

# dbt (Data Build Tool)

떠오르는 ELT 툴!



---

Grepp, Inc

---

한기용

---

keeyonghan@hotmail.com

---

# Contents

- |                           |                       |
|---------------------------|-----------------------|
| 1. ELT의 미래는?              | 8. dbt Seeds          |
| 2. Database Normalization | 9. dbt Sources        |
| 3. dbt 소개                 | 10. dbt Snapshots     |
| 4. dbt 사용 시나리오            | 11. dbt Tests         |
| 5. dbt 설치와 환경 설정          | 12. dbt Documentation |
| 6. dbt Models: Input      | 13. dbt Expectations  |
| 7. dbt Models: Output     | 14. 마무리               |



# dbt Seeds

dbt Seeds란 무엇인가?


## ◆ Seeds 소개

- ❖ 많은 dimension 테이블들은 크기가 작고 많이 변하지 않음
- ❖ Seeds는 이를 파일 형태로 데이터웨어하우스로 로드하는 방법
  - Seeds는 작은 파일 데이터를 지칭 (보통 csv 파일)
- ❖ dbt seed를 실행해서 빌드

## ◆ Seeds 실습 (1)

- ❖ seeds 폴더 밑에 적당히 .csv 파일을 하나 생성
- ❖ 나중에 이 파일 이름으로 테이블이 생성됨
  - **reference\_date**

seeds/reference\_date.csv



```
date
2023-01-01
2023-01-02
2023-01-03
2023-01-04
2023-01-05
2023-01-06
2023-01-07
2023-01-08
2023-01-09
2023-01-10
2023-01-11
2023-01-12
2023-01-13
...
```

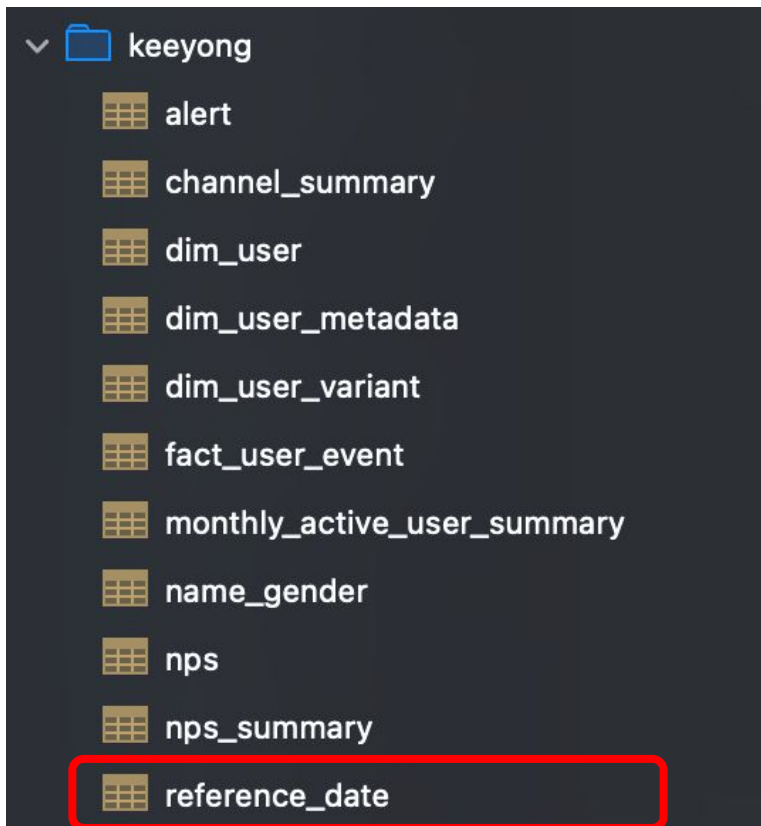
## ◆ Seeds 실습 (2)

### ❖ 다음으로 dbt seed 실행

```
keyyong learn_dbt % dbt seed
22:00:04 Running with dbt=1.4.3
22:00:04 Found 7 models, 0 tests, 0 snapshots, 0 analyses, 327 macros, 0 operations, 1 seed file, 0 sources, 0
exposures, 0 metrics
22:00:04
22:00:10 Concurrency: 1 threads (target='dev')
22:00:10
22:00:10 1 of 1 START seed file keyyong.reference_date ..... [RUN]
22:00:12 1 of 1 OK loaded seed file keyyong.reference_date ..... [INSERT 31 in 2.16s]
22:00:14
22:00:14 Finished running 1 seed in 0 hours 0 minutes and 9.33 seconds (9.33s).
22:00:14
22:00:14 Completed successfully
22:00:14
22:00:14 Done. PASS=1 WARN=0 ERROR=0 SKIP=0 TOTAL=1
```

## ◆ Seeds 실습 (3)

- ❖ 실행 결과 확인
- ❖ 다른 SQL에서 사용시
  - {{ ref("reference\_date") }}





# dbt Sources

dbt Sources란 무엇인가?



## ◆ Staging 테이블을 만들 때 입력 테이블들이 자주 바뀐다면?

- ❖ `models` 밑의 `.sql` 파일들을 일일이 찾아 바꿔주어야함
- ❖ 이 번거로움을 해결하기 위한 것이 **Sources**
  - 입력 테이블에 별칭을 주고 별칭을 **staging** 테이블에서 사용

## ◆ Sources 소개

- ❖ 기본적으로 처음 입력이 되는 **ETL** 테이블을 대상으로 함
  - 별칭 제공
  - 최신 레코드 체크 기능 제공
- ❖ 테이블 이름들에 별명(**alias**)을 주는 것
  - 이를 통해 **ETL**단의 소스 테이블이 바뀌어도 뒤에 영향을 주지 않음
  - 추상화를 통한 변경처리를 용이하게 하는 것
  - 이 별명은 **source** 이름과 새 테이블 이름의 두 가지로 구성됨
    - 예) `raw_data.user_metadata` -> `keeyong, metadata`
- ❖ **Source** 테이블들에 새 레코드가 있는지 체크해주는 기능도 제공

## ◆ Sources 실습 (1)

### ❖ models/sources.yml 파일 생성

```
version: 2
```

```
sources:
```

- name: keeyong  
schema: raw\_data  
tables:
  - name: metadata  
identifier: user\_metadata
  - name: event  
identifier: user\_event
  - name: variant  
identifier: user\_variant

```
keeyong learn_dbt % cd models  
keeyong models % vi sources.yml
```


raw\_data.user\_metadata는  
JINJA에서 **source("keeyong", "metadata")**로 지칭됨

## ◆ Sources 실습 (2)

❖ 아래는 src\_user\_event.sql의 예

- models 밑의 다른 파일들도 적절하게 변경

```
WITH src_user_event AS (  
  SELECT * FROM raw_data.user_event  
)  
SELECT  
  user_id,  
  datestamp,  
  item_id,  
  clicked,  
  purchased,  
  paidamount  
FROM  
  src_user_event
```



The diagram illustrates the mapping between a table name in a SQL query and its corresponding dbt source macro. A red box highlights the text `raw_data.user_event` in the `SELECT * FROM` clause of the `WITH` statement. A grey arrow points from this box to a blue box containing the dbt source macro `{{ source ("keeyong", "event") }}`.

## ◆ Sources 최신성 (Freshness)

- ❖ 특정 데이터가 소스와 비교해서 얼마나 최신성이 떨어지는지 체크하는 기능
- ❖ `dbt source freshness` 명령으로 수행
- ❖ 이를 하려면 `models/sources.yml`의 해당 테이블 밑에 아래 추가

freshness를 결정해주는  
필드

version: 2

sources:

- name: keeyong

schema: raw\_data

tables:

- name: event

identifier: user\_event

**loaded\_at\_field: timestamp**

**freshness:**

**warn\_after: { count: 1, period: hour }**

**error\_after: { count: 24, period: hour }**

지금 `raw_data.user_event`  
테이블에서 `timestamp`의 최대값이  
현재 시간보다 1시간 이상 뒤쳐져  
있지만 24시간은 아니라면 **warning**.  
24시간 이상이라면 **error!**

## ◆ Sources 최신성 (Freshness)

### ❖ dbt source freshness 명령으로 수행

```
keepyong learn_dbt % dbt source freshness
20:33:41 Running with dbt=1.5.1
20:33:41 Found 8 models, 0 tests, 0 snapshots, 0 analyses, 346 macros, 0 operations, 1 seed file, 3
sources, 0 exposures, 0 metrics, 0 groups
20:33:41
20:33:45 Concurrency: 1 threads (target='dev')
20:33:45
20:33:45 1 of 1 START freshness of keepyong.event ..... [RUN]
20:33:47 1 of 1 ERROR STALE freshness of keepyong.event ..... [ERROR ST
ALE in 1.91s]
20:33:47
20:33:47 Done.
```



# dbt Snapshots

dbt Snapshots이란 무엇인가?

## ◆ 데이터베이스에서 스냅샷이란?

- ❖ **Dimension** 테이블은 성격에 따라 변경이 자주 생길 수 있음
- ❖ **dbt**에서는 테이블의 변화를 계속적으로 기록함으로써 과거 어느 시점이건 다시 돌아가서 테이블의 내용을 볼 수 있는 기능을 이야기함
  - 이를 통해 테이블에 문제가 있을 경우 과거 데이터로 롤백 가능
  - 다양한 데이터 관련 문제 디버깅도 쉬워짐



## ◆ SCD Type 2와 dbt (1)

- ❖ Dimension 테이블에서 특정 **entity**에 대한 데이터가 변경되는 경우
- ❖ 예) **employee\_jobs** 테이블
  - 특정 **employee\_id**의 **job\_code**가 바뀌는 경우
  - 변경시간도 같이 추가되어야함

EMPLOYEE_ID	JOB_CODE
E001	J01
E002	J02
E003	J02



E002의 JOB CODE가 J03으로  
변경

## ◆ SCD Type 2와 dbt (2)

- ❖ Dimension 테이블에서 특정 **entity**에 대한 데이터가 변경되는 경우
- ❖ 새로운 Dimension 테이블을 생성 (history/snapshot 테이블)

E002의 JOB  
CODE가 J03으로  
변경



EMPLOYEE_ID	JOB_CODE	DBT_VALID_FROM	DBT_VALID_TO
E002	J02	2020-01-01	2023-02-01
E002	<b>J03</b>	<b>2023-02-01</b>	<b>NULL</b>

이외에도 dbt\_scd\_id,  
dbt\_updated\_at 등의 필드가 추가됨

## ◆ dbt의 스냅샷 처리 방법

- ❖ 먼저 snapshots 폴더에 환경설정이 됨
- ❖ snapshots을 하려면 데이터 소스가 일정 조건을 만족해야함
  - Primary key가 존재해야함
  - 레코드의 변경시간을 나타내는 타임스탬프 필요 (updated\_at, modified\_at 등등)
- ❖ 변경 감지 기준
  - Primary key 기준으로 변경시간이 현재 DW에 있는 시간보다 미래인 경우
- ❖ Snapshots 테이블에는 총 4개의 타임스탬프가 존재
  - dbt\_scd\_id, dbt\_updated\_at
  - valid\_from, valid\_to

## ◆ dbt snapshot 적용해 보기

### ❖ snapshots/scd\_user\_metadata.sql 편집

```
{% snapshot scd_user_metadata %}
```

```
{{  
  config(  
    target_schema='keeyong',  
    unique_key='user_id',  
    strategy='timestamp',  
    updated_at='updated_at',  
    invalidate_hard_deletes=True  
  )  
}}
```

```
SELECT * FROM {{ source('keeyong', 'metadata') }}
```

```
{% endsnapshot %}
```

## ◆ dbt snapshot 실행

### ❖ dbt snapshot

```
keeyong learn_dbt % dbt snapshot
07:16:35 Running with dbt=1.4.3
07:16:35 Found 7 models, 3 tests, 1 snapshot, 0 analyses, 327 macros, 0 operations, 1 seed file, 3 sources, 0 exposures, 0 metrics
07:16:35
07:16:43 Concurrency: 1 threads (target='dev')
07:16:43
07:16:43 1 of 1 START snapshot keeyong.scd_user_metadata ..... [RUN]
07:16:57 1 of 1 OK snapshotted keeyong.scd_user_metadata ..... [success in 13.96s]
07:16:59
07:16:59 Finished running 1 snapshot in 0 hours 0 minutes and 24.40 seconds (24.40s).
07:16:59
07:16:59 Completed successfully
07:16:59
07:16:59 Done. PASS=1 WARN=0 ERROR=0 SKIP=0 TOTAL=1
```

## ◆ keeyong.scd\_user\_metadata 확인해보기

user_id	age	gender	updated_at	dbt_scd_id	dbt_updated_at	dbt_valid_from	dbt_valid_to
1	50-up	female	2023-03-04 04:57:21	0687fc222d1baca5a1a5203152a50c3a	2023-03-04 04:57:21	2023-03-04 04:57:21	NULL
3	50-up	female	2023-03-04 04:57:21	c4bacbbc0d1d81defb6038db82445987	2023-03-04 04:57:21	2023-03-04 04:57:21	NULL
5	0-19	female	2023-03-04 04:57:21	89dae91ac5510bdc21e97495f57b7f3f	2023-03-04 04:57:21	2023-03-04 04:57:21	NULL
7	50-up	female	2023-03-04 04:57:21	c0592442b1c2b407b284e4eb4e773611	2023-03-04 04:57:21	2023-03-04 04:57:21	NULL
9	0-19	male	2023-03-04 04:57:21	73f5943042c43e78a3dbc88cfdfe6433	2023-03-04 04:57:21	2023-03-04 04:57:21	NULL
11	20-49	female	2023-03-04 04:57:21	5694ba404adc403556866c10be737619	2023-03-04 04:57:21	2023-03-04 04:57:21	NULL
13	0-19	female	2023-03-04 04:57:21	cc2a3662cf78c1b23234d2cd4c7b8cc6	2023-03-04 04:57:21	2023-03-04 04:57:21	NULL
15	50-up	male	2023-03-04 04:57:21	cd7eea7d005fb7367e987446b67010e8	2023-03-04 04:57:21	2023-03-04 04:57:21	NULL

## ◆ raw\_data.user\_metadata 업데이트 실습 개요

- ❖ 소스 테이블의 기존 레코드 하나의 age 그룹을 변경
- ❖ 그리고 dbt snapshot 실행
- ❖ keeyong.scd\_user\_metadata에 레코드 추가되었는지 확인

## ◆ raw\_data.user\_metadata 업데이트 실습 (1)

❖ 소스 테이블의 기존 레코드 하나의 age 그룹을 변경

```
SELECT *  
FROM raw_data.user_metadata  
WHERE user_id = 99;
```

user_id	age	gender	updated_at
99	0-19	female	2023-03-04 04:57:21

```
UPDATE raw_data.user_metadata  
SET age = '20-29', updated_at = GETDATE()  
WHERE user_id = 99;
```

user_id	age	gender	updated_at
99	20-29	female	2023-06-22 21:45:53



## ◆ raw\_data.user\_metadata 업데이트 실습 (2)

### ❖ dbt snapshot 실행

```
21:46:55 Running with dbt=1.4.3
21:46:55 Found 8 models, 0 tests, 1 snapshot, 0 analyses, 327 macros, 0 operations, 1 seed file, 3 source
s, 0 exposures, 0 metrics
21:46:55
21:47:01 Concurrency: 1 threads (target='dev')
21:47:01
21:47:01 1 of 1 START snapshot keeyong.scd_user_metadata ..... [RUN]
21:47:05 1 of 1 OK snapshotted keeyong.scd_user_metadata ..... [success in 4.4
2s]
21:47:07
21:47:07 Finished running 1 snapshot in 0 hours 0 minutes and 11.58 seconds (11.58s).
21:47:07
21:47:07 Completed successfully
21:47:07
21:47:07 Done. PASS=1 WARN=0 ERROR=0 SKIP=0 TOTAL=1
```

## ◆ raw\_data.user\_metadata 업데이트 실습 (3)

❖ keeyong.scd\_user\_metadata에 레코드 추가되었는지 확인

```
SELECT * FROM keeyong.scd_user_metadata WHERE user_id = 99;
```

user_id	age	gender	updated_at	dbt_scd_id	dbt_updated_at	dbt_valid_from	dbt_valid_to
99	0-19	female	2023-03-04 04:57:21	e9f93670bcce2a93d907b5c7d18e1f43	2023-03-04 04:57:21	2023-03-04 04:57:21	2023-06-22 21:45:53
99	20-29	female	2023-06-22 21:45:53	1a1295ce6c13e8d16a900bdd8f191473	2023-06-22 21:45:53	2023-06-22 21:45:53	NULL



# dbt Tests

dbt Tests란 무엇인가?

## ◆ Tests 소개

❖ 데이터 품질을 테스트하는 방법

❖ 두 가지가 존재

- 내장 일반 테스트 (“Generic”)

- unique, not\_null, accepted\_values, relationships 등의 테스트 지원
- models 폴더

- 커스텀 테스트 (“Singular”)

- 기본적으로 SELECT로 간단하며 결과가 리턴되면 “실패”로 간주
- tests 폴더

## ◆ Generic Tests 구현

### ❖ models/schema.yml 파일 생성

version: 2

models:

- name: dim\_user\_metadata

columns:

- name: **user\_id**

tests:

- **unique**

- **not\_null**

## ◆ Generic Tests 실행

### ❖ dbt test로 테스트 실행

```
keyyong learn_dbt % dbt test
00:10:33 Running with dbt=1.4.3
00:10:33 Found 7 models, 2 tests, 0 snapshots, 0 analyses, 327 macros, 0 operations, 1 seed file, 3 sources, 0
exposures, 0 metrics
00:10:33
00:10:37 Concurrency: 1 threads (target='dev')
00:10:37
00:10:37 1 of 2 START test not_null_dim_user_metadata_user_id ..... [RUN]
00:10:39 1 of 2 PASS not_null_dim_user_metadata_user_id ..... [PASS in 1.92s]
00:10:39 2 of 2 START test unique_dim_user_metadata_user_id ..... [RUN]
00:10:41 2 of 2 PASS unique_dim_user_metadata_user_id ..... [PASS in 1.81s]
00:10:43
00:10:43 Finished running 2 tests in 0 hours 0 minutes and 9.38 seconds (9.38s).
00:10:43
00:10:43 Completed successfully
00:10:43
00:10:43 Done. PASS=2 WARN=0 ERROR=0 SKIP=0 TOTAL=2
```

## ◆ Singular Tests 구현

### ❖ tests/dim\_user\_metadata.sql 파일 생성

- Primary Key Uniqueness 테스트

```
SELECT
  *
FROM (
  SELECT
    user_id, COUNT(1) cnt
  FROM
    {{ ref("dim_user_metadata") }}
  GROUP BY 1
  ORDER BY 2 DESC
  LIMIT 1
)
WHERE cnt > 1
```

## ◆ Singular Tests 실행

### ❖ dbt test로 테스트 실행

```
keyyong learn_dbt % dbt test
00:16:56 Running with dbt=1.4.3
00:16:56 Found 7 models, 3 tests, 0 snapshots, 0 analyses, 327 macros, 0 operations, 1 seed file, 3 sources, 0
exposures, 0 metrics
00:16:56
00:17:00 Concurrency: 1 threads (target='dev')
00:17:00
00:17:00 1 of 3 START test dim_user_metadata ..... [RUN]
00:17:06 1 of 3 PASS dim_user_metadata ..... [PASS in 5.60s]
00:17:06 2 of 3 START test not_null_dim_user_metadata_user_id ..... [RUN]
00:17:07 2 of 3 PASS not_null_dim_user_metadata_user_id ..... [PASS in 1.70s]
00:17:07 3 of 3 START test unique_dim_user_metadata_user_id ..... [RUN]
00:17:09 3 of 3 PASS unique_dim_user_metadata_user_id ..... [PASS in 1.69s]
00:17:11
00:17:11 Finished running 3 tests in 0 hours 0 minutes and 14.66 seconds (14.66s).
00:17:11
00:17:11 Completed successfully
00:17:11
00:17:11 Done. PASS=3 WARN=0 ERROR=0 SKIP=0 TOTAL=3
```



## ◆ 하나의 테이블 대상으로 실행해보기

### ❖ dbt test --select dim\_user\_metadata

- 만일 디버깅하고 싶다면 `dbt --debug test --select dim_user_metadata`

```
keeyong learn_dbt % dbt test --select dim_user_metadata
08:09:13 Running with dbt=1.4.3
08:09:13 Found 7 models, 3 tests, 1 snapshot, 0 analyses, 327 macros, 0 operations, 1 seed file, 3 sources, 0 exposures, 0 metrics
08:09:13
08:09:17 Concurrency: 1 threads (target='dev')
08:09:17
08:09:17 1 of 3 START test dim_user_metadata ..... [RUN]
08:09:19 1 of 3 PASS dim_user_metadata ..... [PASS in 1.89s]
08:09:19 2 of 3 START test not_null_dim_user_metadata_user_id ..... [RUN]
08:09:21 2 of 3 PASS not_null_dim_user_metadata_user_id ..... [PASS in 1.65s]
08:09:21 3 of 3 START test unique_dim_user_metadata_user_id ..... [RUN]
08:09:22 3 of 3 PASS unique_dim_user_metadata_user_id ..... [PASS in 1.74s]
08:09:24
08:09:24 Finished running 3 tests in 0 hours 0 minutes and 10.90 seconds (10.90s).
08:09:24
08:09:24 Completed successfully
08:09:24
08:09:24 Done. PASS=3 WARN=0 ERROR=0 SKIP=0 TOTAL=3
```



# dbt Documentation

dbt Documentation이란 무엇인가?

## ◆ dbt 문서화 소개

- ❖ 기본 철학은 문서와 소스 코드를 최대한 가깝게 배치하자는 것
- ❖ 문서화 자체는 두 가지 방법이 존재
  - 기존 **.yml** 파일에 문서화 추가 (선호되는 방식)
  - 독립적인 **markdown** 파일 생성
- ❖ 이를 경량 웹서버로 서빙
  - **overview.md**가 기본 홈페이지가 됨
  - 이미지등의 **asset** 추가도 가능

## ◆ models 문서화 하기

❖ description 키를 추가: models/schema.yml, models/sources.yml

version: 2

models:

- name: dim\_user\_metadata

**description: A dimension table with user metadata**

columns:

- name: user\_id

**description: The primary key of the table**

tests:

- unique
- not\_null

## ◆ models 문서 만들기

### ❖ dbt docs generate

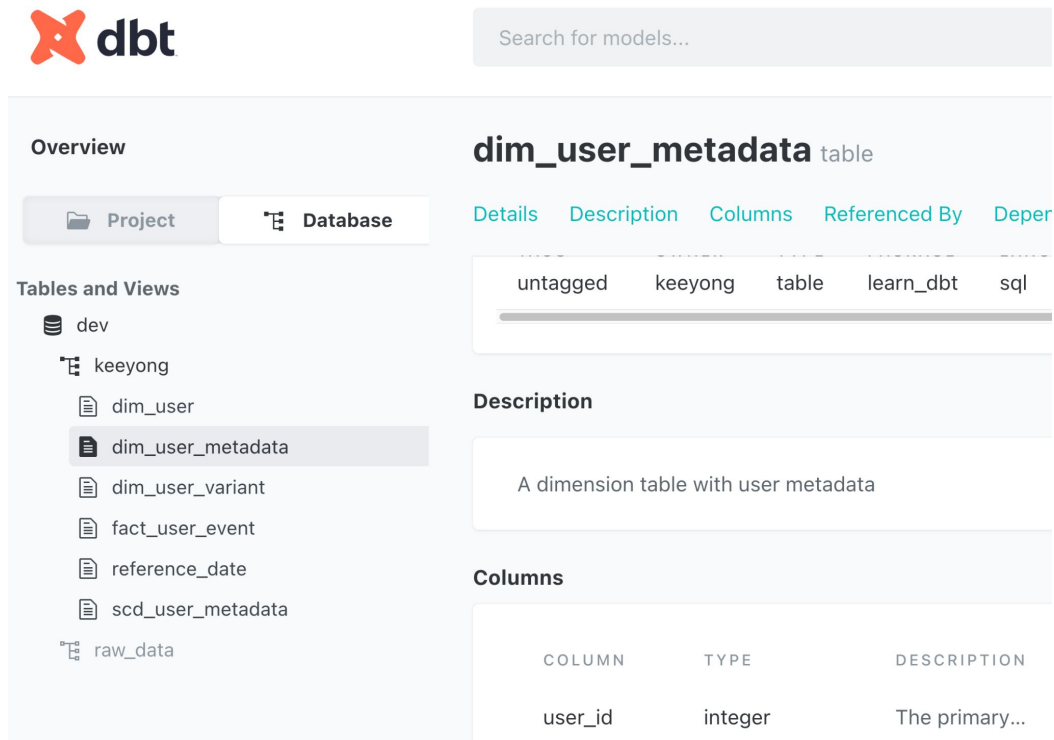
- 사용자 권한이 더 있다면 Redshift로부터 더 많은 정보를 가져다가 보여줌
- 결과 파일은 `target/catalog.json` 파일이 됨

```
keeyong learn_dbt % dbt docs generate
07:54:05 Running with dbt=1.4.3
07:54:06 Found 7 models, 3 tests, 1 snapshot, 0 analyses, 327 macros, 0 operations, 1 seed file, 3 sources, 0 exposures, 0 metrics
07:54:06
07:54:10 Concurrency: 1 threads (target='dev')
07:54:10
07:54:10 Done.
07:54:10 Building catalog
07:54:12
```

```
Warning: The database user "keeyong" has insufficient permissions to
query the "svv_table_info" table. Please grant SELECT permissions on this table
to the "keeyong" user to fetch extended table details from Redshift.
```

## ◆ models 문서 보기

### ❖ dbt docs serve로 웹서버 띄우기



The screenshot displays the dbt documentation interface. At the top left is the dbt logo. To its right is a search bar labeled "Search for models...". Below the logo, there are two tabs: "Project" and "Database". The "Database" tab is selected, showing a list of tables and views under the "dev" schema. The table "dim\_user\_metadata" is highlighted. To the right of the table list, the details for "dim\_user\_metadata" are shown. It is identified as a "table". Below this, there are tabs for "Details", "Description", "Columns", "Referenced By", and "Dependencies". The "Description" tab is selected, showing the text "A dimension table with user metadata". Below the description, the "Columns" section is visible, showing a table with columns: COLUMN, TYPE, and DESCRIPTION. The first row shows "user\_id" as an "integer" with the description "The primary...".

dbt

Search for models...

Overview

Project Database

Tables and Views

dev

keyong

- dim\_user
- dim\_user\_metadata**
- dim\_user\_variant
- fact\_user\_event
- reference\_date
- scd\_user\_metadata

raw\_data

**dim\_user\_metadata** table

Details Description Columns Referenced By Dependencies

untagged keyong table learn\_dbt sql

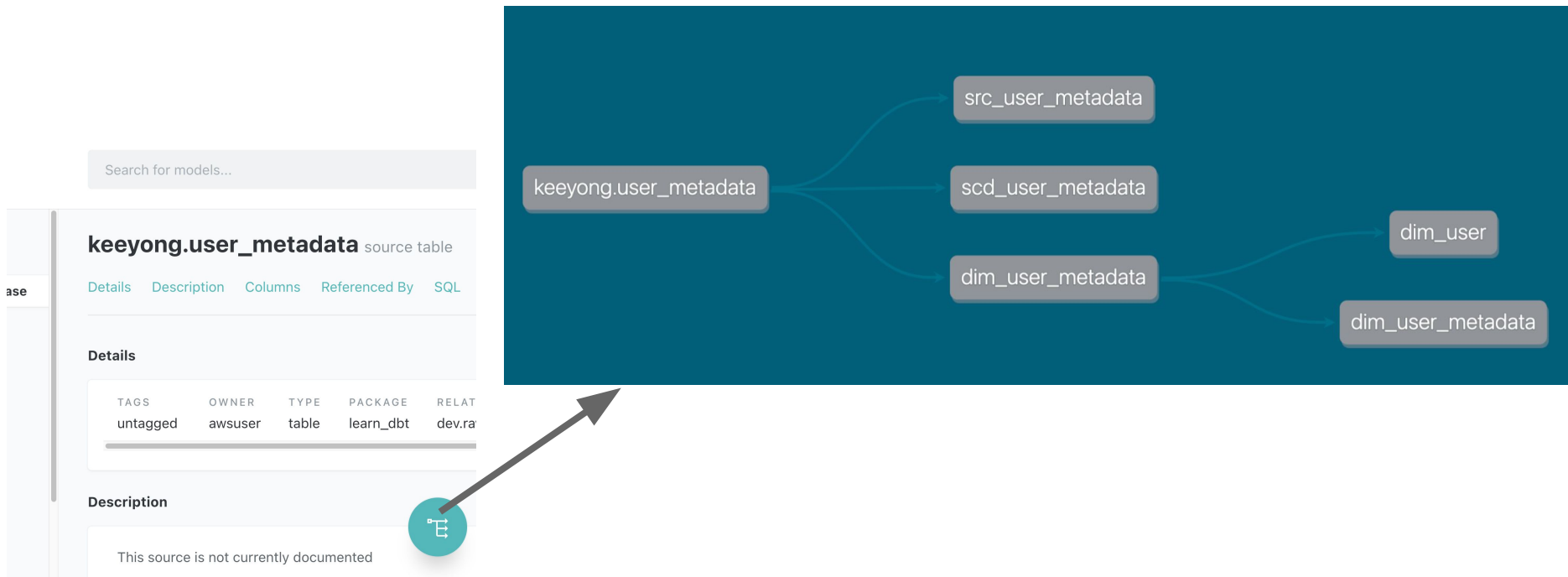
**Description**

A dimension table with user metadata

**Columns**

COLUMN	TYPE	DESCRIPTION
user_id	integer	The primary...

# ◆ Lineage Graph 확인해보기





# dbt Expectations

dbt Expectations이란 무엇인가?



## ◆ dbt Expectations 소개

- ❖ Great Expectations에서 영감을 받아 dbt용으로 만든 dbt 확장판
  - <https://github.com/calogica/dbt-expectations>
- ❖ 설치 후 packages.yml에 등록
  - packages:
    - package: calogica/dbt\_expectations
    - version: [">=0.7.0", "<0.8.0"]
- ❖ 보통은 앞서 dbt 제공 테스트들과 같이 사용
  - models/schema.yml

## ◆ dbt Expectations 함수들 일부

- `expect_column_to_exist`
- `expect_row_values_to_have_recent_data`
- `expect_column_values_to_be_null`
- `expect_column_values_to_not_be_null`
- `expect_column_values_to_be_unique`
- `expect_column_values_to_be_of_type`
- `expect_column_values_to_be_in_set`
- `expect_column_values_to_not_be_in_set`
- `expect_column_values_to_be_between`

# 요약

- ELT 품질의 중요성
- dbt 소개
- 데이터 품질 테스트: Tests
- 아주 유용한 기능: Snapshots
- 용이한 Documentation
- 테스트 기능 Expectations



Grepp Inc

한기용

keeyonghan@hotmail.com



# dbt와 Airflow 연동

dbt를 Airflow를 통해 연동하는 방법을 찾아보자

## ◆ 2가지 옵션

### ❖ dbt Core를 쓴다면 K8s 사용

- dbt Core + dbt Job (Project)를 Docker Image로 만들어 실행

### ❖ dbt Cloud를 쓴다면 [DbtCloudRunJobOperator](#)를 사용

- 이 경우 dbt Job이 하나의 태스크로 실행됨.
- Connector 시스템이 dbt Cloud에서 접근 가능해야함
- [예제 코드](#) 참고