='sql', skip_partitioning=false) -%}
  {%- set materialized = config.get('materialized', default='table') -%}
  {%- set external_location = config.get('external_location', default=none) -%}
  {%- do log("Skip partitioning: " ~ skip_partitioning) -%}
  {%- set partitioned_by = config.get('partitioned_by', default=none) if not skip_partitioning else none -%}
  {%- set bucketed_by = config.get('bucketed_by', default=none) -%}
  {%- set bucket_count = config.get('bucket_count', default=none) -%}
  {%- set field_delimiter = config.get('field_delimiter', default=none) -%}
  {%- set table_type = config.get('table_type', default='hive') | lower -%}
  {%- set format = config.get('format', default='parquet') -%}
  {%- set write_compression = config.get('write_compression', default=none) -%}
  {%- set s3_data_dir = config.get('s3_data_dir', default=target.s3_data_dir) -%}
  {%- set s3_data_naming = config.get('s3_data_naming', default=target.s3_data_naming) -%}
  {%- set s3_tmp_table_dir = config.get('s3_tmp_table_dir', default=target.s3_tmp_table_dir) -%}
  {%- set extra_table_properties = config.get('table_properties', default=none) -%}

  {%- set location_property = 'external_location' -%}
  {%- set partition_property = 'partitioned_by' -%}
  {%- set work_group_output_location_enforced = adapter.is_work_group_output_location_enforced() -%}
  {%- set location = adapter.generate_s3_location(relation,
                                                  s3_data_dir,
                                                  s3_data_naming,
                                                  s3_tmp_table_dir,
                                                  external_location,
                                                  temporary,
                                                  ) -%}
  {%- set native_drop = config.get('native_drop', default=false) -%}

  {%- set contract_config = config.get('contract') -%}
  {%- if contract_config.enforced -%}
    {{ get_assert_columns_equivalent(compiled_code) }}
  {%- endif -%}
```

- **SÉ QUE ESTÁS AHÍ… SOLUCIÓN AL BUG DE DBT.**
- **Y TE VOY A ENCONTRAR**

Crisis 2: Las fact no se podían reprocesar de forma histórica.

Solución:
Especificar True en la tabla temporal.

```
{%- if tmp_relation is not none -%}
  {%- do drop_relation(tmp_relation) -%}
{%- endif -%}

{%- do log('CREATE NON-PARTIONED STAGING TABLE: ' ~ tmp_relation) -%}
{%- do run_query(create_table_as(temporary, tmp_relation, compiled_code, language, true)) -%}

{% set partitions_batches = get_partition_batches(sql=tmp_rel
{% do log('BATCHES TO PROCESS: ' ~ partitions_batches | lengt
```

```
159
160   {%- if tmp_relation is not none -%}
161     {%- do drop_relation(tmp_relation) -%}
162   {%- endif -%}
163
164+  {%- do log('CREATE NON-PARTIONED STAGING TEMP TABLE: ' ~ tmp_relation) -%}      Lucas Trubiano, 3 months
165+  {%- do run_query(create_table_as(true, tmp_relation, compiled_code, language, true)) -%}
166
```

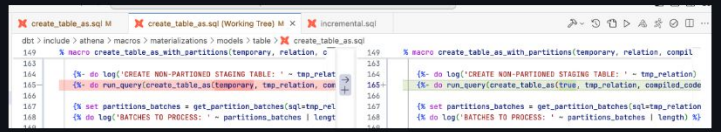
```
n_batches(sql=tmp_relation, as_subquery=False) %}
tions_batches | length) %}
```

lucastrubiano commented on Aug 26 • edited

At my work, I experienced the same issue, and I found that one solution is to set temporary to true on the following line in the file:

`dbt-athena/dbt/include/athena/macros/materializations/models/table/create_table_as.sql`

Line 165: change temporary to true:
`{%- do run_query(create_table_as(true, tmp_relation, compiled_code, language, true)) -%}`



I hope this helps!

nicor88 commented on Aug 27 — Contributor

@lucastrubiano thanks for spotting this issue. Do you mind to raise a PR with your fix? Thanks

Y así fue como sobrevivimos a la deuda técnica

¿Qué aprendimos y para que nos sirvió?

- ✅ Al ser Open Source nos permite manipular todo el código.
- ✅ Comandos de **dbt run**
  - **-d Debug**
  - **-f Full Refresh**
- ✅ Podemos agregar logs con **{{ log() }}**
- ✅ Podemos extender las funcionalidades y hacer cosas inimaginables.
- ✅ Abstraemos la complejidad.
- ✅ Estandarizamos procesos y formas de trabajar.
- ✅ Procesos automáticos y robustos.

/in/lucastrubiano/

Gracias

¿Preguntas?

/in/lucastrubiano/

Antes de irte...

Compartí tu opinión con nosotros