ChatGPT

7-Day Comprehensive Plan to Learn dbt (Data Build Tool)

This 7-day plan assumes you have basic SQL skills and access to a data warehouse (such as Snowflake, BigQuery, Postgres, etc.). dbt lets you transform, test and document data **inside** your warehouse. It turns SQL scripts into modular, version-controlled and tested pipelines [1]. The plan mixes reading with hands-on practice and examples.

Day 0 – What is dbt? Why It Matters

Goals: Get a conceptual overview of dbt (data build tool), understand the problems it was designed to solve, and see a concrete example showing how dbt improves data workflows compared to traditional approaches.

🔍 Before dbt: Manual, Error-Prone Transformations

Before dbt became popular, analysts often wrote raw SQL scripts or scheduled stored procedures to transform data. These scripts were scattered across repos and dashboards, making it hard to maintain consistent business logic. Teams ran queries manually, exported CSV files, uploaded them into BI tools, and manually managed dependencies between tables [2]. Data quality checks were often done by eye or with ad-hoc queries, leading to inconsistent results [3].

What dbt Brings to the Table

dbt is an open-source tool that specializes in the **transform** step of an ELT pipeline. It lets analysts and engineers write modular SQL files and use them to build and maintain data models in their warehouse. Because dbt handles dependency management, compilation, testing and documentation automatically, it boosts productivity and ensures consistency [4] [5].

Key advantages include: * **Centralized, version-controlled transformations:** dbt stores models as SQL files in a Git repository, so your transformation logic is transparent, reviewable and versioned [6]. * **Automated tests and data quality:** You can define tests (e.g., uniqueness, not null, custom rules) and run them automatically. This catches issues early and reduces manual QA work [3]. * **Dependency management with `ref()`:** Instead of hard-coding table names, you reference other models with `ref()`. dbt builds models in the correct order and updates dependencies automatically [5]. * **Documentation and lineage:** dbt generates a documentation site with DAG diagrams and model descriptions, helping teams understand data lineage and context [7]. * **Collaboration and CI/CD:** Because dbt projects live in Git, you can use pull requests to review changes and set up CI pipelines to run tests on each commit [8].

Example: From Manual Script to dbt Model

Suppose your team needs to build a customer dimension table from a raw `customers` table. In a traditional workflow, you might write a SQL script like:

```
-- manual_transformation.sql
SELECT
  id AS customer_id,
  UPPER(first_name || ' ' || last_name) AS customer_name,
  email,
  created_at::date AS signup_date
FROM raw.customers
WHERE status = 'active';
```

You would run this script manually or schedule it with a cron job. To change the logic (e.g., add a new field), you'd edit the script directly and hope no dependencies break.

With dbt, you create a model file `models/dim_customers.sql`:

```
{{ config(materialized='table') }}
SELECT
  id AS customer_id,
  UPPER(first_name || ' ' || last_name) AS customer_name,
  email,
  DATE(created_at) AS signup_date
FROM {{ source('raw', 'customers') }}
WHERE status = 'active';
```

You define the raw table as a source in `schema.yml` and reference it with `source()`. dbt automatically handles the build order, compiles the SQL into your warehouse's syntax and creates the table. You can then add tests in `schema.yml` to ensure `customer_id` is unique and `email` is not null. When you run `dbt run`, the table is created; `dbt test` runs your tests; and `dbt docs generate` builds documentation. If you change the model, dbt tracks the change and you can review it through a pull request.

Deliverables for Day 0

- Understand the pain points of pre-dbt workflows (manual scripts, scattered logic, lack of testing).
- Recognize how dbt centralizes, automates and documents transformations [6] [5].
- Complete a simple model that transforms raw data using `source()` and `ref()`, with tests and documentation.

Day 1 – Introduction and Setup

Goals: Understand what dbt is, install it and create your first project.

📚 Reading & Concepts

- **What is dbt?** dbt (data build tool) operates on top of your warehouse and enables data analysts/engineers to transform, test and document data [1].
- **Command anatomy:** dbt commands consist of an action (`run`, `test`, `docs`, etc.) plus arguments and operators [9].

🛠 Setup

1. **Install dbt.** Use `pip` or `brew` to install the dbt CLI. Example (for Snowflake adapter):

   ```
   pip install dbt-snowflake
   ```

2. **Set up a warehouse profile.** Create a `profiles.yml` file in `~/.dbt` with your warehouse credentials.
3. **Create a dbt project.**

   ```
   dbt init ecommerce_dbt_project
   cd ecommerce_dbt_project
   ```

   This creates the `models/`, `macros/`, and other directories.
4. **Run a sample model.** In `models/example/`, edit `my_first_dbt_model.sql` to:

   ```
   -- models/example/my_first_dbt_model.sql
   select 1 as id, 'hello dbt' as message
   ```

   Then run:

   ```
   dbt run
   ```

   This materializes the view in your warehouse.
5. **Review the DAG.** Run `dbt docs generate` and `dbt docs serve` to explore the project graph locally [10].

Deliverables

- A functioning dbt project connected to your warehouse.
- Generated documentation accessible via `localhost:8080`.

Day 2 – Modeling with dbt and `ref()`

Goals: Learn how dbt models work, how `ref()` manages dependencies, and start building a simple data model.

📚 Reading & Concepts

- **Models:** SQL files in the `models/` folder are compiled and materialized in the warehouse. Use `{{ config(materialized='table') }}` or `view` to choose the materialization.
- `ref()` **function:** Instead of hard-coding table names, use `ref()` to refer to other models. dbt ensures the correct build order.
- **Jinja templating:** dbt models use SQL plus Jinja templating; you can pass variables and logic.

🛠 Practice

1. **Create staging models.** Inside `models/staging/`, create `stg_customers.sql`:

```
{{ config(materialized='view') }}
select id,
       name,
       email,
       created_at
from {{ source('raw', 'customers') }}
```

   Use the `source()` function to reference raw tables (defined in `schema.yml`).
2. **Create transformation models.** Inside `models/marts/`, create `dim_customers.sql`:

```
{{ config(materialized='table') }}
select
    id as customer_id,
    name as customer_name,
    email,
    {{ dbt_utils.date_trunc('day', 'created_at') }} as signup_date
from {{ ref('stg_customers') }}
```

   This uses the `ref()` function to depend on the staging model.
3. **Run and test the DAG.** Execute `dbt run` to build both models. Inspect the DAG in the generated docs.
4. **Example of Jinja logic.** Use a variable to filter data:

```
{{ config(materialized='table') }}
select *
from {{ ref('stg_orders') }}
where order_date >= '{{ var('start_date', '2025-01-01') }}'
```

   Provide `--vars '{start_date: 2025-10-01}'` at runtime.

Deliverables

- Staging and mart models built with `ref()` and Jinja.

- Understanding of dbt's model compilation and dependency graph.

Day 3 – Seeds and Snapshots

Goals: Use seeds to load static data and snapshots to track slowly changing dimensions.

📚 Reading & Concepts

- **Seeds:** `dbt seed` loads CSV files as tables; they're ideal for small lookup/reference tables [11].
- **Snapshots:** `dbt snapshot` captures changes to a dataset over time (similar to slowly changing dimension Type 2) [12].

🛠 Practice

1. **Create a seed.** In the `data/` directory, add `regions.csv`:

   ```
   region_code,region_name
   US,United States
   CA,Canada
   EU,Europe
   ```

   Configure it in `dbt_project.yml`:

   ```
   data-paths: ["data"]
   ```

   Run `dbt seed` to load it. Query the table in your warehouse.
2. **Use the seed in a model.** Create `dim_region.sql`:

   ```
   select region_code, region_name
   from {{ ref('regions') }}
   ```

3. **Create a snapshot.** Define a snapshot to track customer status changes:

   ```
   -- snapshots/snapshot_customers.sql
   {% snapshot snapshot_customers %}
     {{ config(
         target_schema='snapshots',
         unique_key='id',
         strategy='timestamp',
         updated_at='updated_at'
     ) }}
     select *
     from {{ source('raw', 'customers') }}
   {% endsnapshot %}
   ```

Run `dbt snapshot` to materialize history. Inspect the snapshot table.

Deliverables

- Seed table loaded and referenced in a model.
- Snapshot table capturing historical changes.

Day 4 – Testing and Documentation

Goals: Implement dbt tests and generate documentation for your models.

📚 Reading & Concepts

- **Tests:** `dbt test` runs schema and data tests defined on models, sources, snapshots, and seeds [13].
- **Documentation:** `dbt docs` compiles project metadata into a searchable site [10].

🛠 Practice

1. **Define schema tests.** Create `models/staging/schema.yml` with tests:

```
version: 2
models:
  - name: stg_customers
    columns:
      - name: id
        tests:
          - not_null
          - unique
      - name: email
        tests:
          - not_null
```

2. **Define generic tests.** Use `dbt_utils` for accepted values:

```
- name: dim_customers
  columns:
    - name: region_code
      tests:
        - accepted_values:
            values: ['US','CA','EU']
```

3. **Run tests.** Execute `dbt test --select stg_customers`. Explore results.
4. **Create a data (custom) test.** In `tests/`, create `no_future_orders.sql`:

```
-- tests/no_future_orders.sql
select *
from {{ ref('stg_orders') }}
where order_date > current_date
```

Run `dbt test --select no_future_orders`.

5. **Generate docs.** Write descriptions in your YAML files. Run:

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

Navigate to models, sources, and tests in the docs site.

Deliverables

- Passing schema and custom tests.
- Generated docs with model descriptions and test results.

Day 5 – Macros, Packages and Reusability

Goals: Learn to extend dbt with Jinja macros and packages, and reuse logic across models.

📚 Reading & Concepts

- **Macros:** Custom Jinja functions that produce SQL; use `dbt run-operation` to execute macros [14].
- **Packages:** Reusable collections of models and macros (e.g., `dbt-utils`).

🛠 Practice

1. **Install** `dbt-utils`. Add to `packages.yml`:

```
packages:
  - package: dbt-labs/dbt_utils
    version: 1.1.1
```

Run `dbt deps` to install.

2. **Create a macro.** In `macros/grant_select.sql`:

```
-- macros/grant_select.sql
{% macro grant_select(target_role) %}
  grant select on all tables in schema {{ target.schema }} to role {{
target_role }};
{% endmacro %}
```

3. **Run a macro.** Execute:

```
dbt run-operation grant_select --args '{"target_role": "analyst"}'
```

4. **Use macros in models.** Create `models/utilities/get_latest_date.sql`:

```
{% macro get_latest_date(table_name, date_column) %}
  (select max({{ date_column }}) from {{ table_name }})
{% endmacro %}

-- models/marts/latest_order.sql
{{ config(materialized='table') }}
select
    {{ ref('stg_customers') }}.id,
    ({{ get_latest_date(ref('stg_orders'), 'order_date') }}) as
last_order_date
from {{ ref('stg_customers') }}
```

5. **Explore packages.** Use `dbt-utils` macros like `surrogate_key` or `date_trunc`. Example:

```
select {{ dbt_utils.surrogate_key(['customer_id','order_id']) }} as sk
from {{ ref('stg_orders') }}
```

Deliverables

- Custom macro and demonstration of `dbt run-operation`.
- Models using macros and `dbt_utils` package functions.

Day 6 – Incremental Models, Performance and Build Commands

Goals: Learn to build incremental models to handle large datasets and optimize runs.

📚 Reading & Concepts

- **Incremental models:** Build only new or changed data. Use the `is_incremental()` Jinja function to differentiate logic.
- `dbt build`: Executes models, tests, snapshots and seeds following the dependency graph [15].
- **Graph and set operators:** Use `+` to select ancestors/descendants of a model [16].

🛠 Practice

1. **Create an incremental model.** In `models/marts/fct_orders.sql`:

```
{{ config(materialized='incremental', unique_key='order_id') }}
select *
from {{ ref('stg_orders') }}
{% if is_incremental() %}
  where order_date > (select max(order_date) from {{ this }})
{% endif %}
```

Run `dbt run` twice; note that only new records are appended on the second run.

2. **Use the `full-refresh` option.** Force a rebuild when logic changes:

```
dbt run --select fct_orders --full-refresh
```

3. **Selective builds with operators.**
4. Run a model and all its downstream models: `dbt run --select dim_customers+`.
5. Run only models upstream of `fct_orders`: `dbt run --select +fct_orders`.
6. **Build everything.** Execute `dbt build` to run models, tests, snapshots and seeds together [15].
7. **Performance tips.** Use `threads` configuration to parallelize builds; avoid `select *` in models; test in dev schema before production.

Deliverables

- Incremental model that appends new records.
- Examples of graph operators to run targeted subsets of the DAG.

Day 7 – Advanced Features, Orchestration and Review

Goals: Explore advanced dbt features such as exposures, metrics and CI/CD pipelines. Review what you've learned.

📚 Reading & Concepts

- **Exposures:** Define downstream dashboards or reports that depend on dbt models. Exposures help track data lineage to end-user assets.
- **Metrics (dbt Semantic Layer):** (if using dbt Cloud or dbt Core 1.5+); define business metrics centrally.
- **CI/CD:** Automate dbt runs via GitHub Actions or dbt Cloud jobs.
- **Documentation & community:** Use `dbt docs` to keep technical docs updated; join the dbt community for support.

🛠 Practice

1. **Define an exposure.** In `models/marts/exposures.yml`:

```
version: 2
exposures:
```

```
  - name: customer_lifetime_value_dashboard
    type: dashboard
    maturity: medium
    url: https://your-bi-tool.com/dashboards/123
    depends_on:
      - ref('dim_customers')
      - ref('fct_orders')
    owner:
      name: Data Team
      email: data-team@example.com
```

2. **Create a metric.** (Requires dbt 1.5+ with metrics in `metrics/` directory.)

```
version: 2
metrics:
  - name: total_revenue
    model: ref('fct_orders')
    label: Total Revenue
    calculation_method: sum
    expression: revenue
    timestamp: order_date
    time_grains: [day, month, year]
```

3. **Automate tests in CI.** Add a GitHub Actions workflow to run `dbt build` on every pull request. Example job steps:

```
steps:
  - uses: actions/checkout@v3
  - uses: actions/setup-python@v4
    with:
      python-version: '3.11'
  - run: pip install dbt-core dbt-bigquery dbt-utils
  - run: dbt deps
  - run: dbt build --threads 4
```

4. **Schedule jobs in dbt Cloud.** Create a daily job that runs `dbt build`, sends Slack notifications on failure and uploads docs.
5. **Review & Next steps.** Write a retrospective: What worked well? What concepts remain challenging? Consider exploring dbt's Semantic Layer, query policies or performance features (Mesh, serverless). Continue reading docs and engaging with the community.

Deliverables

- Exposure definitions and metrics files.
- A simple CI workflow file or dbt Cloud job configured.

Additional Resources

- **dbt Cheat Sheet:** The Y42 dbt cheat sheet summarizes commands and options with examples; it notes that `dbt seed` loads CSV files into your warehouse [11], `dbt snapshot` tracks historical changes to datasets [12], and `dbt build` runs models, tests, snapshots and seeds according to dependencies [15].
- **Testing & docs:** dbt tests support schema and data tests and can be filtered with `--select` and `--exclude` [17]; `dbt docs` generates searchable documentation sites [10].
- **dbt fundamentals course:** According to dbt Labs, the "dbt Fundamentals" course is about 5 hours long [18] and covers connecting dbt Cloud to a warehouse, modeling, sources, testing and deployment; intermediate courses like "Jinja, Macros and Packages" and "dbt Mesh" are roughly 2 hours each [19].

By following this plan and dedicating several hours per day, you can develop a solid understanding of dbt's core features—models, seeds, snapshots, tests, documentation, macros and incremental builds—and be ready to implement analytics engineering pipelines using best practices.

---

[1] [9] [10] [11] [12] [13] [14] [15] [16] [17] dbt Cheat Sheet | Complete Commands Guide with Examples | Y42 Learning hub
https://www.y42.com/learn/dbt-cheat-sheet

[2] [3] [6] [8] What you beed to know about dbt (Data Build Tool) -
https://bix-tech.com/dbt-data-build-tool/

[4] [5] [7] Data Build Tool (dbt)
https://celerdata.com/glossary/data-build-tool-dbt

[18] [19] Learn dbt with expert-led training | dbt Labs
https://www.getdbt.com/dbt-learn