

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 |

ELT의 미래는?

ETL을 하는 이유는 결국 ELT를 하기 위함이며
이 때 데이터 품질 검증이 중요해짐

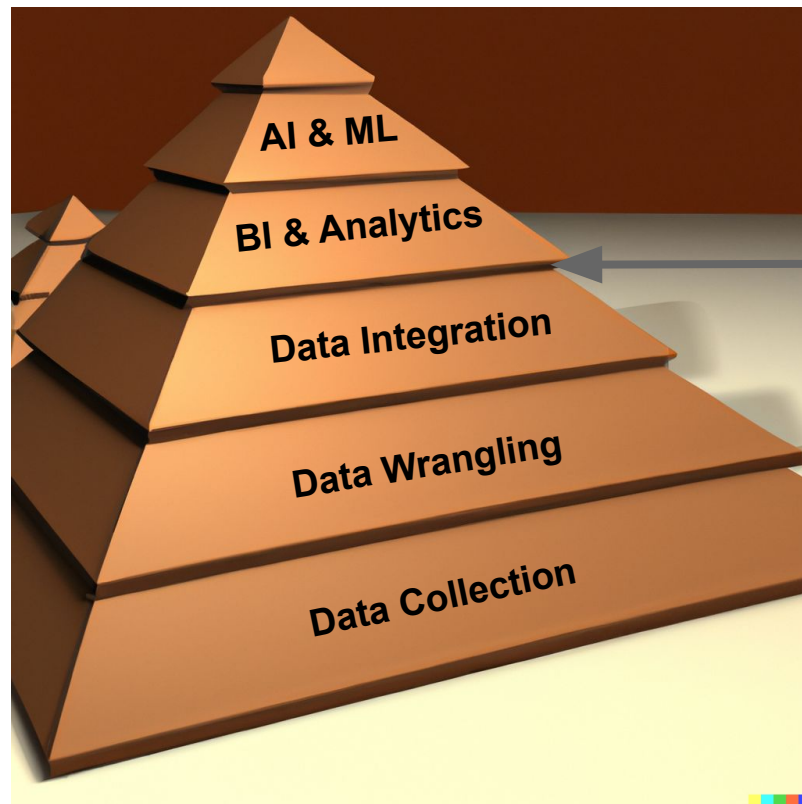
◆ 데이터 품질의 중요성 증대

- ❖ 입출력 체크
 - ❖ 더 다양한 품질 검사
 - ❖ 리니지 체크
 - ❖ 데이터 히스토리 파악
-
- ❖ 데이터 품질 유지 -> 비용/노력 감소와 생산성 증대의 지름길

Database Normalization

1NF, 2NF, 3NF 등의 Normalization과 SCD
Type들에 대해 배워보자

◆ Data Maturity Model and Reality



여기를 경계로
큰 갭이 존재

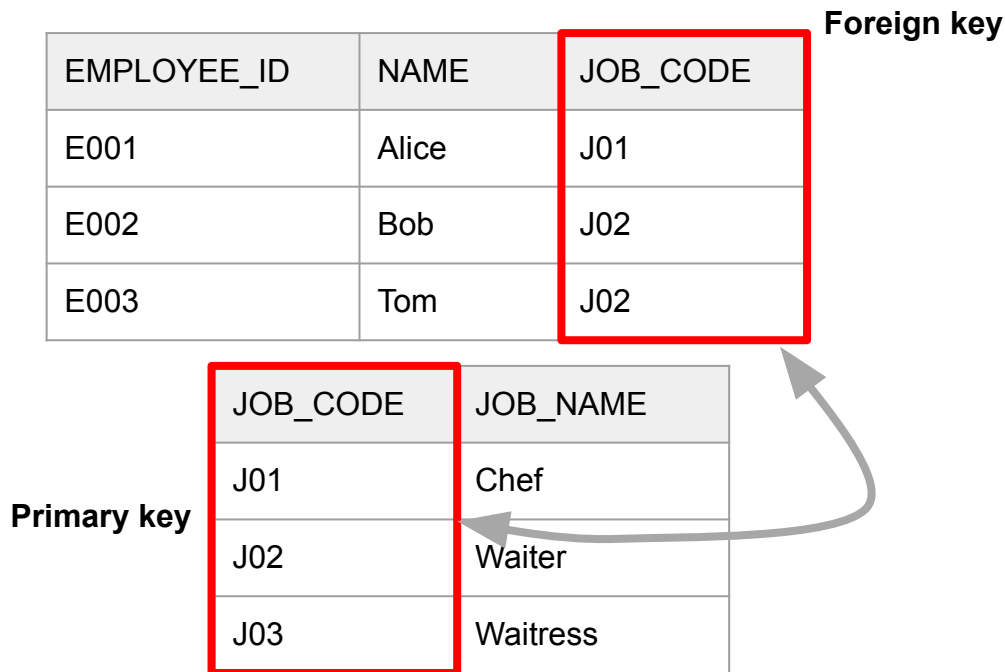
◆ Database Normalization

❖ 데이터베이스를 좀더 조직적이고 일관된 방법으로 디자인하려는 방법

- 데이터베이스 정합성을 쉽게 유지하고 레코드들을 수정/적재/삭제를 용이하게 하는 것

❖ Normalization에 사용되는 개념

- Primary Key
- Composite Key
- Foreign Key



◆ 1NF (First Normal Form)

- ❖ 한 셀에는 하나의 값만 있어야함 (atomicity)
- ❖ **Primary Key**가 있어야함
- ❖ 중복된 키나 레코드들이 없어야함

목표는 중복을 제거하고 **atomicity**를 갖는 것

◆ 1NF (First Normal Form) 예제

❖ Employee 테이블

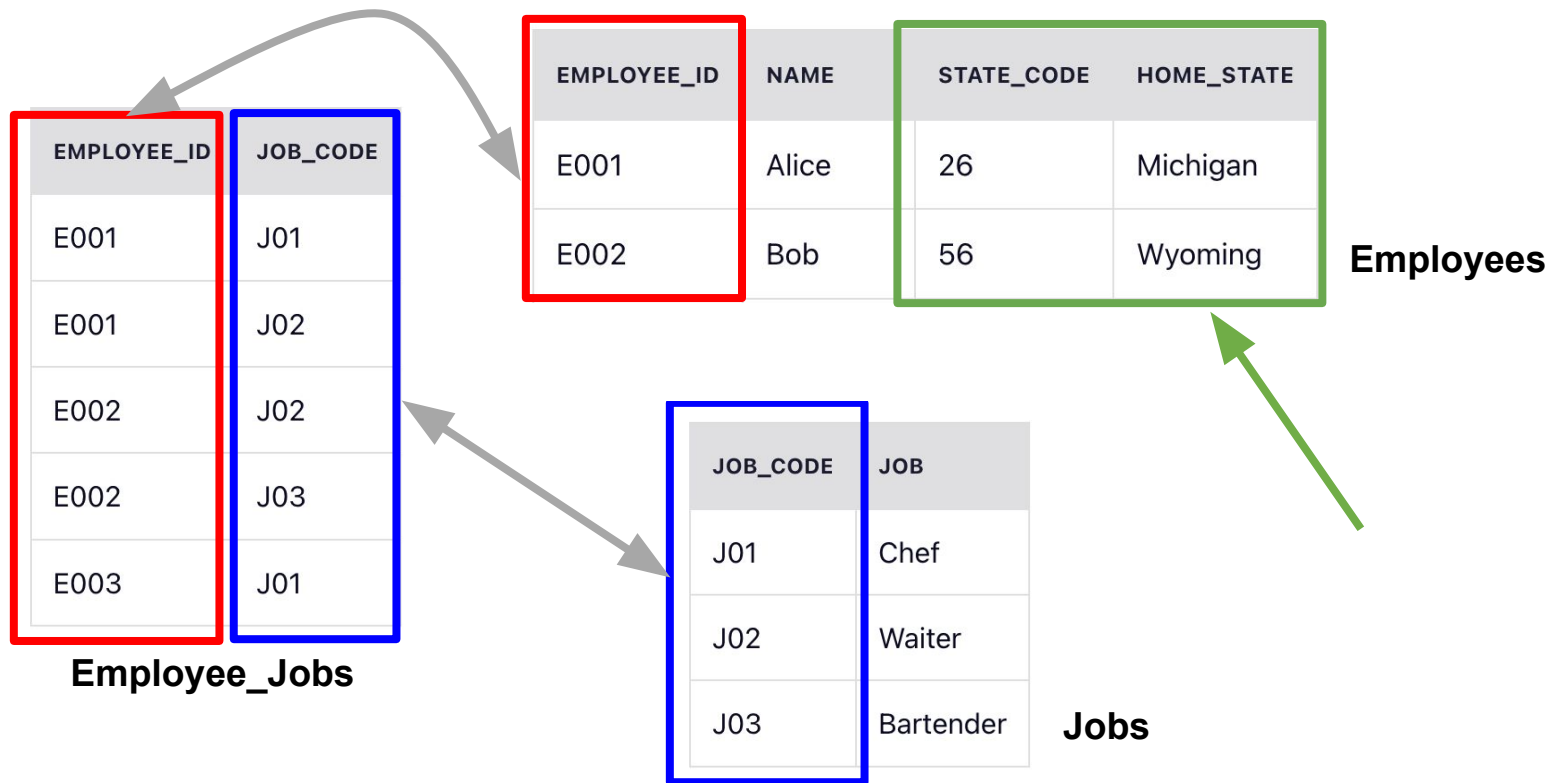
EMPLOYEE_ID	NAME	JOB_CODE	JOB	STATE_CODE	HOME_STATE
E001	Alice	J01	Chef	26	Michigan
E002	Bob	J02	Waiter	56	California
E003	Tom	J02	Waiter	51	Oregon

◆ 2NF (Second Normal Form)

- ❖ 일단 1NF를 만족해야함
- ❖ 다음으로 **Primary Key**를 중심으로 의존결과를 알 수 있어야함
- ❖ 부분적인 의존도가 없어야함
 - 즉 모든 부가 속성들은 **Primary key**를 가지고 찾을 수 있어야함
 - That is, all non-key attributes are fully dependent on a primary key

목표는 중복을 제거하고 **atomicity**를 갖는 것

◆ 2NF (Second Normal Form) - 예제



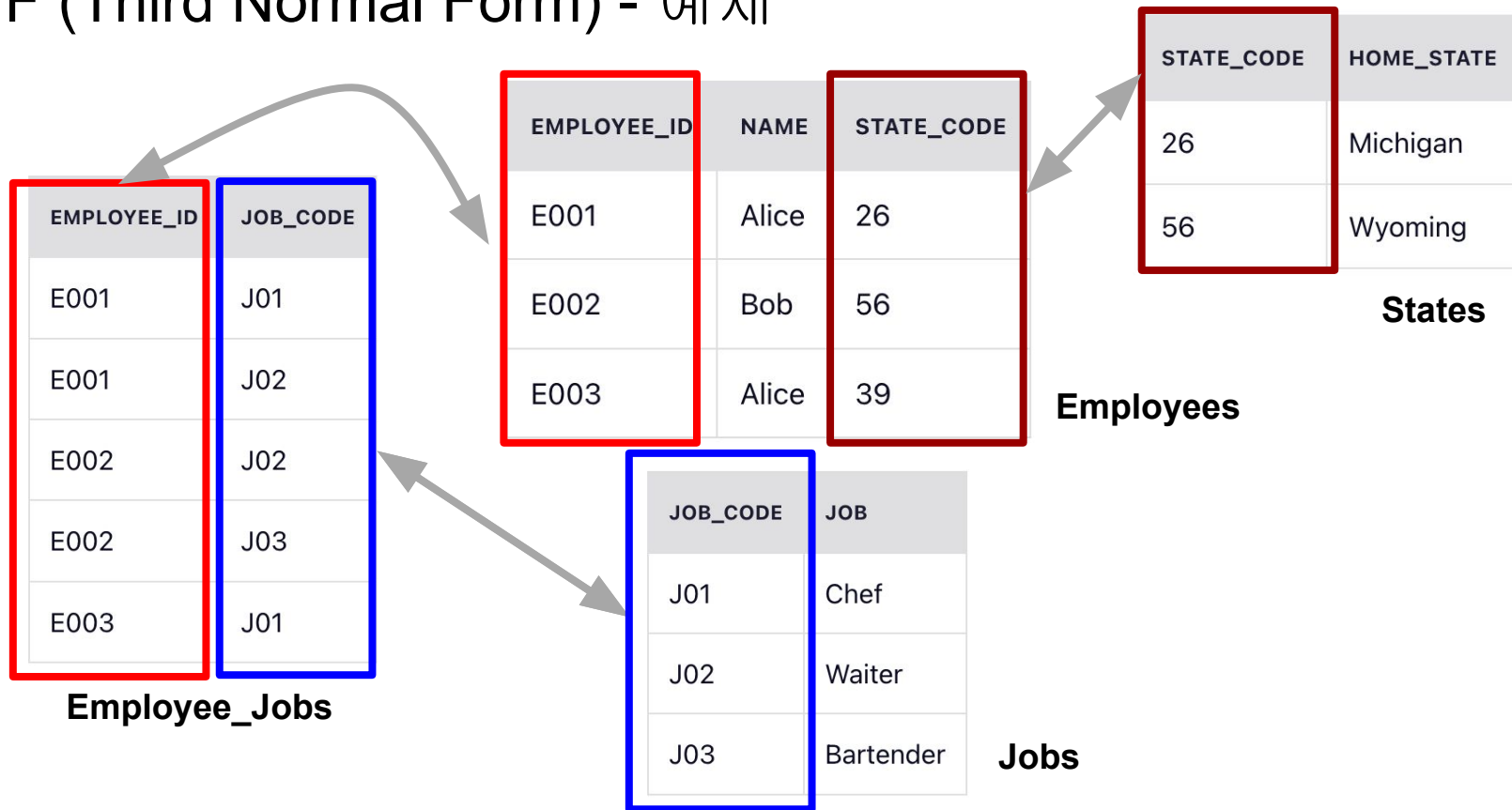
◆ 3NF (Third Normal Form)

- ❖ 일단 2NF를 만족해야함
- ❖ 전이적 부분 종속성을 없어야함
 - 2NF의 예에서 `state_code`과 `home_state`가 같이 `Employees` 테이블에 존재

EMPLOYEE_ID	NAME	STATE_CODE	HOME_STATE
E001	Alice	26	Michigan
E002	Bob	56	Wyoming

Employees

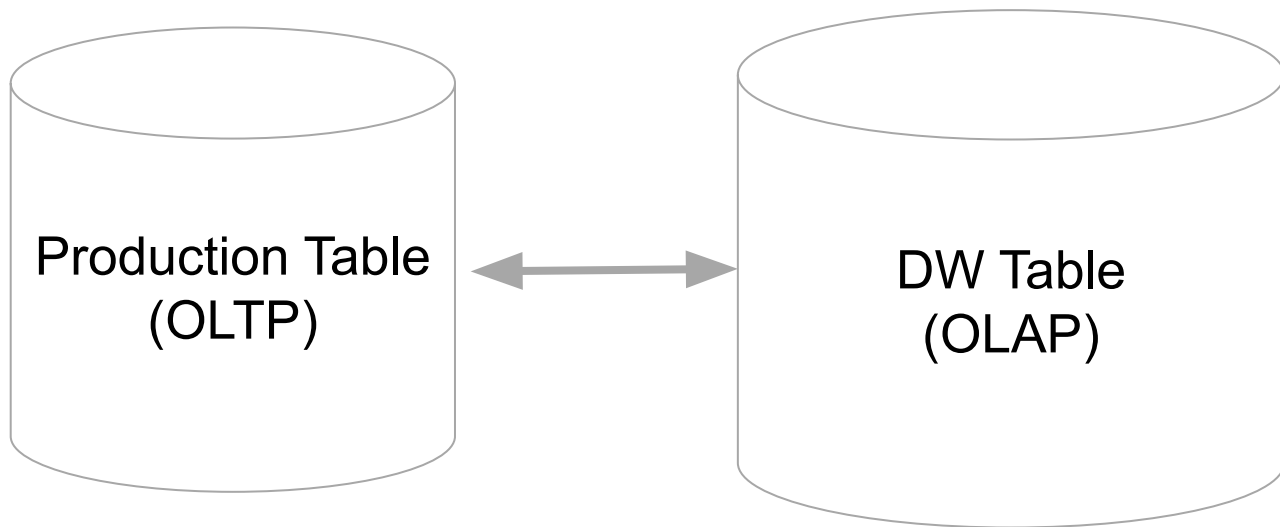
◆ 3NF (Third Normal Form) - 예제



◆ Slowly Changing Dimensions (1)

- ❖ DW나 DL에서는 모든 테이블들의 히스토리를 유지하는 것이 중요함
 - 보통 두 개의 **timestamp** 필드를 갖는 것이 좋음
 - **created_at** (생성시간으로 한번 만들어지면 고정됨)
 - **updated_at** (꼭 필요 마지막 수정 시간을 나타냄)
- ❖ 이 경우 컬럼의 성격에 따라 어떻게 유지할지 방법이 달라짐
 - SCD Type 0
 - SCD Type 1
 - **SCD Type 2**
 - SCD Type 3
 - SCD Type 4

◆ Slowly Changing Dimensions (2)



일부 속성들은 시간을 두고 변하게
되는데 **DW Table** 쪽에 어떻게
반영해야하나?

현재 데이터만
유지

vs.

처음부터 지금까지 히스토리도
유지

◆ SCD Type 0

- ❖ 한번 쓰고 나면 바꿀 이유가 없는 경우들
- ❖ 한번 정해지면 갱신되지 않고 고정되는 필드들
- ❖ 예) 고객 테이블이라면 회원 등록일, 제품 첫 구매일

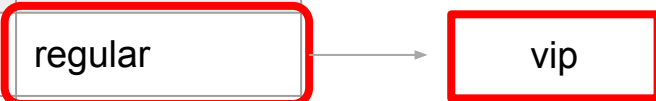
◆ SCD Type 1

- ❖ 데이터가 새로 생기면 덮어쓰면 되는 컬럼들
- ❖ 처음 레코드 생성시에는 존재하지 않았지만 나중에 생기면서 채우는 경우
- ❖ 예) 고객 테이블이라면 연간소득 필드

◆ SCD Type 2

- ❖ 특정 **entity**에 대한 데이터가 새로운 레코드로 추가되어야 하는 경우
- ❖ 예) 고객 테이블에서 고객의 등급 변화
 - tier라는 컬럼의 값이 “regular”에서 “vip”로 변화하는 경우
 - 변경시간도 같이 추가되어야함

customer_id	tier
100	regular
101	regular



◆ SCD Type 3

- ❖ SCD Type 2의 대안으로 특정 **entity** 데이터가 새로운 컬럼으로 추가되는 경우
- ❖ 예) 고객 테이블에서 **tier**라는 컬럼의 값이 “regular”에서 “vip”로 변화하는 경우
 - **previous_tier**라는 컬럼 생성
 - 변경시간도 별도 컬럼으로 존재해야함

◆ SCD Type 4

- ❖ 특정 entity에 대한 데이터를 새로운 Dimension 테이블에 저장하는 경우
- ❖ SCD Type 2의 변종
- ❖ 예) 별도의 테이블로 저장하고 이 경우 아예 일반화할 수도 있음



dbt 소개

Data Build Tool이 무엇인지 알아보자

◆ dbt란 무엇인가?

❖ Data Build Tool (<https://www.getdbt.com/>)

- ELT용 오픈소스: In-warehouse data transformation
- dbt Labs라는 회사가 상용화 (\$4.2B valuation)
- Analytics Engineer라는 말을 만들어냄

❖ 다양한 데이터 웨어하우스를 지원

- Redshift, Snowflake, Bigquery, Spark

❖ 클라우드 버전도 존재




- dbt Cloud




◆ dbt가 서포트해주는 데이터 시스템

- ❖ BigQuery
- ❖ Redshift
- ❖ Snowflake
- ❖ Spark
- ❖ ...

Verified Adapters

Data Platform (click to view setup guide)	latest verified version
AlloyDB	(same as <code>dbt-postgres</code>)
Azure Synapse	1.3.0
BigQuery	1.2.0
Databricks	1.3.0 
Dremio	1.3.0 
Postgres	1.2.0
Redshift	1.2.0
Snowflake	1.2.0
Spark	1.2.0
Starburst & Trino	1.2.0 

 Verification in progress

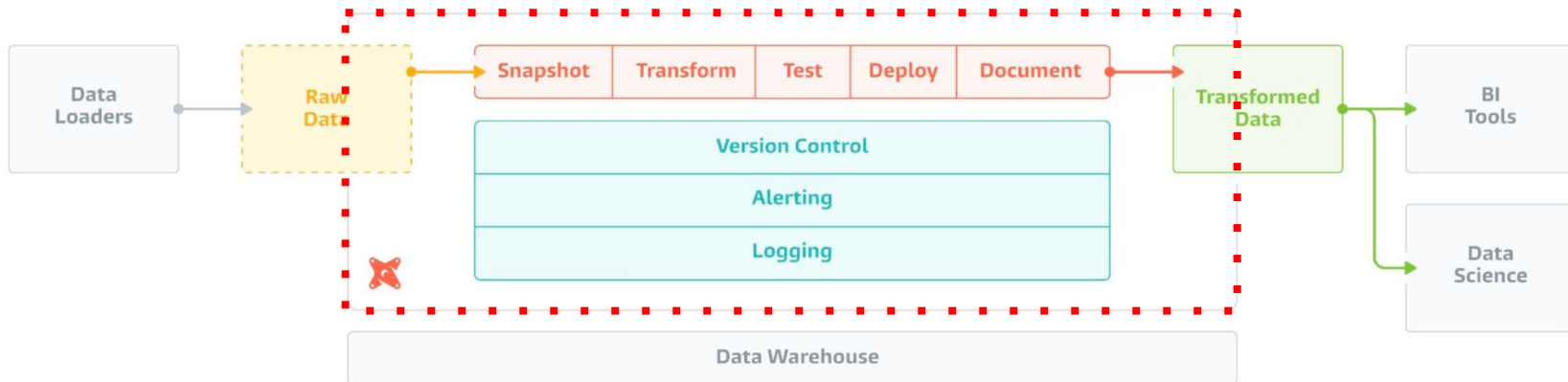
◆ dbt 구성 컴포넌트

❖ 데이터 모델 (models)

- 테이블들을 몇개의 티어로 관리
 - 일종의 CTAS (SELECT 문들), Lineage 트래킹
- Table, View, CTE 등등

❖ 데이터 품질 검증 (tests)

❖ 스냅샷 (snapshots)





dbt 사용 시나리오

dbt가 어떻게 사용될 수 있는지 가상 환경을
생각해보자

- ◆ 다음과 같은 요구조건을 달성해야한다면?
 - ❖ 데이터 변경 사항을 이해하기 쉽고 필요하다면 롤백 가능
 - ❖ 데이터간 리니지 확인 가능
 - ❖ 데이터 품질 테스트 및 에러 보고
 - ❖ **Fact** 테이블의 증분 로드 (Incremental Update)
 - ❖ **Dimension** 테이블 변경 추적 (히스토리 테이블)
 - ❖ 용이한 문서 작성



◆ 보통 사용하는 테크 스택

- ❖ Redshift/Spark/Snowflake/BigQuery
- ❖ dbt
- ❖ Airflow



◆ 무슨 ELT 작업을 해볼까요?

❖ Redshift 사용

❖ AB 테스트 분석을 쉽게 하기 위한 ELT 테이블을 만들어보자

❖ 입력 테이블:

- user_event, user_variant, user_metadata

❖ 생성 테이블: Variant별 사용자별 일별 요약 테이블

- variant_id, user_id, datestamp, age, gender,
- 총 impression, 총 click, 총 purchase, 총 revenue

◆ 입력 데이터들

- ❖ Production DB에 저장되는 정보들을 Data Warehouse로 적재했다고 가정
- ❖ raw_data.user_event
 - 사용자/날짜/아이템별로 impression이 있는 경우 그 정보를 기록하고 impression으로부터 클릭, 구매, 구매시 금액을 기록. 실제 환경에서는 이런 aggregate 정보를 로그 파일등의 소스(하나 이상의 소스가 될 수도 있음)로부터 만들어내는 프로세스가 필요함
- ❖ raw_data.user_variant
 - 사용자가 소속한 AB test variant를 기록한 파일 (control vs. test)
- ❖ raw_data.user_metadata
 - 사용자에 관한 메타 정보가 기록된 파일 (성별, 나이 등등)

◆ 입력 데이터: raw_data.user_event

```
CREATE TABLE raw_data.user_event (
```

```
  user_id int,
```

```
  datestamp timestamp,
```

```
  item_id int,
```

```
  clicked int,
```

```
  purchased int,
```

```
  paidamount int
```

```
);
```



사용자별/날짜별/아이템별

impression/clicked/purchase/paidamount 요약

◆ 입력 데이터: raw_data.user_variant

```
CREATE TABLE raw_data.user_variant (  
  user_id int,  
  variant_id varchar(32) -- control vs. test  
);
```

- 보통은 experiment와 variant 테이블이 별도로 존재함
- 그리고 위의 테이블에도 언제 variant_id로 소속되었는지 타임스탬프 필드가 존재하는 것이 일반적

◆ 입력 데이터: raw_data.user_metadata

```
CREATE TABLE raw_data.user_metadata (  
  user_id int,  
  age varchar(16),  
  gender varchar(16),  
  updated_at timestamp  
);
```

사용자별 메타정보:

이를 이용해 다양한 각도에서 **AB** 테스트
결과를 분석해볼 수 있음

◆ Fact 테이블과 Dimension 테이블

- ❖ **Fact 테이블**: 분석의 초점이 되는 양적 정보를 포함하는 중앙 테이블
 - 일반적으로 매출 수익, 판매량, 이익과 같은 측정 항목 포함. 비즈니스 결정에 사용
 - Fact 테이블은 일반적으로 외래 키를 통해 여러 **Dimension** 테이블과 연결됨
 - 보통 Fact 테이블의 크기가 훨씬 더 큼
- ❖ **Dimension 테이블**: Fact 테이블에 대한 상세 정보를 제공하는 테이블
 - 고객, 제품과 같은 테이블로 Fact 테이블에 대한 상세 정보 제공
 - Fact 테이블의 데이터에 맥락을 제공하여 다양한 방식으로 분석 가능하게 해줌
 - Dimension 테이블은 **primary key**를 가지며, fact 테이블에서 참조 (**foreign key**)
 - 보통 Dimension 테이블의 크기는 훨씬 더 작음

◆ 입력 데이터 요약

❖ user_event, user_variant, user_metadata

Column	Type
user_id	int
timestamp	timestamp
item_id	int
clicked	int
purchased	int
revenue	int

Fact 테이블

Column	Type
user_id	int
variant_id	string

Dimension 테이블

Column	Type
user_id	int
age	int
gender	string
udpated_at	timestamp

Dimension 테이블

◆ 최종 생성 데이터 (ELT 테이블)

❖ SELECT로 표현하면 아래와 같음

```
SELECT
  variant_id,
  ue.user_id,
  datestamp,
  age,
  gender,
  COUNT(DISTINCT item_id) num_of_items, -- 총 impression
  COUNT(DISTINCT CASE WHEN clicked THEN item_id END) num_of_clicks, -- 총 click
  SUM(purchased) num_of_purchases, -- 총 purchase
  SUM(paidamount) revenue           -- 총 revenue
FROM raw_data.user_event ue
JOIN raw_data.user_variant uv ON ue.user_id = uv.user_id
JOIN raw_data.user_metadata um ON uv.user_id = um.user_id
GROUP by 1, 2, 3, 4, 5;
```



dbt 설치와 환경 설정

dbt가 어떻게 사용될 수 있는지 가상 환경을
생각해보자

◆ dbt 사용절차

❖ dbt 설치

- dbt Cloud vs. dbt Core
- git을 보통 사용함

❖ dbt 환경설정

❖ Connector 설정

- Connector가 바로 바탕이 되는 데이터 시스템 (Redshift, Spark, ...)

❖ 데이터 모델링 (tier)

- Raw Data -> Staging -> Core

❖ 테스트 코드 작성

❖ (필요하다면) Snapshot 설정

◆ dbt 설치 옵션

- ❖ Cloud 버전: dbt Cloud
- ❖ 로컬 개발 버전: dbt Core



Get started with dbt Cloud

dbt Cloud is the fastest and most reliable way to deploy dbt. Develop, test, schedule, and investigate data models all in one web-based UI.



Getting started with dbt Core

When you use dbt Core to work with dbt, you will be editing files locally using a code editor, and running projects using a command line interface

◆ dbt 설치: 로컬 버전으로 진행 (1.4.3)

❖ pip3 install dbt-redshift

- 위의 명령은 dbt-core 모듈도 설치해줌
- 환경에 맞는 dbt connector를 설치: Redshift, BigQuery, Snowflake,

```
keyyong grepp % pip3 install dbt-redshift
Collecting dbt-redshift
  Downloading dbt_redshift-1.4.0-py3-none-any.whl (21 kB)
Collecting boto3~=1.26.26
  Downloading boto3-1.26.79-py3-none-any.whl (132 kB)
    _____ 132.7/132.7 kB 1.5 MB/s eta 0:00:00
Collecting dbt-core~=1.4.0
  Downloading dbt_core-1.4.3-py3-none-any.whl (924 kB)
    _____ 924.3/924.3 kB 2.2 MB/s eta 0:00:00
```

◆ dbt 환경 설정 - Connector 연결

❖ Redshift 연결 정보

- `learnde.cduaw970ssvt.ap-northeast-2.redshift.amazonaws.com`
- Schema: dev
- Port: 5439
- Login: 본인의 ID 사용
- Password: 본인의 패스워드 사용

◆ dbt 환경 설정 - dbt init

keeyong dbt % **dbt init learn_dbt**

23:08:41 Running with dbt=1.4.3

Which database would you like to use?

[1] **redshift**

[2] postgres

Enter a number: **1**

host (hostname.region.redshift.amazonaws.com):

learnde.cduaw970ssvt.ap-northeast-2.redshift.amazonaws.com

port [5439]:

user (dev username) **keeyong**

[1] **password**

[2] iam

Desired authentication method option (enter a number) **1**

password (dev password):

dbname (default database that dbt will build objects in): **dev**

schema (default schema that dbt will build objects in) **keeyong**

threads (1 or more) [1] **1**

◆ 잠깐 yml (or yaml) 파일 포맷: 환경설정 파일에 많이 쓰임

Comments start with a

Key-value pairs are separated by a colon and a space

name: John Doe

age: 30

Lists are denoted by a hyphen and a space

hobbies:

- reading

- hiking

Nested key-value pairs are indented with two spaces

contact:

email: john.doe@example.com

phone:

home: 555-1234

work: 555-5678

multi-line string

description: |

This is a

multi-line

string

◆ dbt 환경 설정 - 설치된 파일과 폴더 살펴보기

```
keyyong dbt % ls -tl learn_dbt
total 16
-rw-r--r--  1 [redacted] staff  1330 Feb 24 15:08 dbt_project.yml
drwxr-xr-x  3 [redacted] staff   96 Feb 24 14:47 tests
drwxr-xr-x  3 [redacted] staff   96 Feb 24 14:47 snapshots
drwxr-xr-x  3 [redacted] staff   96 Feb 24 14:47 seeds
drwxr-xr-x  3 [redacted] staff   96 Feb 24 14:47 models
drwxr-xr-x  3 [redacted] staff   96 Feb 24 14:47 macros
drwxr-xr-x  3 [redacted] staff   96 Feb 24 14:47 analyses
-rw-r--r--  1 [redacted] staff   571 Feb 24 14:47 README.md
keyyong dbt % ls -tl ~/.dbt/profiles.yml
-rw-r--r--  1 jobox staff  257 Feb 24 15:09 /Users/[redacted]/.dbt/profiles.yml
```

◆ dbt 환경 설정 - ~/.dbt/profiles.yml

learn_dbt: 프로젝트 이름

outputs:

dev:

dbname: dev

host: learnde.cduaw970ssvt.ap-northeast-2.redshift.amazonaws.com

password: *****

port: 5439

schema: keeyong

threads: 1

type: redshift

user: keeyong

target: **dev**

outputs 밑에 다수의 개발환경을 정의해놓고 필요한 것을 target에 선택

◆ dbt 파일과 폴더 설명

- ❖ dbt_project.yml: 메인 환경 설정 파일
- ❖ models
- ❖ seeds
- ❖ tests
- ❖ snapshots
- ❖ macros
- ❖ analyses
- ❖ README.md

◆ dbt_project.yml

```
name: 'learn_dbt'  
version: '1.0.0'  
config-version: 2
```

~/dbt/profiles.yml
안에 존재해야함

```
profile: 'learn_dbt'
```

```
model-paths: ["models"]  
analysis-paths: ["analyses"]  
test-paths: ["tests"]  
seed-paths: ["seeds"]  
macro-paths: ["macros"]  
snapshot-paths: ["snapshots"]
```

폴더 이름들과 일치해야함

결과들이 저장되는
폴더

```
target-path: "target" # folder to store compiled SQL files  
clean-targets: # directories to be removed by `dbt clean`  
  - "target"  
  - "dbt_packages"
```

models:

```
learn_dbt:
```

```
  example:  
    +materialized: view
```

이 두 라인은 삭제
그리고 models
폴더에 있는 example
서브폴더 삭제

```
keyyong learn_dbt % cd models  
keyyong models % ls -tl  
total 0  
drwxr-xr-x  5 jobox  staff  160 Feb 24 14:47 example  
keyyong models % rm -rf example/
```

dbt Models: Input

dbt Model을 사용해 입력 데이터들을
transform해보자

◆ Model이란?

- ❖ ELT 테이블을 만듬에 있어 기본이 되는 빌딩블록
 - 테이블이나 뷰나 CTE의 형태로 존재
- ❖ 입력, 중간, 최종 테이블을 정의하는 곳
 - 티어 (raw, staging, core, ...)
 - raw => staging (src) => core

```
1330 Feb 24 15:08 dbt_project.yml
96 Feb 24 14:47 tests
96 Feb 24 14:47 snapshots
96 Feb 24 14:47 seeds
96 Feb 24 14:47 models
96 Feb 24 14:47 macros
96 Feb 24 14:47 analyses
571 Feb 24 14:47 README.md
```


◆ 잠깐: View란 무엇인가?

❖ SELECT 결과를 기반으로 만들어진 가상 테이블

- 기존 테이블의 일부 혹은 여러 테이블들을 조인한 결과를 제공함
- CREATE VIEW 이름 AS SELECT ...

❖ View의 장점

- 데이터의 추상화: 사용자는 **View**를 통해 필요 데이터에 직접 접근. 원본 데이터를 알 필요가 없음
- 데이터 보안: **View**를 통해 사용자에게 필요한 데이터만 제공. 원본 데이터 접근 불필요
- 복잡한 쿼리의 간소화: SQL(**View**)를 사용하면 복잡한 쿼리를 단순화.

❖ View의 단점

- 매번 쿼리가 실행되므로 시간이 걸릴 수 있음
- 원본 데이터의 변경을 모르면 실행이 실패함

◆ 잠깐: CTE (Common Table Expression)

```
WITH temp1 AS (  
    SELECT k1, k2  
    FROM t1  
    JOIN t2 ON t1.id = t2.foreign_id  
) , temp2 AS (  
    ...  
)  
SELECT *  
FROM temp1 t1  
JOIN temp2 t2 ON ...
```

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

◆ Model 구성 요소

❖ Input

- 입력(raw)과 중간(staging, src) 데이터 정의
- raw는 CTE로 정의
- staging은 View로 정의

❖ Output

- 최종(core) 데이터 정의
- core는 Table로 정의

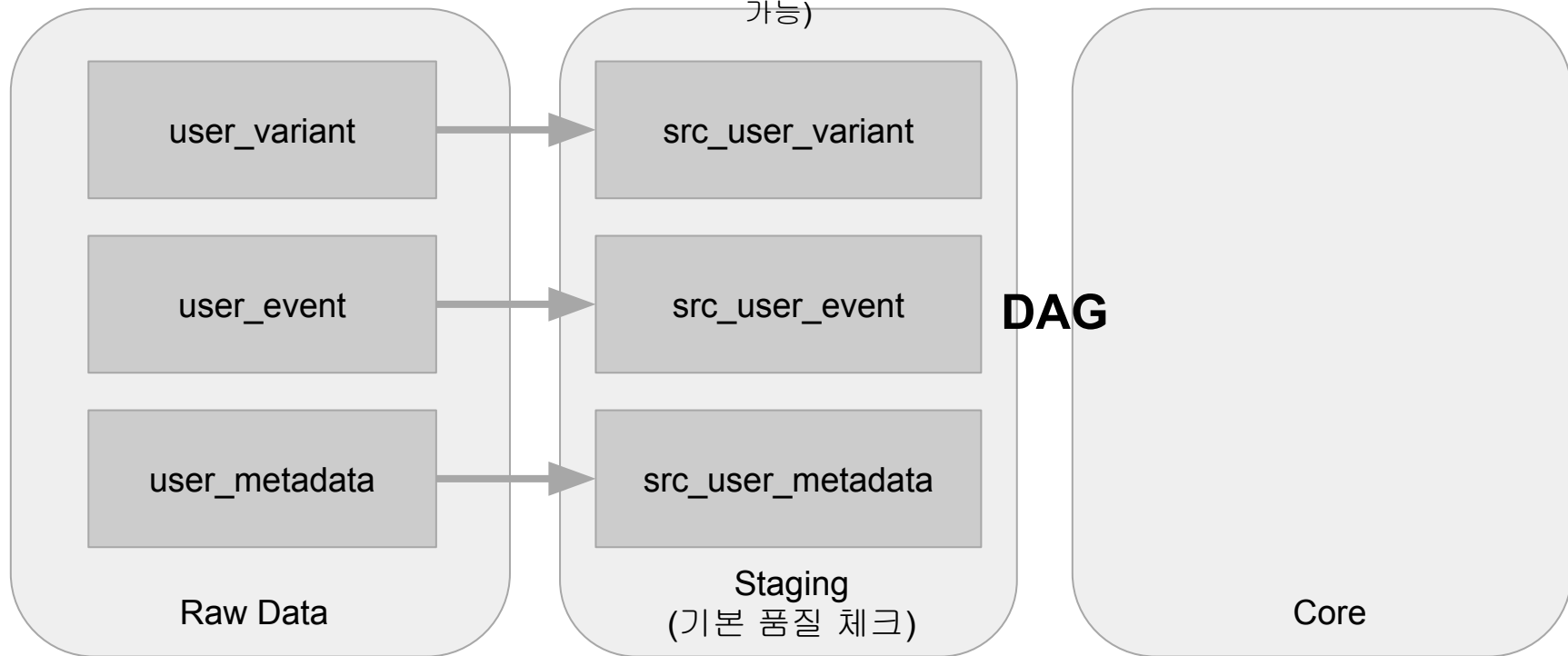
❖ 이 모두는 models 폴더 밑에 sql 파일로 존재

- 기본적으로는 **SELECT + Jinja** 템플릿과 매크로
- 다른 테이블들을 사용 가능 (**reference**)
 - 이를 통해 리니지 파악

◆ 데이터 빌딩 프로세스

models/src/ 폴더 밑에 .sql 파일로 저장
(Table, View, CTE, Append 타입으로 존재
가능)

Models: Output에서 설명



◆ models/src - src_user_event.sql

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

```
keyyong models % mkdir src  
keyyong models % cd src  
keyyong src % vi src_user_event.sql  
keyyong src % vi src_user_metadata.sql  
keyyong src % vi src_user_variant.sql
```

◆ models/src - src_user_variant.sql

```
WITH src_user_variant AS (  
    SELECT * FROM raw_data.user_variant  
)  
SELECT  
    user_id,  
    variant_id  
FROM  
    src_user_variant
```

◆ models/src - src_user_metadata.sql

```
WITH src_user_metadata AS (  
    SELECT * FROM raw_data.user_metadata  
)  
SELECT  
    user_id,  
    age,  
    gender,  
    updated_at  
FROM  
    src_user_metadata
```

◆ Model 빌딩: dbt run

```
keeyong learn_dbt % dbt run
```

```
09:56:39 Running with dbt=1.4.3
```

```
09:56:39 Unable to do partial parsing because profile has changed
```

```
09:56:39 [WARNING]: Configuration paths exist in your dbt_project.yml file which do not apply to any resources.
```

```
There are 1 unused configuration paths:
```

```
- models.learn_dbt.example
```

```
09:56:39 Found 3 models, 0 tests, 0 snapshots, 0 analyses, 327 macros, 0 operations, 0 seed files, 0 sources, 0 exposures, 0 metrics
```

```
09:56:39
```

```
09:56:45 Concurrency: 1 threads (target='dev')
```

```
09:56:45
```

```
09:56:45 1 of 3 START sql view model keeyong.src_user_event ..... [RUN]
```

```
09:56:47 1 of 3 OK created sql view model keeyong.src_user_event ..... [CREATE VIEW in 2.65s]
```

```
09:56:47 2 of 3 START sql view model keeyong.src_user_metadata ..... [RUN]
```

```
09:56:50 2 of 3 OK created sql view model keeyong.src_user_metadata ..... [CREATE VIEW in 2.69s]
```

```
09:56:50 3 of 3 START sql view model keeyong.src_user_variant ..... [RUN]
```

```
09:56:53 3 of 3 OK created sql view model keeyong.src_user_variant ..... [CREATE VIEW in 2.57s]
```

```
09:56:54
```

```
09:56:54 Finished running 3 view models in 0 hours 0 minutes and 14.78 seconds (14.78s).
```

```
09:56:54
```

```
09:56:54 Completed successfully
```

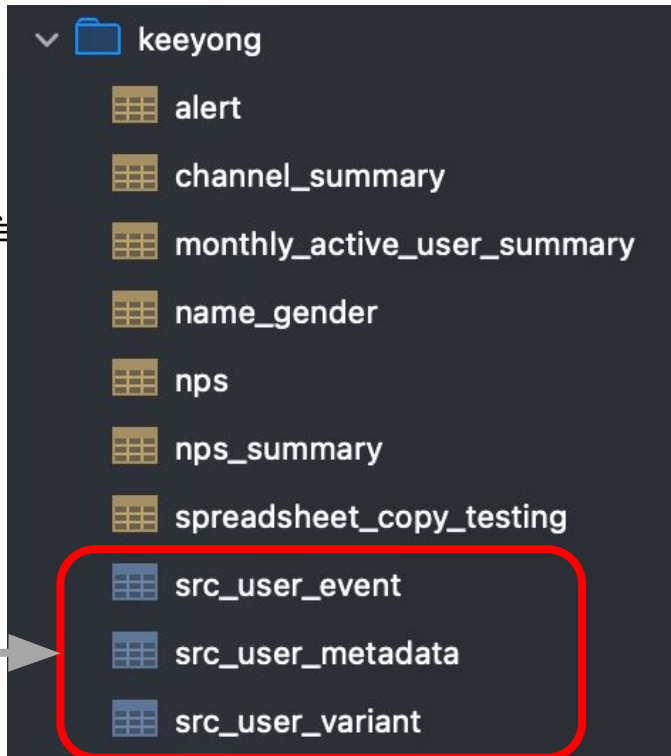
```
09:56:54
```

```
09:56:54 Done. PASS=3 WARN=0 ERROR=0 SKIP=0 TOTAL=3
```


◆ Model 빌딩 확인

- ❖ 해당 스키마 밑에 테이블 생성 여부 확인
- ❖ dbt run은 프로젝트 구성 다양한 SQL 실행
 - 이 SQL들은 DAG로 구성됨
- ❖ dbt run은 보통 다른 더 큰 명령의 일부로 실행
 - dbt test
 - dbt docs generate

View로
만들어짐



dbt Models: Output

최종 출력 데이터를 만드는 과정을 살펴보자

◆ Materialization이란?

- ❖ 입력 데이터(테이블)들을 연결해서 새로운 데이터(테이블) 생성하는 것
 - 보통 여기서 추가 **transformation**이나 데이터 클린업 수행
- ❖ 4가지의 내장 **materialization**이 제공됨
- ❖ 파일이나 프로젝트 레벨에서 가능
- ❖ 역시 **dbt run**을 기타 파라미터를 가지고 실행

◆ 4가지의 Materialization 종류

❖ View

- 데이터를 자주 사용하지 않는 경우

❖ Table

- 데이터를 반복해서 자주 사용하는 경우

❖ Incremental (Table Appends)

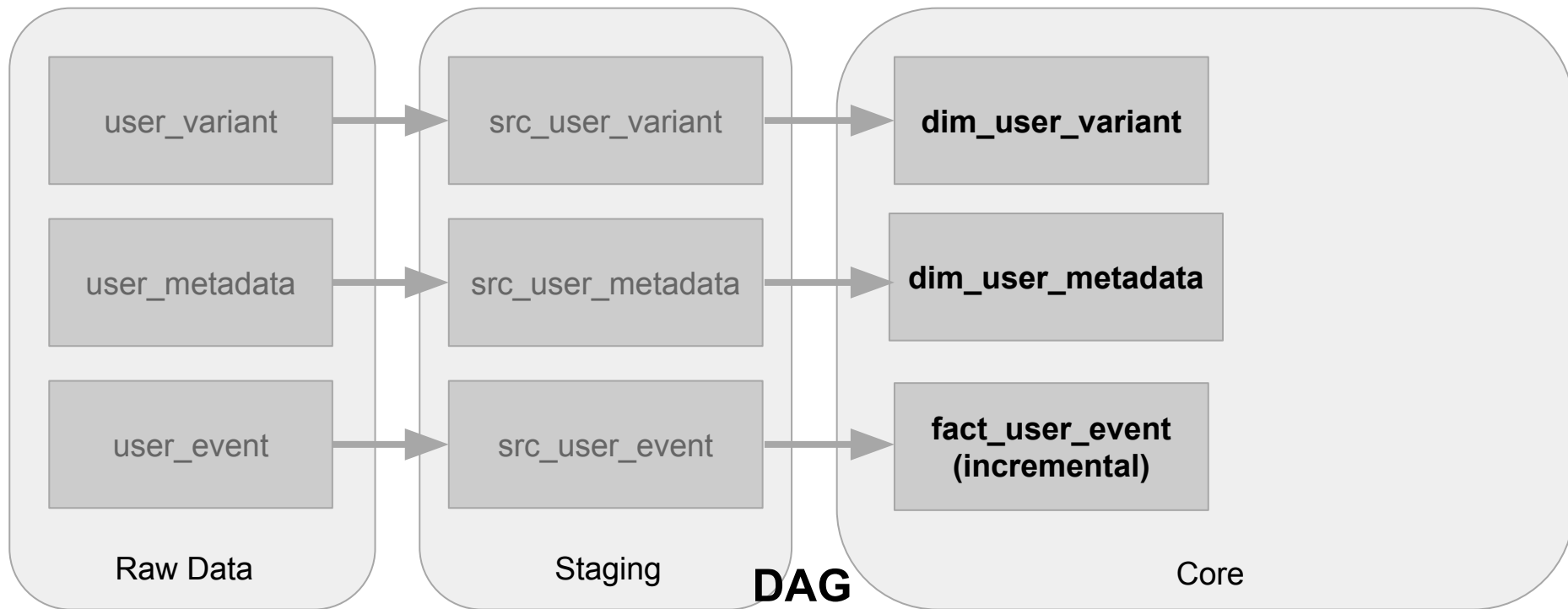
- Fact 테이블
- 과거 레코드를 수정할 필요가 없는 경우

❖ Ephemeral (CTE)

- 한 SELECT에서 자주 사용되는 데이터를 모듈화하는데 사용

◆ 데이터 빌딩 프로세스

models/**dim**/ 폴더 밑에 .sql 파일로 저장



◆ 잠깐 Jinja 템플릿이란?

- ❖ 파이썬이 제공해주는 템플릿 엔진으로 Flask에서 많이 사용
 - Airflow에서도 사용함
- ❖ 입력 파라미터 기준으로 HTML 페이지(마크업)를 동적으로 생성
- ❖ 조건문, 루프, 필터등을 제공

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title> Jinja Template </title>
</head>
<body>
  {% for i in range(100) %}
    <div> hello </div>
  {% endfor %}
</body>
</html>
```

◆ models 밑에 core 테이블들을 위한 폴더 생성

❖ dim 폴더와 fact 폴더 생성

- dim 밑에 각각 dim_user_variant.sql과 dim_user_metadata.sql 생성
- fact 밑에 fact_user_event.sql 생성

❖ 이 모두를 physical table로 생성

```
keeyong models % mkdir dim
keeyong models % cd dim
keeyong dim % vi dim_user_variant.sql
keeyong dim % vi fact_user_event.sql
keeyong dim % vi dim_user_metadata.sql
keeyong models % mkdir fact
keeyong models % cd fact
keeyong fact % vi fact_user_event.sql
```

◆ models/dim - dim_user_variant.sql

❖ Jinja 템플릿과 ref 태그를 사용해서 dbt 내 다른 테이블들을 액세스

```
WITH src_user_variant AS (  
  SELECT * FROM {{ ref('src_user_variant') }}  
)  
SELECT  
  user_id,  
  variant_id  
FROM  
  src_user_variant
```



필요하면 여기서
더 transformation
수행

◆ models/dim - dim_user_metadata.sql

- ❖ 설정에 따라 view/table/CTE 등으로 만들어져서 사용됨
 - materialized라는 키워드로 설정

```
WITH src_user_metadata AS (  
  SELECT * FROM {{ ref('src_user_metadata') }}  
)  
SELECT  
  user_id,  
  age,  
  gender,  
  updated_at  
FROM  
  src_user_metadata
```

◆ models/fact - fact_user_event.sql

❖ Incremental Table로 빌드 (materialized = 'incremental')

```
{{
  config(
    materialized = 'incremental',
    on_schema_change='fail'
  )
}}
WITH src_user_event AS (
  SELECT * FROM {{ ref("src_user_event") }}
)
SELECT
  user_id,
  datestamp,
  item_id,
  clicked,
  purchased,
  paidamount
FROM
  src_user_event
```

incremental_strategy도 사용가능

- append
- merge
- insert_overwrite

이 경우 unique_key와 merge_update_columns 필드를 사용하기도 함

스키마가 바뀐 경우 대응 방법 지정

- append_new_columns
- ignore
- sync_all_columns
- fail

◆ 다음으로 model의 materialized format 결정

- ❖ 최종 Core 테이블들은 view가 아닌 table로 빌드
- ❖ dbt_project.yml을 편집

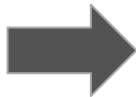
models:

learn_dbt:

example:

+materialized: view

↑
example 폴더 밑에 있는
테이블들은 모두 view로
빌드



models:

learn_dbt:

+materialized: view

dim:

+materialized: table

↑
dim 폴더 밑에 있는
테이블들은 모두 table로
빌드

이 프로젝트의
테이블들은 기본적으로
view로 빌드

◆ Model 빌딩: dbt run (dbt compile도 있음)

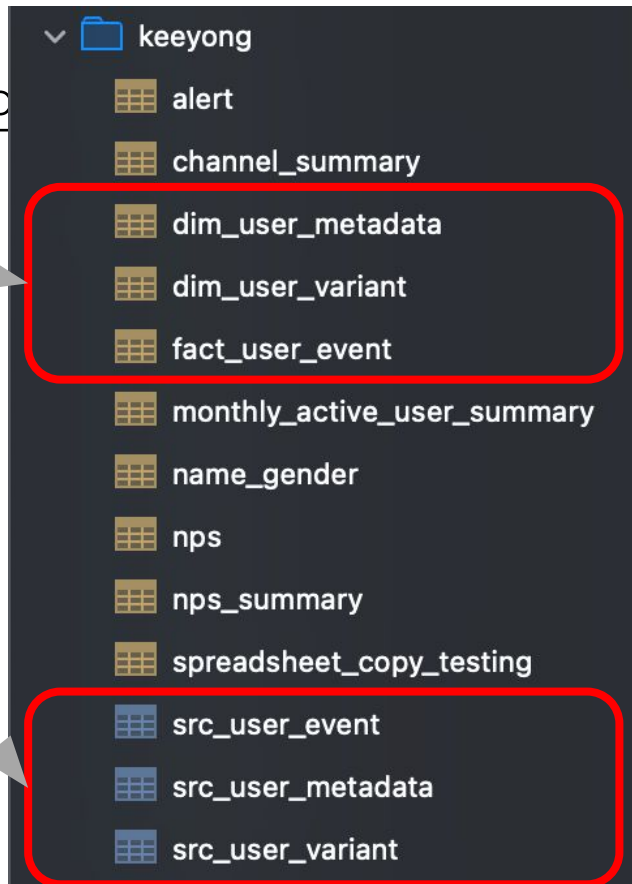
```
keyyong learn_dbt % dbt run
19:19:17 Running with dbt=1.5.1
19:19:17 Found 6 models, 0 tests, 0 snapshots, 0 analyses, 346 macros, 0 operations, 0 seed files, 0 sources, 0 exposures
, 0 metrics, 0 groups
19:19:17
19:19:23 Concurrency: 1 threads (target='dev')
19:19:23
19:19:23 1 of 6 START sql view model keyyong.src_user_event ..... [RUN]
19:19:28 1 of 6 OK created sql view model keyyong.src_user_event ..... [SUCCESS in 4.85s]
19:19:28 2 of 6 START sql view model keyyong.src_user_metadata ..... [RUN]
19:19:32 2 of 6 OK created sql view model keyyong.src_user_metadata ..... [SUCCESS in 4.36s]
19:19:32 3 of 6 START sql view model keyyong.src_user_variant ..... [RUN]
19:19:37 3 of 6 OK created sql view model keyyong.src_user_variant ..... [SUCCESS in 4.39s]
19:19:37 4 of 6 START sql incremental model keyyong.fact_user_event ..... [RUN]
19:19:45 4 of 6 OK created sql incremental model keyyong.fact_user_event ..... [SUCCESS in 8.37s]
19:19:45 5 of 6 START sql table model keyyong.dim_user_metadata ..... [RUN]
19:19:50 5 of 6 OK created sql table model keyyong.dim_user_metadata ..... [SUCCESS in 4.64s]
19:19:50 6 of 6 START sql table model keyyong.dim_user_variant ..... [RUN]
19:19:54 6 of 6 OK created sql table model keyyong.dim_user_variant ..... [SUCCESS in 4.45s]
19:19:56
19:19:56 Finished running 3 view models, 1 incremental model, 2 table models in 0 hours 0 minutes and 38.55 seconds (38.5
5s).
19:19:56
19:19:56 Completed successfully
19:19:56
19:19:56 Done. PASS=6 WARN=0 ERROR=0 SKIP=0 TOTAL=6
```

◆ dbt compile vs. dbt run

- ❖ dbt compile은 SQL 코드까지만 생성하고 실행하지는 않음
- ❖ dbt run은 생성된 코드를 실제 실행함

◆ Model 빌딩 확인

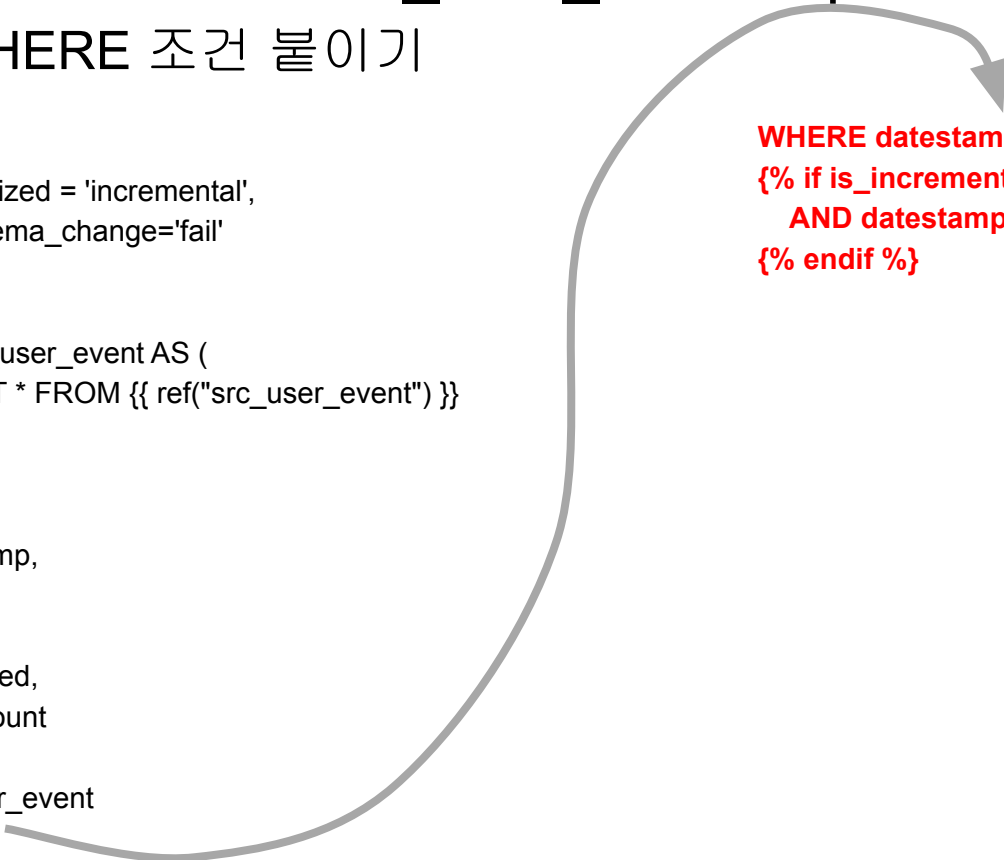
- ❖ 해당 스키마 밑에 테이블 생성 여부 확인
- ❖ Core 테이블들은 Table
- ❖ Staging 테이블들은 View



◆ models/fact - fact_user_event.sql

❖ WHERE 조건 붙이기

```
{{
  config(
    materialized = 'incremental',
    on_schema_change='fail'
  )
}}
WITH src_user_event AS (
  SELECT * FROM {{ ref("src_user_event") }}
)
SELECT
  user_id,
  timestamp,
  item_id,
  clicked,
  purchased,
  paidamount
FROM
  src_user_event
```



WHERE timestamp is not NULL
{% if is_incremental() %}
 AND timestamp > (SELECT max(timestamp) from {{ this }})
{% endif %}

◆ raw_data.user_event에 새 레코드 추가후 dbt run 수행

❖ 적당한 Redshift 클라이언트 툴에서 아래 수행

```
INSERT INTO raw_data.user_event VALUES (100, '2023-06-10', 100, 1, 0, 0);
```

❖ 다음으로 dbt run을 수행

❖ compiled SQL을 확인해서 정말 Incremental하게 업데이트되었는지 확인

❖ 최종적으로 Redshift 클라이언트 툴에서 다시 확인

```
SELECT * FROM keeyong.fact_user_event WHERE datestamp = '2023-06-10';
```


◆ Model 빌딩: Compile 결과 확인

❖ learn_dbt/target/compiled/learn_dbt/models/fact

- fact_user_event.sql의 내용은 아래와 같음

```
WITH src_user_event AS (  
    SELECT * FROM "dev"."keeyong"."src_user_event"  
)  
SELECT  
    user_id,  
    datestamp,  
    item_id,  
    clicked,  
    purchased,  
    paidamount  
FROM  
    src_user_event  
WHERE datestamp is not NULL  
    AND datestamp > (SELECT max(datestamp) from "dev"."keeyong"."fact_user_event")
```

◆ src 테이블들을 CTE로 변환해보기

❖ src 테이블들을 굳이 빌드할 필요가 있나?

❖ dbt_project.yml 편집

models:

learn_dbt:

Config indicated by + and applies to all files under models

+materialized: view

dim:

+materialized: table

src:

+materialized: ephemeral

❖ src 테이블들 (View) 삭제



```
DROP VIEW keeyong.src_user_event;  
DROP VIEW keeyong.src_user_metadata;  
DROP VIEW keeyong.src_user_variant
```

❖ dbt run 실행

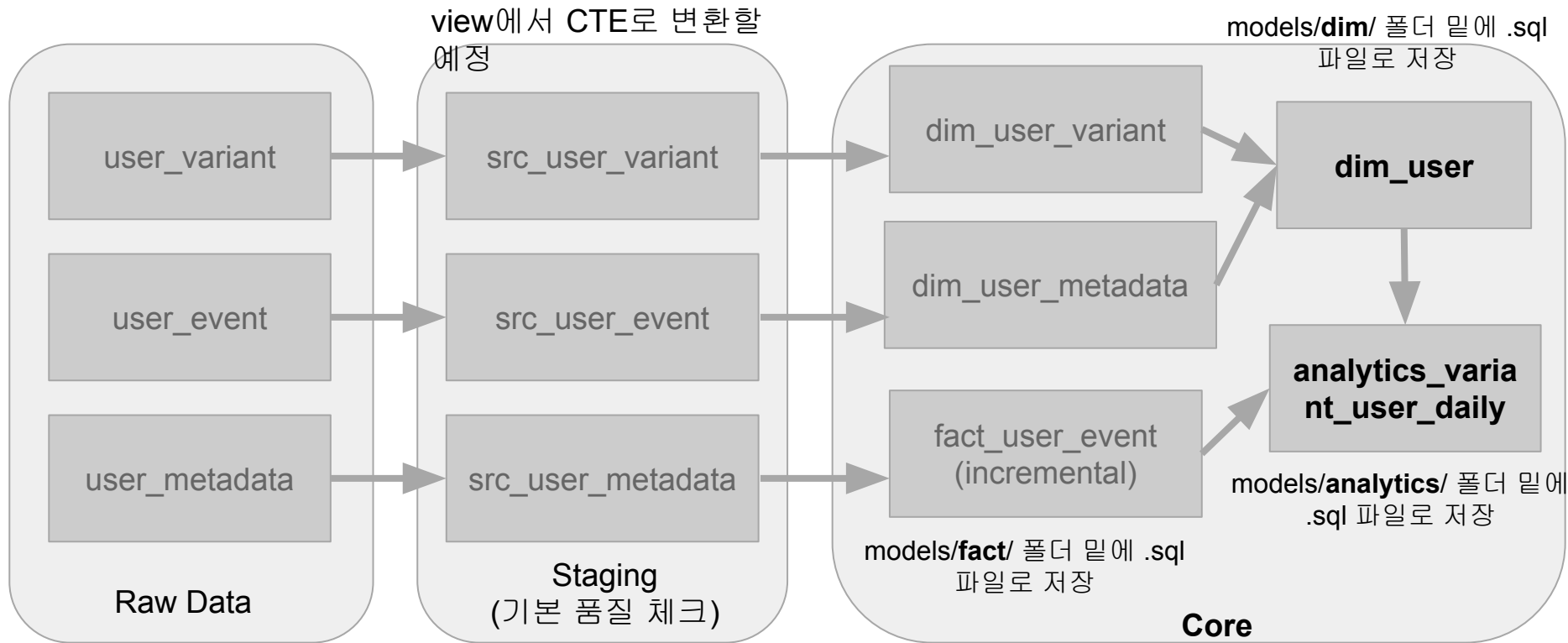
- 이제 SRC 테이블들은 CTE 형태로 임베드되어서 빌드됨

◆ Model 빌딩: dbt run (dbt compile도 있음)

```
keeyong learn_dbt % dbt run
20:44:59 Running with dbt=1.5.1
20:45:00 Unable to do partial parsing because a project config has changed
20:45:00 Found 6 models, 0 tests, 0 snapshots, 0 analyses, 346 macros, 0 operations, 0 seed files, 0 sources, 0 exposures, 0 metrics, 0 groups
20:45:00
20:45:06 Concurrency: 1 threads (target='dev')
20:45:06
20:45:06 1 of 3 START sql incremental model keeyong.fact_user_event ..... [RUN]
20:45:11 1 of 3 OK created sql incremental model keeyong.fact_user_event ..... [SUCCESS in 4.53s]
20:45:11 2 of 3 START sql table model keeyong.dim_user_metadata ..... [RUN]
20:45:15 2 of 3 OK created sql table model keeyong.dim_user_metadata ..... [SUCCESS in 4.53s]
20:45:15 3 of 3 START sql table model keeyong.dim_user_variant ..... [RUN]
20:45:20 3 of 3 OK created sql table model keeyong.dim_user_variant ..... [SUCCESS in 4.52s]
20:45:21
20:45:21 Finished running 1 incremental model, 2 table models in 0 hours 0 minutes and 21.12 seconds (21.12s).
20:45:21
20:45:21 Completed successfully
20:45:21
20:45:21 Done. PASS=3 WARN=0 ERROR=0 SKIP=0 TOTAL=3
```

src 테이블 빌드와 관련된 부분들이 빠져있음

◆ 데이터 빌딩 프로세스



◆ models/dim - dim_user.sql

❖ dim_user_variant와 dim_user_metadata를 조인

```
WITH um AS (  
    SELECT * FROM {{ ref("dim_user_metadata") }}  
) , uv AS (  
    SELECT * FROM {{ ref("dim_user_variant") }}  
)  
SELECT  
    uv.user_id,  
    uv.variant_id,  
    um.age,  
    um.gender  
FROM uv  
LEFT JOIN um ON uv.user_id = um.user_id
```

◆ models/analytics - analytics_variant_user_daily.sql

❖ dim_user와 fact_user_event를 조인 - analytics 폴더를 models 밑에

```
WITH
  u AS (
    SELECT * FROM {{ ref("dim_user") }}
  ), ue AS (
    SELECT * FROM {{ ref("fact_user_event") }}
  )
SELECT
  variant_id,
  ue.user_id,
  timestamp,
  age,
  gender,
  COUNT(DISTINCT item_id) num_of_items, -- 총 impression
  COUNT(DISTINCT CASE WHEN clicked THEN item_id END) num_of_clicks, -- 총 click
  SUM(purchased) num_of_purchases, -- 총 purchase
  SUM(paidamount) revenue -- 총 revenue
FROM ue LEFT JOIN u ON ue.user_id = u.user_id
GROUP BY 1, 2, 3, 4, 5
```

◆ Model 빌딩: dbt run

```
keyyong models % dbt run
22:36:39 Running with dbt=1.4.3
22:36:39 Found 8 models, 0 tests, 0 snapshots, 0 analyses, 327 macros, 0 operations, 0 seed files, 0 sources, 0 exposures
, 0 metrics
22:36:39
22:36:45 Concurrency: 1 threads (target='dev')
22:36:45
22:36:45 1 of 5 START sql incremental model keyyong.fact_user_event ..... [RUN]
22:36:48 1 of 5 OK created sql incremental model keyyong.fact_user_event ..... [INSERT 0 0 in 3.37s]
22:36:48 2 of 5 START sql table model keyyong.dim_user_metadata ..... [RUN]
22:36:51 2 of 5 OK created sql table model keyyong.dim_user_metadata ..... [SELECT in 3.12s]
22:36:51 3 of 5 START sql table model keyyong.dim_user_variant ..... [RUN]
22:36:54 3 of 5 OK created sql table model keyyong.dim_user_variant ..... [SELECT in 2.97s]
22:36:54 4 of 5 START sql table model keyyong.dim_user ..... [RUN]
22:36:57 4 of 5 OK created sql table model keyyong.dim_user ..... [SELECT in 3.06s]
22:36:57 5 of 5 START sql view model keyyong.analytics_variant_user_daily ..... [RUN]
22:37:00 5 of 5 OK created sql view model keyyong.analytics_variant_user_daily ..... [CREATE VIEW in 2.64s]
22:37:02
22:37:02 Finished running 1 incremental model, 3 table models, 1 view model in 0 hours 0 minutes and 22.38 seconds (22.38
s).
22:37:02 Completed successfully
22:37:02
22:37:02 Done. PASS=5 WARN=0 ERROR=0 SKIP=0 TOTAL=5
```

최종 테이블까지 생성

◆ 데모: dbt Models: Input & Output

❖ 앞서 내용들을 전체적으로 직접 실행해보자



숙제

4일차 숙제를 알아보자

◆ 오늘의 숙제

- ❖ 오늘 배운 내용을 다 따라해보고 최종 dbt run 실행 화면 보내기
 - <https://github.com/learndataeng/learn-dbt> 참고
- ❖ 최종 analytics 테이블의 타입을 View에서 Table로 바꾸기