dbt Cheat Sheet for Data Engineers

Build, Test, and Document Your Data Pipelines Like a Pro.

Abhishek Agarwal
Data Engineer

What is dbt?

Concept	Description	Why it matters
Definition	A SQL-based transformation tool	Enables modular, version-controlled data transformations
Core philosophy	Transform data in-warehouse, SQL-first	Leverages warehouse compute power
Key benefits	Version control, testing, documentation	Improves data quality & collaboration

dbt Project Structure

Folder/File	Purpose	Typical contents
models/	SQL files for transformations	SELECT statements & CTEs
tests/	Custom test SQL or configs	Data integrity & schema checks
macros/	Reusable SQL snippets	Jinja macros for templating
snapshots/	Track slowly changing data states	Historical versions of tables
dbt_project.yml	Main project config	Settings for models, vars, etc.

Key dbt Concepts

Concept	Description	Example/Notes
Models	SQL SELECT statements defining transformations	materialized as tables, views, or incremental
Sources	Raw input tables declared in YAML	Enable lineage tracking
Seeds	Static CSV files loaded into warehouse	Useful for reference data
Tests	Assertions on data quality	Unique, not_null, relationships
Macros	Reusable Jinja functions	DRY (Don’t Repeat Yourself) logic

Materializations

Type	Description	Use Case
Table	Creates a physical table	Fast access, but storage-heavy
View	Creates a virtual table (no storage)	Always fresh, but slower queries
Incremental	Updates only new/changed data	Efficient for large datasets
Ephemeral	Inlined SQL snippets, no table created	For intermediate transformations

Writing Models & SQL Best Practices

Tip	Description	Example
Use CTEs	Modularize complex queries	WITH customer_orders AS (...)
Ref function	Reference other models safely	{{ ref('model_name') }}
Avoid select *	Explicit columns for clarity & performance	SELECT id, name FROM ...
Use variables	Parametrize with Jinja	{{ var('date_filter') }}
Comment generously	Document assumptions & logic	-- Filter only active customers

Testing in dbt

Test Type	Purpose	How to declare
Schema tests	Check uniqueness, non-nullability, etc.	YAML config under models
Data tests	Custom SQL assertions	SQL files in tests/ folder
Generic tests	Reusable test macros (unique, not_null)	Declared in YAML with test names
Custom tests	Tailored business logic tests	SQL in tests folder, run at build

Documentation & Lineage

Feature	Description	Benefit
Auto-generated docs	Graph visualization of models & lineage	Understand data flow
Descriptions	Add context on tables/columns	Improves discoverability
Exposure	Declare key outputs & metrics	Communicate business metrics
Docs site	Host interactive documentation website	Shareable with stakeholders

dbt Cloud vs Open Source

Aspect	dbt Open Source	dbt Cloud
Hosting	Self-managed on own infra	Managed SaaS
Scheduling	Manual or external orchestration	Built-in scheduler + alerting
IDE	Command line / external editors	Web-based IDE with autocomplete
Collaboration	Git integration	Git + team management + docs
Pricing	Free	Paid plans

Best Practices & Tips

Tip / Strategy	Description	Example
Modular models	Build reusable, composable models	One model per logical step
Use version control	Keep all code in Git	Branching & PRs for changes
Automate tests	Run tests with every build	CI/CD pipelines
Document thoroughly	Keep docs updated and detailed	Use dbt docs & descriptions
Use snapshots wisely	Track slowly changing dimensions	Avoid performance hits

Follow for more content like this

Abhishek Agarwal

Data Engineer