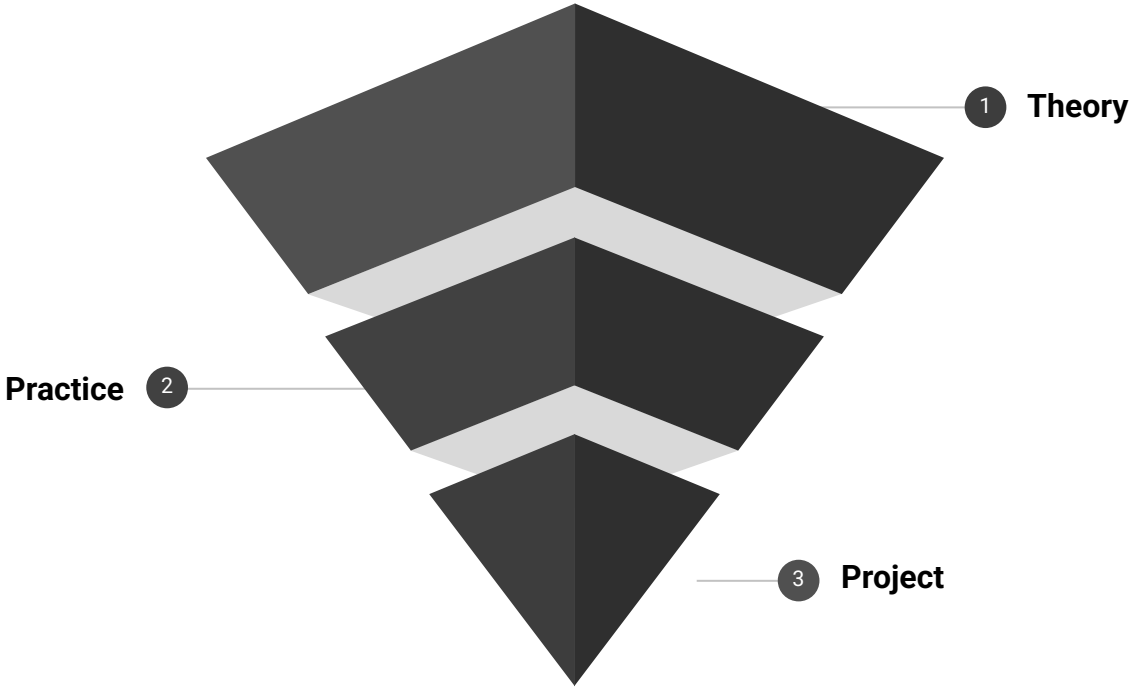
Course Introduction

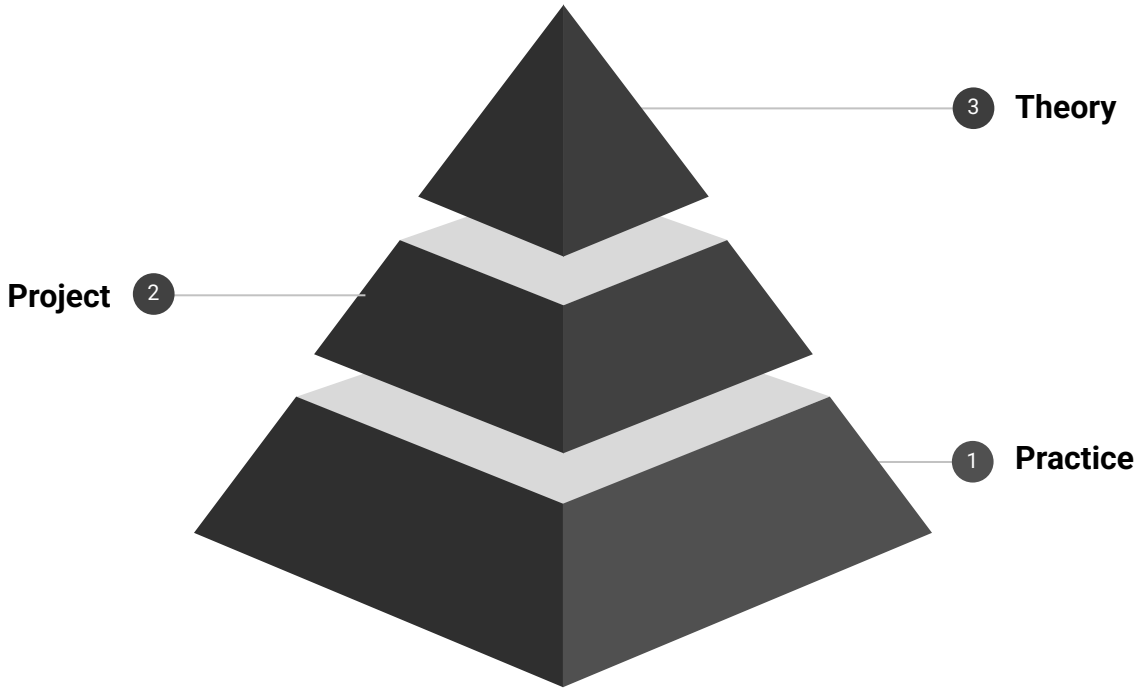
OBJECTIVES

- Data-maturity model
- dbt and data architectures
- Data warehouses, data lakes, and lakehouses
- ETL and ELT procedures
- dbt fundamentals
- Analytics Engineering

TOP-DOWN

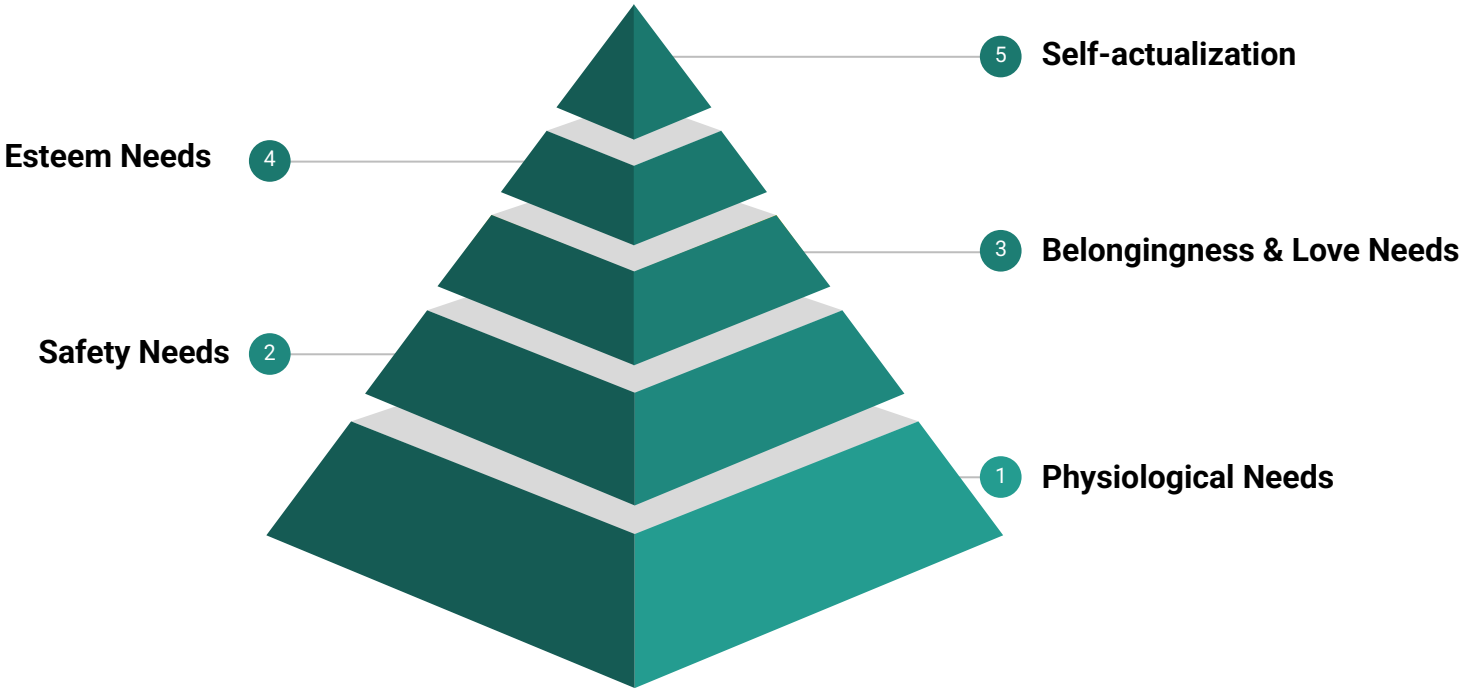


BOTTOM-UP

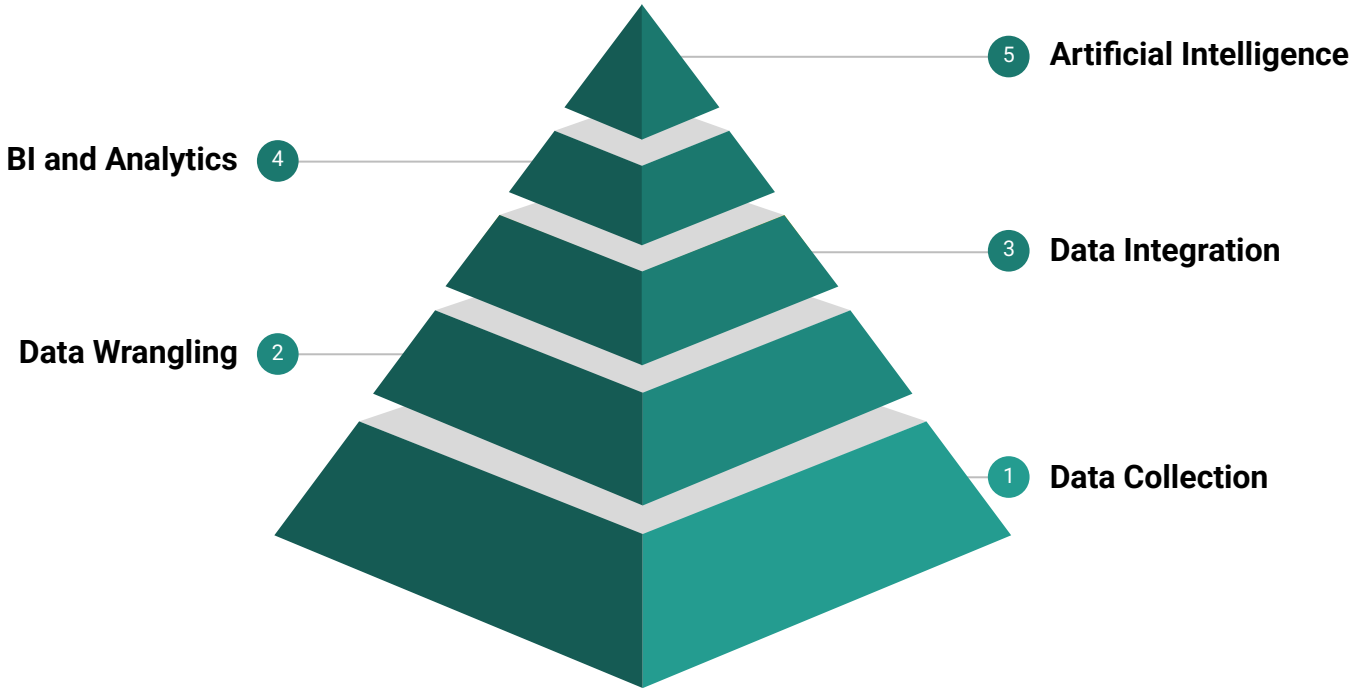


Data Maturity Model

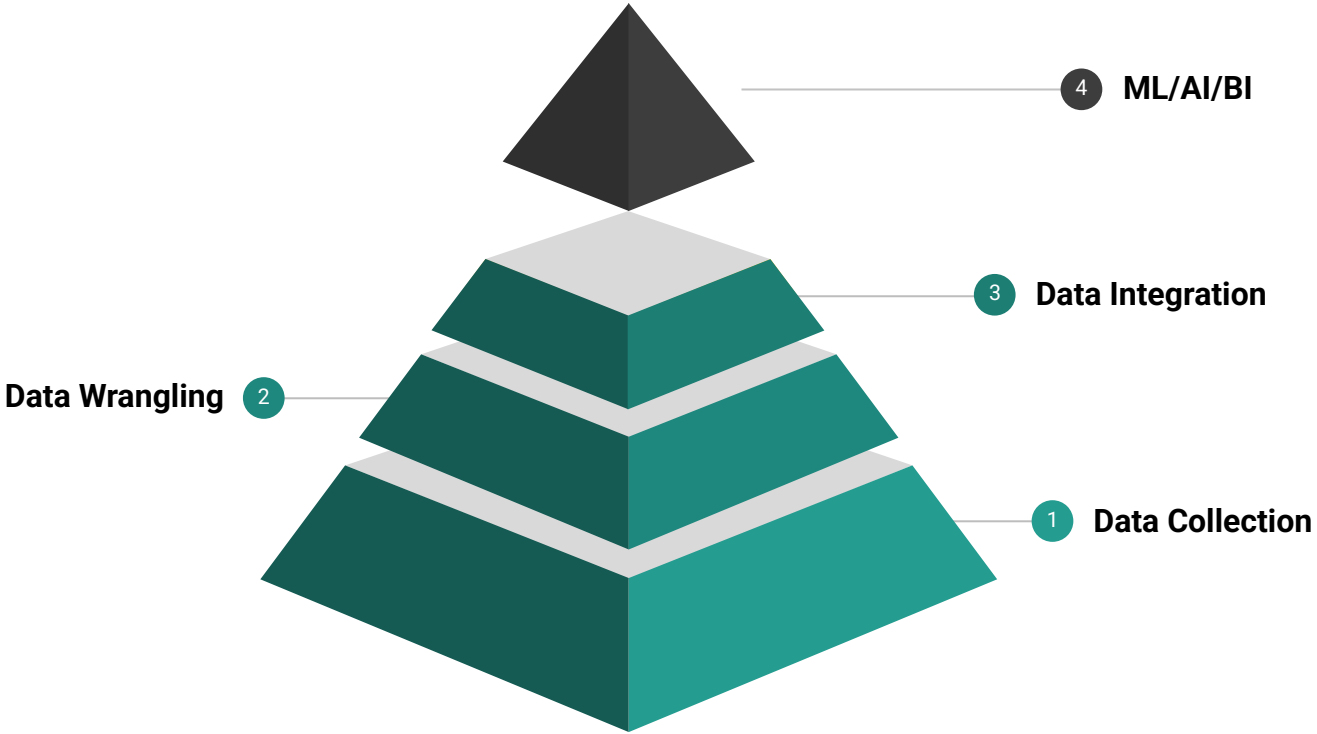
Maslow's Hierarchy of Needs



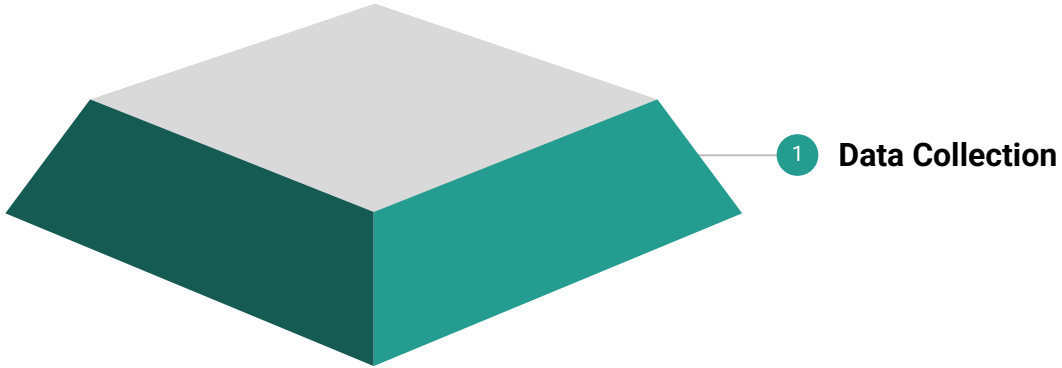
Data-Maturity Model



Typical Data Architecture



Data Collection



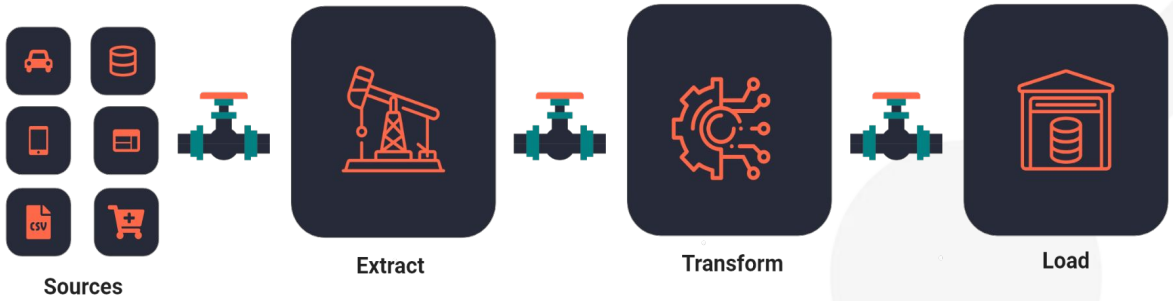
Data Wrangling

Data Integration

ETL - ELT#

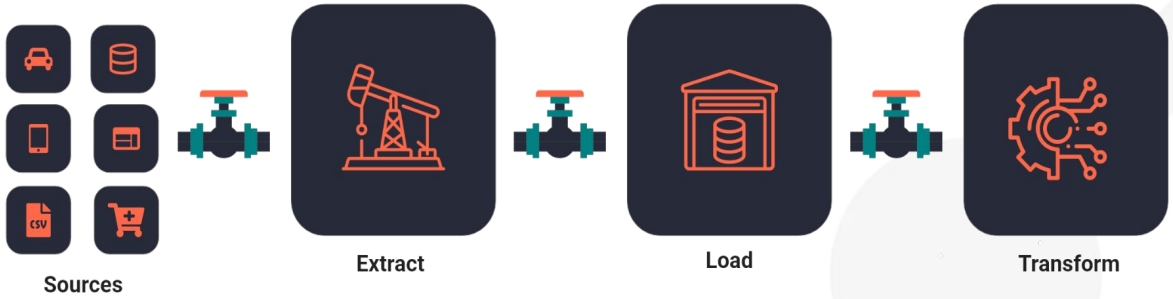
ETL

ETL
Extract, Transform, Load



ELT

ELT
Extract, Load, Transform



Data Warehouse

Data Warehouse

On-Premise

ORACLE IBM teradata.

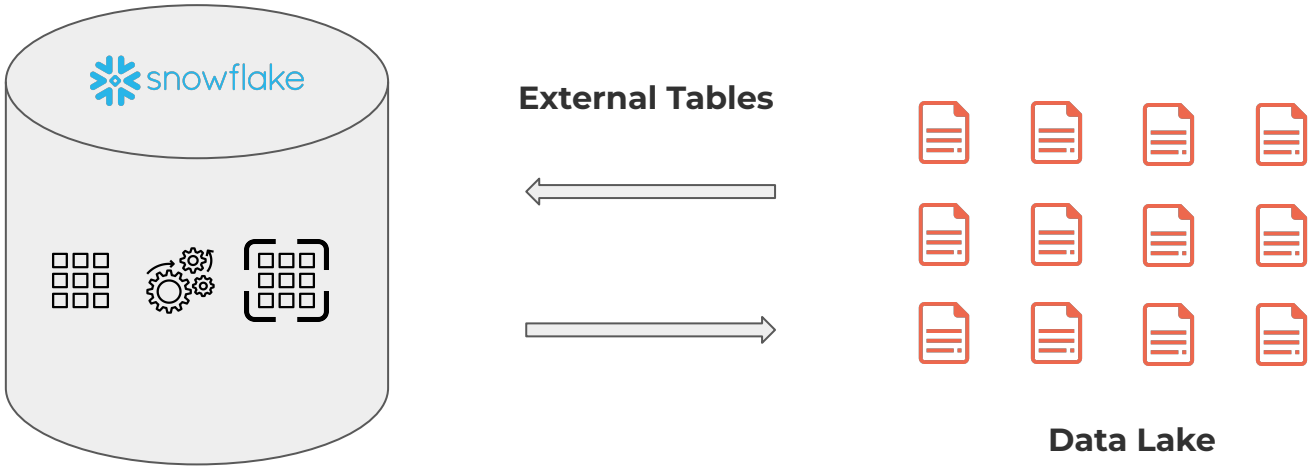
Cloud

amazon REDSHIFT Google BigQuery snowflake

PNG Credits: https://databricks.com/wp-content/uploads/2020/01/data-lakehouse-new-1024x538.png

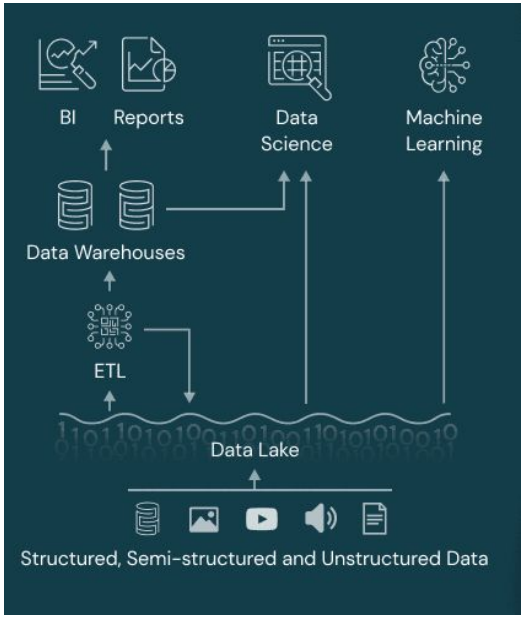
External Tables

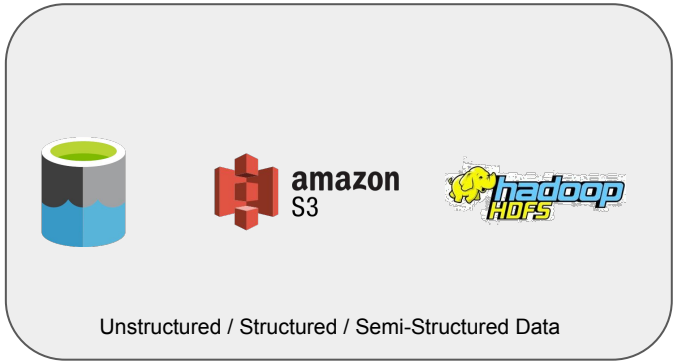
External Tables



Data Lake

Data Lake

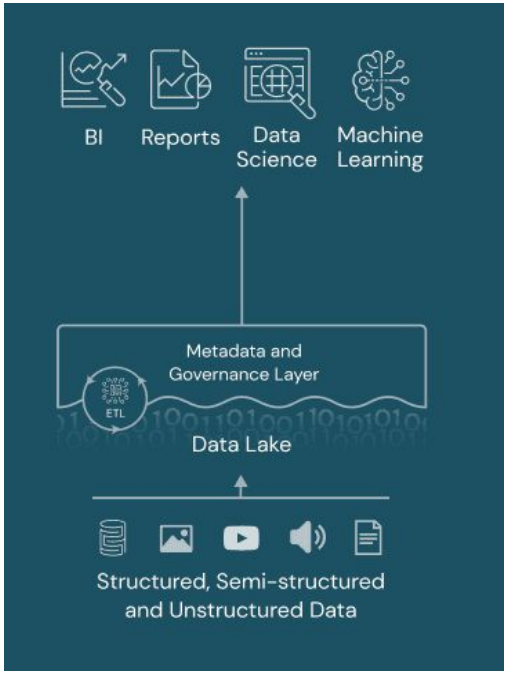




PNG Credits: https://databricks.com/wp-content/uploads/2020/01/data-lakehouse-new-1024x538.png

Data Lakehouse

Data Lakehouse



PNG Credits: https://databricks.com/wp-content/uploads/2020/01/data-lakehouse-new-1024x538.png

Slowly Changing Dimensions

SCD Type 0

Not updating the DWH table when a Dimension changes

Source	DWH
Host ID | Name | Fax Number #1 | John Doe | 1-408-999 8888	Host ID | Name | Fax Number #1 | John Doe | 555-123-4567

SCD Type 1

Updating the DWH table when a Dimension changes, overwriting the original data

	Source	DWH
No Air-conditioning	Host ID: #1, Air-conditioning: No	Host ID: #1, Air-conditioning: No
Installed Air-conditioning	Host ID: #1, Air-conditioning: Yes	Host ID: #1, Air-conditioning: No
DWH updated	Host ID: #1, Air-conditioning: Yes	Host ID: #1, Air-conditioning: Yes

SCD Type 2

Keeping full history - Adding additional (historic data) rows for each dimension change

	Source	DWH
Current rental price ($300)	Apartment ID: #1, Price: $300	Host Key: 1000, Apartment ID: #1, Price: $300, Start_Date: 2020-01-01T00:00:00, End_Date: NULL
Change in the rental price ($450)	Apartment ID: #1, Price: $450	Host Key: 1000, Apartment ID: #1, Price: $300, Start_Date: 2020-01-01T00:00:00, End_Date: 2021-01-01T00:00:00
DWH updated	Apartment ID: #1, Price: $450	Host Key: 1000, Apartment ID: #1, Price: $300, Start_Date: 2020-01-01T00:00:00, End_Date: 2021-01-01T00:00:00 Host Key: 1001, Apartment ID: #1, Price: $450, Start_Date: 2021-01-01T00:00:00, End_Date: NULL

SCD Type 2

Keeping full history - Adding additional (historic data) rows for each dimension change

Current rental price ($300)

Change in the rental price ($450)

DWH updated

Source

Apartment ID	Price
#1	$300

Apartment ID	Price
#1	$450

Apartment ID	Price
#1	$450

DWH

Host Key	Apartment ID	Price	Start_Date	End_Date	Is_Current
1000	#1	$300	2020-01-01T00:00:00	9999-12-12T00:00:00	Y

Host Key	Apartment ID	Price	Start_Date	End_Date	Is_Current
1000	#1	$300	2020-01-01T00:00:00	9999-12-12T00:00:00	N

Host Key	Apartment ID	Price	Start_Date	End_Date	Is_Current
1000	#1	$300	2020-01-01T00:00:00	2021-01-01T00:00:00	N
1001	#1	$450	2021-01-01T00:00:00	9999-12-12T00:00:00	Y

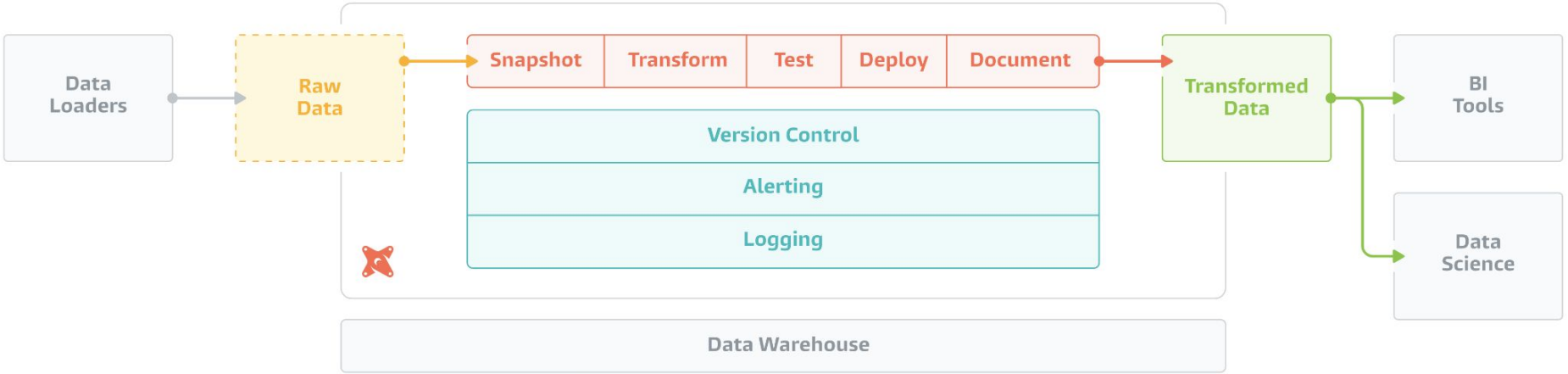
SCD Type 3

Keeping limited data history - adding separate columns for original and current value

	Source	DWH
Listed as Private	Apartment ID: #1, Type: Private	Apartment ID: #1, Previous Type: Private, Current Type: Private
Host changed Private to Entire	Apartment ID: #1, Type: Entire	Apartment ID: #1, Previous Type: Private, Current Type: Entire
Host changed Entire to Shared	Apartment ID: #1, Type: Shared	Apartment ID: #1, Previous Type: Entire, Current Type: Shared

dbt Overview

dbt Overview



PNG Credits: https://data.solita.fi/wp-content/uploads/2021/06/dbt-transform.png

Analytics Engineering

Common Table Expression (CTE)

CTE

Syntax

```sql
WITH <name_of_the_result_set> ([column_names])
AS
(
   <cte_query>
)
<reference_the_CTE>
```

CTE

Example

```
-- STEP 1
WITH raw_listings AS (

-- STEP 2
        SELECT * FROM [source].[listings]
)

-- STEP 3
SELECT
    id AS listing_id,
    listing_url,
    name AS listing_name,
    room_type,
    minimum_nights,
    host_id,
    price AS price_str,
    created_at,
    updated_at

FROM raw_listings
```

PROJECT OVERVIEW

Analytics Engineering with Airbnb

ANALYTICS ENGINEERING WITH AIRBNB

- Simulating the life of an Analytics Engineer in Airbnb
- Loading, Cleansing, Exposing data
- Writing test, automations and documentation
- Data source: Inside Airbnb: Berlin

TECH STACK

REQUIREMENTS

- Modeling changes are easy to follow and revert
- Explicit dependencies between models
- Explore dependencies between models
- Data quality tests
- Error reporting
- Incremental load of fact tables
- Track history of dimension tables
- Easy-to-access documentation

NEXT STEPS - SETUP

- Snowflake registration
- Dataset import
- dbt installation
- dbt setup, snowflake connection

That’s it :)

SNOWFLAKE

Trial Account Setup

SNOWFLAKE AUTHENTICATION

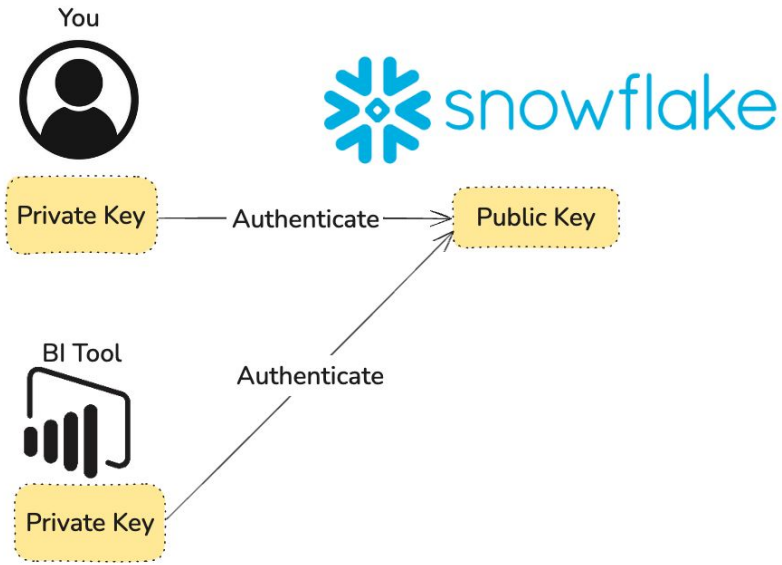
❌ Username & Password + Multi-Factor Authentication Application

❌ Personal Access Token (Requires Network Policies)

❌ Single Sign-On (SSO)

✅ Keypair-Based Authentication

Public-Key Authentication (PKI)



SNOWFLAKE AUTO-SETUP APP

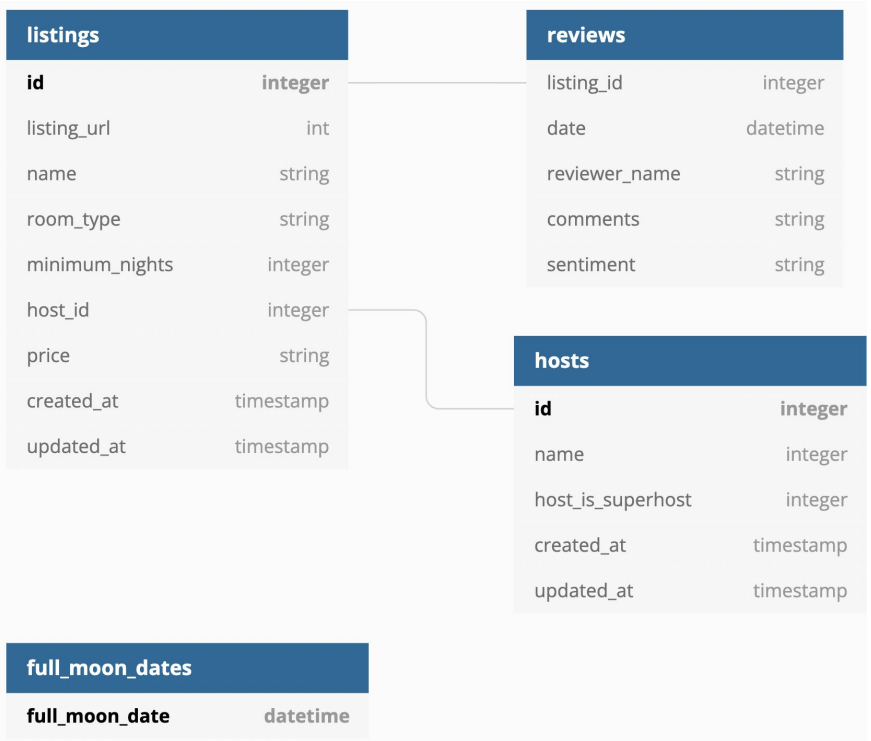
- **Imports the database tables**: listings, hosts, reviews
- **dbt Snowflake user creation** and permissions setup
- **Preset user creation** and permission setup

https://bit.ly/dbt-course-setup

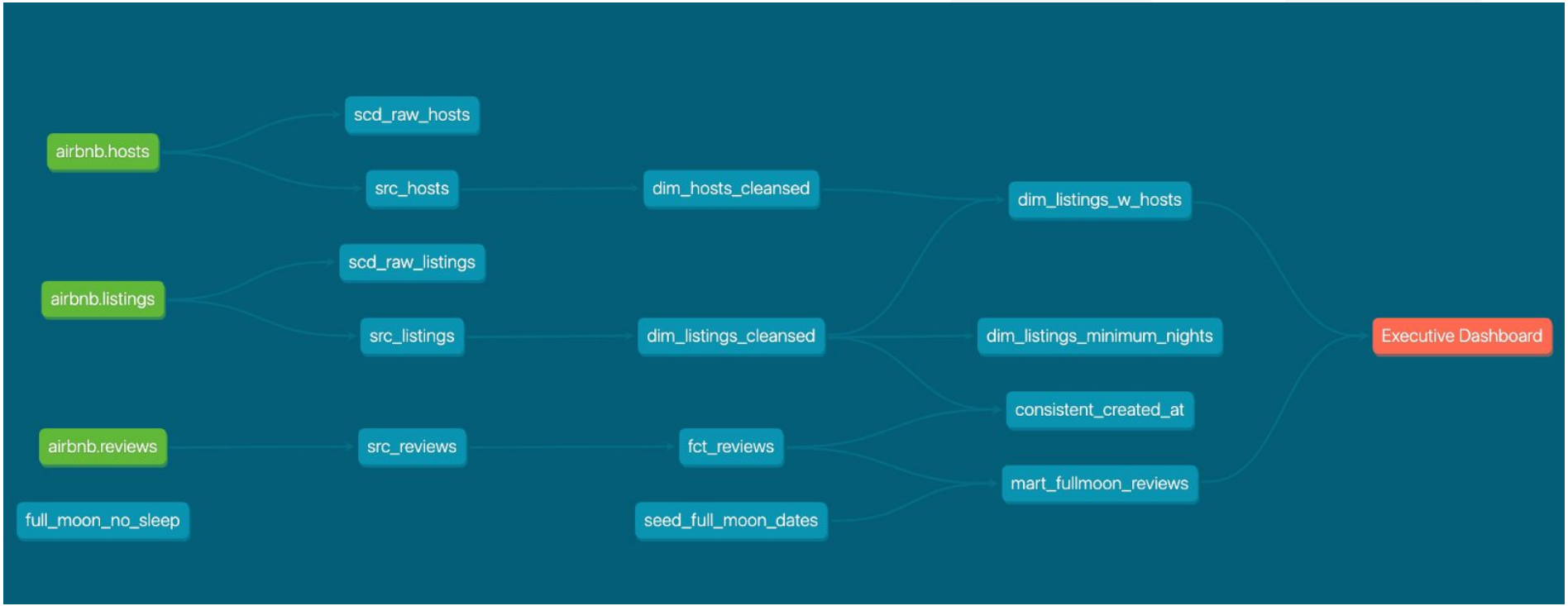
DATA FLOW

Overview

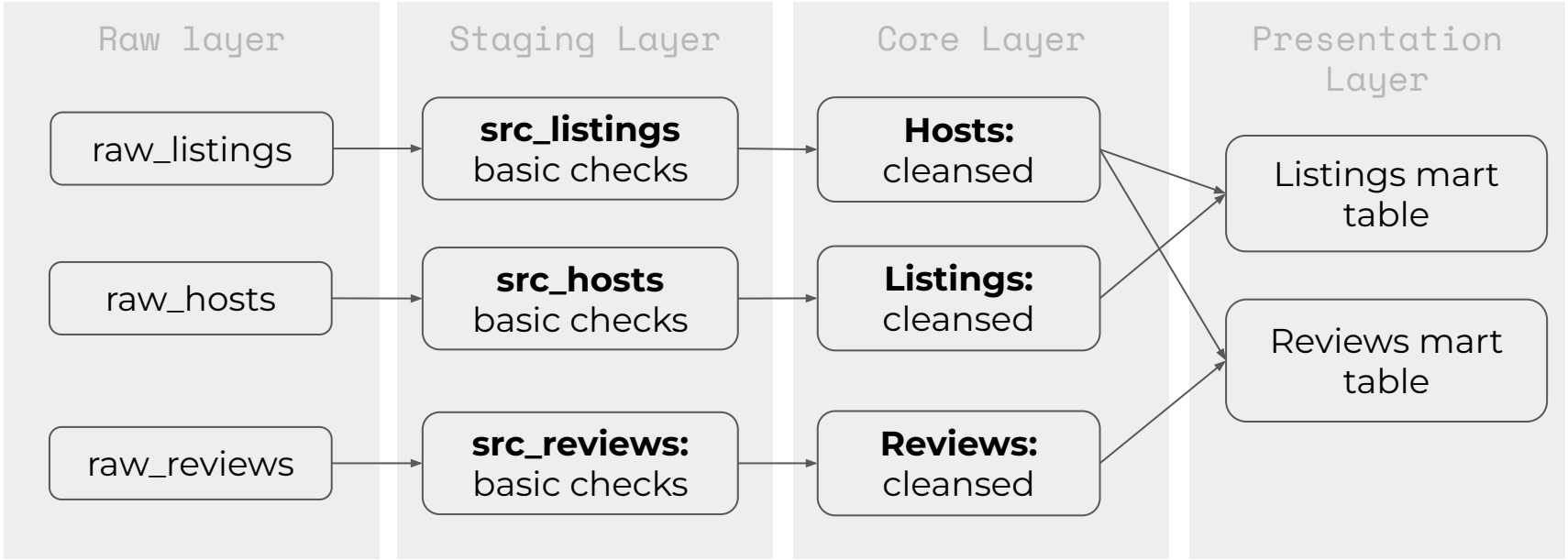
INPUT DATA MODEL



DATA FLOW OVERVIEW



DATA FLOW OVERVIEW



DBT SETUP

Mac

VIRTUALENV SETUP

- Install Python 3.11 and the Python virtualenv package
- Create a virtualenv
- Activate virtualenv

DBT SETUP

Windows

INSTALLING DBT

- Install Python3
- Create a virtualenv
- Activate virtualenv
- Install dbt and the dbt-snowflake connector

DBT SETUP

dbt init and connecting to Snowflake

SNOWFLAKE CONNECTION ISSUES?

- Ensure VPN is switched off
- Ensure you are not behind a corporate proxy or firewall
- Try to change network or laptop

MODELS

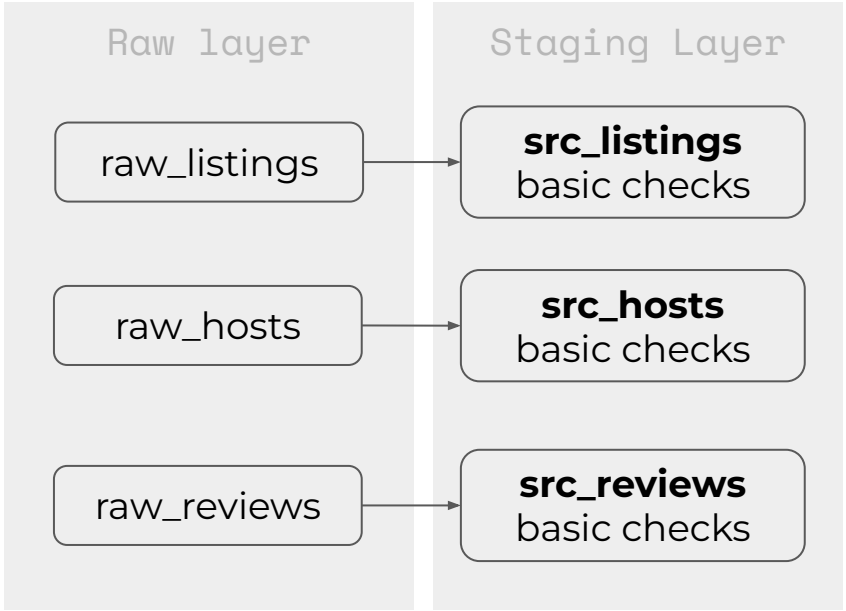
LEARNING OBJECTIVES

- Understand the data flow of our project
- Understand the concept of Models in dbt
- Create three basic models:
  - src_listings
  - src_reviews: guided exercises
  - src_hosts: individual lab

MODELS OVERVIEW

- Models are the basic building block of your business logic
- Materialized as tables, views, etc...
- They live in SQL files in the `models` folder
- Models can reference each other and use templates and macros

DATA FLOW PROGRESS



GUIDED EXERCISE

src_reviews.sql

Create a new model in the `models/src/` folder called `src_reviews.sql`.

- Use a CTE to reference the AIRBNB.RAW.RAW_REVIEWS table
- SELECT every column and every record, and rename the following columns:
  - *date* to *review_date*
  - *comments* to *review_text*
  - *sentiment* to *review_sentiment*
- Execute `dbt run` and verify that your model has been created

(You can find the solution among the resources)

MATERIALIZATIONS

LEARNING OBJECTIVES

- Understand how models can be connected
- Understand the four built-in materializations
- Understand how materializations can be configured on the file and project level
- Use *dbt run* with extra parameters

MATERIALIZATIONS

MATERIALISATIONS OVERVIEW

View

Use it
- You want a lightweight representation
- You don't reuse data too often

Don't use it
- You read from the same model several times

Table

Use it
- You read from this model repeatedly

Don't use it
- Building single-use models
- Your model is populated incrementally

Incremental (table appends)

Use it
- Fact tables
- Appends to tables

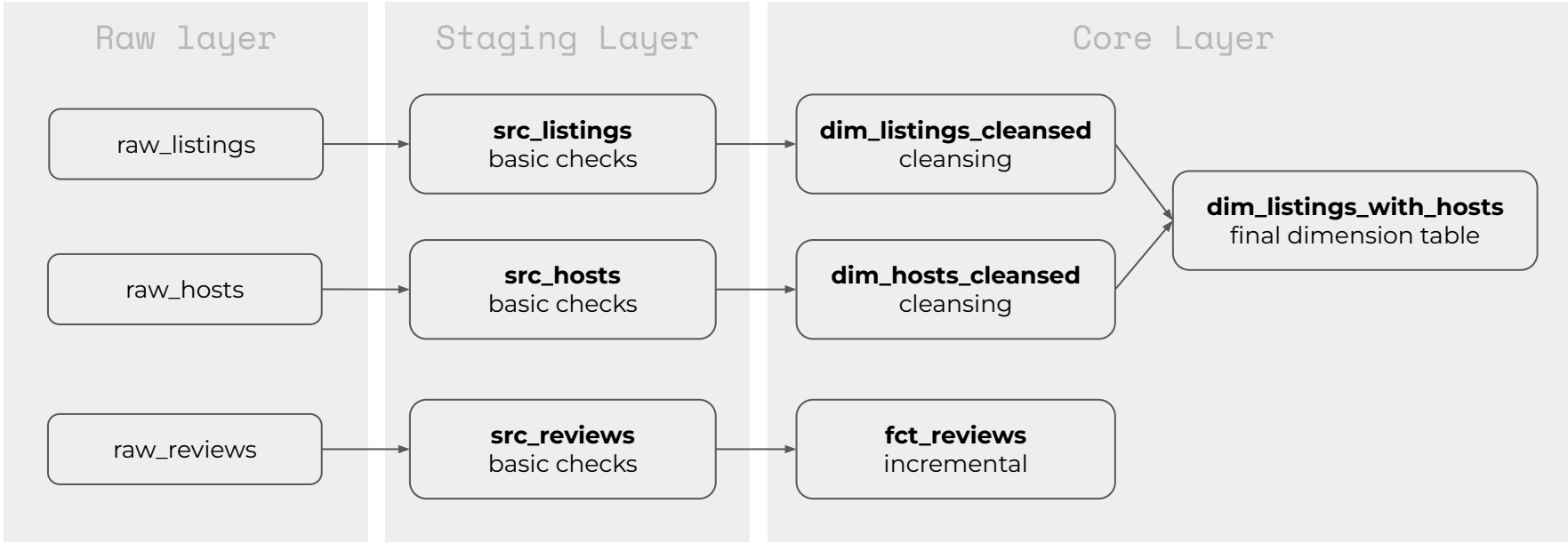
Don't use it
- You want to update historical records

Ephemeral (CTEs)

Use it
- You merely want an alias to your date

Don't use it
- You read from the same model several times

DATA FLOW PROGRESS



GUIDED EXERCISE

dim_hosts_cleansed.sql

Create a new model in the `models/dim/` folder called `dim_hosts_cleansed.sql`.

- Use a CTE to reference the `src_hosts` model
- SELECT every column and every record, and add a cleansing step to *host_name*:
  - If *host_name* is not null, keep the original value
  - If *host_name* is null, replace it with the value *'Anonymous'*
  - Use the *NVL(column_name, default_null_value)* function
- Execute `dbt run` and verify that your model has been created

(You can find the solution among the resources)

SOURCES & SEEDS

LEARNING OBJECTIVES

- Understand the difference between seeds and sources
- Understand source-freshness
- Integrate sources into our project

SOURCES AND SEEDS OVERVIEW

- Seeds are local files that you upload to the data warehouse from dbt
- Sources is an abstraction layer on the top of your input tables
- Source freshness can be checked automatically

SNAPSHOTS

LEARNING OBJECTIVES

- Understand how dbt handles type-2 slowly changing dimensions
- Understand snapshot strategies
- Learn how to create snapshots on top of our *listings* and *hosts* models

SNAPSHOTS

Overview

TYPE-2 SLOWLY CHANGING DIMENSIONS

host_id	host_name	email
1	Alice	alice.airbnb@gmail.com
2	Bob	**bob.airbnb@gmail.com**

TYPE-2 SLOWLY CHANGING DIMENSIONS

host_id	host_name	email
1	Alice	alice.airbnb@gmail.com
2	Bob	**bobs.new.address@gmail.com**

TYPE-2 SLOWLY CHANGING DIMENSIONS

host_id	host_name	email	dbt_valid_from	dbt_valid_to
1	Alice	alice.airbnb@gmail.com	2022-01-01 00:00:00	null
2	Bob	bob.airbnb@gmail.com	2022-01-01 00:00:00	2022-03-01 12:53:20
3	Bob	**bobs.new.address@gmail.com**	2022-03-01 12:53:20	null

CONFIGURATION AND STRATEGIES

- Snapshots live in the *snapshots* folder
- Strategies:
    - *Timestamp:* A **unique key** and an **updated_at** field is defined on the source model. These columns are used for determining changes.
    - *Check:* Any change in a set of columns (or all columns) will be picked up as an update.

GUIDED EXERCISE

scd_raw_hosts.sql

Create a new snapshot in the `snapshots/` folder called `scd_raw_hosts.sql`.

- Set the target table name to *scd_raw_hosts*
- Set the output schema to *dev*
- Use the *timestamp* strategy, figure out the unique key and updated_at column to use
- Execute `dbt snapshot` and verify that your snapshot has been created

(You can find the solution among the resources)

TESTS

LEARNING OBJECTIVES

- Understand **Unit Tests** and **Data Tests**
- Understand how these tests can be defined
- Configure built-in generic Data Tests
- Create your own singular Data Tests
- Create Unit Tests for your data

TESTS OVERVIEW

- dbt has two types of tests: **Unit Tests** and **Data Tests** and it supports **Contracts**
- **Unit Tests** test transformations with a small sample of mock data you provide
- **Data Tests** test data integrity and quality on the actual data
- **Contracts** enforce the schema of models (such as column names, types and constraints)

DATA TESTS OVERVIEW

- There are two types of data tests: ***singular*** and ***generic***
- There are four built-in generic tests:
  - unique
  - not_null
  - accepted_values
  - Relationships
- Singular data tests are SQL queries stored in tests which are expected to return an empty result set
- *You can define your own custom generic tests*
- *You can import tests from dbt packages (will discuss later)*

GUIDED EXERCISE

TEST dim_hosts_cleansed

Create a generic data tests for the `dim_hosts_cleansed` model.

- *host_id:* Unique values, no nulls
- *host_name* shouldn't contain any null values
- *Is_superhost* should only contain the values *t* and *f*.
- Execute `dbt test` to verify that your tests are passing
- **Bonus: Figure out which tests to write for `fct_reviews` and implement them**

(You can find the solution among the resources)

GUIDED EXERCISE

TEST dim_hosts_cleansed

Create a generic data test for the `dim_hosts_cleansed` model.

- *host_id:* Unique values, no nulls
- *host_name* shouldn't contain any null values
- *Is_superhost* should only contain the values *t* and *f*.
- Execute `dbt test` to verify that your tests are passing
- **Bonus: Figure out which tests to write for `fct_reviews` and implement them**

(You can find the solution among the resources)

JINJA AND MACROS (this section is work in progress)

LEARNING OBJECTIVES

- Understand the basic building blocks and control structures of Jinja
- Use macros to add new functionality to your project

JINJA

- A **templating language** that lets you write dynamic, reusable SQL code.

Think of it as "**programming inside your SQL**" - you can use:

- Variables
- Conditionals and Loops
- Functions
- Macros

JINJA - BASIC BUILDING BLOCKS

- Comments: `{# This is a comment #}`
  - *Instantly removed - won't appear anywhere*
- Statements: `{% set my_name = "Zoltan" %}`
- Expressions: `{{ my_name }}`
  - *Outputs* `"Zoltan"`

(SOME) BUILT-IN VARIABLES IN DBT

- `{{ this }}`
  - References the model being built
- `{{ target.name }}`
  - The name of our target - the default is defined in *dbt_profiles.yml* (`dev`)
- `{{ target.database }}`
  - The target database
- `{{ target.schema }}`
  - The target schema

JINJA - MACRO (FUNCTION) EXAMPLES

- `{{ log("Debug message", info=True) }}` Prints to the console
- `{{ var("user_name") }}` Get variable value
- `{{ ref('dim_listings_cleansed') }}` References another model - **dbt specific**
- `{{ source('airbnb', 'listings') }}` References a source table - **dbt specific**
- `{{ config('dim_listings_cleansed') }}` References another model - **dbt specific**
- `{{ adapter.get_columns_in_relation }}` Adapter-level functionality - **dbt specific**

MACROS (FUNCTIONS) IN DBT

- Macros are jinja macros created in the ***macros*** folder
- There are many built-in macros in DBT
- You can use macros in various places, such as:
  - model definitions
  - tests
  - other macros

CUSTOM BUILDING BLOCKS IN DBT

- `{% test … %}` Test Definition
- `{% snapshot %}` Snapshot definition
- `{% docs %}` Documentation block
- `{% materialization %}` Custom materialization

JINJA - DEFINING MACROS

```
{% macro select_positive_values(model, column_name) %}

SELECT *

FROM {{ model }}

WHERE {{ column_name }} > 0

{% endmacro %}
```

JINJA - IF/ELSE

```
SELECT * FROM {{ ref('src_reviews') }} WHERE review_text IS NOT NULL

{% if is_incremental() %}

  AND review_date > (SELECT MAX(review_date) FROM {{ this }})

  {{ log('Loading incrementally - new reviews only', info=True) }}

{% else %}

  {{ log('Loading all reviews - first run or full refresh', info=True) }}

{% endif %}
```

JINJA - LOOPS

```
{% set columns = ['listing_id', 'listing_name', 'price'] %}

SELECT

{% for col in columns %}

  {{ col }}{% if not loop.last %}, {% endif %}

{% endfor %}

FROM {{ ref('dim_listings_cleansed') }}
```

WHITESPACE CONTROL

- `{% statement %}`: **No** whitespace control
- `{%- statement %}`: Strip whitespace **BEFORE**
- `{% statement -%}`: Strip whitespace **AFTER**
- `{%- statement -%}` Strip whitespace **BEFORE** and **AFTER**

JINJA, CUSTOM TESTS AND PACKAGES

LEARNING OBJECTIVES

- Understand how macros are created
- Use macros to implement your own generic tests
- Find and install third-party dbt packages

MACROS, CUSTOM TESTS AND PACKAGES

- Macros are jinja templates created in the *macros* folder
- There are many built-in macros in DBT
- You can use macros in model definitions and tests
- A special macro, called *test*, can be used for implementing your own generic tests
- dbt packages can be installed easily to get access to a plethora of macros and tests

DOCUMENTATION

LEARNING OBJECTIVES

- Understand how to document models
- Use the documentation generator and server
- Add assets and markdown to the documentation
- Discuss dev vs. production documentation serving

DOCUMENTATION OVERVIEW

- Documentations can be defined two ways:
  - In yaml files (like *schema.yml*)
  - In standalone markdown files
- Dbt ships with a lightweight documentation web server
- For customizing the landing page, a special file, *overview.md* is used
- You can add your own assets (like images) to a special project folder

ANALYSES, HOOKS AND EXPOSURES

LEARNING OBJECTIVES

- Understand how to store ad-hoc analytical queries in dbt
- Work with dbt hooks to manage table permissions
- Build a dashboard in Preset
- Create a dbt exposure to document the dashboard

HOOKS

- Hooks are SQLs that are executed at predefined times
- Hooks can be configured on the project, subfolder, or model level
- Hook types:
  - on_run_start: executed at the start of *dbt {run, seed, snapshot}*
  - on_run_end: executed at the end of *dbt {run, seed, snapshot}*
  - pre-hook: executed before a model/seed/snapshot is built
  - post-hook: executed after a model/seed/snapshot is built

dbt Fusion

&

The Official VSCode Extension

A new Engine for dbt

- Completely rewritten from the ground up in Rust
  - Magnitudes-Faster Parsing
  - “Code Comprehension Engine”
- Released as Source-available from dbt Labs
- Released along an Official Visual Studio Code Extension

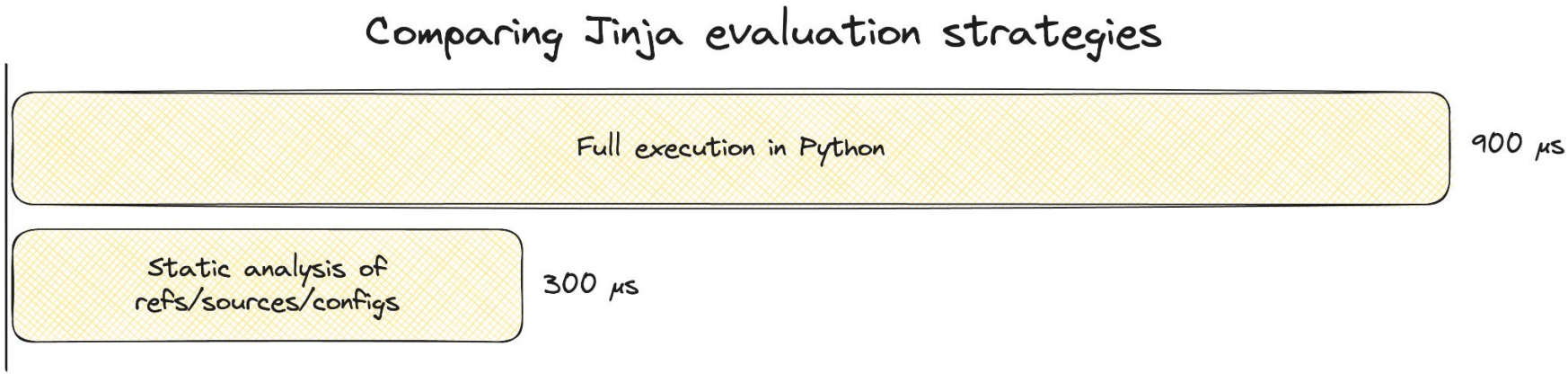
LEARNING OBJECTIVES

- Understand how the Parser and Compiler work
- Feature Matrix - dbt Fusion vs dbt Core
- License differences
- Get hands-on with dbt Fusion and the VSCode Extension

dbt Fusion

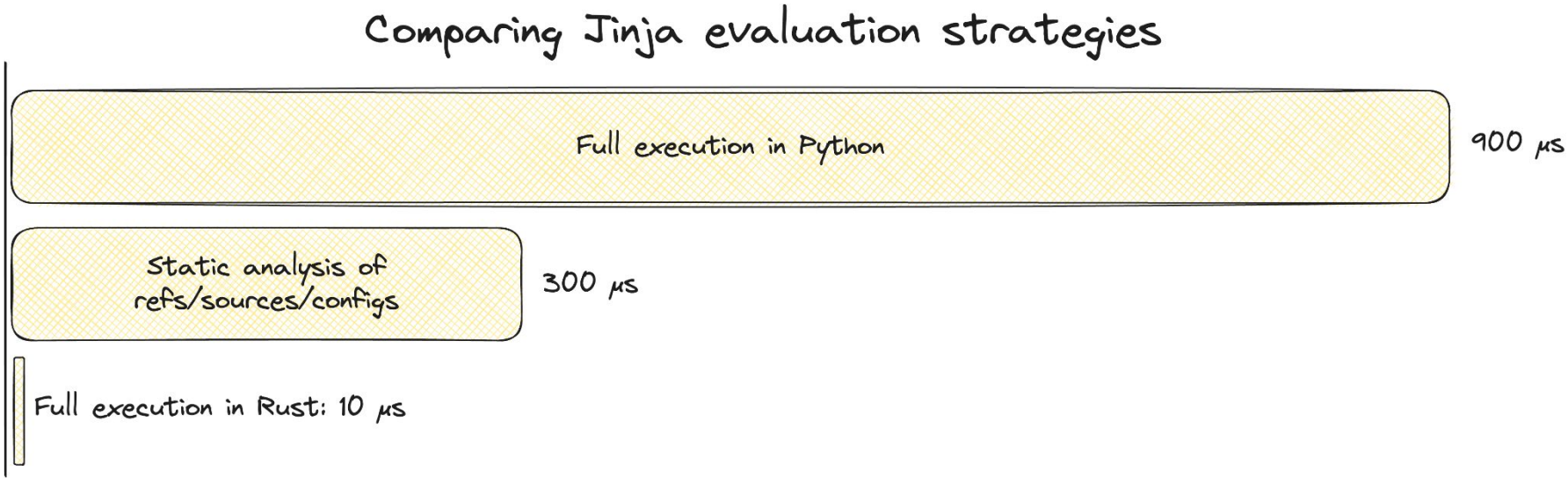
Technical Deep Dive

PARSING



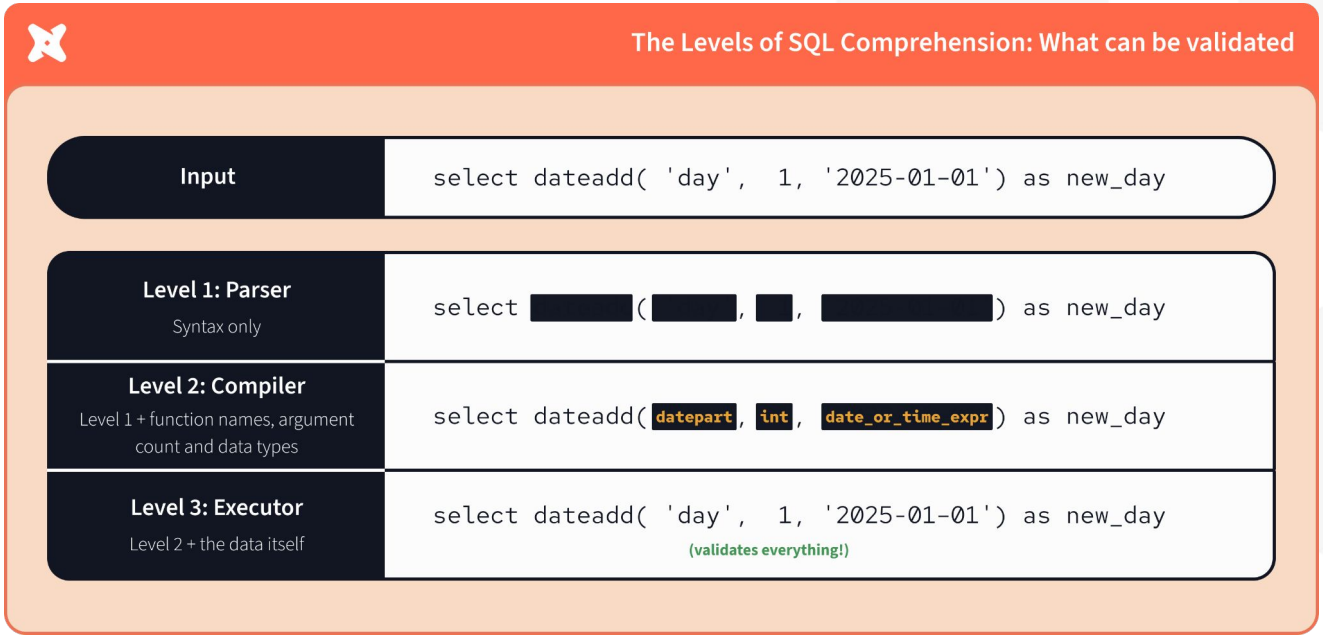
Source: https://docs.getdbt.com/blog/faster-project-parsing-with-rust

PARSING WITH RUST



Source: https://docs.getdbt.com/blog/faster-project-parsing-with-rust

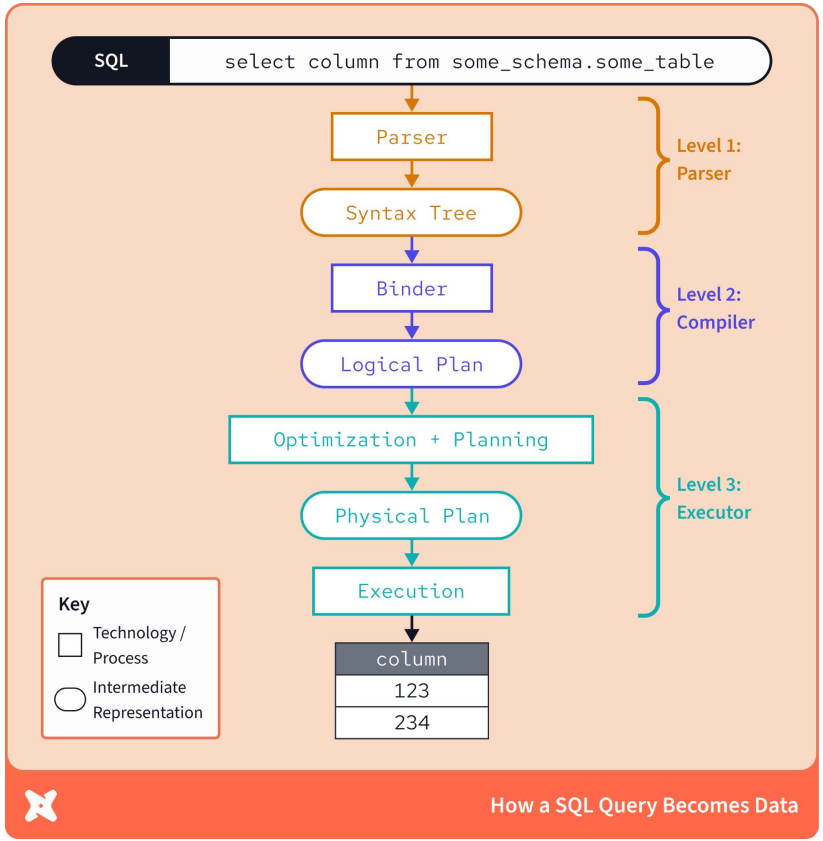
SQL COMPREHENSION



Source: https://docs.getdbt.com/blog/faster-project-parsing-with-rust

SQL COMPREHENSION

- **dbt Core**: let the Data Warehouse compile the SQL code
- **dbt Fusion**: let's compile and comprehend as much as possible without touching the DWH



Source: https://docs.getdbt.com/blog/faster-project-parsing-with-rust

dbt Fusion License

- dbt Core: **Apache 2**
- dbt Fusion: Mostly Elastic License v2 (**ELv2**)
    - You can use it for free, but you cannot provide it as a service for free
- Might include proprietary licensed enterprise features later
- For you most probably the license change won't make a difference

Source: https://www.getdbt.com/licenses-faq

VISUAL STUDIO CODE EXTENSION

Let’s go ahead and install the **VSCode Extension**, which will:

- Install dbt Fusion
- Upgrade your dbt project to be 100% Fusion compatible
- Compile your project for SQL Comprehension

HERO

ORCHESTRATION

THE ORCHESTRATION LANDSCAPE

Apache Airflow

Azure Data Factory

PREFECT

dagster

dbt Cloud

DAGSTER - SIMILAR DATA CONCEPTS

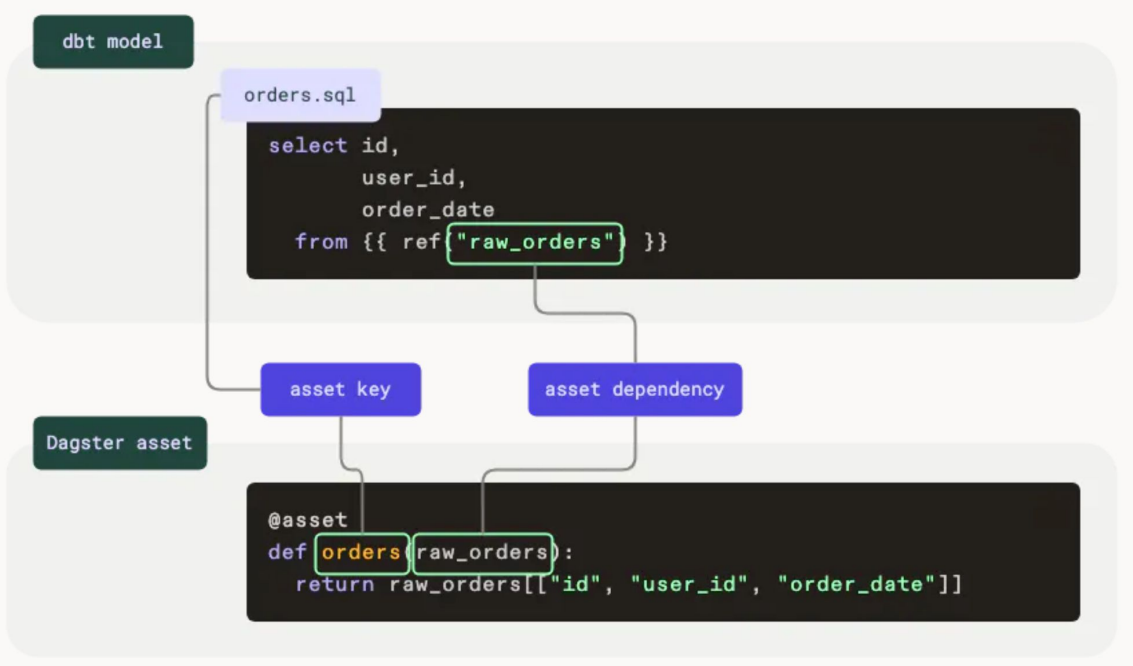


Image source: https://docs.dagster.io/integrations/dbt

GREAT UI

dagster

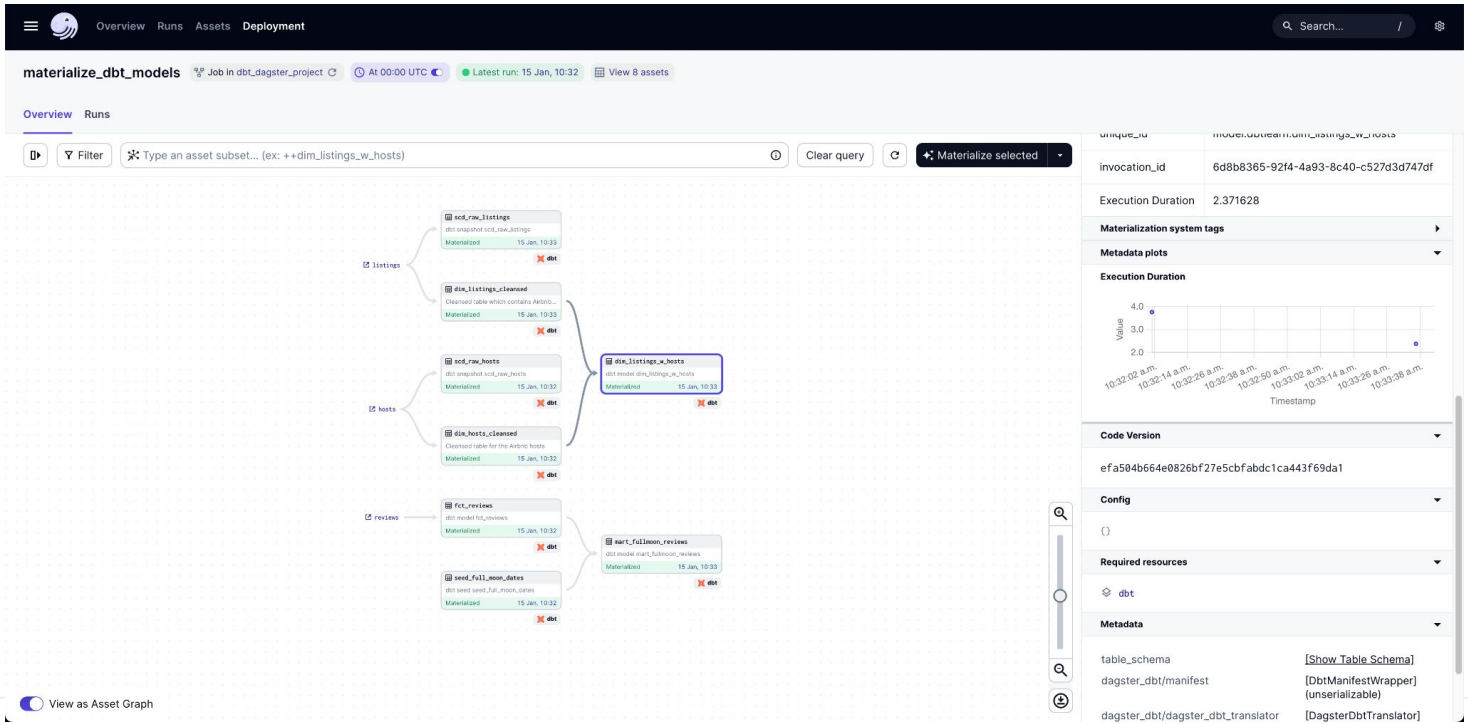


Image source: https://docs.dagster.io/integrations/dbt

EASY TO DEBUG

dagster

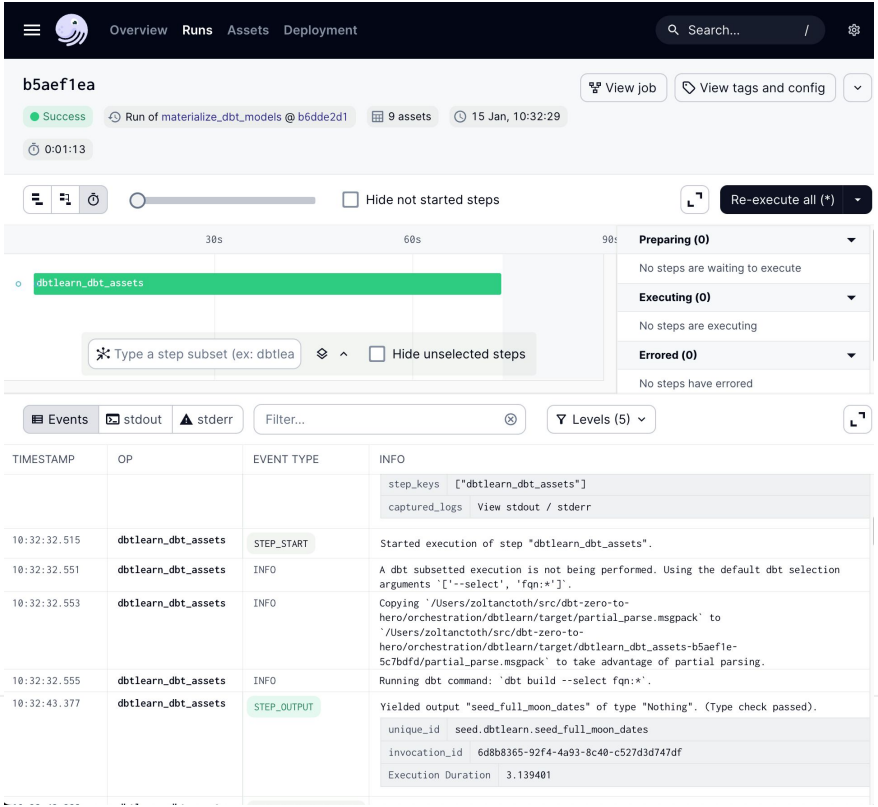


Image source: https://docs.dagster.io/integrations/dbt

CONGRATS!

FROM THIS POINT ON THE SLIDES ARE WORK IN PROGRESS