dbt SQL data transformation

START HERE

`dbt init <project>`
Create a project

`dbt debug`
Check connection

`dbt deps`
Install packages

`dbt build`
Seed, run, snapshot, test

`dbt run -s <model>`
Build one model

`dbt test -s <model>`
Test one model

`dbt show -s <model>`
Preview rows

`dbt docs generate`
Build the docs site

Sources raw tables → `source()` → **Staging models** views → `ref()` → **Marts** tables → `dbt test` → **Docs** lineage graph

CONCEPTS

A model is one SELECT in models/*.sql; dbt wraps it in CREATE VIEW or CREATE TABLE and runs models in dependency order.

Never hard-code table names: ref() and source() build the DAG that selection and run order depend on.

dbt_project.yml describes the project; profiles.yml (in ~/.dbt by default) holds connections and stays out of git.

PROJECT

`dbt init <project>`
Scaffold a project and profile

`dbt debug`
Check profile, project and connection

`dbt deps`
Install packages from packages.yml
Run again after changing packages.yml; writes package-lock.yml.

`dbt deps --add-package <pkg>@<ver>`
Add a package and install it

`dbt parse`
Parse the project, write the manifest

`dbt clean`
Delete target/ and dbt_packages/
Removes the folders listed in clean-targets.

PROJECT FILES

Path	Holds
`dbt_project.yml`	name, profile, folder configs
`profiles.yml`	targets and credentials
`models/*.sql`	one SELECT per model
`models/*.yml`	tests, docs, sources
`seeds/*.csv`	small static tables
`snapshots/`	SCD type-2 history
`macros/*.sql`	reusable Jinja
`packages.yml`	package dependencies
`target/`	compiled SQL, artifacts

PROJECT FILES (cont.)

models/schema.yml

```
models:
  - name: orders
    columns:
      - name: id
        data_tests:
          - unique
          - not_null
```

JINJA IN MODELS

`{{ ref('<model>') }}`
Reference another model or seed

`{{ source('<src>', '<table>') }}`
Reference a raw table declared as a source

`{{ config(materialized='table') }}`
Set config at the top of a model
view (default), table, incremental, ephemeral.

`{{ var('<name>', '<default>') }}`
Read a project variable

`{{ this }}`
The model's own relation

`{% if is_incremental() %}`
Filter only on incremental runs

`{{ env_var('<VAR>') }}`
Read an environment variable

`{{ target.name }}`
Current target, e.g. dev or prod

RUN AND BUILD

`dbt build`
Seed, run, snapshot and test in DAG order
A failing test skips the models downstream of it.

`dbt run`
Build all models

`dbt test`
Run data and unit tests

`dbt seed`
Load seeds/*.csv into tables

`dbt snapshot`
Record changes in snapshot tables

`dbt source freshness`
Check sources against freshness rules

`dbt compile -s <model>`
Print the rendered SQL of a model
Compiled SQL also lands in target/compiled/.

RUN AND BUILD (cont.)

`dbt show -s <model> --limit <n>`
Preview a model's rows

`dbt show --inline "<sql>"`
Run an ad-hoc query with Jinja

`dbt ls -s <selector>`
List what a selection matches
Dry-run any --select before running it.

`dbt run-operation <macro> --args '<yaml>'`
Call a macro

`dbt retry`
Rerun only what failed last time

`dbt docs generate`
Build catalog and docs site

`dbt docs serve --port <port>`
Serve the docs locally

SELECTION

`dbt run -s <model>`
Select one model (--select)

`dbt run -s <model>+`
Model and everything downstream

`dbt run -s +<model>`
Model and everything upstream

`dbt run -s +<model>+`
Upstream, model and downstream

`dbt run -s 1+<model>`
Model and direct parents only

`dbt run -s "<a> <b>"`
Union: either selection

`dbt run -s "tag:<t>,path:<dir>"`
Intersection: both match

`dbt run --exclude <model>`
Leave nodes out

`dbt run -s tag:<tag>`
By tag

`dbt run -s path:models/<dir>`
By folder

`dbt build -s source:<src>+`
Everything fed by a source

`dbt test -s test_type:unit`
Only unit tests

COMMON FLAGS

`-t <target>`
Use another target from profiles.yml
Same as --target; the default comes from the profile.

COMMON FLAGS (cont.)

`--full-refresh`
Rebuild incremental models and seeds

`--vars '{<key>: <value>}'`
Set variables (YAML)

`--threads <n>`
Run n models in parallel

`-x`
Stop at the first failure

`--profiles-dir <dir>`
Look for profiles.yml in <dir>

`-d`
Debug logging incl. SQL

STATE AND CI

--state points at another run's target/ (its manifest.json); copy it aside first, since each run overwrites target/.

`dbt build -s state:modified+ --state <dir>`
Build changed nodes and children

`dbt build -s result:error+ --state <dir>`
Rebuild failures and their children

`dbt run -s <model> --defer --state <dir>`
Read unbuilt parents from prod
Unselected refs resolve to the relations in the state manifest.

RECIPES

Iterate on one model

```
dbt compile -s <model>
dbt run -s <model>
dbt test -s <model>
dbt show -s <model> --limit 20
```

Slim CI against prod

```
# prod/ holds prod's manifest.json
dbt deps
dbt build -s state:modified+ \
  --defer --state prod/ -t ci
```

Fix and resume a failed build

```
dbt build
# fix the failing model, then
dbt retry
```

Incremental model

```
{{ config(materialized='incremental',
          unique_key='id') }}
select * from {{ source('app', 'events') }}
{% if is_incremental() %}
where ts > (select max(ts) from {{ this }})
{% endif %}
```