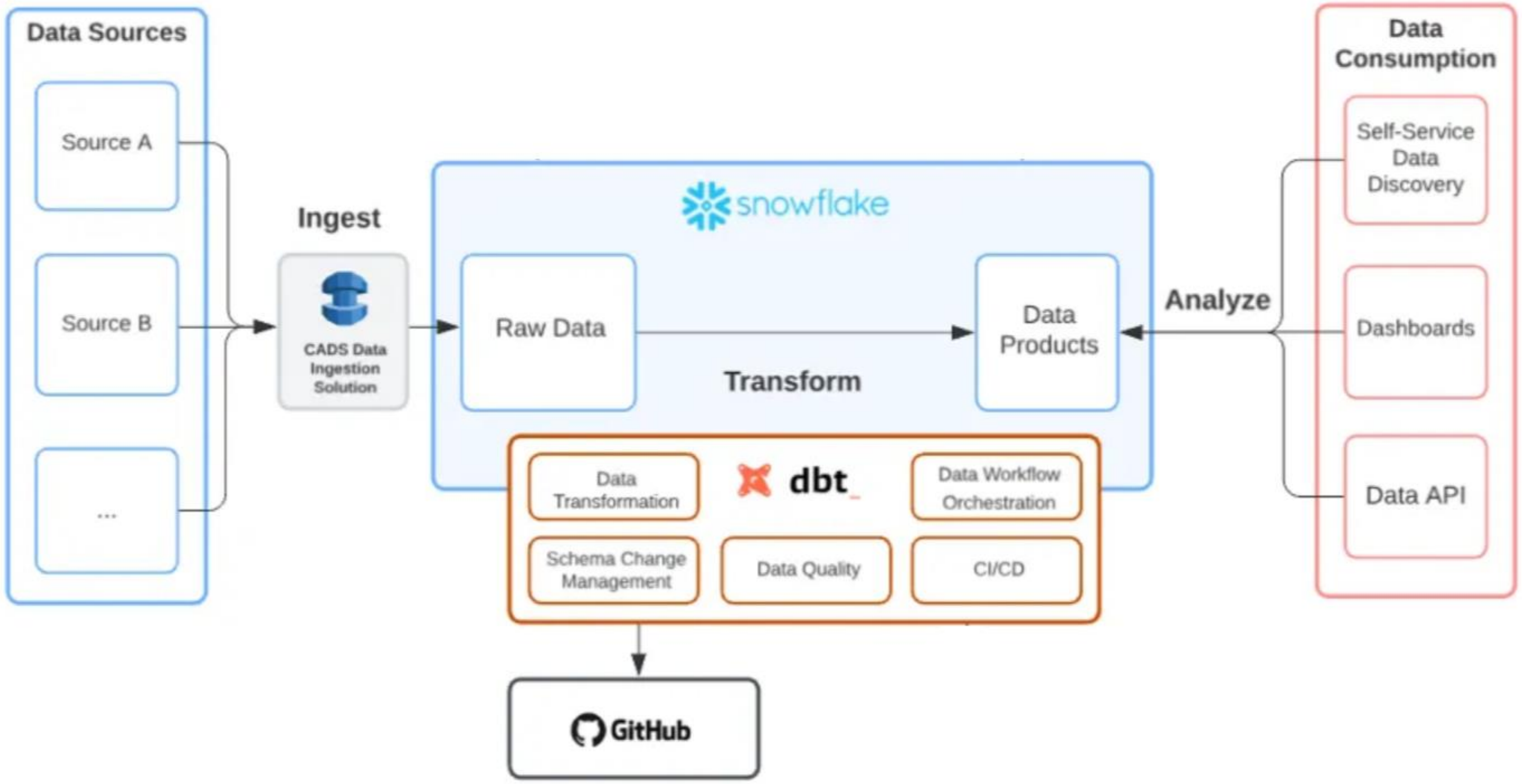

Data Sources
Source A
Source B
...
Ingest
CADS Data Ingestion Solution
snowflake
Raw Data
Transform
Data Products
Analyze
dbt
Data Transformation
Data Workflow Orchestration
Schema Change Management
Data Quality
CI/CD
GitHub
Data Consumption
Self-Service Data Discovery
Dashboards
Data API

DBT Core Vs DBT Cloud

dbt

DBT Core Vs DBT Cloud

Feature	dbt Core	dbt Cloud
Type	Open-source data transformation tool	Fully managed SaaS solution for dbt
Infrastructure	Self-managed (install locally or on your infra)	Fully managed by dbt Labs
Scalability	Manual scaling (you manage compute)	Automatic scaling with managed compute and cloud integrations
Ease of Use	Command-line interface (CLI only)	User-friendly web interface with visual IDE
Functionality	Core SQL compilation, Jinja templating, database adapters	All core features + job orchestration, logging, scheduling, and alerting
Interface	CLI-based interface	Web-based IDE, integrated docs, SSO, collaboration tools
Cost	Free and open source	Paid (subscription-based plans, with a free tier for small teams)

dbt project components

Understanding the dbt project components

	Component	Description
1	**Models**	SQL/Python files that contain logic that either transforms raw data into a dataset that is ready for analytics or, more often, is an intermediate step in such a transformation.
2	**Snapshots**	Capture the state of your data at specific points in time, useful for tracking historical changes.
3	**Seeds**	CSV files containing static data that you can load into your data warehouse using dbt.
4	**Data Tests**	SQL queries that validate your data, ensuring quality and consistency.
5	**Macros**	Reusable blocks of SQL code that can be invoked across your project to avoid repetition.
6	**Docs**	Documentation for your project, helping others understand your data models and transformations.
7	**Sources**	Define and describe the raw data tables or views loaded into your warehouse by ETL tools.
8	**Analysis**	SQL queries for ad-hoc analysis or reporting, not part of the main transformation pipeline.

dbt_project.yml file

Understanding the dbt_project.yml file

YAML Key	Simple Description
Name	The name of your project.
version	The version number of your project.
require- dbt-version	Limits which dbt versions can be used with your project.
profile	Tells dbt how to connect to your data (based on the profile.yml file).
model-paths	Folder where your model and source files are saved.
seed-paths	Folder where your seed (static CSV) files are saved.
test-paths	Folder where your test files are stored.
analysis-paths	Folder where your analysis files are located.
macro-paths	Folder where your reusable SQL code (macros) is stored.
snapshot-paths	Folder where your snapshot files are saved (for tracking historical data).
docs-paths	Folder where your documentation files live.
vars	Extra values (variables) you want to use in your project.

- ❖ Understanding the dbt_project.yml file is key to working with dbt.
- ❖ It acts as the configuration file that tells dbt how your project is set up and where to find important resources.

DBT Models

DBT Models

What are DBT Models?

DBT models are SQL select statements that transform your data. Each model:

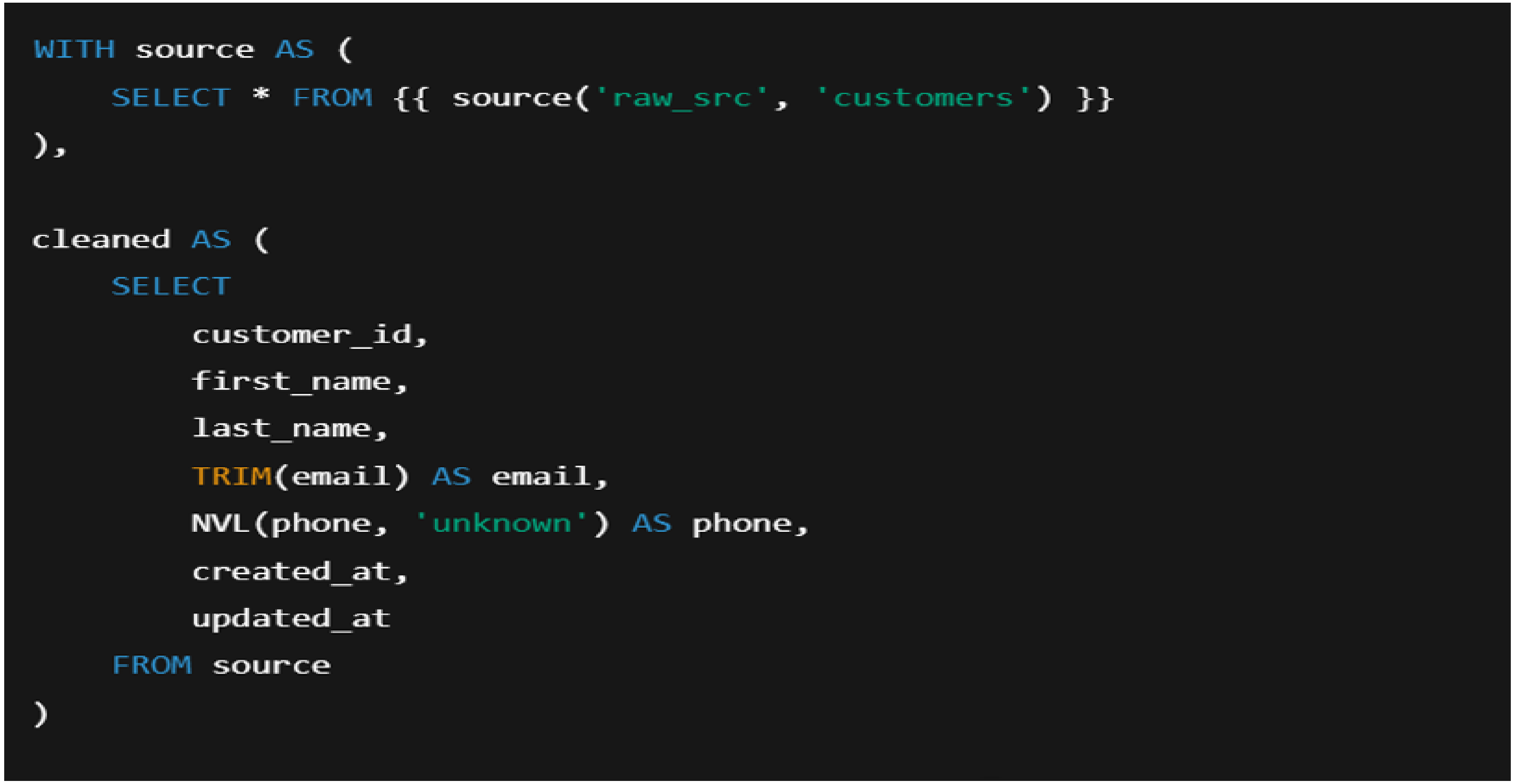
- Is defined in a .sql file
- Contains a single SELECT statement
- Produces a table or view in your data warehouse
- Can reference other models, creating a dependency graph

Models are the core building blocks of DBT projects and represent the transformations applied to your data.

Creating Your First Model

Let's create a simple model that transforms customer data:

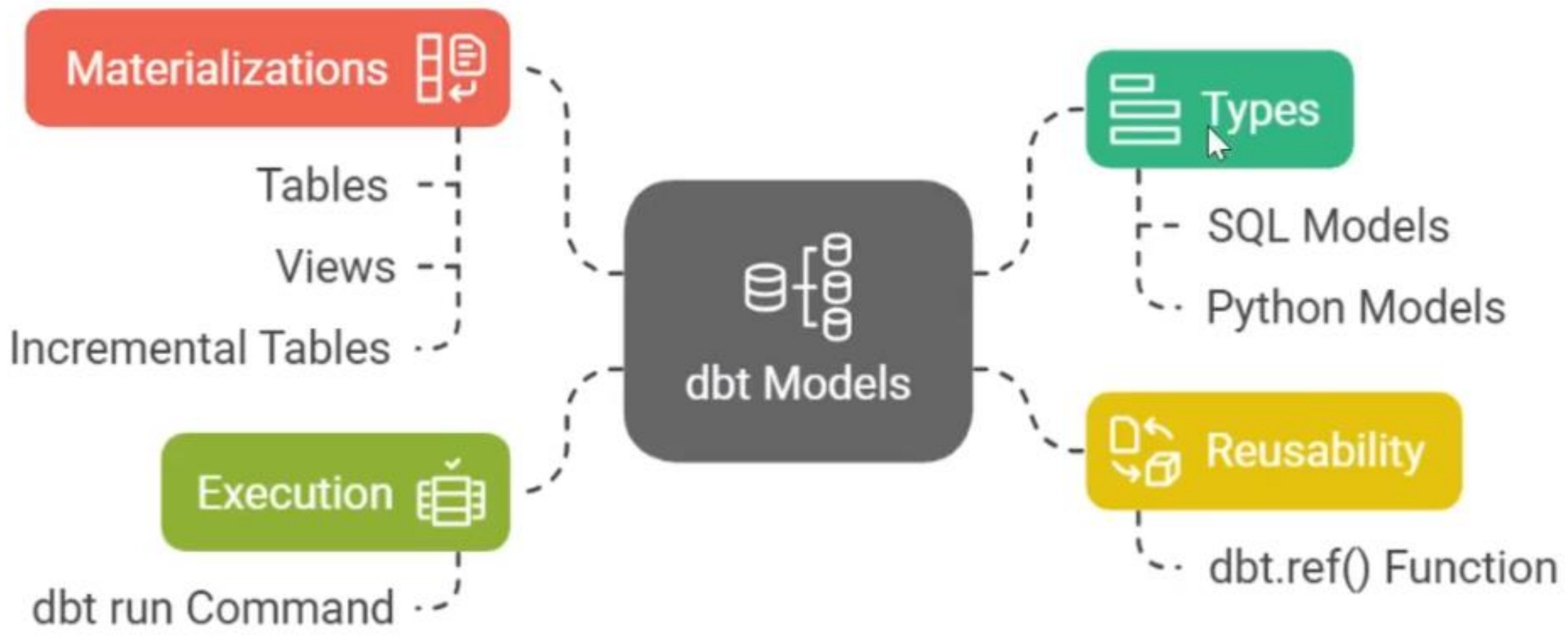
Create a new file models/stg_customers.sql:



```sql
WITH source AS (
    SELECT * FROM {{ source('raw_src', 'customers') }}
),

cleaned AS (
    SELECT
        customer_id,
        first_name,
        last_name,
        TRIM(email) AS email,
        NVL(phone, 'unknown') AS phone,
        created_at,
        updated_at
    FROM source
)
```

dbt Models components



Materializations

- Materializations defines standard of the objects to be created in the data warehouse
- There are four types of materializations built in dbt
    - Table
    - View
    - Incremental
    - Ephemeral

Table

When **table** is used in the materialized property, the model will be created as a table in the database

- Can be used for simple transformations and in order to query faster
- New records in the source will not be automatically refreshed until the table is rebuilt
- Table takes long time to build

View

When **view** is used in the materialized property, the model will be created as a View in the database

- View does not have a storage in the schema. View is build using the source and the transformation logic provided in the model

- New records in the source can be refreshed

- Views can be used for the simple transformations like, type casting and renaming the columns

Ephemeral

When **ephemeral** is used in the materialized property, the model will not create any physical objects in the database. The model holds the logic within dbt and is used as a reusable static logics

- Can be used as reusable transformations
- Performance is good
- Harder to debug in case of errors/ issues

Incremental

When **incremental** is used in the materialized property, the model will be created as a table in the database on the first execution, and the data will be either updated or inserted on following executions

- Can be used for complex transformations
- Performance is good compared to Table and View

Models

Models are basically .sql file where we define the select statements to transform the data in a data warehouse

Command to execute Models:

```
dbt run
```

When a model is executed, the statements in the models are created as a table or view in the data warehouse

Materialization Strategies

Choose the right materialization based on:

1. **Data Volume**:
   - Small data: Views
   - Large data: Tables or incremental models
2. **Update Frequency**:
   - Frequent updates: Views or incremental models
   - Infrequent updates: Tables
3. **Query Complexity**:
   - Simple transformations: Views
   - Complex transformations: Tables
4. **Query Patterns**:
   - Ad-hoc exploration: Views
   - Repeated reporting: Tables