Contents

Guida Completa a dbt (Data Build Tool): Dalle Basi all'Analytics Engineering Avanzato

Benvenuto in questa guida completa a **dbt (Data Build Tool)**. Se lavori con i dati, probabilmente hai sentito parlare di dbt come della tecnologia che ha rivoluzionato il modo in cui i team di analisi trasformano le informazioni.

Questa guida è stata progettata per trasformarti da un principiante a un professionista competente, capace di implementare flussi di lavoro di *Analytics Engineering* robusti, testati e documentati.

---

1. Introduzione: Cos'è dbt e perché è importante

Cos'è dbt?

dbt (Data Build Tool) è un framework di sviluppo open-source che consente ai data analyst e agli ingegneri di trasformare i dati all'interno del proprio data warehouse semplicemente scrivendo istruzioni `SELECT` in SQL.

In termini tecnici, dbt si occupa della parte di **Trasformazione** nel processo **ELT** (Extract, Load, Transform). A differenza dei vecchi strumenti ETL (Extract, Transform, Load) che trasformavano i dati *prima* di caricarli, dbt lavora direttamente sopra il tuo database (Snowflake, BigQuery, Redshift, Databricks, PostgreSQL, ecc.), sfruttando la loro potenza di calcolo.

Perché è diventato lo standard del settore?

Prima di dbt, la trasformazione dei dati era spesso un caos di script SQL manuali, procedure stoccate (stored procedures) difficili da mantenere o complessi strumenti "drag-and-drop".

dbt introduce le **best practice dell'ingegneria del software** nel mondo degli analytics: * **Version Control:** Tutto il codice è su Git. * **Modularità:** Puoi scomporre query enormi in piccoli blocchi riutilizzabili. * **Testing:** Puoi testare automaticamente se i tuoi dati sono corretti. * **Documentazione:** Genera automaticamente siti web che spiegano cosa significano le tue tabelle. * **Ambienti:** Gestione nativa di ambienti di Sviluppo, Staging e Produzione.

2. Concetti Chiave

Per padroneggiare dbt, devi comprendere i suoi pilastri fondamentali.

A. I Modelli (Models)

In dbt, un **modello** è semplicemente un file `.sql` che contiene una singola istruzione `SELECT`. Non scriverai mai `CREATE TABLE` o `INSERT`. dbt si occupa di avvolgere la tua `SELECT` nel codice necessario per creare una tabella o una vista.

B. La funzione `ref()`

È il concetto più importante. Invece di scrivere nomi di tabelle hard-coded (es. `FROM mio_database.schema.tabella`), userai la funzione `{{ ref('nome_modello') }}`. Questo permette a dbt di: 1. Capire le dipendenze tra i modelli (Lineage). 2. Costruire i modelli nell'ordine corretto. 3. Cambiare database/schema automaticamente a seconda dell'ambiente (sviluppo o produzione).

C. Materializzazioni (Materializations)

Definiscono come il modello viene salvato nel database. Le principali sono: * **View (Vista):** (Default) Non occupa spazio, la query viene eseguita ogni volta che la interroghi. * **Table (Tabella):** I dati vengono fisicamente scritti sul disco. Più veloce da interrogare, ma richiede tempo per essere costruita. * **Incremental:** Aggiunge solo i nuovi dati ai dati esistenti. Fondamentale per dataset enormi. * **Ephemeral:** Il modello non viene creato nel database, ma viene usato come una "Common Table Expression" (CTE) nei modelli che lo richiamano.

D. Jinja e Macro

dbt usa **Jinja**, un linguaggio di templating per Python. Ti permette di usare logica di programmazione (cicli `for`, istruzioni `if`, variabili) dentro il tuo SQL. Le **Macro** sono funzioni SQL riutilizzabili, simili alle funzioni nei linguaggi di programmazione.

E. Test e Documentazione

dbt permette di definire test sui dati in un file `.yml`. * **Test generici:** `unique` (univocità), `not_null` (nessun valore nullo), `accepted_values` (valori ammessi), `relationships` (integrità referenziale). * **Test singolari:** Query SQL personalizzate che, se restituiscono record, indicano un errore.

3. Guida Pratica: Creare il tuo primo progetto

In questa sezione vedremo come configurare e far girare un progetto dbt.

1. Inizializzazione

Dopo aver installato dbt tramite `pip`, esegui:

```
dbt init mio_progetto_dati
```

Questo comando creerà la struttura delle cartelle necessaria.

2. Configurazione delle Sorgenti (`sources.yml`)

Prima di trasformare i dati, dobbiamo dire a dbt dove si trovano i dati grezzi (raw data). Creiamo un file in `models/sources.yml`:

```yaml
version: 2

sources:
  - name: shopify
    database: raw_database
    schema: shopify_data
    tables:
      - name: orders
      - name: customers
```

3. Creazione del primo modello (Staging)

Creiamo un file `models/staging/stg_orders.sql`. Qui puliamo i dati grezzi.

```sql
with orders as (
    select
        id as order_id,
        user_id as customer_id,
        order_date,
        status as order_status
    from {{ source('shopify', 'orders') }}
)

select * from orders
```

4. Creazione di un modello trasformato (Marts)

Creiamo `models/marts/fct_orders.sql`. Qui uniamo i dati per il business usando `ref()`.

```sql
with orders as (
    select * from {{ ref('stg_orders') }}
),

customers as (
    select * from {{ ref('stg_customers') }}
),

final as (
    select
        orders.order_id,
        customers.customer_id,
```

```
        orders.order_date,
        orders.order_status,
        customers.first_name,
        customers.last_name
    from orders
    left join customers using (customer_id)
)

select * from final
```

5. Aggiungere Test e Documentazione

Creiamo `models/marts/schema.yml`:

```
version: 2

models:
  - name: fct_orders
    description: "Tabella dei fatti che contiene tutti gli ordini completati."
    columns:
      - name: order_id
        description: "Chiave primaria dell'ordine"
        tests:
          - unique
          - not_null
      - name: order_status
        tests:
          - accepted_values:
              values: ['placed', 'shipped', 'completed', 'returned']
```

6. Esecuzione

Per compilare ed eseguire i modelli nel database:

```
dbt run
```

Per eseguire i test definiti:

```
dbt test
```

Per generare la documentazione:

```
dbt docs generate
dbt docs serve
```

4. Usi Avanzati e Best Practice

A. Analisi dell'impatto (Lineage)

Quando la documentazione viene generata, dbt crea un grafico delle dipendenze (DAG). Questo è vitale per capire: "Se cambio questa colonna nella tabella grezza, quali dashboard si romperanno?".

B. Gestione degli Snapshot

dbt offre una funzionalità chiamata **Snapshots** per gestire le *Slowly Changing Dimensions* (SCD). Ad esempio, se vuoi tenere traccia di come cambia lo stato di un ordine nel tempo (e non solo vedere lo stato attuale), dbt può creare automaticamente una tabella storica.

C. Utilizzo delle Macro

Se devi ripetere la stessa logica SQL (es. convertire valute o pulire stringhe) in 20 modelli diversi, scrivi una Macro in `macros/valuta.sql`:

```
{% macro converti_a_euro(colonna_prezzo, tasso_cambio) %}
    ({{ colonna_prezzo }} * {{ tasso_cambio }}) as prezzo_euro
{% endmacro %}
```

E usala nel tuo modello:

```
select
    id,
    {{ converti_a_euro('prezzo_usd', 0.92) }}
from {{ ref('ordini') }}
```

D. Best Practice di Strutturazione

Un progetto professionale dbt è solitamente diviso in tre livelli: 1. **Staging:** Pulizia minima, ridenominazione colonne, cast di tipi. Un modello per ogni tabella sorgente. 2. **Intermediate:** Logica complessa, aggregazioni intermedie. 3. **Marts:** Dati pronti per essere consumati dagli strumenti di BI (Tableau, PowerBI, Looker). Divisi per area di business (Marketing, Finance, Sales).

---

5. Casi d'Uso: Quando usare dbt

1. Migrazione a un Modern Data Warehouse

Se stai passando da un database on-premise a Snowflake o BigQuery, dbt è lo strumento ideale per ricostruire la tua logica di business in modo pulito e documentato.

2. Team di Analytics in crescita

Quando più persone lavorano sugli stessi dati, dbt impedisce che ognuno crei la propria "versione della verità". Centralizza la logica di business.

3. Data Quality Assurance

Se hai problemi di dati inconsistenti o duplicati, i test automatizzati di dbt bloccano i dati errati prima che arrivino alle dashboard dei decision-maker.

4. Automazione del Catalogo Dati

Per le aziende che devono rispettare normative (come il GDPR), dbt genera automaticamente la documentazione del lineage dei dati, mostrando esattamente da dove proviene ogni informazione.

---

6. Conclusione

dbt ha cambiato le regole del gioco perché ha capito che il problema principale dei dati non è più lo spazio di archiviazione o la velocità di calcolo, ma la **complessità del codice** e la **fiducia nei dati**.

Utilizzando dbt, smetti di essere un semplice "scrittore di query" e diventi un **Analytics Engineer**, un professionista che applica il rigore del software engineer al mondo dei dati.

I prossimi passi consigliati: 1. Installa **dbt Core** localmente. 2. Connettilo a un database di prova (PostgreSQL o Google BigQuery Sandbox). 3. Prova a convertire un vecchio script SQL manuale in modelli dbt modulari. 4. Esplora **dbt Cloud** se cerchi un'interfaccia web e un orchestratore integrato.

I dati sono il petrolio del futuro, ma dbt è la raffineria che li rende utilizzabili. Buona trasformazione!