Contents

Guida Pratica a dbt: Percorso di Esercizi da Principiante ad Esperto **1**
1. Introduzione . . . 2
   - Cos'è dbt? . . . 2
   - Perché seguire questo percorso? . . . 2
   - Prerequisiti per gli esercizi . . . 2
2. Concetti Chiave da Ricordare . . . 2
3. Livello 1: Le Fondamenta (Setup e Modellazione) . . . 3
   - Esercizio 1: "Hello World" - Setup del Progetto . . . 3
   - Esercizio 2: Definizione delle Sources . . . 3
   - Esercizio 3: Staging Models (Pulizia Base) . . . 3
   - Esercizio 4: Il primo Fact Model (Marts) . . . 4
4. Livello 2: Qualità e Scalabilità (Intermedio) . . . 5
   - Esercizio 5: Testing (La cintura di sicurezza) . . . 5
   - Esercizio 6: Documentazione . . . 5
   - Esercizio 7: Introduzione a Jinja (DRY - Don't Repeat Yourself) . . . 5
   - Esercizio 8: Creazione di una Macro personalizzata . . . 5
5. Livello 3: Tecniche Avanzate . . . 7
   - Esercizio 9: Materializzazione Incrementale . . . 7
   - Esercizio 10: Snapshots (SCD Type 2) . . . 7
   - Esercizio 11: Utilizzo di Pacchetti (dbt Packages) . . . 8
6. Casi d'Uso e Best Practices . . . 9
   - Quando usare cosa? . . . 9
   - Struttura consigliata di un progetto . . . 9
7. Conclusione . . . 9
   - Prossimi Passi . . . 9

Guida Pratica a dbt: Percorso di Esercizi da Principiante ad Esperto

Questa guida è strutturata come un percorso formativo pratico ("workbook") per padroneggiare **dbt (data build tool)**. A differenza dei tutorial passivi, questo documento richiede di scrivere codice e risolvere problemi reali che un Analytics Engineer affronta quotidianamente.

Il percorso è diviso in **3 Livelli**: 1. **Base**: Fondamenta, setup e modellazione semplice. 2. **Intermedio**: Testing, documentazione, Jinja e macro. 3. **Avanzato**: Materializzazioni incrementali, snapshot e gestione dei pacchetti.

1. Introduzione

Cos'è dbt?

dbt è uno strumento di trasformazione dei dati che permette agli analisti e agli ingegneri di trasformare i dati nel loro warehouse scrivendo semplici istruzioni `SELECT`. dbt gestisce il processo di trasformazione da "grezzo" a "pronto per l'analisi" (la "T" in ELT).

Perché seguire questo percorso?

La teoria non basta. dbt è uno strumento che si basa fortemente su convenzioni e pattern. Eseguire questi esercizi ti permetterà di sviluppare la "memoria muscolare" necessaria per strutturare progetti scalabili, comprendere il grafo delle dipendenze (DAG) e utilizzare la potenza di Jinja.

Prerequisiti per gli esercizi

Per svolgere gli esercizi, avrai bisogno di: 1. **Conoscenza di SQL** (SELECT, JOIN, GROUP BY). 2. **Python installato** (per installare dbt-core). 3. **Un Data Warehouse**: Puoi usare un account gratuito di Snowflake, BigQuery, o per semplicità locale, **DuckDB** o **PostgreSQL**. 4. **Dataset di Esempio**: Useremo concettualmente lo schema "Jaffle Shop" (Clienti, Ordini, Pagamenti).

---

2. Concetti Chiave da Ricordare

Prima di iniziare, tieni a mente questi quattro pilastri:

Concetto	Descrizione
Model	Un singolo file `.sql` che contiene una query `SELECT`. dbt lo materializza come tabella o vista.
Materialization	La strategia con cui dbt salva il modello nel DB (table, view, incremental, ephemeral).
Ref()	La funzione più importante. Sostituisce il nome della tabella fisica creando una dipendenza nel DAG. Esempio: `{{ ref('stg_customers') }}`.
Source	Riferimento ai dati grezzi caricati nel warehouse da strumenti esterni (Fivetran, Airbyte, script).

3. Livello 1: Le Fondamenta (Setup e Modellazione)

In questa fase, imparerai a configurare un progetto e a creare la struttura base: *Sources -> Staging -> Marts*.

Esercizio 1: "Hello World" - Setup del Progetto

Obiettivo: Installare dbt e inizializzare un progetto vuoto.

1. Crea una cartella `dbt_bootcamp`.
2. Crea un ambiente virtuale Python e installa l'adattatore per il tuo DB (es. `pip install dbt-duckdb` o `dbt-postgres`).
3. Esegui `dbt init my_project`.
4. Configura il file `profiles.yml` (se non guidato automaticamente) per connetterti al database.
5. Esegui `dbt debug` per verificare la connessione.

Criterio di successo: Il comando `dbt debug` restituisce "All checks passed!".

Esercizio 2: Definizione delle Sources

Scenario: Hai tre tabelle grezze nel tuo database (schema `raw`): `raw_customers`, `raw_orders`, `raw_payments`. **Obiettivo:** Dire a dbt dove si trovano i dati grezzi.

1. Crea un file `models/staging/src_jaffle_shop.yml`.
2. Definisci la source `jaffle_shop` con le tre tabelle.

Codice Suggerito (YAML):

```yaml
version: 2

sources:
  - name: jaffle_shop
    database: raw  # O il nome del tuo DB
    schema: public # O il tuo schema
    tables:
      - name: customers
      - name: orders
```

Esercizio 3: Staging Models (Pulizia Base)

Obiettivo: Creare modelli di staging (vista 1:1 con la source) per rinominare colonne e castare tipi di dati.

1. Crea `models/staging/stg_customers.sql`.
2. Usa la funzione `{{ source('jaffle_shop', 'customers') }}`.
3. Seleziona ID, nome e cognome. Rinomina `id` in `customer_id`.

Codice Suggerito (SQL):

```sql
with source as (
    select * from {{ source('jaffle_shop', 'customers') }}
),
renamed as (
    select
        id as customer_id,
        first_name,
        last_name
    from source
)
select * from renamed
```

4. Esegui `dbt run`. Verifica che la vista sia stata creata nel DB.

Esercizio 4: Il primo Fact Model (Marts)

Scenario: Il business vuole una tabella `dim_customers` che mostri il primo ordine e l'ordine più recente per ogni cliente. **Obiettivo:** Usare `ref()` per unire i modelli di staging.

1. Crea `models/marts/core/dim_customers.sql`.
2. Fai una CTE che seleziona da `{{ ref('stg_customers') }}`.
3. Fai una CTE che seleziona da `{{ ref('stg_orders') }}` (crealo se non l'hai fatto, simile all'Es 3).
4. Esegui una LEFT JOIN tra clienti e un'aggregazione degli ordini (MIN data, MAX data, COUNT ordini).

Concetto chiave: Non scrivere mai `FROM raw.public.customers`. Usa sempre `ref` o `source`.

4. Livello 2: Qualità e Scalabilità (Intermedio)

Ora che i dati fluiscono, dobbiamo assicurarci che siano corretti e che il codice sia mantenibile.

Esercizio 5: Testing (La cintura di sicurezza)

Obiettivo: Impedire che dati sporchi rompano la pipeline.

1. Apri (o crea) il file `models/marts/core/schema.yml`.
2. Aggiungi test generici (`unique`, `not_null`) alle chiavi primarie dei modelli creati.
3. Aggiungi un test `accepted_values` sullo stato degli ordini (es. 'placed', 'shipped', 'completed', 'return_pending', 'returned').

Codice Suggerito (YAML):

```yaml
models:
  - name: stg_orders
    columns:
      - name: order_id
        tests:
          - unique
          - not_null
      - name: status
        tests:
          - accepted_values:
              values: ['placed', 'shipped', 'completed', 'returned']
```

4. Esegui `dbt test`.
5. *Bonus:* Inserisci intenzionalmente un dato duplicato nel raw data e vedi il test fallire.

Esercizio 6: Documentazione

Obiettivo: Generare un catalogo dati automatico.

1. Nello `schema.yml`, aggiungi il campo `description` per le tabelle e le colonne principali.
2. Esegui `dbt docs generate`.
3. Esegui `dbt docs serve`.
4. Esplora il sito locale generato e visualizza il **Lineage Graph** (il grafo delle dipendenze).

Esercizio 7: Introduzione a Jinja (DRY - Don't Repeat Yourself)

Scenario: Hai bisogno di convertire i centesimi in dollari/euro in molte tabelle. **Obiettivo:** Usare una logica inline con Jinja.

1. Modifica un modello (es. `stg_payments.sql`).
2. Invece di scrivere `amount / 100`, usa una variabile o matematica semplice Jinja.
3. Esempio base: `sql select order_id, {{ 100 + 50 }} as calculated_value, -- Esempio banale amount / 100 as amount_usd from ...`

Esercizio 8: Creazione di una Macro personalizzata

Scenario: La conversione da centesimi a valuta principale è usata ovunque. Creiamo una funzione riutilizzabile. **Obiettivo:** Scrivere una macro.

1. Crea un file `macros/cents_to_dollars.sql`.
2. Definisci la macro:

```
{% macro cents_to_dollars(column_name, decimal_places=2) %}
round( 1.0 * {{ column_name }} / 100, {{ decimal_places }})
{% endmacro %}
```

3. Torna nel modello `stg_payments.sql` e usala: `sql select payment_id, {{ cents_to_dollars('amount') }} as amount_usd from ...`
4. Esegui `dbt run` e controlla il codice SQL compilato nella cartella `target/compiled/`.

5. Livello 3: Tecniche Avanzate

Qui separiamo i principianti dagli esperti. Tratteremo performance e storicizzazione.

Esercizio 9: Materializzazione Incrementale

Scenario: La tabella `raw_events` ha milioni di righe. Ricostruire la tabella (`table` materialization) ogni volta è troppo lento e costoso. **Obiettivo:** Configurare un modello incrementale che processi solo i nuovi dati.

1. Crea `models/marts/core/fct_events.sql`.
2. Configura la materializzazione all'inizio del file.

Codice Suggerito (SQL):

```
{{
    config(
        materialized='incremental',
        unique_key='event_id'
    )
}}

select
    event_id,
    event_time,
    event_type
from {{ ref('stg_events') }}

{% if is_incremental() %}
  -- Questa clausola viene eseguita solo nelle run successive alla prima
  where event_time > (select max(event_time) from {{ this }})
{% endif %}
```

3. Esegui `dbt run`. La prima volta creerà la tabella.
4. Aggiungi nuovi dati fittizi alla source.
5. Esegui `dbt run` di nuovo. Controlla i log: dovresti vedere che ha processato solo le nuove righe.

Esercizio 10: Snapshots (SCD Type 2)

Scenario: Gli ordini cambiano stato (`status`) nel tempo, ma il database sovrascrive il valore. Il business vuole sapere quanto tempo un ordine è rimasto in stato "shipped". **Obiettivo:** Tracciare la storia dei cambiamenti (Slowly Changing Dimensions).

1. Crea un file `snapshots/orders_snapshot.sql`.

Codice Suggerito (SQL):

```
{% snapshot orders_snapshot %}

{{
    config(
      target_database='analytics',
      target_schema='snapshots',
      unique_key='order_id',

      strategy='timestamp',
      updated_at='updated_at',
    )
```

```
}}

select * from {{ source('jaffle_shop', 'orders') }}

{% endsnapshot %}
```

Nota: Se non hai una colonna ***updated_at****, puoi usare la strategia* ***check*** *specificando le colonne da monitorare.*

2. Esegui `dbt snapshot`.
3. Modifica un record nel database raw (cambia lo status di un ordine).
4. Esegui di nuovo `dbt snapshot`.
5. Interroga la tabella `snapshots.orders_snapshot`. Vedrai due righe per quell'ordine: una con `dbt_valid_to` valorizzato (la vecchia versione) e una con `dbt_valid_to` NULL (la versione corrente).

Esercizio 11: Utilizzo di Pacchetti (dbt Packages)

Scenario: Hai bisogno di funzioni complesse (es. generare un surrogate key o una pivot table) ma non vuoi scriverle da zero. **Obiettivo:** Installare e usare `dbt-utils`.

1. Crea un file `packages.yml` nella root del progetto.
2. Aggiungi: `yaml packages: - package: dbt-labs/dbt_utils version: 1.1.1`
3. Esegui `dbt deps`.
4. Usa una funzione del pacchetto in un modello. Esempio: generare una chiave univoca basata su più colonne.

```
select
    {{ dbt_utils.generate_surrogate_key(['customer_id', 'order_date']) }} as id_univoco,
    *
from {{ ref('stg_orders') }}
```

6. Casi d'Uso e Best Practices

Dopo aver completato gli esercizi, ecco come applicare queste conoscenze nel mondo reale:

Quando usare cosa?

1. **View vs Table**: Usa le **View** per i modelli di staging (leggeri, 1:1). Usa le **Table** per i Marts (modelli finali ad alto valore che vengono interrogati spesso da BI tool come Tableau/PowerBI).
2. **Ephemeral**: Usa la materializzazione **Ephemeral** per blocchi di logica intermedi molto complessi che non vuoi inquinino il database come viste, ma che servono solo per pulire il codice (funzionano come una CTE iniettata).
3. **Seeds**: Usa i **Seeds** (file CSV nella cartella `seeds/`) per dati statici di piccole dimensioni, come tabelle di decodifica (es. codici paese -> nome paese) o target di vendita manuali. Esegui `dbt seed` per caricarli.

Struttura consigliata di un progetto

Una struttura caotica è il nemico numero uno. Segui questa gerarchia:

- `staging/`:
  - Una cartella per ogni sistema sorgente (es. `stripe`, `salesforce`).
  - Solo pulizia leggera, rinomina colonne, casting tipi.
- `intermediate/` (Opzionale):
  - Join complessi, logica di business pesante.
  - Non esposti agli utenti finali.
- `marts/`:
  - Divisi per area di business (`finance`, `marketing`, `sales`).
  - Tabelle pronte per la BI (Fact e Dimension tables).

7. Conclusione

Hai completato il percorso da principiante ad avanzato. Ecco cosa hai imparato:

1. **Setup**: Come inizializzare e connettere dbt.
2. **Il DAG**: Come concatenare i modelli con `ref` invece di query statiche.
3. **Qualità**: Come garantire l'affidabilità dei dati con i test.
4. **Ingegneria**: Come usare Jinja per rendere il codice dinamico e DRY.
5. **Ottimizzazione**: Come gestire grandi volumi di dati con modelli incrementali.
6. **Storicizzazione**: Come tracciare i cambiamenti nel tempo con gli Snapshots.

Prossimi Passi

Per diventare un vero "Analytics Engineer" Senior: * Integra dbt in una pipeline **CI/CD** (GitHub Actions o GitLab CI). * Impara a usare **dbt Cloud** per l'orchestrazione dei job. * Approfondisci **dbt Mesh** per gestire progetti multi-team in grandi aziende.