

**Trasforma i dati nel tuo data
warehouse con il framework DBT
e Fabric**

DATA
SATURDAYS



Sponsors



CONSORZIO
UNIVERSITARIO
DI PORDENONE
MOLTIPLICATORE DI VALORE



altitudo

 **beanTech**
IT moves your business

OVERNET.
upgrade your digital skills

[stesi]
Powered by Innovation

DATA
SATURDAYS

 **hn0va**
community

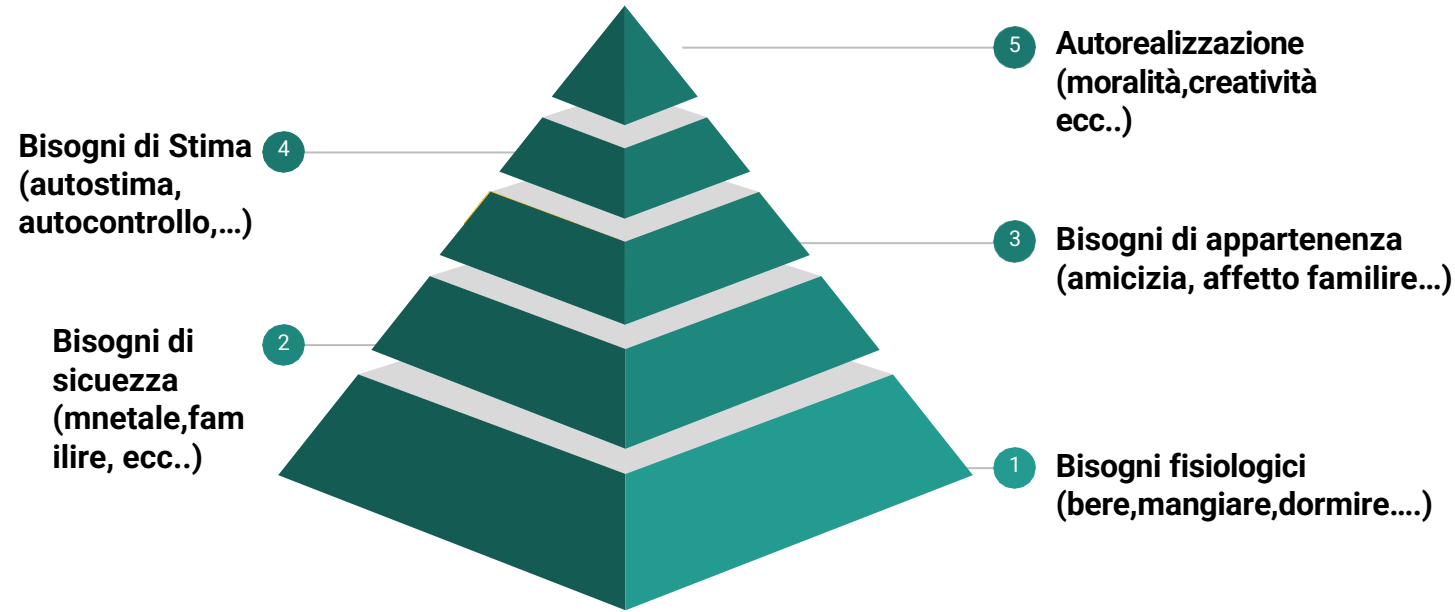
About me

- Nel 2023 Fondatore e CTO Regolo Farm ci occupiamo di Business Intelligence e Power Platform e AI. Realizzato un verticale per la insurance data analytics per le compagnie assicurative
- Docente all'Università di Pordenone per i corsi IFTS di analisi Big Data
- Community Lead di 1nn0va (www.innovazionefvg.net)
- MCP,MCSA,MCSE dal 2017 MCT e dal 2014 MVP per SQL Server e poi per Fabric e relatore in diverse conferenze sul tema.
 - Marco.pozzan@regolofarm.com
 - [@marcopozzan.it](https://www.linkedin.com/company/marcopozzan) (linked-in)
 - [@regolo farm](https://www.linkedin.com/company/regolo-farm) (linked-in)
 - www.marcopozzan.it



- Architettura Dati Classica
- Overview di DBT
- DBT e strutture dati
- Demo - Analytics Engineering

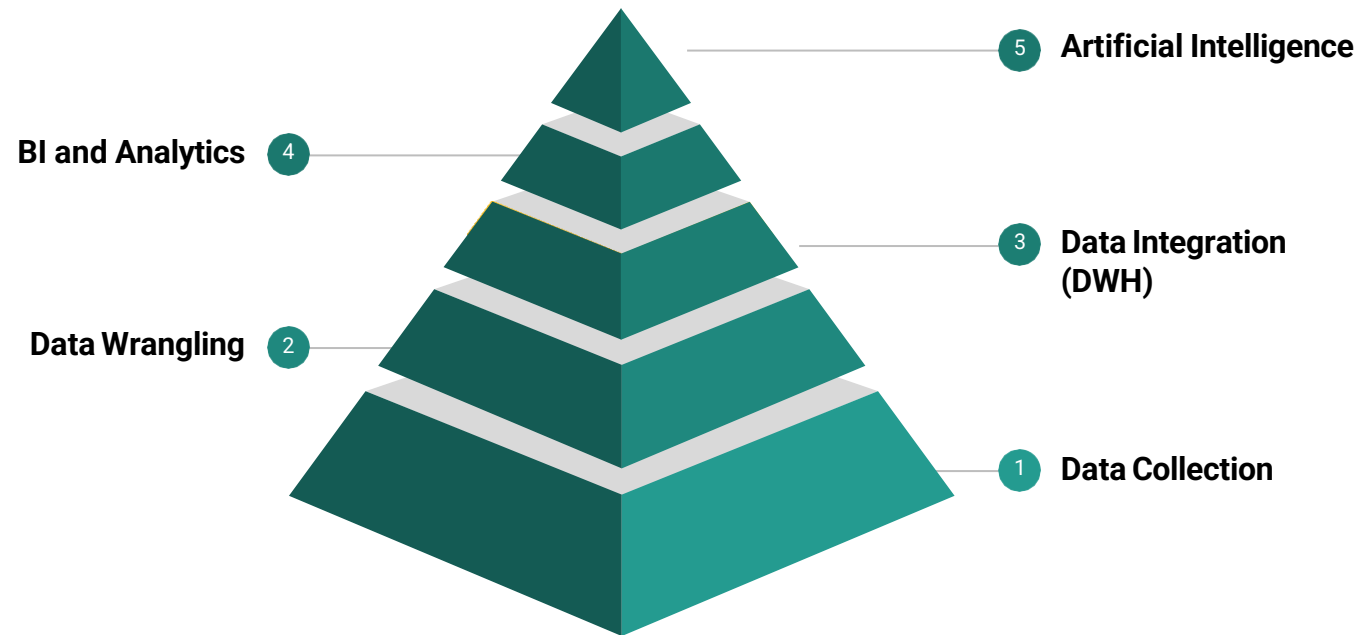
Piramide di maslow's



- Quando la categoria è soddisfatta, si passa al livello successivo della gerarchia.
- Ma non si può salire se non si soddisfano le esigenze del livello inferiore.
- Se si salta troppo velocemente a uno stadio avanzato, non si tratta di una piramide solida. Quindi fallimento!!!!
- Perché vi dico questo? 😊

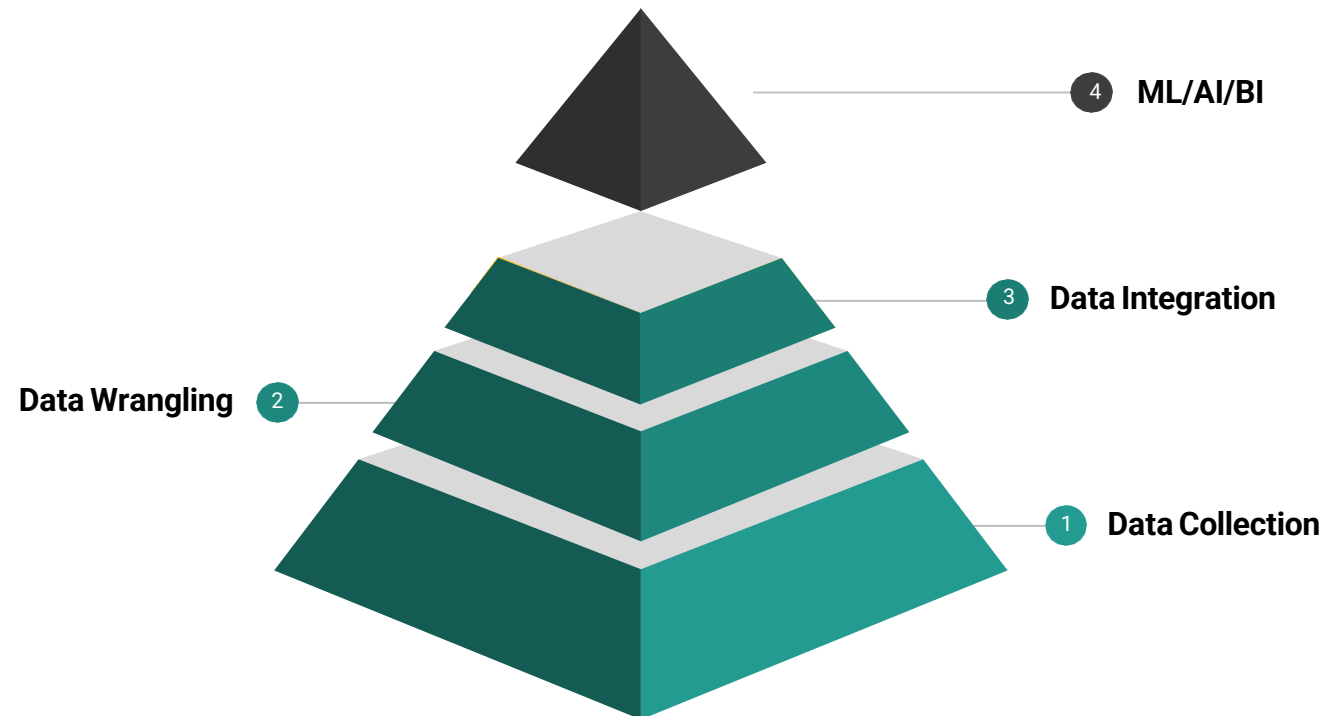
Data maturity model

La nostra gerarchia dei bisogni dei dati è simile a quella di maslow.



Queste sono le cinque fasi dell'evoluzione dei dati che ogni organizzazione orientata ai dati deve padroneggiare per avere successo.

Ora rimuoviamo la parte superiore della piramide non sono per noi rilevanti in questo momento



Tuttavia, le tre fasi rimanenti sono estremamente critiche e rappresentano la fase di ETL (prime tre fasi della piramide dei bisogni)

- Lo scopo di un data warehouse è quello di fungere da tecnologia per l'analisi dei dati e il reporting.
- Non è altro che un database che ci permette di fare analisi ottimizzate ad alte prestazioni sui nostri dati e con cui possiamo creare dimensioni, fatti e gestire la demoralizzazione.
- I data warehouse di dati non sono altro che un motore di prestazioni che ci permette di eseguire carichi di lavoro analitici sui nostri dati utilizzando il sql.
- È probabile che utilizziate i data warehouse per scopi di reporting o dashboard, quindi è molto importante mantenere i dati nel data warehouse ben strutturati e puliti o puliti in modo da non dover lavorare giorni e settimane per rispondere a domande molto semplici quando il vostro manager vi chiede qualcosa.





Di seguito è riportato l'albero completo dei file del progetto.

Non preoccuparti se ti sembrano troppe informazioni da assimilare in una volta sola 😊

serve solo a darti una visione completa di ciò che è DBT.

Ci concentreremo su ciascuna sezione, una per una

```
jaffle_shop
├── README.md
├── analyses
├── seeds
│   └── employees.csv
├── dbt_project.yml
├── macros
│   └── cents_to_dollars.sql
├── models
│   ├── intermediate
│   │   ├── finance
│   │   │   ├── _int_finance__models.yml
│   │   │   └── int_payments_pivoted_to_orders.sql
│   ├── marts
│   │   ├── finance
│   │   │   ├── _finance__models.yml
│   │   │   ├── orders.sql
│   │   │   └── payments.sql
│   │   └── marketing
│   │       ├── _marketing__models.yml
│   │       └── customers.sql
│   └── staging
│       ├── jaffle_shop
│       │   ├── _jaffle_shop__docs.md
│       │   ├── _jaffle_shop__models.yml
│       │   ├── _jaffle_shop__sources.yml
│       │   ├── base
│       │   │   ├── base_jaffle_shop__customers.sql
│       │   │   └── base_jaffle_shop__deleted_customers.sql
│       │   ├── stg_jaffle_shop__customers.sql
│       │   └── stg_jaffle_shop__orders.sql
│       ├── stripe
│       │   ├── _stripe__models.yml
│       │   ├── _stripe__sources.yml
│       │   └── stg_stripe__payments.sql
│       └── utilities
│           └── all_dates.sql
├── packages.yml
├── snapshots
├── tests
│   └── assert_positive_value_for_total_amount.sql
```

Staging

Il livello di staging è dove inizia il nostro viaggio. Questa è la base del nostro progetto, dove portiamo tutti i singoli componenti che useremo per costruire i nostri modelli più complessi e utili nel progetto.

Useremo un'analogia per lavorare con dbt

- pensare in modo modulare in termini di atomi, molecole e output più complessi come proteine o cellule.
- In questo contesto, se i dati del nostro sistema sorgente sono il **brodo primordiale**, allora puoi pensare allo strato di staging come a una condensazione e raffinazione di questo materiale nei singoli atomi con cui in seguito costruiremo strutture più intricate e utili.

La struttura delle cartelle è estremamente importante in dbt: chiave per comprendere il knowledge graph codificato nel nostro progetto.

Possiamo usare la nostra struttura delle cartelle come mezzo di selezione nella sintassi del selettore dbt .

- Ad esempio se volessimo eseguire tutti i modelli che si basano sui nostri dati Stripe, possiamo facilmente eseguirli `dbt build --select staging.stripe+`

```
models/staging
├── jaffle_shop
│   ├── _jaffle_shop__docs.md
│   ├── _jaffle_shop__models.yml
│   ├── _jaffle_shop__sources.yml
│   ├── base
│   │   ├── base_jaffle_shop__customers.sql
│   │   └── base_jaffle_shop__deleted_customers.sql
│   ├── stg_jaffle_shop__customers.sql
│   └── stg_jaffle_shop__orders.sql
└── stripe
    ├── _stripe__models.yml
    ├── _stripe__sources.yml
    └── stg_stripe__payments.sql
```

Creare un modello coerente di denominazione dei file è fondamentale in dbt .

I nomi file devono essere univoci e corrispondere al nome del modello:

- Consigliamo di inserire quante più informazioni chiare possibili nel nome file
- mettere un prefisso per il layer in cui esiste il modello,
- informazioni di raggruppamento specifiche sull'entità o la trasformazione nel modello.
 - ✓ **stg_[source]__[entity]s.sql**- la doppia sottolineatura tra sistema sorgente ed entità aiuta a distinguere visivamente le parti separate nel caso di un nome sorgente con più parole.. Pensalo come una virgola di Oxford , la chiarezza extra vale di gran lunga la punteggiatura extra.
 - ✓ **Plurale**. Vogliamo appoggiarci alla chiarezza e alla natura dichiarativa di SQL quando possibile. Pertanto, a meno che non ci sia un singolo ordine nella tua **orders** , il plurale è il modo corretto per descrivere cosa c'è in una tabella con più righe.

Uno dei pattern più comuni all'interno del layer di staging lo vediamo a lato. Abbiamo organizzato il nostro modello in due CTE: uno estrae una tabella sorgente tramite la macro sorgente e l'altro applica le nostre trasformazioni.

I successivi livelli di trasformazione varieranno notevolmente da modello a modello, ognuno dei nostri modelli di staging seguirà esattamente lo stesso schema. Dobbiamo assicurarci che lo schema sia solido come una roccia e coerente.

```
with
source as (
    select * from {{ source('stripe','payment') }}
),
renamed as (
    select
        -- ids
        id as payment_id,
        orderid as order_id,

        -- strings
        paymentmethod as payment_method,
        case
            when payment_method in ('stripe', 'paypal', 'credit_card', 'gift_card') then 'credit'
            else 'cash'
        end as payment_type,
        status,

        -- numerics
        amount as amount_cents,
        amount / 100.0 as amount,

        -- booleans
        case
            when status = 'successful' then true
            else false
        end as is_completed_payment,

        -- dates
        date_trunc('day', created) as created_date,

        -- timestamps
        created::timestamp_ltz as created_at

    from source

    select * from renamed
```

Sulla base di quanto sopra, i tipi più standard di trasformazioni del modello di staging sono:

- ✓ Rinomina
- ✓ Fusione di tipo
- ✓ Calcoli di base (ad esempio centesimi in dollari)
- ✓ Categorizzazione (utilizzando la logica condizionale per raggruppare i valori in gruppi o valori booleani, come nelle case when istruzioni precedenti)

✗ Join : l'obiettivo dei modelli di staging è di pulire e preparare i singoli concetti conformi alla sorgente per l'utilizzo a valle. Stiamo creando la versione più utile di una tabella del sistema sorgente, che possiamo utilizzare come nuovo componente modulare per il nostro progetto.

Nella nostra esperienza, i join sono quasi sempre una cattiva idea in questo caso

✗ **Aggregazioni** : le aggregazioni comportano raggruppamenti, e non lo faremo in questa fase. Ricorda: i modelli di staging sono il tuo posto per creare i blocchi di costruzione che utilizzerai per tutto il resto del tuo progetto: **se iniziamo a modificare la grana delle nostre tabelle raggruppando in questo livello, perderemo l'accesso ai dati sorgente di cui probabilmente avremo bisogno a un certo punto.** Vogliamo solo che i nostri singoli concetti siano puliti e pronti per l'uso e gestiremo i valori di aggregazione a valle.

✓ **Unione di fonti disparate ma simmetriche** . Un esempio tipico qui sarebbe se gestissi più piattaforme di e-commerce in vari territori tramite una piattaforma SaaS. Avresti schemi perfettamente identici, ma tutti caricati separatamente nel tuo warehouse. In questo caso, è più facile ragionare sui nostri ordini se tutti i nostri negozi sono uniti insieme, quindi vorremmo gestire l'unione in un modello di base prima di procedere con le nostre solite trasformazioni del modello di staging

Staging Materializziamo con viste

Nel nostro **dbt_project.yml** sotto, possiamo vedere che abbiamo configurato l'intera directory di staging per essere materializzata come viste. Poiché non sono concepiti come artefatti finali, ma piuttosto come elementi costitutivi per modelli successivi, i modelli di staging dovrebbero in genere essere materializzati come viste per due motivi principali:

- Qualsiasi modello downstream che fa riferimento ai nostri modelli di staging otterrà sempre i dati più aggiornati possibili da tutte le viste dei componenti che sta assemblando e materializzando
- Evita di sprecare spazio nel DWH su modelli che non sono destinati a essere interrogati dai consumatori di dati e che quindi non devono funzionare in modo rapido o efficiente

```
# dbt_project.yml

models:
  jaffle_shop:
    staging:
      +materialized: view
```

I modelli di staging dovrebbero avere una relazione 1 a 1 con le nostre tabelle di origine. Ciò significa che per ogni tabella di sistema di origine avremo un singolo modello di staging che vi fa riferimento, fungendo da punto di ingresso, ovvero da staging , per l'uso a valle.

Staging Materializziamo con viste

View	Table	Incremental (table appends)	Ephemeral (CTEs)
Use it <ul style="list-style-type: none">- You want a lightweight representation- You don't reuse data too often Don't use it <ul style="list-style-type: none">- You read from the same model several times	Use it <ul style="list-style-type: none">- You read from this model repeatedly Don't use it <ul style="list-style-type: none">- Building single-use models- Your model is populated incrementally	Use it <ul style="list-style-type: none">- Fact tables- Appends to tables Don't use it <ul style="list-style-type: none">- You want to update historical records	Use it <ul style="list-style-type: none">- You merely want an alias to your data Don't use it <ul style="list-style-type: none">- You read from the same model several times

Supponiamo di voler creare una trasformazione intermedia e non si vuole pubblicare nel DWH quindi si crea una CTE dietro le quinte per poi usarla per una seconda trasformazione

Intermediate

Una volta che avremo i nostri atomi pronti per lavorare, inizieremo a riunirli in forme molecolari più intricate e connesse. Lo strato intermedio è dove vivono queste molecole, creando forme diverse con scopi specifici sulla strada verso le proteine e le cellule più complesse

```
models/intermediate
├─ finance
│   ├── _int_finance__models.yml
│   └─ int_payments_pivoted_to_orders.sql
```

✓ Sottodirectory basate su raggruppamenti aziendali. A differenza dello strato di staging, qui ci spostiamo verso la conformità aziendale, suddividendo i nostri modelli in sottodirectory non in base al loro sistema di origine, ma in base alla loro area di interesse aziendale.

✓ **Nomi dei file: int_[entity]s_[verb]s.sql**- la varietà di trasformazioni che possono verificarsi all'interno dello strato intermedio rende più difficile dettare come nominarle. **Il miglior principio guida è pensare ai verbi (ad esempio pivoted, aggregated_to_user, joined, fanned_out_by_quantity, funnel_created, ecc.) nello strato intermedio.**

Ad esempio un modello intermedio per far pivottare i pagamenti, quindi chiamiamo il nostro modello int_payments_pivoted_to_orders. **È facile per chiunque capire rapidamente cosa sta accadendo in quel modello, anche se non conosce SQL.**

✗ **Esposti agli utenti finali.** I modelli intermedi in non dovrebbero essere esposti nello schema di produzione principale. **Non sono destinati all'output su target finali come dashboard o applicazioni, quindi tenerli separati dai modelli che lo sono, in modo da poter controllare più facilmente la governance dei dati e la rilevabilità.**

```
-- int_payments_pivoted_to_orders.sql

{% set payment_methods = ['bank_transfer', 'credit_card', 'coupon', 'gift_card'] -%}

with

payments as (

    select * from {{ ref('stg_stripe__payments') }}

),

pivot_and_aggregate_payments_to_order_grain as (

    select
        order_id,
        {% for payment_method in payment_methods -%}

            sum(
                case
                    when payment_method = '{{ payment_method }}' and
                        status = 'success'
                    then amount
                    else 0
                end
            ) as {{ payment_method }}_amount,

        {%- endfor -%}
        sum(case when status = 'success' then amount end) as total_amount

    from payments

    group by 1

)

select * from pivot_and_aggregate_payments_to_order_grain
```


- ✓ Materializzati come viste in uno schema personalizzato con autorizzazioni speciali. Un'opzione più solida è quella di materializzare i tuoi modelli intermedi come viste in uno schema personalizzato specifico , al di fuori del tuo schema di produzione principale. Ciò:
- ti offre una visione più approfondita dello sviluppo
 - una risoluzione dei problemi più semplice man mano che il numero e la complessità dei tuoi modelli aumentano, pur rimanendo facili da implementare
 - occupando uno spazio trascurabile.

Mantieni il tuo DWH in ordine! 😊

Alcuni dei casi d'uso più comuni dei modelli intermedi includono:

- ✓ **Semplificazione strutturale.** Riunendo un numero ragionevole (in genere da 4 a 6) di entità o concetti (modelli di staging o forse altri modelli intermedi) che saranno uniti a un altro modello intermedio con scopi simili per generare un mart, anziché avere 10 join nel nostro mart, possiamo unire due modelli intermedi che ospitano ciascuno un pezzo della complessità, il che ci offre maggiore leggibilità, flessibilità, superficie di test e approfondimenti sui nostri componenti.
- ✓ **Ri-graining.** I modelli intermedi sono spesso usati per distribuire o comprimere i modelli alla giusta grana
- ✓ **Isolare operazioni complesse.** È utile spostare qualsiasi parte di logica particolarmente complessa o difficile da comprendere nei propri modelli intermedi. Semplifica anche i modelli successivi che possono fare riferimento a questo concetto in un modo più chiaramente leggibile

- ✓ Ri-graining. I modelli intermedi sono spesso usati per distribuire o comprimere i modelli alla giusta grana composita
- ✓ Isolare operazioni complesse. È utile spostare qualsiasi parte di logica particolarmente complessa o difficile da comprendere nei propri modelli intermedi. Ciò non solo li rende più facili da perfezionare e risolvere i problemi, ma semplifica anche i modelli successivi che possono fare riferimento a questo concetto in un modo più chiaramente leggibile

Mart

Questo è il livello in cui tutto si unisce e iniziamo a organizzare tutti i nostri atomi (modelli di staging) e molecole (modelli intermedi) in cellule a pieno titolo che hanno identità e scopo.

Tutti i nostri mart sono pensati per rappresentare un'entità o un concetto specifico nella sua grana.

Ad esempio, un ordine, un cliente, un territorio, un evento clic, un pagamento: ognuno di questi sarebbe rappresentato con un mart distinto e ogni riga rappresenterebbe un'istanza discreta di questi concetti. (Creare gli stessi dati in più posti è più efficiente in questo paradigma rispetto al riunire ripetutamente questi concetti)

```
models/marts
├── finance
│   ├── _finance__models.yml
│   ├── orders.sql
│   └── payments.sql
└── marketing
    ├── _marketing__models.yml
    └── customers.sql
```

- ✓ **Raggruppa per reparto o area di interesse.** Se hai meno di 10 mart circa, potresti non aver bisogno di sottocartelle, quindi, come per il livello intermedio, non ottimizzare troppo presto. Nel nostro livello mart, non ci preoccupiamo più dei dati conformi alla fonte, quindi il raggruppamento per reparti (marketing, finanza, ecc.) è la struttura più comune in questa fase.
- ✓ **Assegna un nome all'entità.** Usa un inglese per assegnare un nome al file in base al concetto che costituisce la grana del mart customers, orders.
- ✗ **Costruisci lo stesso concetto in modo diverso per team diversi.** finance_orders ed marketing_orders è in genere considerato un anti-pattern.

```
-- orders.sql
```

```
with
```

```
orders as (
```

```
    select * from {{ ref('stg_jaffle_shop__orders' ) }}
```

```
),
```

```
order_payments as (
```

```
    select * from {{ ref('int_payments_pivoted_to_orders') }}
```

```
),
```

```
orders_and_order_payments_joined as (
```

```
    select
```

```
        orders.order_id,
```

```
        orders.customer_id,
```

```
        orders.order_date,
```

```
        coalesce(order_payments.total_amount, 0) as amount,
```

```
        coalesce(order_payments.gift_card_amount, 0) as gift_card_amount
```

```
    from orders
```

```
    left join order_payments on orders.order_id = order_payments.order_id
```

```
)
```

```
select * from orders_and_payments_joined
```

```
-- customers.sql

with

customers as (

    select * from [{ ref('stg_jaffle_shop__customers')}]

),

orders as (

    select * from [{ ref('orders')}]

),

customer_orders as (

    select

        customer_id,
        min(order_date) as first_order_date,
        max(order_date) as most_recent_order_date,
        count(order_id) as number_of_orders,
        sum(amount) as lifetime_value

    from orders

    group by 1

),

customers_and_customer_orders_joined as (

    select

        customers.customer_id,
        customers.first_name,
        customers.last_name,
        customer_orders.first_order_date,
        customer_orders.most_recent_order_date,
        coalesce(customer_orders.number_of_orders, 0) as number_of_orders,
        customer_orders.lifetime_value

    from customers

    left join customer_orders on customers.customer_id = customer_orders.customer_id

)

select * from customers_and_customer_orders_joined
```




Demo ANALYTICS ENGINEERING WITH AIRBNB

Requisiti:

- Le modifiche alla modellazione sono facili da seguire e ripristinare
- Dipendenze esplicite tra modelli
- Esplorare le dipendenze tra modelli
- Test di qualità dei dati
- Segnalazione degli errori
- Caricamento incrementale delle tabelle dei fatti
- Documentazione di facile accesso



Cosa vedremo?

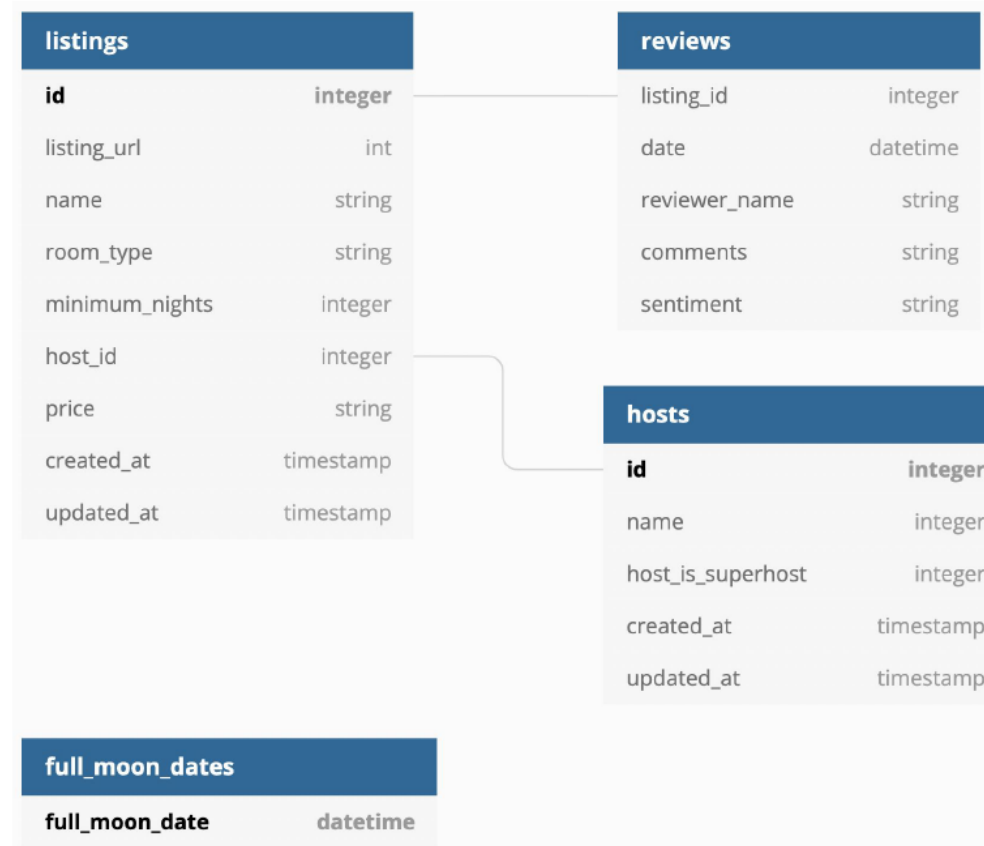
- Caricamento, pulizia, esposizione dei dati
- Scrittura di test, automazioni e documentazione
- Fonte dei dati: Inside Airbnb: Roma (<https://insideairbnb.com/rome/>)

12 December, 2024 ([Explore](#))

Country/City	File Name	Description
Rome	listings.csv.gz	Detailed Listings data
Rome	calendar.csv.gz	Detailed Calendar Data
Rome	reviews.csv.gz	Detailed Review Data
Rome	listings.csv	Summary information and metrics for listings in Rome (good for visualisations).
Rome	reviews.csv	Summary Review data and Listing ID (to facilitate time based analytics and visualisations linked to a listing).

Per connettersi a Fabric Bisogna creare un Service Principal

1. ClientId:feca4d00-63fe-47b6-95d1-4fe2ca30dd48
2. Secretid: YGy8Q~~WATQCFfV~LpuyiwFyQ4OxCGECjEhGya~Z



Configurazione Ambiente

Prima bisogna installare python (<https://www.python.org/downloads/>)

- 1) pip install dbt-core dbt-fabric -- Installazione dbt con supporto fabric
- 2) dbt --version --verifica versione dbt
- 2.1) pip install --upgrade dbt-fabric -- Eventualmente per aggiornare il connettore
- 3) python -m venv dbt-env (C:\Users\MarcoPozzan-REGOLOFA\dbt-env)
- 4) dbt-env\Scripts\activate
- 5) dbt init -- Inizializza un nuovo progetto DBT su Fabric. (C:\Users\MarcoPozzan-REGOLOFA\datasat25pn)


```
13:00:14 Using profiles.yml file at C:\Users\MarcoPozzan-REGOLOFA\.dbt\profiles.yml
13:00:14 Using dbt_project.yml file at C:\Users\MarcoPozzan-REGOLOFA\datasat25\dbt_project.yml
13:00:14 adapter type: fabric
13:00:14 adapter version: 1.9.0
13:00:14 Configuration:
13:00:14   profiles.yml file [OK found and valid]
13:00:14   dbt_project.yml file [OK found and valid]
13:00:14 Required dependencies:
13:00:14   - git [OK found]

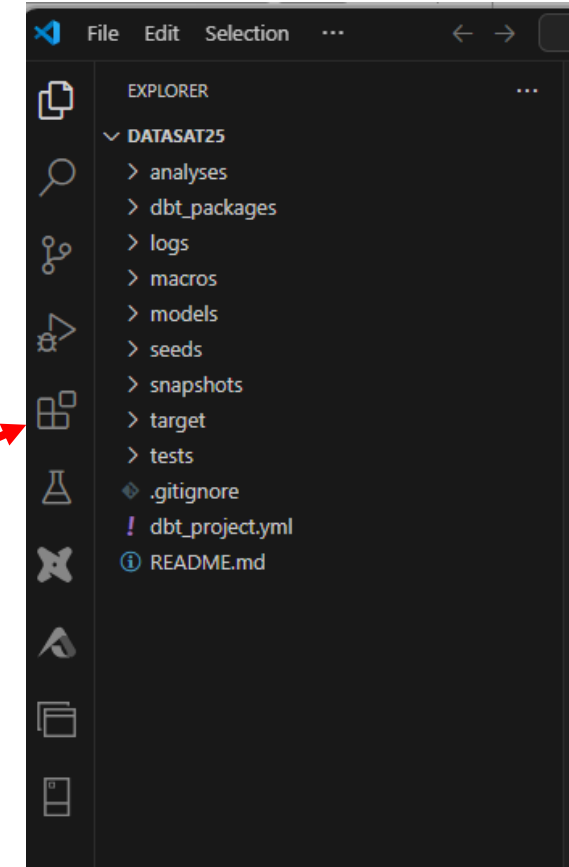
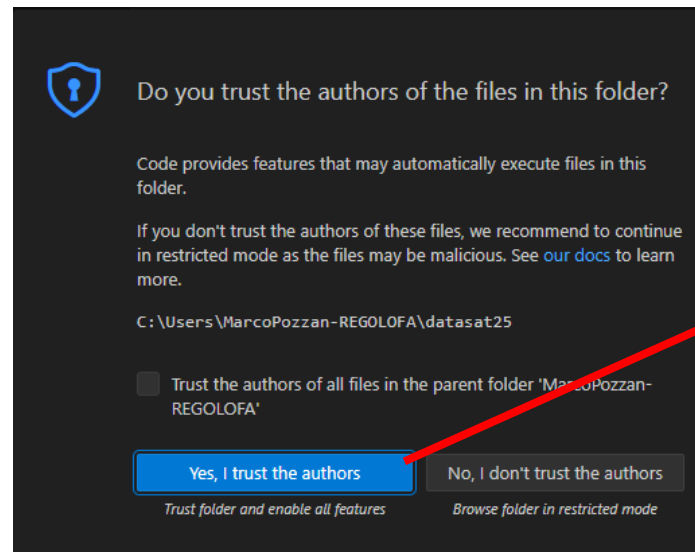
13:00:14 Connection:
13:00:14   server: ehauyipajrjepfj3id6yqalcnm-qm3wev3hxs4unlbubqeqpbwlu.datawarehouse.fabric.microsoft.com
13:00:14   database: airbnb
13:00:14   schema: dev
13:00:14   UID: None
13:00:14   client_id: feca4d00-63fe-47b6-95d1-4fe2ca30dd48
13:00:14   authentication: ActiveDirectoryServicePrincipal
13:00:14   encrypt: True
13:00:14   trust_cert: False
13:00:14   retries: 3
13:00:14   login_timeout: 0
13:00:14   query_timeout: 0
13:00:14   trace_flag: False
13:00:14 Registered adapter: fabric=1.9.0
13:00:17 Connection test: [OK connection ok]

13:00:17 All checks passed!
```

Prima bisogna installare il plug-in per visual studio code (dbt power User)

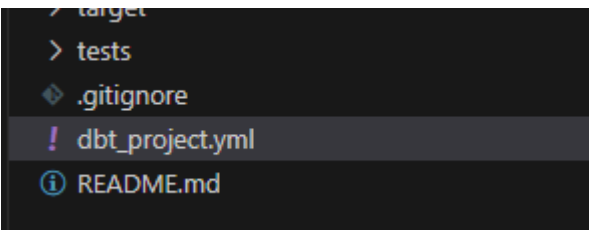
1) Ci si posiziona nell'ambiente inizializzato nel passo precedente **datasat25pn**

2) Si digita code . `(dbt-env) C:\Users\MarcoPozzan-REGOLOFA\datasat25> code .|`



Configurazione DBT_PROJECT.YML

DEMO – DBT_PROJECT.YML



```
# Name your project! Project names should contain only lowercase characters
# and underscores. A good package name should reflect your organization's
# name or the intended use of these models
name: 'datasat25'
version: '1.0.0'

# This setting configures which "profile" dbt uses for this project.
profile: 'datasat25'

# These configurations specify where dbt should look for different types of files.
# The `model-paths` config, for example, states that models in this project can be
# found in the "models/" directory. You probably won't need to change these:
model-paths: ["models"]
analysis-paths: ["analyses"]
test-paths: ["tests"]
seed-paths: ["seeds"]
macro-paths: ["macros"]
snapshot-paths: ["snapshots"]

clean-targets:      # directories to be removed by `dbt clean`
- "target"
- "dbt_packages"

# Configuring models
# Full documentation: https://docs.getdbt.com/docs/configuring-models

# In this example config, we tell dbt to build all models in the example/
# directory as views. These settings can be overridden in the individual model
# files using the `{{ config(...) }}` macro.
models:
  datasat25:
    # Config indicated by + and applies to all files under models/example/
    example:
      +materialized: view
```

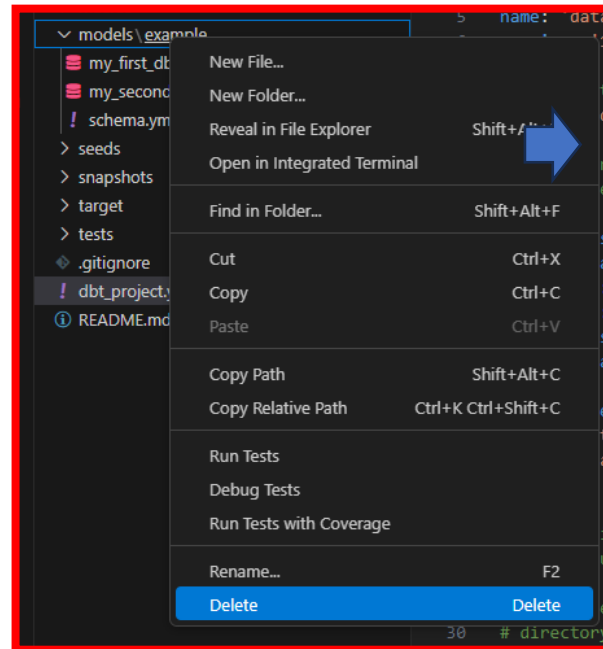
Definizione dei percorsi
dove recuperare i comandi

Tutto ciò che è in queste
cartelle di codice sorgente
creato da noi ma che non
serve per il progetto verrà
ripulito con dbt clean

Lo modificherete spesso
perché determina come
vengono materializzati i
modelli

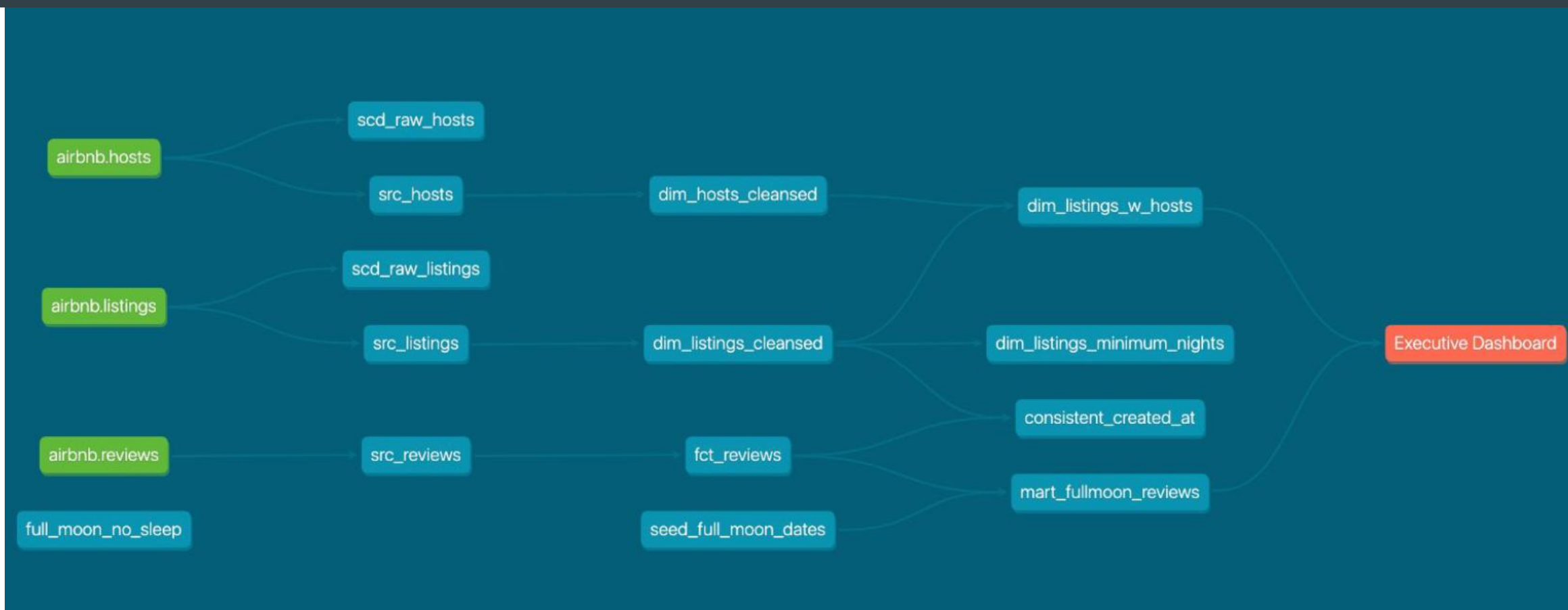
```
1 # files using the `{{ config(...) }}` macro.
2
3 models:
4   +materialized: view
5
```

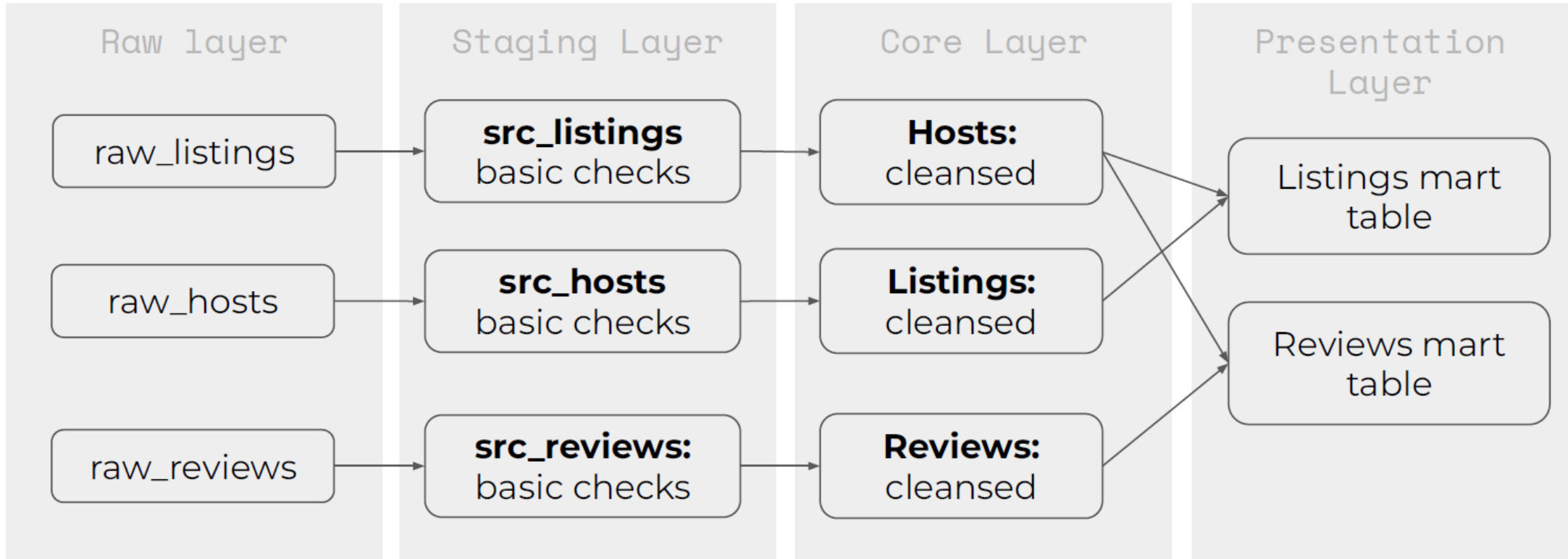
Rimuoviamo la sessione Examples



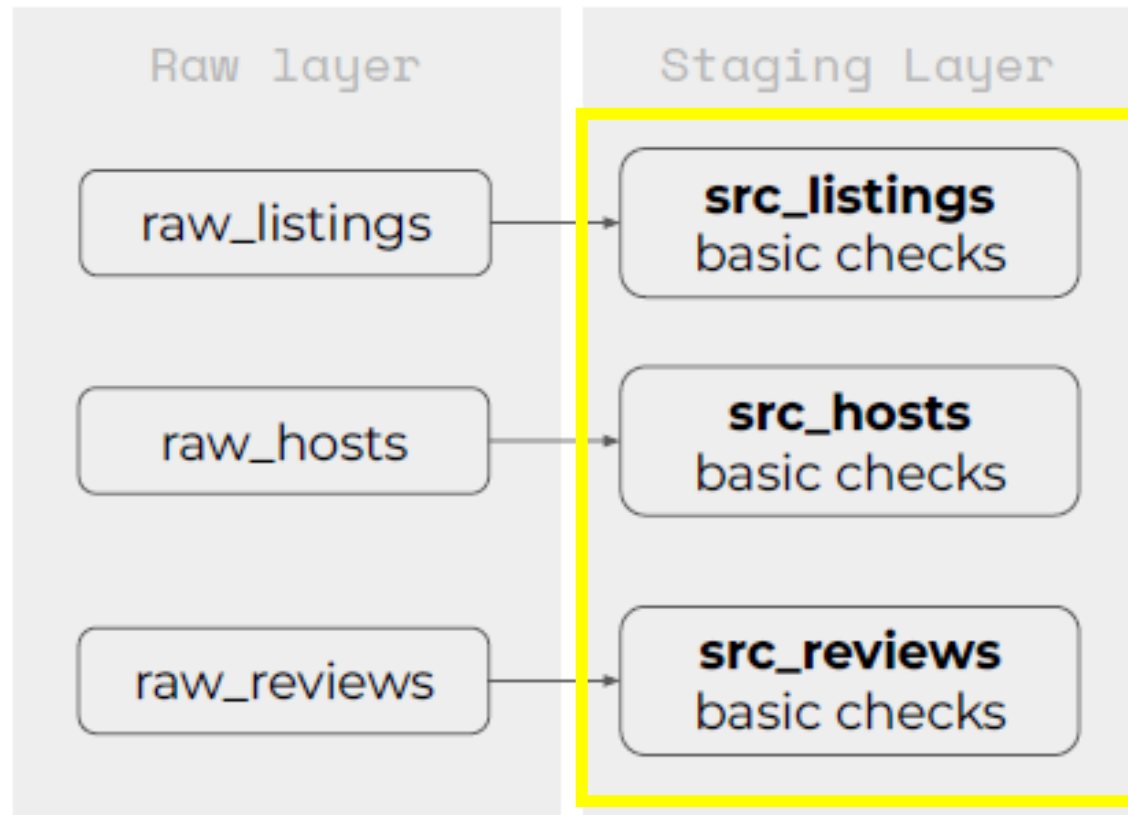
Rimuoviamo la cartella Examples

Flusso Dati





Staging



models/src/src_listings.sql

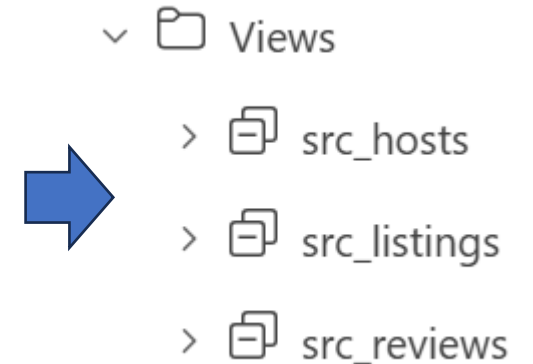
```
WITH raw_listings AS (
  SELECT
    *
  FROM
    airbnb.RAW.raw_listings
)
SELECT
  id AS listing_id,
  name AS listing_name,
  listing_url,
  room_type,
  minimum_nights,
  host_id,
  price AS price_str,
  created_at,
  updated_at
FROM
  raw_listings
```

models/src/src_reviews.sql

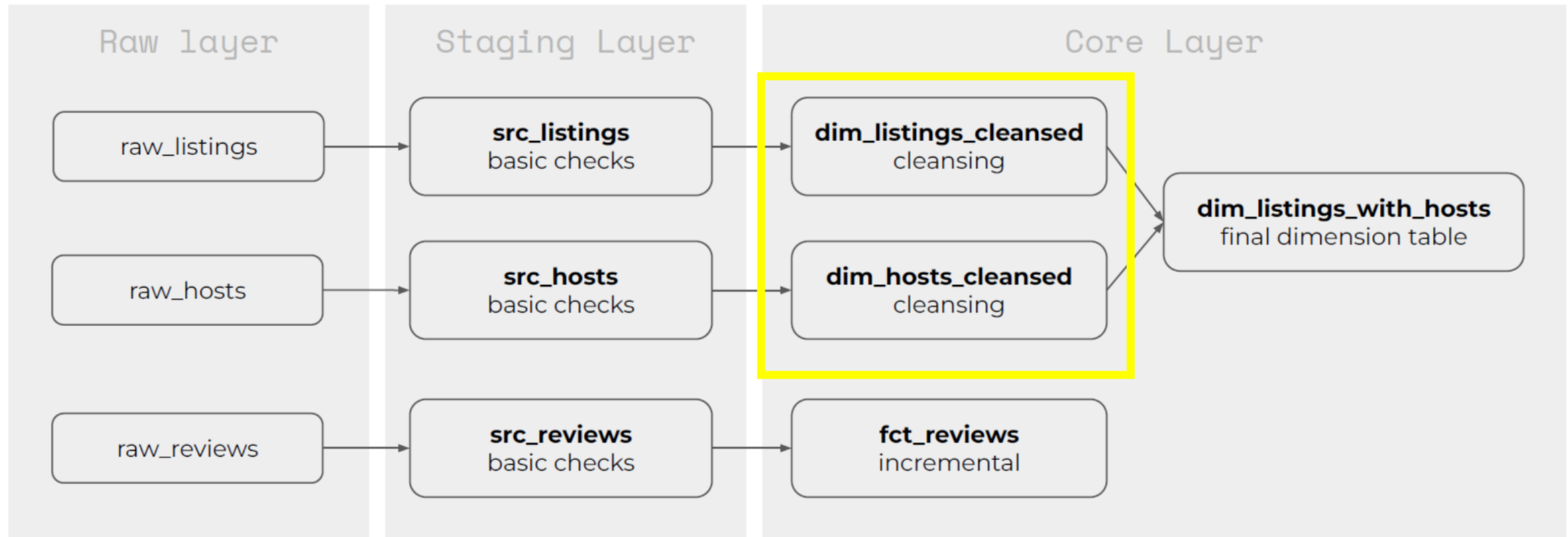
```
WITH raw_reviews AS (
  SELECT
    *
  FROM
    airbnb.RAW.raw_reviews
)
SELECT
  listing_id,
  date AS review_date,
  reviewer_name,
  comments AS review_text,
  sentiment AS review_sentiment
FROM
  raw_reviews
```

models/src/src_hosts.sql

```
WITH raw_hosts AS (
  SELECT
    *
  FROM
    airbnb.RAW.raw_hosts
)
SELECT
  id AS host_id,
  name AS host_name,
  is_superhost,
  created_at,
  updated_at
FROM
  raw_hosts
```



Intermediate Dim e Fact



models/dim/dim_listings_cleansed.sql :

```
WITH src_listings AS (
  SELECT
    *
  FROM
    {{ ref('src_listings') }}
)
SELECT
  listing_id,
  listing_name,
  room_type,
  CASE
    WHEN minimum_nights = 0 THEN 1
    ELSE minimum_nights
  END AS minimum_nights,
  host_id,
  price_str AS price,
  created_at,
  updated_at
FROM
  src_listings
```

models/dim/dim_hosts_cleansed.sql

```
WITH src_hosts AS (
  SELECT
    *
  FROM
    {{ ref('src_hosts') }}
)
SELECT
  host_id,
  COALESCE(
    host_name,
    'Anonymous'
  ) AS host_name,
  is_superhost,
  created_at,
  updated_at
FROM
  src_hosts
```

Jinja
(Template Engine)



- ✓ Tables
 - > dim_hosts_cleansed
 - > dim_listings_cleansed

VOGLIAMO MATERIALIZZARE CON TABELLE

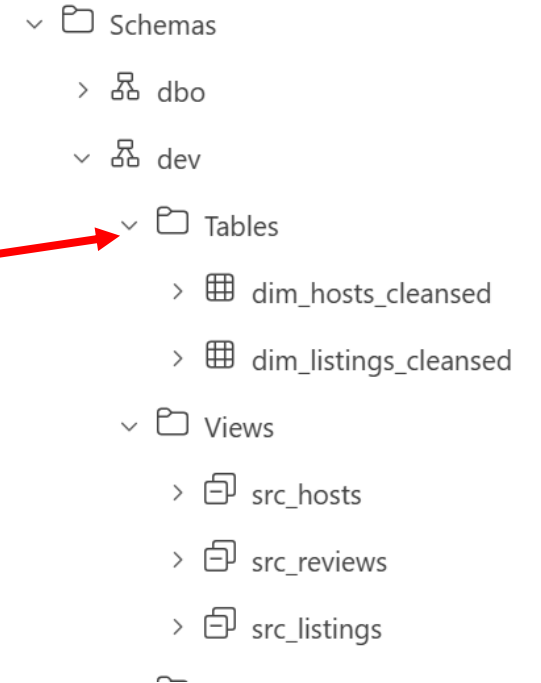
Possibilità di cambiare la tipologia di materializzazione. Per Fare questo dobbiamo modificare il file **dbt_project.yml**. Indicando di materializzare tutto come view. Il +materialized view rinforza il default e quindi tutto quello che c'è dentro il ramo datasat25 è impostato come view.

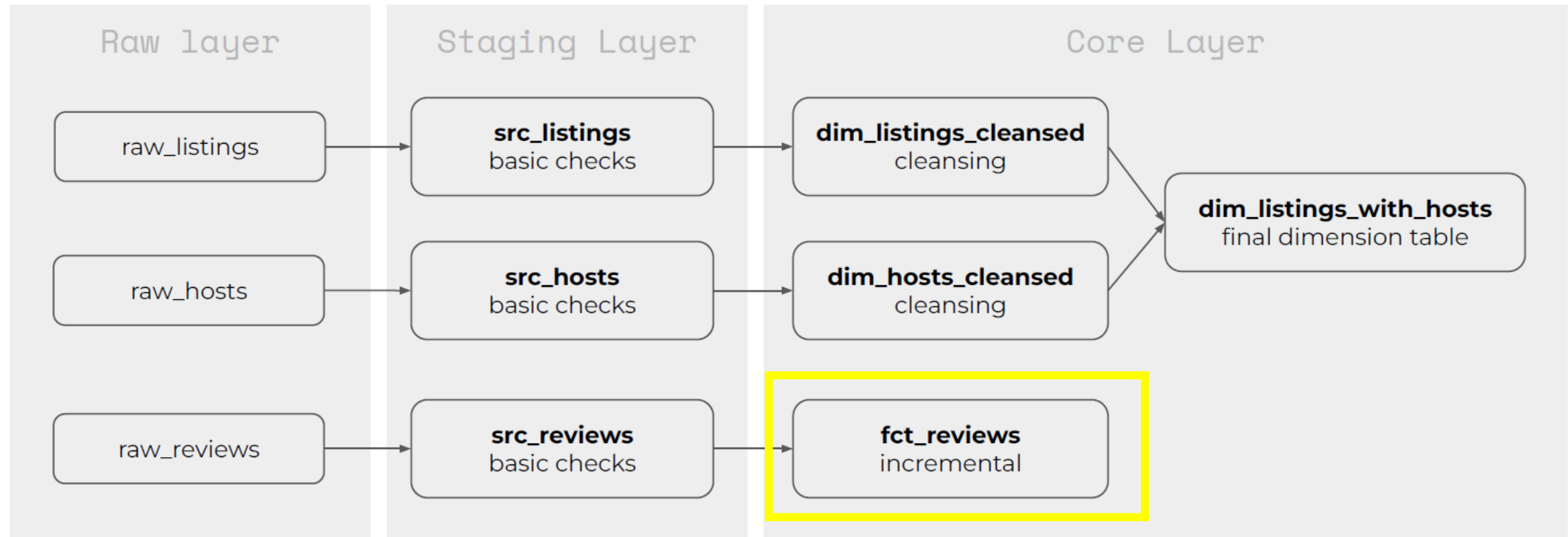
```
models:
  datasat25:
    # Config indicated by + and applies to all files under models/example/
    +materialized: view
```

Se vogliamo che tutto ciò che è sotto **dim** venga materializzato come table aggiungiamo la seguente parte.

```
models:
  datasat25:
    # Config indicated by + and applies to all files under models/example/
    +materialized: view
  dim:
    +materialized: table
```

```
(dbt-env) C:\Users\MarcoPozzan-REGOLOFA\datasat25>dbt run
21:06:17 Running with dbt=1.9.1
21:06:18 Registered adapter: fabric=1.9.0
21:06:18 Found 5 models, 469 macros
21:06:18
21:06:18 Concurrency: 4 threads (target='dev')
21:06:18
21:06:19 1 of 5 START sql view model dev.src_hosts ..... [RUN]
21:06:19 2 of 5 START sql view model dev.src_listings ..... [RUN]
21:06:19 3 of 5 START sql view model dev.src_reviews ..... [RUN]
21:06:21 1 of 5 OK created sql view model dev.src_hosts ..... [OK in 1.30s]
21:06:21 4 of 5 START sql table model dev.dim_hosts_cleansed ..... [RUN]
21:06:21 3 of 5 OK created sql view model dev.src_reviews ..... [OK in 1.41s]
21:06:21 2 of 5 OK created sql view model dev.src_listings ..... [OK in 1.66s]
21:06:21 5 of 5 START sql table model dev.dim_listings_cleansed ..... [RUN]
21:06:30 4 of 5 OK created sql table model dev.dim_hosts_cleansed ..... [OK in 9.59s]
21:06:30 5 of 5 OK created sql table model dev.dim_listings_cleansed ..... [OK in 9.21s]
21:06:30
21:06:30 Finished running 2 table models, 3 view models in 0 hours 0 minutes and 12.29 seconds (12.29s).
21:06:30 Completed successfully
21:06:30
21:06:30 Done. PASS=5 WARN=0 ERROR=0 SKIP=0 TOTAL=5
(dbt-env) C:\Users\MarcoPozzan-REGOLOFA\datasat25>
```





fct/fct_reviews.sql

```

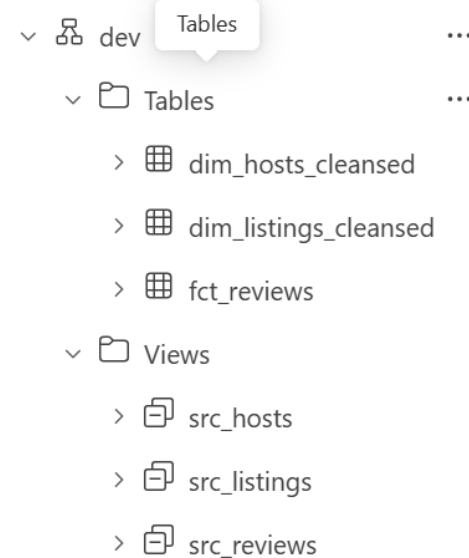
-- Add documentation or tests
{{
  config(
    materialized = 'incremental',
    on_schema_change='fail'
  )
}}

WITH src_reviews AS (
  SELECT * FROM {{ ref('src_reviews') }}
)

SELECT * FROM src_reviews
WHERE review_text is not null

{% if is_incremental() %}
  AND review_date > (select max(review_date) from {{ this }})
{% endif %}

```



vogliamo che sia una materializzazione del tipo incrementale. Si specifica con un comando particolare all'inizio del file. Specifichiamo anche come deve comportarsi con il cambiamento dello schema

la condizione dice che sono interessato solo a quei documenti in cui la data di revisione di review_date nella tabella originale è più attuale della nostra ultima data di revisione.

Ora, notate per un attimo quanta libertà vi lascia questo, perché qui, se si vuole avere una logica più sofisticata, come lavorare su un timore aggiornato o su qualsiasi cosa che abbia a che fare con gli ID, o altro.

```
(dbt-env) C:\Users\MarcoPozzan-REGOLOFA\datsat25>dbt run
21:23:12 Running with dbt=1.9.1
21:23:13 Registered adapter: fabric=1.9.0
21:23:13 Found 6 models, 469 macros
21:23:13
21:23:13 Concurrency: 4 threads (target='dev')
21:23:13
21:23:15 1 of 6 START sql view model dev.src_hosts ..... [RUN]
21:23:15 2 of 6 START sql view model dev.src_listings ..... [RUN]
21:23:15 3 of 6 START sql view model dev.src_reviews ..... [RUN]
21:23:15 2 of 6 OK created sql view model dev.src_listings ..... [OK in 0.46s]
21:23:15 4 of 6 START sql table model dev.dim_listings_cleansed ..... [RUN]
21:23:16 1 of 6 OK created sql view model dev.src_hosts ..... [OK in 1.22s]
21:23:16 5 of 6 START sql table model dev.dim_hosts_cleansed ..... [RUN]
21:23:16 3 of 6 OK created sql view model dev.src_reviews ..... [OK in 1.44s]
21:23:16 6 of 6 START sql incremental model dev.fct_reviews ..... [RUN]
21:23:18 5 of 6 OK created sql table model dev.dim_hosts_cleansed ..... [OK in 1.85s]
21:23:18 4 of 6 OK created sql table model dev.dim_listings_cleansed ..... [OK in 2.62s]
21:23:23 6 of 6 OK created sql incremental model dev.fct_reviews ..... [OK in 7.36s]
21:23:23
21:23:23 Finished running 1 incremental model, 2 table models, 3 view models in 0 hours 0 minutes and 10.42 seconds (10.42s
).
21:23:24
21:23:24 Completed successfully
21:23:24
21:23:24 Done. PASS=6 WARN=0 ERROR=0 SKIP=0 TOTAL=6
```



```
1 SELECT count(*)
2 FROM [airbnb].[dev].[fct_reviews]
3 where listing_id = 3176
```

Messages		Results	
	123	untitled1	
1	147		

INSERT INTO RAW.raw_reviews

VALUES (3176, CURRENT_TIMESTAMP, 'Zoltan', 'excellent stay!', 'positive')



```
(dbt-env) C:\Users\MarcoPozzan-REGOLOFA\datasat25>dbt run
21:36:47 Running with dbt=1.9.1
21:36:47 Registered adapter: fabric=1.9.0
21:36:48 Found 6 models, 469 macros
21:36:48
21:36:48 Concurrency: 4 threads (target='dev')
21:36:48
21:36:49 1 of 6 START sql view model dev.src_hosts ..... [RUN]
21:36:49 2 of 6 START sql view model dev.src_listings ..... [RUN]
21:36:49 3 of 6 START sql view model dev.src_reviews ..... [RUN]
21:36:51 3 of 6 OK created sql view model dev.src_reviews ..... [OK in 1.45s]
21:36:51 2 of 6 OK created sql view model dev.src_listings ..... [OK in 1.47s]
21:36:51 4 of 6 START sql incremental model dev.fct_reviews ..... [RUN]
21:36:51 5 of 6 START sql table model dev.dim_listings_cleansed ..... [RUN]
21:36:51 1 of 6 OK created sql view model dev.src_hosts ..... [OK in 1.63s]
21:36:51 6 of 6 START sql table model dev.dim_hosts_cleansed ..... [RUN]
21:36:53 6 of 6 OK created sql table model dev.dim_hosts_cleansed ..... [OK in 2.16s]
21:36:53 5 of 6 OK created sql table model dev.dim_listings_cleansed ..... [OK in 2.43s]
21:36:58 4 of 6 OK created sql incremental model dev.fct_reviews ..... [OK in 7.28s]
21:36:58
21:36:58 Finished running 1 incremental model, 2 table models, 3 view models in 0 hours 0 minutes and 10.12 seconds (10.12s
).
21:36:58
21:36:58 Completed successfully
21:36:58
21:36:58 Done. PASS=6 WARN=0 ERROR=0 SKIP=0 TOTAL=6
```



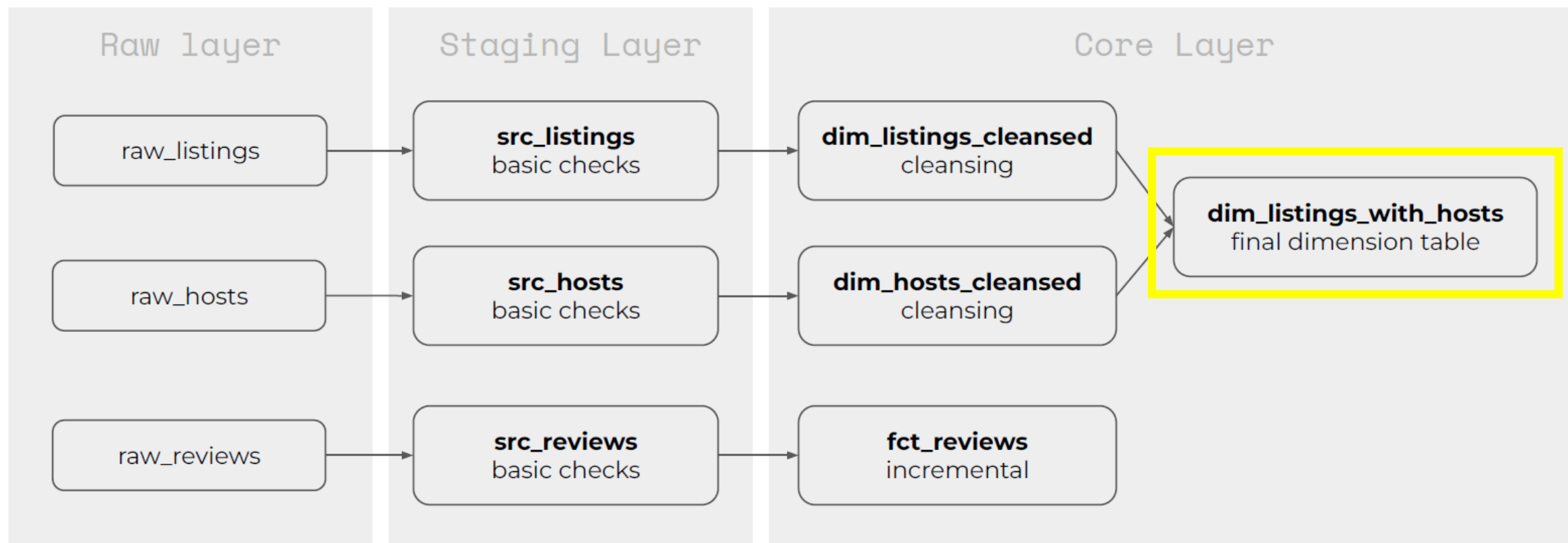
```
1 SELECT count(*)
2 FROM [airbnb].[dev].[fct_reviews]
3 where listing_id = 3176
```

Messages Results

123	untitled1
1	148

Se vogliamo che tutte le tabelle incrementali vengano caricate da zero con un full refresh posso mandare il comando qui sotto

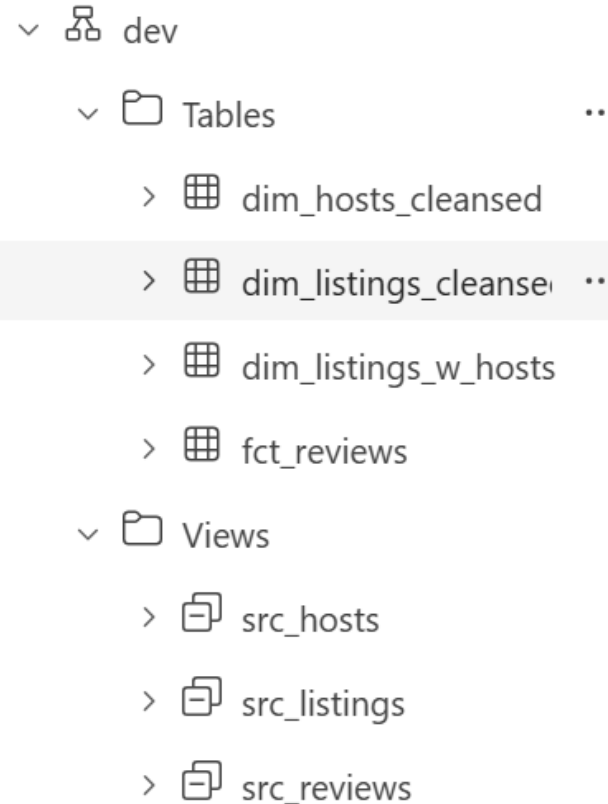
```
(dbt-env) C:\Users\MarcoPozzan-REGOLOFA\datasat25>dbt run --full refresh
```



dim/dim_listings_w_hosts.sql

```
WITH
l AS (
  SELECT
    *
  FROM
    {{ ref('dim_listings_cleansed') }}
),
h AS (
  SELECT *
  FROM {{ ref('dim_hosts_cleansed') }}
)

SELECT
  l.listing_id,
  l.listing_name,
  l.room_type,
  l.minimum_nights,
  l.price,
  l.host_id,
  h.host_name,
  h.is_superhost as host_is_superhost,
  l.created_at,
  GREATEST(l.updated_at, h.updated_at) as updated_at
FROM l
LEFT JOIN h ON (h.host_id = l.host_id)
```



DEMO – CARICAMENTO FILE SEED NEL DWH

Quando si fa modellazione, i dati di input possono provenire da due fonti.

- O da un qualsiasi strumento ETL personalizzato o da un intero processo,
- oppure è un set di dati più piccolo che si desidera inviare dal proprio laptop direttamente al data warehouse.
(seed)

Basta copiare il file nella directory Seed e verranno caricati nel DWH








```
# found in the "models/" directory. You probably won't need to change these!  
model-paths: ["models"]  
analysis-paths: ["analyses"]  
test-paths: ["tests"]  
seed-paths: ["seeds"]  
macro-paths: ["macros"]  
snapshot-paths: ["snapshots"]
```

Poi si esegue il comando per caricare i seed

```
(dbt-env) C:\Users\MarcoPozzan-REGOLOFA\datasat25>dbt seed|
```



```
(dbt-env) C:\Users\MarcoPozzan-REGOLOFA\datasat25>dbt seed
22:37:30 Running with dbt=1.9.1
22:37:30 Registered adapter: fabric=1.9.0
22:37:31 Found 7 models, 1 seed, 469 macros
22:37:31
22:37:31 Concurrency: 4 threads (target='dev')
22:37:31
22:37:38 1 of 1 START seed file dev.seed_full_moon_dates ..... [RUN]
22:37:47 1 of 1 OK loaded seed file dev.seed_full_moon_dates ..... [INSERT 272 in 8.87s]
22:37:47
22:37:47 Finished running 1 seed in 0 hours 0 minutes and 16.34 seconds (16.34s).
22:37:47
22:37:47 Completed successfully
22:37:47
22:37:47 Done. PASS=1 WARN=0 ERROR=0 SKIP=0 TOTAL=1
```

- ✓  dev
 - ✓  Tables
 - >  dim_hosts_cleansed
 - >  dim_listings_cleansed
 - >  dim_listings_w_hosts ...
 - >  fct_reviews
 - >  seed_full_moon_dates

Mart

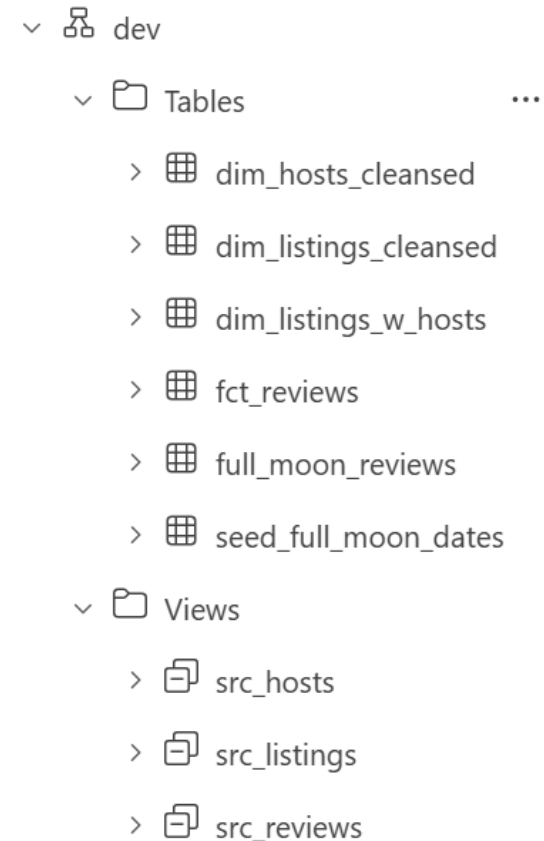
(Ovvero viste e tabelle visibili
ai BI Tools)

models/mart/full_moon_reviews.sql

```
{{ config(
  materialized = 'table',
) }}

WITH fct_reviews AS (
  SELECT * FROM {{ ref('fct_reviews') }}
),
full_moon_dates AS (
  SELECT * FROM {{ ref('seed_full_moon_dates') }}
)

SELECT
  r.*,
  CASE
    WHEN fm.full_moon_date IS NULL THEN 'not full moon'
    ELSE 'full moon'
  END AS is_full_moon
FROM
  fct_reviews
  r
  LEFT JOIN full_moon_dates
    fm
  ON ((CAST(r.review_date AS Date) = DATEADD(DAY, 1, fm.full_moon_date)))
```



TEST

- Esistono due tipi di test: singolari e generici
- I **test singolari** sono query SQL archiviate in test che dovrebbero restituire un set di risultati vuoto
- Sono disponibili quattro test **generici integrati**:
 - unique
 - non_nullo
 - valori_accettati
 - Relazioni
- È possibile definire i propri test generici personalizzati o importare test da pacchetti dbt (**schema.yml**)

models/schema.yml

```
version: 2

models:
  Add documentation or tests
  - name: dim_listings_cleansed
    description: description: Cleansed table which contains Airbnb listings.
    columns:
      - name: listing_id
        description: Primary key for the listing
        tests:
          - not_null

      - name: room_type
        tests:
          - accepted_values:
              values: ['Entire home/apt',
                    'Private room',
                    'Shared room',
                    'Hotel room']
              , '' ]
```

```
(dbt-env) C:\Users\MarcoPozzan-REGOLOFA\datasat25>dbt test
```

```
(dbt-env) C:\Users\MarcoPozzan-REGOLOFA\datsat25>dbt test
14:48:01 Running with dbt=1.9.1
14:48:02 Registered adapter: fabric=1.9.0
14:48:02 Found 8 models, 1 seed, 1 test, 469 macros
14:48:02
14:48:02 Concurrency: 4 threads (target='dev')
14:48:02
14:48:04 1 of 1 START test accepted_values_dim_listings_cleansed_room_type__Entire_home_apartment__Private_room__Shared_room__Hotel_room__ [RUN]
14:48:05 1 of 1 PASS accepted_values_dim_listings_cleansed_room_type__Entire_home_apartment__Private_room__Shared_room__Hotel_room__ [PASS in 1.25s]
14:48:05
14:48:05 Finished running 1 test in 0 hours 0 minutes and 2.78 seconds (2.78s).
14:48:05
14:48:05 Completed successfully
14:48:05
14:48:05 Done. PASS=1 WARN=0 ERROR=0 SKIP=0 TOTAL=1
```



schema.yml



DEMO – TEST (SINGULAR)

tests/dim_listings_mininum_nights.sql

```
SELECT
  *
FROM
  {{ ref('dim_listings_cleansed') }}
WHERE minimum_nights < 1
```

```
(dbt-env) C:\Users\MarcoPozzan-REGOLOFA\datasat25>dbt test
```



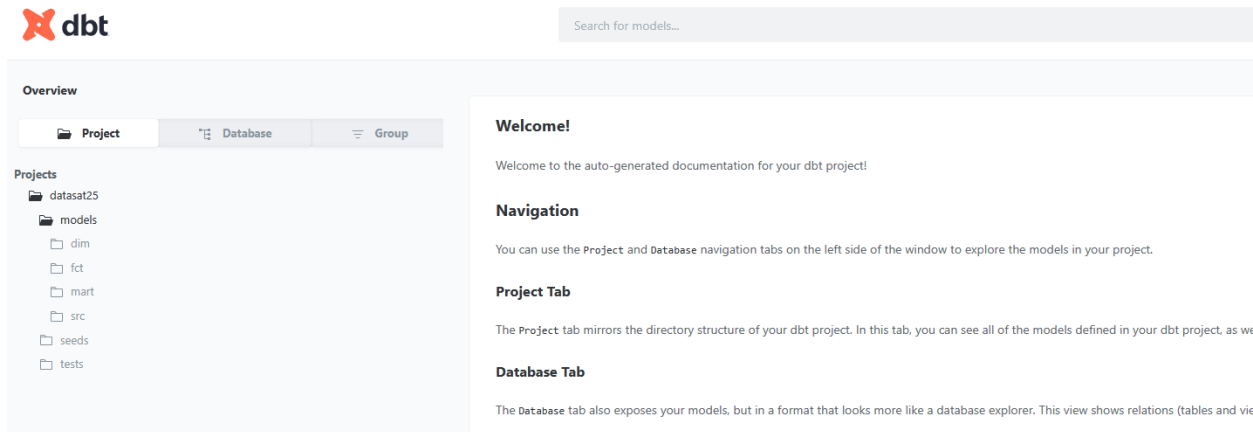
```
(dbt-env) C:\Users\MarcoPozzan-REGOLOFA\datsat25>dbt test
14:48:01 Running with dbt=1.9.1
14:48:02 Registered adapter: fabric=1.9.0
14:48:02 Found 8 models, 1 seed, 1 test, 469 macros
14:48:02
14:48:02 Concurrency: 4 threads (target='dev')
14:48:02
14:48:04 1 of 1 START test accepted_values_dim_listings_cleansed_room_type__Entire_home_apartment__Private_room__Shared_room__Hotel_room__ [RUN]
14:48:05 1 of 1 PASS accepted_values_dim_listings_cleansed_room_type__Entire_home_apartment__Private_room__Shared_room__Hotel_room__ [PASS in 1.25s]
14:48:05
14:48:05 Finished running 1 test in 0 hours 0 minutes and 2.78 seconds (2.78s).
14:48:05
14:48:05 Completed successfully
14:48:05
14:48:05 Done. PASS=1 WARN=0 ERROR=0 SKIP=0 TOTAL=1
```

- Si esegue il comando **dbt docs generate**

```
(dbt-env) C:\Users\MarcoPozzan-REGOLOFA\datasat25>dbt docs generate
20:07:18 Running with dbt=1.9.1
20:07:19 Registered adapter: fabric=1.9.0
20:07:19 Found 8 models, 1 seed, 3 data tests, 469 macros
20:07:19
20:07:19 Concurrency: 4 threads (target='dev')
20:07:19
20:07:21 Building catalog
20:07:22 Catalog written to C:\Users\MarcoPozzan-REGOLOFA\datasat25\target\catalog.json

(dbt-env) C:\Users\MarcoPozzan-REGOLOFA\datasat25>|
```

- Si esegue il comando **dbt docs serve**





GRAZIE