



# BigQuery의 모든 것

(기획자, 마케터, 신입 데이터 분석가를 위한)  
- 입문편 -

[ 쏘카 데이터 그룹, 타다 데이터팀 변성윤 ]

# 목차

---

- #1. BigQuery
  - #2. SQL 이해하기
  - #3. 실습 (1)
  - #4. 데이터 시각화
  - #5. 실습 (2)
  - #6. BigQuery 심화 지식
  - #7. 실습 (3)
  - #8. 스케줄 쿼리
  - #9. Table 생성하기
  - #10. 쿼리 결과 다운로드하기
  - #11. DML Query
  - #12. Firebase Log
  - #13. Data Pipeline
  - #14. Python에서 BigQuery 사용하기
  - #15. 데이터셋 공유하기
  - #16. 정리
- (총 271쪽...)

# 제 간단한 소개

---

## 변성윤

- 인하대 경영학과
- 마케팅/광고 => 공기업 => 창업 => 데이터 순서로 커리어 전환 중
- 현 직장 : **쏘카** 데이터 그룹 타다 데이터팀  
타다 관련 업무도 하고, 사내 데이터 문화 조성, 팀원 성장 등
- 전 직장 : 레트리카(~~데이터 분석가로 입사했는데 데이터 엔지니어도 시킴~~)  
2017년부터 BigQuery 사용
- 개발 블로그 : <https://zzsza.github.io/>
- 유튜브 : DeepNol(<https://www.youtube.com/channel/UCdLZ0MsYS4hmqFgOYCB6C9w>)

# BigQuery

# DB? SQL? MYSQL? BigQuery?

---

DB : 데이터를 보관하는 장소  
실 서비스 데이터 저장소에서 무언가를 많이 할 경우,  
실 서비스에 문제가 갈 수 있기 때문에 다양한 전략을 사용

(복제본을 만들어서 거기서 쿼리를 날린다거나..)

허나 사람이 많아질수록 관리해야 되는 부분이 많아지기 때문에  
아예 “분석용 데이터 저장소”를 만들고 거기에 데이터를 저장하는 경우도 존재

# DB? SQL? MYSQL? BigQuery?

---

데이터 저장소 : Database라고 부르고  
MYSQL, MSSQL, BigQuery 등 다양한 종류가 있음

그리고 BigQuery는 데이터 분석할 때 많은 장점이 있는 데이터 저장소

SQL은 RDBMS에서 데이터를 추출할 때 사용하는 문법

(TMI : NoSQL도 있는데 일단... 이해를 위해 넘어감니다..)

# BigQuery란?

---

**Google Cloud Platform의 Data Warehouse  
(데이터 창고)**

"SQL을 통해 데이터를 추출할 수 있는 공간"

# BigQuery의 장점

---

## 1. 난이도

SQL 기반으로 데이터를 추출 가능  
(=코딩이 아닌 그나마 간단한 SQL로 데이터 추출 가능)

## 2. 속도

다른 데이터베이스는 Index 또는 서버의 성능에 따라 속도가 느리지만 BigQuery는 Index 개념이 없으므로 신경쓰지 않아도 됨

## 3. Firebase를 사용할 경우 앱 데이터를 쉽게 획득 가능

사용 기기, 위치(시 단위까지 표현), OS 버전, 이벤트 행동 획득 가능  
(안드로이드의 경우 app\_remove까지 확인 가능)

## 4. 서버를 따로 구축할 필요 없음. 관리가 필요 없음

(AWS Redshift는 인스턴스를 띄우고 관리해야..)

# BigQuery란?

---

BigQuery의 2가지 문법 체계

#LegacySQL

- 과거에 주로 사용하던 문법

#StandardSQL

- 최근에 주로 사용하는 문법

**정리 : StandardSQL을 사용해주세요!**

Tip) BigQuery 검색시 <BigQuery StandardSQL + 검색어> 이런식으로!

# BigQuery Beta

---

BigQuery에 Beta 기능이 매우 많음

Beta는 바뀔 가능성도 있고 문서가 이상하게 작성되어 있는 경우도 있음

항상 신경을..!

# BigQuery의 비용

Region마다 금액이 다름(예전엔 안그랬는데..)

## 가격 책정 요약

다음은 BigQuery 가격을 요약한 표입니다. 이 작업에는 BigQuery의 [할당량 및 한도](#)가 적용됩니다.

작업	가격	세부정보
활성 저장소	\$0.020 per GB	매월 10GB까지는 무료입니다. 자세한 내용은 <a href="#">저장소 가격 책정</a> 을 참조하세요.
장기 저장소	\$0.010 per GB	매월 10GB까지는 무료입니다. 자세한 내용은 <a href="#">저장소 가격 책정</a> 을 참조하세요.
Storage API	\$1.10 per TB	BigQuery Storage API는 <a href="#">무료 등급</a> 에 포함되지 않습니다.
스트리밍 삽입	\$0.010 per 200 MB	삽입에 성공한 행의 요금만 청구됩니다. 최소 크기 1KB를 적용하여 개별 행을 계정 <a href="#">가격 책정</a> 을 참조하세요.
쿼리(분석)	\$5.00 per TB	매월 1TB까지는 무료입니다. 자세한 내용은 <a href="#">주문형 가격 책정</a> 을 참조하세요. 대량 사용 고객은 <a href="#">정액제</a> 를 이용할 수도 있습니다.

## 가격 책정 요약

다음은 BigQuery 가격을 요약한 표입니다. 이 작업에는 BigQuery의 [할당량 및 한도](#)가 적용됩니다.

작업	가격	세부정보
활성 저장소	\$0.023 per GB	매월 10GB까지는 무료입니다. 자세한 내용은 <a href="#">저장소 가격 책정</a> 을 참조하세요.
장기 저장소	\$0.016 per GB	매월 10GB까지는 무료입니다. 자세한 내용은 <a href="#">저장소 가격 책정</a> 을 참조하세요.
Storage API	Unavailable	BigQuery Storage API는 <a href="#">무료 등급</a> 에 포함되지 않습니다.
스트리밍 삽입	\$0.012 per 200 MB	삽입에 성공한 행의 요금만 청구됩니다. 최소 크기 1KB를 적용하여 개별 행을 계정 <a href="#">가격 책정</a> 을 참조하세요.
쿼리(분석)	\$8.55 per TB	매월 1TB까지는 무료입니다. 자세한 내용은 <a href="#">주문형 가격 책정</a> 을 참조하세요. 대량 사용 고객은 <a href="#">정액제</a> 를 이용할 수도 있습니다.

<https://cloud.google.com/bigquery/pricing>

# BigQuery를 어떻게 사용할까?

---

#1

기획 / 마케팅 / 신사업 등 다양한 팀에서 모두  
"데이터에 기반한 의사 결정을 진행"

각자 팀에서 스스로 쿼리를 날려 데이터를 추출

#2

"가설"을 설정한 후, 데이터를 통해 가설이 진짜 맞는지 확인

#3

머신러닝용 Feature 생성

# 데이터 이해하기

---

현재 BigQuery에 어떤 데이터가 쌓이고 있는지 점검

## #1. 앱 로그 데이터

- 유저가 앱에 들어와서 회원 가입 - 페이지 확인, 컨텐츠 확인 등등의 데이터
- "과정"을 알 수 있는 데이터
- Firebase에서 BigQuery로 데이터 적재

## #2. 결제 데이터

- 결제가 있는 서비스라면 최종 결제 금액 데이터

## #3. 유저 데이터

- 유저의 정보(성별, 연령대 등)

## #4. 공공 데이터

- 날씨, 좌표 등

# BigQuery(SQL) 이해하기

---

SQL : Structured Query Langauge

RDBMS의 데이터를 관리하기 위해 설계된 특수 목적의 프로그래밍 언어

# BigQuery(SQL) 이해하기

---

SQL : Structured Query Langauge

RDBMS의 데이터를 관리하기 위해 설계된 특수 목적의 프로그래밍 언어

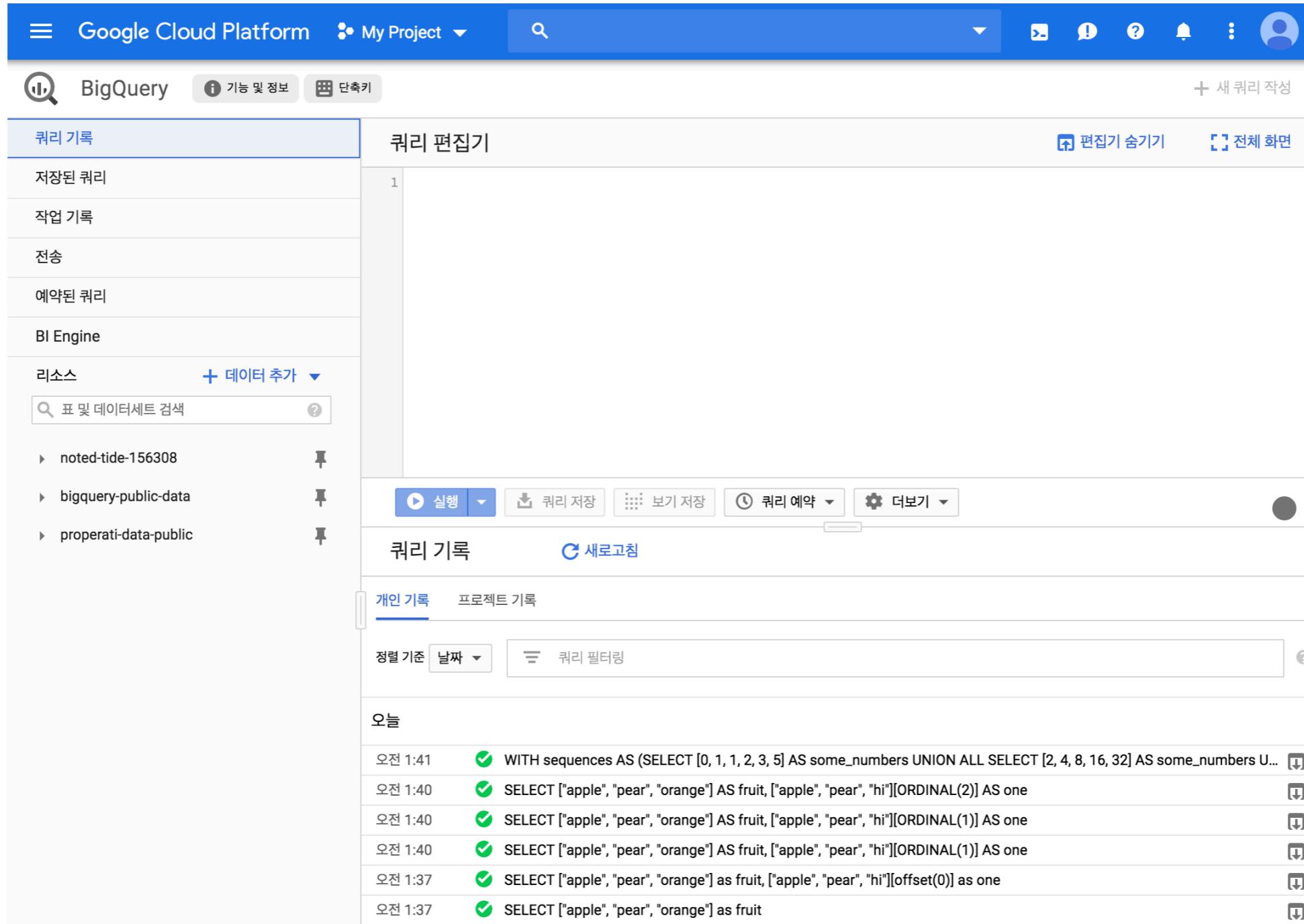
- RDBMS -

데이터를 관계로 표현

모든 데이터를 2차원의 표현(마치 엑셀과 유사)

# BigQuery Console

New UI  
<https://console.cloud.google.com/bigquery>



The screenshot shows the new BigQuery UI interface. At the top, there's a navigation bar with the Google Cloud Platform logo, 'My Project' dropdown, search bar, and various status icons. Below the navigation bar, the main area has a left sidebar with sections like '쿼리 기록' (Query History), '작업 기록' (Job History), '전송' (Transfer), '예약된 쿼리' (Scheduled Queries), and 'BI Engine'. The main panel is titled '쿼리 편집기' (Query Editor) and contains a large text area with a query ID '1'. Below the editor are buttons for '실행' (Run), '쿼리 저장' (Save Query), '보기 저장' (Save View), '쿼리 예약' (Schedule Query), and '더보기' (More). To the right of the editor, there are buttons for '편집기 숨기기' (Hide Editor) and '전체 화면' (Full Screen). The bottom section shows a timeline with recent queries, including:

- 오전 1:41 WITH sequences AS (SELECT [0, 1, 1, 2, 3, 5] AS some\_numbers UNION ALL SELECT [2, 4, 8, 16, 32] AS some\_numbers U... [down]
- 오전 1:40 SELECT ["apple", "pear", "orange"] AS fruit, ["apple", "pear", "hi"][[ORDINAL(2)] AS one [down]
- 오전 1:40 SELECT ["apple", "pear", "orange"] AS fruit, ["apple", "pear", "hi"][[ORDINAL(1)] AS one [down]
- 오전 1:40 SELECT ["apple", "pear", "orange"] AS fruit, ["apple", "pear", "hi"][[ORDINAL(1)] AS one [down]
- 오전 1:37 SELECT ["apple", "pear", "orange"] as fruit, ["apple", "pear", "hi"][[offset(0)] as one [down]
- 오전 1:37 SELECT ["apple", "pear", "orange"] as fruit [down]

# BigQuery Console

Legacy UI :

<https://bigquery.cloud.google.com>

The screenshot shows the Google BigQuery Legacy UI. At the top, there's a navigation bar with the 'Google BigQuery' logo, a 'Try the new UI' button, and a user profile icon labeled '성윤'. Below the navigation bar, the main interface has a sidebar on the left containing links for 'Query History', 'Job History', 'Scheduled Queries', and 'Transfers'. It also includes a 'Filter by ID or label' input field and a dropdown menu for 'My Project' which lists datasets like 'babynames', 'kaggle\_taxi', 'kaggle\_Web\_Traffic\_Time\_Series...', 'kaggle\_zillow\_2nd', 'test\_data', 'zillow\_prize', and 'bigquery-public-data'. A section titled 'Public Datasets' is also visible. The main content area on the right displays a message in Korean: '여긴 Default가 legacySQL standard 사용하고 싶으면 아래처럼!' followed by a sample SQL query: '#standardSQL  
SELECT \*  
FROM `table`  
LIMIT 100'. The entire interface is set against a light gray background.

여긴 Default가 legacySQL  
standard 사용하고 싶으면 아래처럼!

```
#standardSQL  
SELECT *  
FROM `table`  
LIMIT 100
```

# BigQuery Console

## 쿼리 기록(Query history) : 쿼리 기록(개인 기록 / 프로젝트 기록)

BigQuery 기능 및 정보 단축키 새 쿼리 작성

쿼리 기록 저장된 쿼리 작업 기록 전송 예약된 쿼리 BI Engine 리소스 + 데이터 추가 표 및 데이터세트 검색

1

쿼리 편집기 편집기 숨기기 전체 화면

▶ noted-tide-156308  
▶ bigquery-public-data

실행 쿼리 저장 보기 저장 쿼리 예약 더보기

쿼리 기록 새로고침 개인 기록 프로젝트 기록 정렬 기준 날짜 쿼리 필터링

19. 5. 6.

오후 10:13	✓	SELECT * -- SELECT distinct zip_code, city, county FROM `bigquery-public-data.geo_us_boundaries.us_zip_codes` WHERE state_code = 'NY' order by city	⬇️
오후 10:13	✓	SELECT * -- SELECT distinct zip_code, city, county FROM `bigquery-public-data.geo_us_boundaries.us_zip_codes` WHERE state_code = 'NY'	⬇️
오후 9:54	✓	SELECT * -- SELECT distinct zip_code, city, county FROM `bigquery-public-data.geo_us_boundaries.us_zip_codes` WHERE state_code = 'NY'	⬇️
오후 9:28	✓	SELECT * -- SELECT distinct zip_code, city, county FROM `bigquery-public-data.geo_us_boundaries.us_zip_codes` where city like '%New York%' LIMIT 1000	⬇️
오후 9:27	✓	SELECT distinct zip_code, city, county FROM `bigquery-public-data.geo_us_boundaries.us_zip_codes` where city like '%New York%' LIMIT 1000	⬇️
오후 9:27	✓	SELECT distinct city, county FROM `bigquery-public-data.geo_us_boundaries.us_zip_codes` where city like '%New York%' LIMIT 1000	⬇️
오후 9:27	✓	SELECT distinct city, county FROM `bigquery-public-data.geo_us_boundaries.us_zip_codes` where city like '%New York' LIMIT 1000	⬇️
오후 9:26	✓	SELECT distinct city, county FROM `bigquery-public-data.geo_us_boundaries.us_zip_codes` where zip_code='10001' LIMIT 1000	⬇️
오후 9:23	✓	SELECT distinct city, county FROM `bigquery-public-data.geo_us_boundaries.us_zip_codes` LIMIT 1000	⬇️
오후 9:23	✓	SELECT distinct county FROM `bigquery-public-data.geo_us_boundaries.us_zip_codes` LIMIT 1000	⬇️
오후 9:23	!	SELECT distinct country FROM `bigquery-public-data.geo_us_boundaries.us_zip_codes` LIMIT 1000	⬇️

# BigQuery Console

## 쿼리 기록(Query history) : 쿼리 기록(개인 기록 / 프로젝트 기록)

BigQuery 기능 및 정보 단축키 새 쿼리 작성

쿼리 기록 쿼리 편집기 편집기 숨기기 전체 화면

저장된 쿼리 작업 기록 전송 예약된 쿼리 BI Engine 리소스 데이터 추가 표 및 데이터세트 검색

noted-tide-156308 bigquery-public-data

실행 쿼리 저장 보기 저장 쿼리 예약 더보기

쿼리 기록 새로고침 개인 기록 프로젝트 기록 정렬 기준 날짜 쿼리 필터링

19. 5. 6.

오후 10:13	✓	snugyun01@gmail....	SELECT * – SELECT distinct zip_code, city, county FROM `bigquery-public-data.geo_us_boundaries.us_zip_codes` WHERE state_c...	⬇️
오후 10:13	✓	snugyun01@gmail....	SELECT * – SELECT distinct zip_code, city, county FROM `bigquery-public-data.geo_us_boundaries.us_zip_codes` WHERE state_c...	⬇️
오후 9:54	✓	snugyun01@gmail....	SELECT * – SELECT distinct zip_code, city, county FROM `bigquery-public-data.geo_us_boundaries.us_zip_codes` WHERE state_c...	⬇️
오후 9:28	✓	snugyun01@gmail....	SELECT * – SELECT distinct zip_code, city, county FROM `bigquery-public-data.geo_us_boundaries.us_zip_codes` where city like '...	⬇️
오후 9:27	✓	snugyun01@gmail....	SELECT distinct zip_code, city, county FROM `bigquery-public-data.geo_us_boundaries.us_zip_codes` where city like '%New York%'	⬇️
오후 9:27	✓	snugyun01@gmail....	SELECT distinct city, county FROM `bigquery-public-data.geo_us_boundaries.us_zip_codes` where city like '%New York%' LIMIT 10...	⬇️
오후 9:27	✓	snugyun01@gmail....	SELECT distinct city, county FROM `bigquery-public-data.geo_us_boundaries.us_zip_codes` where city like '%New York' LIMIT 1000	⬇️
오후 9:26	✓	snugyun01@gmail....	SELECT distinct city, county FROM `bigquery-public-data.geo_us_boundaries.us_zip_codes` where zip_code='10001' LIMIT 1000	⬇️
오후 9:23	✓	snugyun01@gmail....	SELECT distinct city, county FROM `bigquery-public-data.geo_us_boundaries.us_zip_codes` LIMIT 1000	⬇️
오후 9:23	✓	snugyun01@gmail....	SELECT distinct county FROM `bigquery-public-data.geo_us_boundaries.us_zip_codes` LIMIT 1000	⬇️
오후 9:23	!	snugyun01@gmail....	SELECT distinct country FROM `bigquery-public-data.geo_us_boundaries.us_zip_codes` LIMIT 1000	⬇️

# BigQuery Console

쿼리 클릭하면 쿼리 기록이 나타남

The screenshot shows the BigQuery Console interface. On the left, there's a sidebar with tabs for '쿼리 기록' (Query History), '저장된 쿼리' (Saved Queries), '작업 기록' (Job History), '전송' (Transfer), '예약된 쿼리' (Scheduled Queries), and 'BI Engine'. Below this is a '리소스' section with a search bar for '표 및 데이터세트 검색' and a dropdown for '데이터 추가'. A list of datasets is shown: 'noted-tide-156308' and 'bigquery-public-data'. The main area is titled '쿼리 편집기' (Query Editor) and shows a single query entry with ID '1'. The query code is:

```
1 SELECT *
2 -- SELECT distinct zip_code, city, county
3 FROM `bigquery-public-data.geo_us_boundaries.us_zip_codes`
4 WHERE state_code = 'NY'
5 order by city
```

Below the editor are buttons for '실행' (Run), '쿼리 저장' (Save Query), '보기 저장' (Save View), '쿼리 예약' (Schedule Query), and '더보기' (More). The status bar at the bottom says '쿼리 성공' (Query Success) and '4.016초 내에 쿼리 완료됨' (Query completed within 4.016 seconds). It also shows the time '오후 10:13'. To the right, there are links for '편집기에서 쿼리 열기' (Open query in editor) and '전체 화면' (Full screen). The entire query history section is highlighted with a red border.

# BigQuery Console

저장된 쿼리(Saved queries) : 개인 쿼리 / 프로젝트 쿼리(팀끼리 쿼리 공유)

The screenshot shows the BigQuery Console interface. On the left, there is a sidebar with the following items:

- 쿼리 기록
- 저장된 쿼리** (highlighted with an orange box)
- 작업 기록
- 전송
- 예약된 쿼리
- BI Engine

Below the sidebar, there is a search bar labeled "표 및 데이터세트 검색" and a button "+ 데이터 추가".

The main area is titled "쿼리 편집기" and contains the following elements:

- A table header with columns: 1, 편집기 숨기기, 전체 화면.
- A table body with one row, indexed at 1.
- Buttons at the top of the table: 실행 (Execute), 쿼리 저장 (Save Query), 보기 저장 (Save View), 쿼리 예약 (Schedule Query), 더보기 (More).
- A section titled "저장된 쿼리" with tabs: 개인 쿼리 (highlighted with an orange box) and 프로젝트 쿼리.
- A "쿼리 필터링" (Query Filtering) section with a dropdown menu.
- A query card for "뉴욕 주소 데이터 추출 쿼리" (New York Address Data Extraction Query) with a download icon.

# BigQuery Console

저장된 쿼리(Saved queries) : 쿼리 클릭시 쿼리 상세 정보를 볼 수 있음

The screenshot shows the BigQuery Console interface. On the left, there's a sidebar with navigation links: '쿼리 기록' (selected), '저장된 쿼리' (highlighted with an orange box), '작업 기록', '전송', '예약된 쿼리', 'BI Engine', and '리소스'. Below the sidebar, there's a search bar labeled '표 및 데이터세트 검색' and a dropdown menu '데이터 추가 ▾'. The main area is titled '쿼리 편집기' and shows a list of saved queries. The first query, '1' (highlighted with an orange box), is titled '저장된 쿼리'. It has three tabs: '개인 쿼리' (selected) and '프로젝트 쿼리'. Below these tabs is a '쿼리 필터링' section. The query details are shown in a blue box with the title '뉴욕 주소 데이터 추출 쿼리'. It includes a user selection dropdown ('개인(나만 수정 가능)'), a link sharing toggle ('링크 공유: 사용 안함'), a delete button ('삭제'), and a '편집기에서 쿼리 열기' button. The query code itself is:

```
1 SELECT *
2 FROM `bigquery-public-data.geo_us_boundaries.us_zip_codes`
3 WHERE state_code = 'NY'
4 order by city
```

# BigQuery Console

## 작업 기록(Job history) : 데이터 로드, 데이터 추출

BigQuery 기능 및 정보 단축키 새 쿼리 작성

쿼리 기록 저장된 쿼리 작업 기록 전송 예약된 쿼리 BI Engine 리소스 + 데이터 추가 표 및 데이터세트 검색

1 편집기 숨기기 전체 화면

작업 기록 실행 쿼리 저장 보기 저장 쿼리 예약 더보기

작업 기록 새로고침 개인 기록 프로젝트 기록

작업 필터링

18. 5. 18.

오전 12:23 noted-tide-156308:test\_data.leadme\_sample(으)로 업로드된 파일 로드

18. 2. 2.

오후 10:23 noted-tide-156308:kaggle\_zillow\_2nd.home\_attributes\_2007(으)로 gs://zgsza\_data/kaggle/zillow\_2nd/home\_attributes\_history/home\_attributes\_2007\_v1.cs...

오후 10:11 noted-tide-156308:kaggle\_zillow\_2nd.home\_attributes\_2007(으)로 gs://zgsza\_data/kaggle/zillow\_2nd/home\_attributes\_history/home\_attributes\_2007\_v1.cs...

오후 10:07 noted-tide-156308:kaggle\_zillow\_2nd.home\_attributes\_2007(으)로 gs://zgsza\_data/kaggle/zillow\_2nd/home\_attributes\_history/home\_attributes\_2007\_v1.cs...

오후 10:06 noted-tide-156308:kaggle\_zillow\_2nd.home\_attributes\_2007(으)로 gs://zgsza\_data/kaggle/zillow\_2nd/home\_attributes\_history/home\_attributes\_2007\_v1.cs...

오후 10:05 noted-tide-156308:kaggle\_zillow\_2nd.saleprice\_info(으)로 gs://zgsza\_data/kaggle/zillow\_2nd/saleprice/saleprice\_with\_buyer\_seller\_info\_2007-01-01to2017-...

오후 10:04 noted-tide-156308:kaggle\_zillow\_2nd.saleprice\_info(으)로 gs://zgsza\_data/kaggle/zillow\_2nd/saleprice/saleprice\_with\_buyer\_seller\_info\_2007-01-01to2017-...

오후 10:03 noted-tide-156308:kaggle\_zillow\_2nd.saleprice\_info(으)로 gs://zgsza\_data/kaggle/zillow\_2nd/saleprice/saleprice\_with\_buyer\_seller\_info\_2007-01-01to2017-...

오후 10:02 noted-tide-156308:kaggle\_zillow\_2nd.saleprice\_info(으)로 gs://zgsza\_data/kaggle/zillow\_2nd/saleprice/saleprice\_with\_buyer\_seller\_info\_2007-01-01to2017-...

<https://cloud.google.com/bigquery/docs/jobs-overview>

# BigQuery Console

## 클릭하면 상세 정보 보임

BigQuery 기능 및 정보 단축키 새 쿼리 작성

쿼리 기록 저장된 쿼리 작업 기록 전송 예약된 쿼리 BI Engine 리소스 + 데이터 추가 표 및 데이터세트 검색

작업 기록 1 편집기 숨기기 전체 화면

작업 ID: noted-tide-156308:US.bquijob\_2bed0392\_1636eb2de93  
작업자: snugyun01@gmail.com  
위치: 미국(US)  
생성 시간: 2018. 5. 18. 오전 12:23:30  
시작 시간: 2018. 5. 18. 오전 12:23:30  
종료 시간: 2018. 5. 18. 오전 12:23:32  
기간: 1.3초  
대상 테이블: noted-tide-156308:test\_data.leadme\_sample  
스키마 자동 감지: true  
소스 형식: CSV  
최대 불량 레코드: 0

부하 작업 반복

작업 기록 새고침

실행 쿼리 저장 보기 저장 쿼리 예약 더보기

# BigQuery Console

과거 작업을 반복할 수 있음! 테이블 만들기는 우선 Pass

The screenshot shows the BigQuery Console interface for creating a new table. The main area is titled "테이블 만들기" (Table Creation). It includes sections for "소스" (Source), "대상" (Destination), "스키마" (Schema), and "파티션 및 클러스터 설정" (Partition and Clustering settings).

**소스**: 다음 항목으로 테이블 만들기: 파일 선택: 파일 형식: 업로드 탐색 CSV

**대상**: 프로젝트 이름: My Project 데이터세트 이름: test\_data 테이블 유형: 기본 테이블 테이블 이름: leadme\_sample

**스키마**: 자동 감지:  스키마 및 입력 매개변수: 스키마가 자동으로 생성됩니다.

**파티션 및 클러스터 설정**: 분할: 파티션 없음

**고급 옵션**: 클러스팅 순서(선택사항): Clustering order determines the sort order of the data. Clustering can only be used on a partitioned table, and works with tables partitioned either by column or ingestion time. 클러스팅 순서를 정의하는 쉼표로 구분된 필드 목록입니다. 최대 4개를 입력할 수 있습니다.

작업 기록: noted-tide-156308:test\_data

작업 ID: 사용자: 위치: 생성 시간: 시작 시간: 종료 시간: 기간: 대상 테이블: 스키마 자동 감지: 소스 형식: 최대 불량 레코드:

테이블 만들기 취소

# BigQuery Console

전송(Transfer) : 데이터를 다른 곳에서 BigQuery로 옮김

예약된 쿼리(Scheduled queries) : 특정 시간대에 반복적으로 쿼리 실행

BI Engine : 데이터 시각화 해주는 도구

여기선 일단 이게 뭔지 정도만 알고 넘어갑시다

BigQuery

쿼리 기록

저장된 쿼리

작업 기록

전송

예약된 쿼리

BI Engine

리소스 + 데이터 추가 ▾

표 및 데이터세트 검색 ?

noted-tide-156308

bigquery-public-data

- austin\_311
- austin\_bikeshare
- austin\_crime
- austin\_incidents
- austin\_waste
- baseball
- bitcoin\_blockchain
- bls
- census\_bureau\_construction
- census\_bureau\_international
- census\_bureau\_usa
- census\_fips\_codes
- chicago\_crime
- chicago\_taxi\_trips
- cloud\_storage\_aoo\_index

실행 ▾

쿼리 저장

보기 저장

쿼리 예약 ▾

더보기 ▾

socar-data

+ 데이터세트 만들기

프로젝트 고정

이 프로젝트에는 데이터세트가 없습니다.

위의 제어 메뉴를 사용하여 데이터세트를 만들고 리소스 트리를 구성해 보세요.



# BigQuery Console

## 데이터셋과 테이블

BigQuery 기능 및 정보 단축키 새 쿼리 작성

저장되지 않은 쿼리 수정됨

편집기 숨기기 전체 화면

쿼리 기록 저장된 쿼리 작업 기록 전송 예약된 쿼리 BI Engine

리소스 + 데이터 추가 표 및 데이터세트 검색

처리 위치: US

실행 쿼리 저장 보기 저장 쿼리 예약 더보기

[ 데이터셋 ] [ 테이블 ]

bigquery-public-data:austin\_bikeshare

+ 테이블 만들기 데이터세트 공유 데이터세트 삭제

설명 라벨

없음 없음

데이터세트 정보

데이터세트 ID: bigquery-public-data:austin\_bikeshare  
생성 시간: 2017. 5. 17. 오전 4:25:37  
기본 표 만료: 사용 안함  
최종 수정 시간: 2018. 8. 23. 오전 5:36:57  
데이터 위치: US

[ 데이터셋 클릭시 나오는 부분(데이터셋 정보) ]

# BigQuery Console

## 테이블 관련 정보

BigQuery    기능 및 정보    단축키    + 새 쿼리 작성

쿼리 기록    저장되지 않은 쿼리    설정됨    편집기 숨기기    전체 화면

저장된 쿼리

작업 기록

전송

예약된 쿼리

BI Engine

리소스    + 데이터 추가 ▾

표 및 데이터세트 검색    ?

noted-tide-156308

bigquery-public-data

- austin\_311
- austin\_bikeshare
- bikeshare\_stations

bikeshare\_trips

처리 위치: US    실행    쿼리 저장    보기 저장    쿼리 예약    더보기

테이블 쿼리    테이블 복사    테이블 삭제    내보내기

스키마    세부정보    미리보기    [ 테이블 스키마 ]

필드 이름	유형	모드	설명
trip_id	INTEGER	NULLABLE	Numeric ID of bike trip
subscriber_type	STRING	NULLABLE	Type of the Subscriber
bikeid	INTEGER	NULLABLE	ID of bike used
start_time	TIMESTAMP	NULLABLE	Start timestamp of trip
start_station_id	INTEGER	NULLABLE	Numeric reference for start station
start_station_name	STRING	NULLABLE	Station name for start station
end_station_id	INTEGER	NULLABLE	Numeric reference for end station
end_station_name	STRING	NULLABLE	Station name for end station
duration_minutes	INTEGER	NULLABLE	Time of trip in minutes

스키마 수정

# BigQuery Console

## 테이블 관련 정보

BigQuery 기능 및 정보 단축키 + 새 쿼리 작성

쿼리 기록 저장된 쿼리 작업 기록 전송 예약된 쿼리 BI Engine 리소스 + 데이터 추가 표 및 데이터세트 검색 처리 위치: US 실행 쿼리 저장 보기 저장 쿼리 예약 더보기 편집기 숨기기 전체 화면

저장되지 않은 쿼리 수정됨 1

bikeshare\_trips 테이블 쿼리 테이블 복사 테이블 삭제 내보내기

스키마 세부정보 미리보기

[ 테이블 세부정보 ]

설명 라벨

없음 없음

테이블 정보

테이블 ID	bigquery-public-data:austin_bikeshare.bikeshare_trips
테이블 크기	115.56MB
장기 저장소 크기	115.56MB
열 개수	1,077,929
생성 시간	2017. 5. 25. 오후 12:55:42
표 만료	만료되지 않음
최종 수정 시간	2019. 2. 12. 오전 9:31:45
데이터 위치	US

# BigQuery Console

## 테이블 관련 정보 : SELECT \* 하지 말고 미리보기 활용!

The screenshot shows the BigQuery web interface. On the left, a sidebar lists various datasets and tables under the 'bigquery-public-data' project. The 'bikeshare\_trips' table is selected and highlighted with a blue border. The main panel displays the table's schema and a preview of the first 10 rows of data. The schema includes columns: 행 (row index), trip\_id, subscriber\_type, bikeid, start\_time, end\_time, start\_lng, start\_lat, end\_lng, end\_lat, and duration. The preview shows data from October 2014, with trips starting at locations like Toomey Rd @ South Lamar and Pease Park. The top right corner features a large button labeled '[ 테이블 데이터 미리보기 ]' (Table Data Preview) with a blue border.

BigQuery

기능 및 정보

단축키

+ 새 쿼리 작성

쿼리 기록

저장된 쿼리

작업 기록

전송

예약된 쿼리

BI Engine

리소스 + 데이터 추가 ▾

표 및 데이터세트 검색 ?

noted-tide-156308

bigquery-public-data

- austin\_311
- austin\_bikeshare
  - bikeshare\_stations
  - bikeshare\_trips**
- austin\_crime
- austin\_incidents
- austin\_waste
- baseball
- bitcoin\_blockchain
- bls
- census\_bureau\_construction
- census\_bureau\_international
- census\_bureau\_usa
- census\_fips\_codes
- chicago\_crime

처리 위치: US

실행 ▾

쿼리 저장

보기 저장

쿼리 예약 ▾

더보기 ▾

bikeshare\_trips

테이블 쿼리

테이블 복사

테이블 삭제

내보내기 ▾

스키마 세부정보 미리보기

행	trip_id	subscriber_type	bikeid	start_time	end_time	start_lng	start_lat	end_lng	end_lat	duration
1	9900285987	24-Hour Kiosk (Austin B-cycle)	446	2014-10-26 15:12:00 UTC		2712	Toomey Rd @ South Lamar			
2	9900285989	24-Hour Kiosk (Austin B-cycle)	203	2014-10-26 15:12:00 UTC		2712	Toomey Rd @ South Lamar			
3	9900285991	24-Hour Kiosk (Austin B-cycle)	101	2014-10-26 15:12:00 UTC		2712	Toomey Rd @ South Lamar			
4	9900286140	24-Hour Kiosk (Austin B-cycle)	242	2014-10-26 18:12:00 UTC		2541	State Capitol @ 14th & Colorado			
5	9900286143	24-Hour Kiosk (Austin B-cycle)	924	2014-10-26 18:12:00 UTC		2541	State Capitol @ 14th & Colorado			
6	9900286214	Annual Membership (Austin B-cycle)	24	2014-10-26 20:12:00 UTC		2712	Toomey Rd @ South Lamar			
7	9900286338	24-Hour Kiosk (Austin B-cycle)	101	2014-10-27 10:12:00 UTC		2712	Toomey Rd @ South Lamar			
8	13575843	Walk Up	302	2017-01-29 16:42:52 UTC		3464	Pease Park			
9	9900286942	24-Hour Kiosk (Austin B-cycle)	660	2014-10-28 14:12:00 UTC		2712	Toomey Rd @ South Lamar			
10	9900287148	24-Hour Kiosk (Austin B-cycle)	428	2014-10-28 19:12:00 UTC		2712	Toomey Rd @ South Lamar			

페이지당 행 수: 100 ▾ 1 - 100 / 1077929

첫 페이지 |< < > > 마지막 페이지

# BigQuery Console

## 테이블 쿼리 클릭시 쿼리 포맷 생성

BigQuery 기능 및 정보 단축키 새 쿼리 작성

쿼리 기록 저장된 쿼리 작업 기록 전송 예약된 쿼리 BI Engine 리소스 + 데이터 추가 표 및 데이터세트 검색

저장되지 않은 쿼리 수정됨 편집기 숨기기 전체 화면

! 1 SELECT FROM `bigquery-public-data.austin\_bikeshare.bikeshare\_trips` LIMIT 1000

실행 쿼리 저장 보기 저장 쿼리 예약 더보기 테이블 쿼리 테이블 복사 테이블 삭제 내보내기 !

bikeshare\_trips

행	trip_id	subscriber_type	bikeid	start_time	start_station_id	start_station_name	end_stat
1	9900285987	24-Hour Kiosk (Austin B-cycle)	446	2014-10-26 15:12:00 UTC	2712	Toomey Rd @ South Lamar	
2	9900285989	24-Hour Kiosk (Austin B-cycle)	203	2014-10-26 15:12:00 UTC	2712	Toomey Rd @ South Lamar	
3	9900285991	24-Hour Kiosk (Austin B-cycle)	101	2014-10-26 15:12:00 UTC	2712	Toomey Rd @ South Lamar	
4	9900286140	24-Hour Kiosk (Austin B-cycle)	242	2014-10-26 18:12:00 UTC	2541	State Capitol @ 14th & Colorado	
5	9900286143	24-Hour Kiosk (Austin B-cycle)	924	2014-10-26 18:12:00 UTC	2541	State Capitol @ 14th & Colorado	
6	9900286214	Annual Membership (Austin B-cycle)	24	2014-10-26 20:12:00 UTC	2712	Toomey Rd @ South Lamar	
7	9900286338	24-Hour Kiosk (Austin B-cycle)	101	2014-10-27 10:12:00 UTC	2712	Toomey Rd @ South Lamar	
8	13575843	Walk Up	302	2017-01-29 16:42:52 UTC	3464	Pease Park	
9	9900286942	24-Hour Kiosk (Austin B-cycle)	660	2014-10-28 14:12:00 UTC	2712	Toomey Rd @ South Lamar	
10	9900287148	24-Hour Kiosk (Austin B-cycle)	428	2014-10-28 19:12:00 UTC	2712	Toomey Rd @ South Lamar	

# SQL 이해하기

# BigQuery(SQL) 이해하기

SQL : Structured Query Language

RDBMS의 데이터를 관리하기 위해 설계된 특수 목적의 프로그래밍 언어

이걸 보시는 분은 대부분 **SELECT**만 하실거에요

데이터 선택 : SELECT

데이터 조작 (DML : Data Manipulation Language) : INSERT, UPDATE, DELETE

데이터 정의 (DDL : Data Definition Language) : CREATE, ALTER, DROP

데이터 제어 (DCL : Data Control Language)

BigQuery엔 없음

( TMI : 보통 DML에 SELECT가 속하는데,

BigQuery에선 DML에 INSERT, UPDATE, DELETE만 포함합니다 )

# SELECT의 큰 구조

---

행렬  
(Row, Column)

	Column 1	Column 2	Column 3
Row 1			
Row 2			
Row 3			
Row 4			

# SELECT의 큰 구조

---

SELECT [컬럼 이름]  
FROM [테이블 이름]



나는 [테이블 이름]에서  
[컬럼 이름]을 선택한다

# SELECT의 큰 구조

---

**SELECT [컬럼 이름]  
FROM [테이블 이름]  
WHERE [조건]**



나는 [테이블 이름]에서  
[조건]을 가지는  
[컬럼 이름]을 선택한다

# SELECT의 큰 구조

---

SELECT [컬럼 이름]  
FROM [테이블 이름]  
WHERE [조건]  
GROUP BY [그룹화할 컬럼]



나는 [테이블 이름]에서  
[조건]을 가지는 것들 중  
[그룹화할 컬럼]으로 모아서(그룹화해서)  
[컬럼 이름]을 선택한다

# SELECT의 큰 구조

---

SELECT [컬럼 이름]  
FROM [테이블 이름]  
WHERE [조건]  
GROUP BY [그룹화할 컬럼]  
**HAVING** [그룹화한 뒤 컬럼]  
LIMIT [제한할 개수]



나는 [테이블 이름]에서  
[조건]을 가지는 것들 중  
[그룹화할 컬럼]으로 모은 **후**(그룹화해서)  
[그룹화한 뒤 컬럼] 조건을 가지는  
[제한할 개수]만 선택  
[컬럼 이름]을 선택한다

# SELECT ~ FROM

---

가장 중심이 되는 요소

**SELECT** 뒤에는 찾고 싶은 대상을(Column)을 나열하고,  
**FROM** 뒤에는 찾을 대상이 있는 공간(Table)을 작성하면 됩니다

예시 : "서랍에 있는 연필을 찾고 싶은 경우"

SELECT 연필  
FROM 서랍

(모든 컬럼을 보고 싶다면 \* 사용)

# WHERE

---

조건을 주고 싶은 경우 WHERE 뒤에 작성

예시 : "서랍에 있는 연필 중 길이가 10cm 이상인 연필을 찾고 싶은 경우"

```
SELECT 연필  
FROM 서랍  
WHERE 길이 >= 10cm
```

# WHERE

---

조건을 하나 더 주고 싶은 경우 AND 뒤에 작성

예시 : "서랍에 있는 연필 중 길이가 10cm 이상이고 색상이 빨간색인  
연필을 찾고 싶은 경우"

```
SELECT 연필  
FROM 서랍  
WHERE 길이 >= 10cm AND 색상 = “빨간색”
```

# WHERE

---

특정 컬럼에 여러 조건을 지정하고 싶은 경우 : OR 사용

예시 : "서랍에 있는 연필 중 길이가 10cm 이상이고 색상이 빨간색이거나 파란색인 연필을 찾고 싶은 경우"

```
SELECT 연필  
FROM 서랍  
WHERE 길이 >= 10cm AND (색상 = “빨간색” OR 색상 = “파란색”)
```

# WHERE

---

특정 컬럼에 여러 조건을 지정하고 싶은 경우 : IN 사용

예시 : "서랍에 있는 연필 중 길이가 10cm 이상이고 색상이 빨간색이거나 파란색인 연필을 찾고 싶은 경우"

```
SELECT 연필
FROM 서랍
WHERE 길이 >= 10cm AND 색상 IN ("빨간색", "파란색")  
 WHERE 색상 = "빨간색" OR 색상 = "파란색"
```

# Mini Quiz

---

user\_log Table

user_id(int)	event(string)	event_date(string)
1	login_facebook	2019-05-14
1	write_posting	2019-05-14
1	write_comment	2019-05-14
1	view_posting	2019-05-14
1	view_posintg	2019-05-14
2	login_facebook	2019-05-14
2	view_posting	2019-05-14
2	view_posting	2019-05-14
2	write_comment	2019-05-14
2	logout	2019-05-14
2	login_facebook	2019-05-15
3	login_google	2019-05-15
3	write_posting	2019-05-15
3	view_posting	2019-05-15
3	purchase_item	2019-05-18
3	write_comment	2019-05-17
1	view_posting	2019-05-17
4	view_posintg	2019-05-17
5	purchase_item	2019-05-16

# Mini Quiz

---

1번 유저의 모든 이벤트 로그를 확인하는 쿼리

Column은 무엇인가요?

Table은 무엇인가요?

WHERE 조건은 무엇인가요?

# Mini Quiz

---

1번 유저의 모든 이벤트 로그를 확인하는 쿼리

Column은 무엇인가요?

Table은 무엇인가요?

WHERE 조건은 무엇인가요?

```
SELECT user_id, event, event_date  
FROM user_log  
WHERE user_id = 1
```

[ 결과 ]

user_id	event	event_date
1	login_facebook	2019-05-14
1	write_posting	2019-05-14
1	write_comment	2019-05-14
1	view_posting	2019-05-14
1	view_posintg	2019-05-14

# Mini Quiz

1번 유저의 모든 이벤트 로그를 확인하는 쿼리

Column은 무엇인가요?

Table은 무엇인가요?

WHERE 조건은 무엇인가요?

SELECT user\_id, event, event\_date

FROM user\_log

WHERE user\_id = 1

[ 결과 ]

user_id	event	event_date
1	login_facebook	2019-05-14
1	write_posting	2019-05-14
1	write_comment	2019-05-14
1	view_posting	2019-05-14
1	view_posintg	2019-05-14

같은 열이 반복

1번 유저가 2019-05-14에  
이벤트들을 몇 번 했는지 궁금하다면?

# GROUP BY

GROUP BY [컬럼 이름] 으로 사용

주로 집계 함수, 연산(SUM, COUNT)와 같이 쓰임

## GROUP BY

```
SELECT event, event_date, COUNT(DISTINCT user_id) AS unique  
FROM A  
GROUP BY event, event_date
```

### 예시 데이터

user_id	event	event_date
1	login_facebook	2018-03-12
2	write_posting	2018-03-13
1	write_comment	2018-03-12
3	login_facebook	2018-03-12



user_id	event	event_date
1	login_facebook	2018-03-12
3	login_facebook	2018-03-12
1	write_comment	2018-03-12
2	write_posting	2018-03-13



### 결과

event	event_date	unique
login_facebook	2018-03-12	2
write_comment	2018-03-12	1
write_posting	2018-03-13	1

# GROUP BY

---

1번 유저의 일자별 이벤트 횟수를 구하는 쿼리

Column은 무엇인가요?

Table은 무엇인가요?

WHERE 조건은 무엇인가요?

# GROUP BY

---

1번 유저의 일자별 이벤트 횟수를 구하는 쿼리

Column은 무엇인가요?

Table은 무엇인가요?

WHERE 조건은 무엇인가요?

```
SELECT user_id, event, event_date,  
       COUNT(DISTINCT user_id) AS unique,  
       COUNT(user_id) AS total  
  FROM user_log  
 WHERE user_id = 1  
 GROUP BY user_id, event, event_date
```

# GROUP BY

---

1번 유저의 일자별 이벤트 횟수를 구하는 쿼리

Column은 무엇인가요?

Table은 무엇인가요?

WHERE 조건은 무엇인가요?

```
SELECT user_id, event, event_date,  
       COUNT(DISTINCT user_id) AS unique,  
       COUNT(user_id) AS total  
  FROM user_log  
 WHERE user_id = 1  
 GROUP BY user_id, event, event_date
```

집계 함수(COUNT, SUM) 전의 컬럼은  
GROUP BY 뒤에 꼭 있어야 함!  
(그래야 그 앞의 값으로 COUNT)

# COUNT(DISTINCT ~)

---

COUNT : 개수를 카운트하는 함수  
COUNT(개수를 셀 컬럼)

DISTINCT : 고유한 값  
COUNT(DISTINCT user\_id) 고유한 user\_id를 센다

AS unique : 센 후 이름을 unique로 한다

```
SELECT user_id, event, event_date,  
       COUNT(DISTINCT user_id) AS unique,  
       COUNT(user_id) AS total  
  FROM user_log  
 WHERE user_id = 1  
 GROUP BY user_id, event, event_date
```

# ORDER BY

---

ORDER BY [컬럼] 컬럼 기준으로 오름차순(작은 값 -> 큰 값)

만약 내림차순을 하려면 ORDER BY [컬럼] DESC

```
SELECT user_id, event, event_date,  
       COUNT(DISTINCT user_id) AS unique,  
       COUNT(user_id) AS total  
  FROM user_log  
 WHERE user_id = 1  
 GROUP BY user_id, event, event_date  
ORDER BY event_date
```

# HAVING

---

[ 결과 ]

만약 이 상태에서 2번 이상 한 행동을 알고싶다면?  
(hint : total)

user_id	event	event_date	unique	total
1	login_facebook	2019-05-14	1	1
1	write_posting	2019-05-14	1	1
1	write_comment	2019-05-14	1	1
1	view_posting	2019-05-14	1	2

# HAVING

[ 결과 ]

만약 이 상태에서 2번 이상 한 행동을 알고싶다면?  
(hint : total)

user_id	event	event_date	unique	total
1	login_facebook	2019-05-14	1	1
1	write_posting	2019-05-14	1	1
1	write_comment	2019-05-14	1	1
1	view_posting	2019-05-14	1	2

```
SELECT user_id, event, event_date,  
       COUNT(DISTINCT user_id) AS unique,  
       COUNT(user_id) AS total
```

```
FROM user_log  
WHERE user_id = 1  
GROUP BY user_id, event, event_date  
HAVING total >= 2  
ORDER BY event_date
```

HAVING [그룹화한 이후 컬럼의 조건]

# 짧은 정리 다시 보기

---

SELECT [컬럼 이름]  
FROM [테이블 이름]  
WHERE [조건]  
GROUP BY [그룹화할 컬럼]  
**HAVING** [그룹화한 뒤 컬럼]  
LIMIT [제한할 개수]

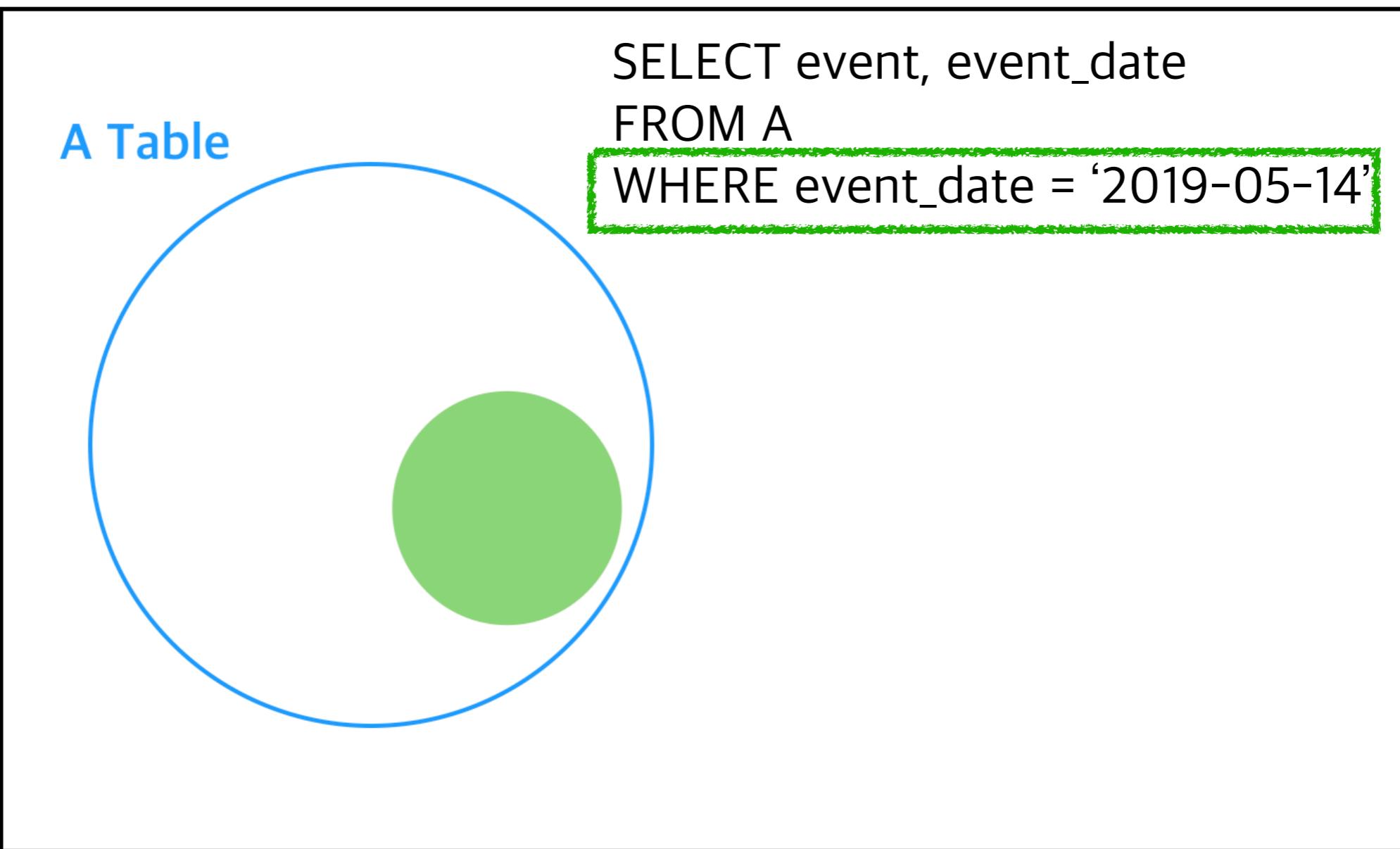


나는 [테이블 이름]에서  
[조건]을 가지는 것들 중  
[그룹화할 컬럼]으로 모은 후(그룹화해서)  
[그룹화한 뒤 컬럼] 조건을 가지는  
[제한할 개수]만 선택  
[컬럼 이름]을 선택한다

# 집합처럼 생각해보기

---

## Database



# 쿼리를 어떻게 짜야할까요?

---

1. 보고 싶은 지표가 무엇인지 고민한다
2. 보고 싶은 지표를 구체화한다(ex : 유저가 얼마나 A 제품을 구입하는가?)
  - 2-1. 사내에 이미 해당 지표를 구하는 쿼리가 있는지 찾아본다 => 있으면 바로 쿼리 맛보기
3. 보고 싶은 지표가 있는 데이터가 있는 Table 찾기(여러 Table일 수 있음)
  - 3-1. 하나의 Table에서 모든 데이터가 나올 것 같은 경우 => 바로 쿼리를 짠다
  - 3-2. 여러 Table에서 데이터를 연결해야 할 것 같은 경우 => 어떻게 연결할지 고민한다
4. 구한 결과값이 정말 맞는지 확인한다(데이터 정합성 체크)  
=> 원본 데이터를 하나씩 비교해보는 방법도 있고 다양한 방법이 존재
5. 쿼리를 더 좋게 만들 방법을 고민한다(쿼리 속도, 가독성 등)

# 쿼리를 어떻게 짜야할까요?

---

1. 보고 싶은 지표가 무엇인지 고민한다
2. 보고 싶은 지표를 구체화한다(ex : 유저가 얼마나 A 제품을 구입하는가?)  
2-1. 사내에 이미 해당 지표를 구하는 쿼리가 있는지 찾아본다 => 있으면 바로 쿼리 맛보기
3. 보고 싶은 지표가 있는 데이터가 있는 Table 찾기(여러 Table일 수 있음)  
3-1. 하나의 Table에서 모든 데이터가 나올 것 같은 경우 => 바로 쿼리를 짠다  
3-2. 여러 Table에서 데이터를 연결해야 할 것 같은 경우 => 어떻게 연결할지 고민한다
4. 구한 결과값이 정말 맞는지 확인한다(데이터 정합성 체크)  
=> 원본 데이터를 하나씩 비교해보는 방법도 있고 다양한 방법이 존재
5. 쿼리를 더 좋게 만들 방법을 고민한다(쿼리 속도, 가독성 등)

JOIN

# 실습 (1)

# JOIN 하기 전에 실습

## [ 공개 데이터셋 ]

The screenshot shows the BigQuery web interface. On the left, there's a sidebar with various navigation options: 쿼리 기록, 저장된 쿼리, 작업 기록, 전송, 예약된 쿼리, BI Engine, and 리소스. Below the BI Engine option is a button labeled '+ 데이터 추가 ▾'. A dropdown menu is open under this button, showing '표 및 데이터세트' and '프로젝트 고정'. Under '표 및 데이터세트', there's a list of datasets: noted-tide-156308, babynames, kaggle\_taxi, kaggle\_Web\_Traffic\_Time\_Seri..., kaggle\_zillow\_2nd, test\_data, zillow\_prize, and bigquery-public-data. The 'noted-tide-156308' dataset is highlighted with an orange box. At the top right of the interface, there are buttons for 실행 (Execute), 쿼리 저장 (Save Query), 보기 저장 (Save View), 쿼리 예약 (Schedule Query), and 더보기 (More). The main area displays the contents of the selected dataset, with the message 'noted-tide-156308' and a note '리소스 트리를 사용하여'.

# JOIN 하기 전에 실습

[ 다양한 공개 데이터가 있음! ]

X      솔루션 검색

## 데이터세트

필터링 기준      검색결과 107개

유형  
데이터세트 (1)

카테고리  
광고 (7)  
분석 (6)  
빅데이터 (9)  
기후 (21)  
데이터베이스 (1)  
개발자 도구 (2)  
경제학 (26)  
백과사전 (27)  
유전체학 (1)  
건강 (8)  
머신러닝 (1)  
지도 (1)  
공공안전 (13)  
과학 및 연구 (42)  
소설 (3)  
교통 (1)  
기타 (11)

검색 결과

Argentina Real Estate Listings Properati  Monthly property listing data for Argentina since 2016	Austin Crime Data City of Austin  City of Austin crime data for 2014 and 2015	Bitcoin Cash Cryptocurrency Dataset Google Cloud Public Datasets Pro...  The Bitcoin Cash blockchain loaded to BigQuery and updated daily	Brazil Real Estate Listings Properati  Monthly property listing data for Brazil since 2016
Bureau of Labor Statistics U.S. Bureau of Labor Statistics  U.S. economic statistics for inflation, prices and unemployment	ChEMBL Data Google Patents Public Datasets	Chicago Crime Data City of Chicago  Chicago Police Department crime data from 2001 to present	Chicago Taxi Trips City of Chicago  Chicago taxi trips from 2013 to present
Chile Real Estate Listings Properati  Monthly property listing data for Chile since 2016	Cloud-to-Ground Lightning Strikes NOAA  Aggregated lightning strike data from 1987 to 2018	Columbia Real Estate Listings Properati  Monthly property listing data for Columbia since 2016	Cooperative Patent Classification Data Google Patents Public Datasets
CPA Global Technical Standards ETSI Data	crypto_zcash Google Cloud Public Datasets Pro...	Dash Cryptocurrency Google Cloud Public Datasets Pro...	Disclosed Standard Essential Patents (dSEP) Data

# JOIN 하기 전에 실습

[ bike 검색 후 아무거나 클릭 ]

The screenshot shows a search interface with a search bar containing the text "bike". Below the search bar, the text "Marketplace > 'bike'" is displayed. The main heading is "Datasets" followed by "Filter by". Under "Filter by", there are two sections: "TYPE" and "CATEGORY". In the "TYPE" section, "Datasets" is selected. In the "CATEGORY" section, items listed are "Encyclopedic (1)", "Public safety (1)", and "Transportation (1)". On the right side, there are two dataset cards. The first card is for "San Francisco Ford GoBike Share" from the "City and County of San Francisco", describing it as trip data for the Bay Area's Ford GoBike bike share system. The second card is for "NYC Citi Bike Trips" from the "City of New York", describing it as New York City bike share trips since 2013. The "NYC Citi Bike Trips" card is highlighted with a thick orange border.

Marketplace > "bike"

## Datasets

Filter by

2 results

TYPE

Datasets ×

CATEGORY

Encyclopedic (1)

Public safety (1)

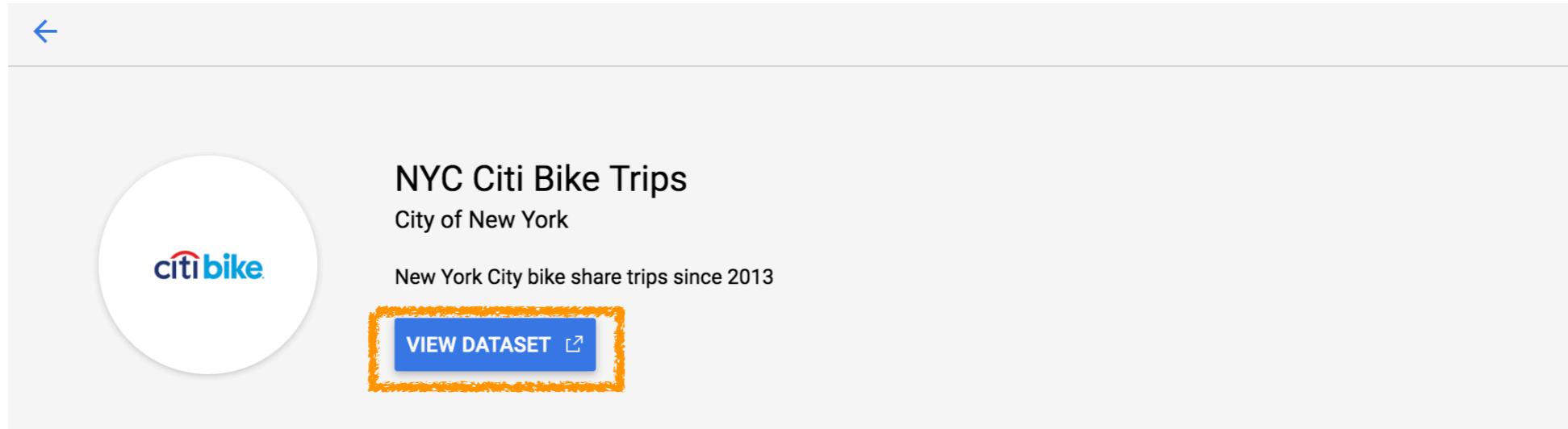
Transportation (1)

San Francisco Ford GoBike Share  
City and County of San Francisco  
Trip data the Bay Area's Ford GoBike bike share system

NYC Citi Bike Trips  
City of New York  
New York City bike share trips since 2013

# JOIN 하기 전에 실습

## [ VIEW DATASET 클릭 ]



Type	Overview
Datasets	Citi Bike is the nation's largest bike share program, with 10,000 bikes and 600 stations across Manhattan, Brooklyn, Queens, and Jersey City. This dataset includes Citi Bike trips since Citi Bike launched in September 2013 and is updated daily. The data has been processed by Citi Bike to remove trips that are taken by staff to service and inspect the system, as well as any trips below 60 seconds in length, which are considered false starts.
Last updated	1/10/19, 5:24 AM
Category	This public dataset is hosted in Google BigQuery and is included in BigQuery's 1TB/mo of free tier processing. This means that each user receives 1TB of free BigQuery processing every month, which can be used to run queries on this public dataset. Watch this short video to learn how to get started quickly using BigQuery to access public datasets. <a href="#">What is BigQuery</a> .
Dataset source	<a href="#">Citi Bike New York</a>
Cloud service	BigQuery
Region	US
Update frequency	Monthly

**Overview**

Citi Bike is the nation's largest bike share program, with 10,000 bikes and 600 stations across Manhattan, Brooklyn, Queens, and Jersey City. This dataset includes Citi Bike trips since Citi Bike launched in September 2013 and is updated daily. The data has been processed by Citi Bike to remove trips that are taken by staff to service and inspect the system, as well as any trips below 60 seconds in length, which are considered false starts.

This public dataset is hosted in Google BigQuery and is included in BigQuery's 1TB/mo of free tier processing. This means that each user receives 1TB of free BigQuery processing every month, which can be used to run queries on this public dataset. Watch this short video to learn how to get started quickly using BigQuery to access public datasets. [What is BigQuery](#).

### Samples

Try the sample queries below in the BigQuery UI.

#### What are the most popular Citi Bike stations?

First, let's look at the most popular Citi Bike stations, including their name, location, and number of trips. [Run this query](#).

#### What are the most popular routes by subscriber type?

Next, let's look at the most popular routes by subscriber type, where "Subscribers" are Citibike members and "Customers" are one-off users. This query uses CONCAT to get the route. [Run this query](#).

#### What are the top routes by gender?

This query looks at top routes by gender. Here we get the top female routes in 2016. This query can easily be edited for different years and genders. [Run this query](#).

For more examples using NYC data in BigQuery see [this tutorial](#) on YouTube.

# JOIN 하기 전에 실습

[ bigquery-public-data가 생김! ]

The screenshot shows the BigQuery web interface. On the left, there is a sidebar with various navigation links: 쿼리 기록, 저장된 쿼리, 작업 기록, 전송, 예약된 쿼리, BI Engine, and 리소스. Below these is a search bar labeled '표 및 데이터세트 검색'. Under '리소스', there is a tree view of datasets. A specific dataset, 'bigquery-public-data', is expanded, revealing its contents: austin\_311, austin\_bikeshare, austin\_crime, austin\_incidents, austin\_waste, baseball, bitcoin\_blockchain, bls, census\_bureau\_construction, census\_bureau\_international, census\_bureau\_usa, and census\_fips\_codes. The 'austin\_bikeshare' dataset is selected and highlighted with a red border. The main panel on the right displays the details for the selected dataset. The title is 'bigquery-public-data:austin\_bikeshare'. Below it is a '설명' (Description) section with a '없음' (None) status. The '데이터세트 정보' (Dataset Information) section contains the following data:

데이터세트 ID	bigquery-public-data:austin_bikeshare
생성 시간	2017. 5. 17. 오전 4:25:37
기본 표 만료	사용 안함
최종 수정 시간	2018. 8. 23. 오전 5:36:57
데이터 위치	US

# JOIN 하기 전에 실습

데이터 설명 : bikeshare trip data  
**bikeshare\_stations Table**

bikeshare\_stations

테이블 쿼리 테이블 복사 테이블 삭제 내보내기 ▾

스키마 세부정보 미리보기

행	station_id	name	status	latitude	longitude	location
1	3793	Rio Grande & 28th	active	30.29333	-97.74412	(30.29333, -97.74412)
2	3291	11th & San Jacinto	active	30.27193	-97.73854	(30.27193, -97.73854)
3	4058	Hollow Creek & Barton Hills	active	30.26139	-97.77234	(30.26139, -97.77234)
4	3797	21st & University	active	30.28354	-97.73953	(30.28354, -97.73953)
5	3838	Nueces & 26th	active	30.29068	-97.74292	(30.29068, -97.74292)
6	2544	East 6th & Pedernales St.	active	30.25895	-97.71475	(30.25895, -97.71475)
7	3390	Brazos & 6th	active	30.26754	-97.74154	(30.26754, -97.74154)
8	2823	Capital Metro HQ - East 5th at Broadway	active	30.2563	-97.71007	(30.2563, -97.71007)
9	2711	Barton Springs @ Kinney Ave	active	30.262	-97.76118	(30.262, -97.76118)
10	3685	Henderson & 9th	active	30.27217	-97.75246	(30.27217, -97.75246)

# JOIN 하기 전에 실습

데이터 설명 : bikeshare trip data  
**bikeshare\_trips Table** ( 지금은 이거만 사용해요 )

bikeshare\_trips

퀵 쿼리 테이블 복사 테이블 삭제 내보내기 ▾

행	trip_id	subscriber_type	bikeid	start_time	start_station_id	start_station_name	end_st
1	9900285987	24-Hour Kiosk (Austin B-cycle)	446	2014-10-26 15:12:00 UTC	2712	Toomey Rd @ South Lamar	
2	9900285989	24-Hour Kiosk (Austin B-cycle)	203	2014-10-26 15:12:00 UTC	2712	Toomey Rd @ South Lamar	
3	9900285991	24-Hour Kiosk (Austin B-cycle)	101	2014-10-26 15:12:00 UTC	2712	Toomey Rd @ South Lamar	
4	9900286140	24-Hour Kiosk (Austin B-cycle)	242	2014-10-26 18:12:00 UTC	2541	State Capitol @ 14th & Colorado	
5	9900286143	24-Hour Kiosk (Austin B-cycle)	924	2014-10-26 18:12:00 UTC	2541	State Capitol @ 14th & Colorado	
6	9900286214	Annual Membership (Austin B-cycle)	24	2014-10-26 20:12:00 UTC	2712	Toomey Rd @ South Lamar	
7	9900286338	24-Hour Kiosk (Austin B-cycle)	101	2014-10-27 10:12:00 UTC	2712	Toomey Rd @ South Lamar	
8	13575843	Walk Up	302	2017-01-29 16:42:52 UTC	3464	Pease Park	
9	9900286942	24-Hour Kiosk (Austin B-cycle)	660	2014-10-28 14:12:00 UTC	2712	Toomey Rd @ South Lamar	
10	9900287148	24-Hour Kiosk (Austin B-cycle)	428	2014-10-28 19:12:00 UTC	2712	Toomey Rd @ South Lamar	

# JOIN 하기 전에 실습

---

- #1. 총 trip의 개수를 구해주세요
- #2. 일자별 trip의 개수를 구해주세요 (hint : DATE() 함수)
- #3. subscriber\_type별 trip 개수를 구해주세요
- #4. 시작 station과 도착 station별 개수를 구해주세요(제일 많은 구간은?)

# JOIN 하기 전에 실습

---

#1. 총 trip의 개수를 구해주세요

```
SELECT count(trip_id)
FROM `bigquery-public-data.austin_bikeshare.bikeshare_trips`
```

문제 출제 의도 : COUNT(column 이름)으로 개수를 COUNT할 수 있음

# JOIN 하기 전에 실습

---

#2. 일자별 trip의 개수를 구해주세요 (hint : DATE() 함수)

#2

```
SELECT DATE(start_time) as date,  
count(trip_id) as count  
FROM `bigquery-public-data.austin_bikeshare.bikeshare_trips`  
GROUP BY date  
order by date
```

문제 출제 의도 : DATE(TIMESTAMP인 start\_date)를 하면 date를 뽑을 수 있음

# JOIN 하기 전에 실습

---

#3. subscriber\_type별 trip 개수를 구해주세요

#3

```
SELECT subscriber_type,  
       count(trip_id) as count  
  FROM `bigquery-public-data.austin_bikeshare.bikeshare_trips`  
 GROUP BY subscriber_type  
 ORDER BY count DESC
```

# JOIN 하기 전에 실습

---

#4. 시작 station과 도착 station별 개수를 구해주세요(제일 많은 구간은?)

#4

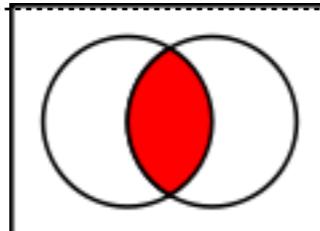
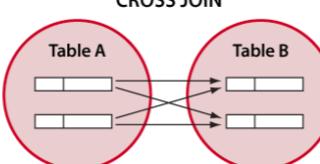
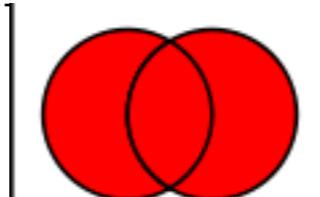
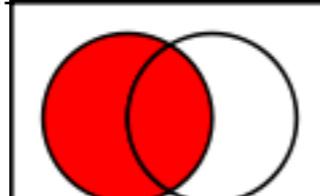
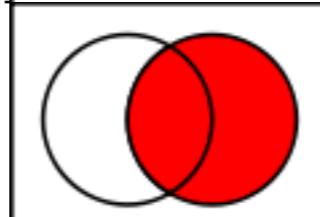
```
SELECT start_station_id, end_station_id, count(trip_id) as trip_cnt  
FROM `bigquery-public-data.austin_bikeshare.bikeshare_trips`  
GROUP BY start_station_id, end_station_id  
order by trip_cnt desc
```

문제 출제 의도 : 2가지 이상 GROUP BY

# JOIN

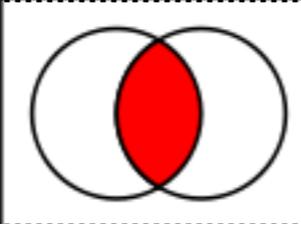
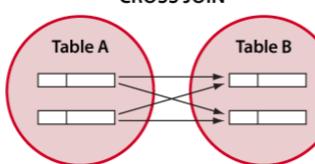
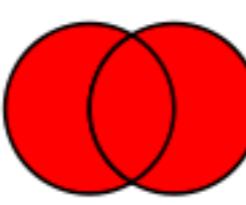
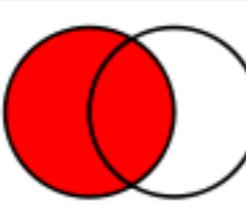
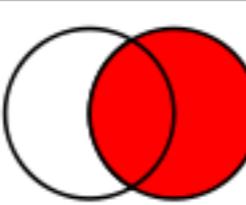
JOIN : 데이터끼리 “연결”

2개 이상의 Table을 조합해 새로운 Table을 만듬  
(엑셀의 vlookup과 유사)

종류	이미지	설명
INNER JOIN		두 집합의 "교집합"
CROSS JOIN		두 집합의 "곱집합"(요소끼리 곱함)
FULL [OUTER] JOIN		양쪽 모두 데이터를 붙임
LEFT [OUTER] JOIN		왼쪽 기준으로 붙임
RIGHT [OUTER] JOIN		오른쪽 기준으로 붙임

# JOIN

만약 SQL이 처음이시면  
INNER JOIN, LEFT JOIN만 사용하시다가  
점점 확장 추천!

종류	이미지	설명
INNER JOIN		두 집합의 "교집합"
CROSS JOIN		두 집합의 "곱집합"(요소끼리 곱함)
FULL [OUTER] JOIN		양쪽 모두 데이터를 붙임
LEFT [OUTER] JOIN		왼쪽 기준으로 붙임
RIGHT [OUTER] JOIN		오른쪽 기준으로 붙임

# JOIN 예시

---

austin\_bikeshare 데이터셋에서  
출발지(start\_station\_id)와 도착지(end\_station\_id)의  
좌표를 불이고 싶은 경우

# JOIN 예시

---

austin\_bikeshare 데이터셋에서  
출발지(start\_station\_id)와 도착지(end\_station\_id)의  
좌표를 불리고 싶은 경우  
[ #1. 출발지 정보 추가 ]

```
SELECT
  left_side.*,
  middle_side.latitude as start_lat, middle_side.longitude as start_lng
FROM `bigquery-public-data.austin_bikeshare.bikeshare_trips` AS left_side
LEFT JOIN
  (SELECT * FROM `bigquery-public-data.austin_bikeshare.bikeshare_stations`) as
middle_side
ON left_side.start_station_id = middle_side.station_id
```

# JOIN 예시

---

austin\_bikeshare 데이터셋에서  
출발지(start\_station\_id)와 도착지(end\_station\_id)의  
좌표를 불리고 싶은 경우  
#1. 출발지 정보 추가  
[ #2. 도착지 정보 추가 ]

```
SELECT
    left_side.*,
    middle_side.latitude as start_lat, middle_side.longitude as start_lng,
    right_side.latitude as end_lat, right_side.longitude as end_lng
FROM `bigquery-public-data.austin_bikeshare.bikeshare_trips` AS left_side
LEFT JOIN
    (SELECT * FROM `bigquery-public-data.austin_bikeshare.bikeshare_stations`) as
middle_side
    ON left_side.start_station_id = middle_side.station_id
LEFT JOIN
    (SELECT * FROM `bigquery-public-data.austin_bikeshare.bikeshare_stations`) as
right_side
    ON left_side.end_station_id = right_side.station_id
```

# JOIN 예시

---

[ 앞 쿼리와 동일함! ]

앞 쿼리는 JOIN 뒤에 (SELECT ~ FROM) 할 수 있는 것을 보여주려고 사용

SELECT

```
left_side.*,
middle_side.latitude as start_lat, middle_side.longitude as start_lng,
right_side.latitude as end_lat, right_side.longitude as end_lng
FROM `bigquery-public-data.austin_bikeshare.bikeshare_trips` AS left_side
LEFT JOIN `bigquery-public-data.austin_bikeshare.bikeshare_stations` as
middle_side
ON left_side.start_station_id = middle_side.station_id
```

**LEFT JOIN**

```
`bigquery-public-data.austin_bikeshare.bikeshare_stations` ) as right_side
ON left_side.end_station_id = right_side.station_id
```

# 시간 데이터 다루기

---

**DATE** : Date만 표시하는 데이터, 2019-05-15

**DATETIME** : Date와 TIME까지 표시하는 데이터, 2019-05-15 14:00:00

그 외에 TIMESTAMP, TIME도 있음



데이터의 timezone을 꼭 확인할 것!

(UTC 인지? 만약 다른 시간대라면 변경할 때 'Asia/Seoul'을 사용해야 함)

## CURRENT\_DATE([time\_zone])

: 현재 날짜

Ex) CURRENT\_DATE('Asia/Seoul')

## EXTRACT(part FROM date\_expression)

: DATE에서 값(일자, 요일, 주차, 월, 분기, 연도 등) 추출

Ex) EXTRACT(YEAR FROM <date type column>) : “연도” 추출

```
SELECT EXTRACT(DAY FROM DATE '2019-12-25') as the_day;
```

the_day
25

# 시간 데이터 다루기

[ DATE ]

## DATE\_ADD(date\_expression, INTERVAL INT date\_part)

: Date에서 INT 간격만큼 더함(DAY, WEEK, MONTH, QUATER, YEAR 등)

Ex) DATE\_ADD(<date type column>, INTERVAL 1 DAY) : 1일 뒤 날짜

```
SELECT DATE_ADD(DATE "2019-03-25", INTERVAL 5 DAY) as five_days_later;
+-----+
| five_days_later |
+-----+
| 2019-03-30      |
+-----+
```

## DATE\_DIFF(date\_expression\_a, date\_expression\_b, date\_part)

: a-b date diff

```
SELECT DATE_DIFF(DATE '2010-07-07', DATE '2008-12-25', DAY) as days_diff;
+-----+
| days_diff |
+-----+
| 559       |
+-----+
```

**DATE\_TRUNC(date\_expression, date\_part)**

: 날짜를 지정한 단위로 자름

```
SELECT DATE_TRUNC(DATE '2008-12-25', MONTH) as month;  
+-----+  
| month |  
+-----+  
| 2008-12-01 |  
+-----+
```

## FORMAT\_DATE(format\_string, date\_expr)

: date\_expr를 format\_string처럼 변경  
결과는 String

```
SELECT FORMAT_DATE("%x", DATE "2019-12-25") as US_format;
+-----+
| US_format |
+-----+
| 12/25/19 |
+-----+
```

## PARSE\_DATE(format\_string, date\_string)

: string을 format\_string처럼 변경  
결과는 Date

```
SELECT PARSE_DATE("%x", "12/25/19") as parsed;
+-----+
| parsed |
+-----+
| 2019-12-25 |
+-----+
```

# 시간 데이터 다루기

[ DATE ]

## FORMAT\_STRING 종류

%A	요일 이름입니다.
%a	요일 이름의 약어입니다.
%B	월 이름입니다.
%b	월 이름의 약어입니다. 는 %h
%C	10진수(00-99)로 표현한 세기(연도를 100으로 나누어 정수로 자른 것)입니다.
%D	%m/%d/%y 형식으로 표현한 날짜입니다.
%d	한 달의 일을 10진수(01-31)로 표현한 것입니다.
%e	한 달의 일을 10진수(1-31)로 표현한 것입니다. 한 자릿수 앞에는 공백이 옵니다.
%F	%Y-%m-%d 형식으로 표현한 날짜입니다.
%G	ISO 8601 연도를 세기와 함께 10진수로 표현한 것입니다.
%g	ISO 8601 연도를 세기 없이 10진수(00-99)로 표현한 것입니다.
%j	한 해의 일을 10진수(001-366)로 표현한 것입니다.
%m	월을 10진수(01-12)로 표현한 것입니다.
%n	줄바꿈 문자입니다.
%t	탭 문자입니다.
%U	한 해의 주 번호(일요일이 일주일의 첫 번째 날임)를 10진수(00-53)로 표현한 것입니다.
%u	요일(월요일이 일주일의 첫 번째 날임)을 10진수(1-7)로 표현한 것입니다.
%V	한 해의 주 번호(월요일이 일주일의 첫 번째 날임)를 10진수(01-53)로 표현한 것입니다. 새해에 1월 1일이 포함된 주의 일수가 4일 이상인 경우, 그 주가 첫 번째 주이고, 그렇지 않은 경우에는 그 주가 이전 연도의 53번째 주이고 그 다음 주가 첫 번째 주입니다.
%W	한 해의 주 번호(월요일이 일주일의 첫 번째 날임)를 10진수(00-53)로 표현한 것입니다.
%w	요일(일요일이 일주일의 첫 번째 날임)을 10진수(0-6)로 표현한 것입니다.
%x	날짜를 MM/DD/YY 형식으로 표현한 것입니다.
%Y	연도를 세기와 함께 10진수로 표현한 것입니다.
%y	연도를 세기 없이 10진수(00-99)로 표현한 것입니다. 앞의 0 표기 여부는 선택할 수 있습니다. %C와 함께 사용할 수 있습니다. %C를 지정하지 않으면, 00-68년은 2000년대이고 69-99년은 1900년대입니다.
%E4Y	4자릿수 연도(0001 ... 9999)입니다. %Y는 연도를 완전히 렌더링하는 데 필요한 만큼 문자 수를 생성한다는 점에 유의하세요.

# 시간 데이터 다루기

[ DATETIME ]

---

DATE랑 거의 비슷합니다  
(TIME을 더 보여준다 정도의 차이)

공식 문서엔 DATETIME 함수 설명에 EXTRACT가 없지만 사용 가능

```
SELECT EXTRACT(HOUR FROM CAST('2019-10-01 14:00:00' AS DATETIME)) as a,
```

### CURRENT\_DATETIME([time\_zone])

: 현재 날짜

Ex) CURRENT\_DATE('Asia/Seoul')

### EXTRACT(part FROM datetime\_expression)

: DATETIME에서 값(일자, 요일, 주차, 월, 분기, 연도, 시간 등) 추출

Ex) EXTRACT(YEAR FROM <datetime type column>) : “연도” 추출

```
SELECT EXTRACT(HOUR FROM CAST('2019-12-25 14:00:00' AS DATETIME) as hour;
```

hour
14

# 시간 데이터 다루기

[ DATETIME ]

---

**DATETIME\_ADD(datetime\_expression, INTERVAL INT part)**

**DATETIME\_DIFF(datetime\_expression\_a, datetime\_expression\_b, part)**

**DATETIME\_TRUNC(datetime\_expression, part) : 시간 자르기**

Ex) : 2019-05-19 19:03:22을 HOUR로 자르면 2019-05-19 19:00:00

**FORMAT\_DATETIME(format\_string, datetime\_expression)**

**PARSE\_DATETIME(format\_string, string)**

# EXCEPT

---

1. SELECT할 컬럼 중 딱 2개만 제외하고 싶은 경우
2. JOIN하고 특정 컬럼을 제외하고 SELECT하고 싶은 경우

```
SELECT  
    * EXCEPT (a, b, c)  
FROM TABLE
```

```
SELECT  
    A.* EXCEPT (column a),  
    B.* EXCEPT(column b)      ( 은근히 편해요 )  
FROM TABLE as A  
LEFT JOIN TABLE as B  
ON A.id=B.id
```

# 데이터 시각화

# 시각화 하는 다양한 방법

---

1. 쿼리를 날린 후, 스프레드시트로 저장한 후 스프레드시트로 시각화
  2. Tableau에 BigQuery를 연결해서 시각화
  3. 쿼리를 날린 후, 데이터를 내 컴퓨터에 저장해서 Python으로 시각화  
(Matplotlib, Seaborn, Plotly 등)
  4. Data Studio를 사용해서 시각화
- 등등.. 다양한 방법이 존재

# 시각화 하는 다양한 방법

---

## [ 방법들의 장단점 ]

1. 쿼리를 날린 후, 스프레드시트로 저장한 후 스프레드시트로 시각화  
**엑셀과 유사해 편하고, 공유도 쉬움. 단, 시각화가 어렵고 16,000행까지 가능**
2. Tableau에 BigQuery를 연결해서 시각화  
**시각화를 정말 쉽게 할 수 있음. 단, 유료**
3. 쿼리를 날린 후, 데이터를 내 컴퓨터에 저장해서 Python으로 시각화  
(Matplotlib, Seaborn, Plotly 등)  
**파이썬, 시각화 라이브러리에 능숙해야 함**
4. Data Studio를 사용해서 시각화  
**쿼리 결과에서 클릭 한번하고 진행 가능. 스프레드시트 상위 호환**

# 시각화 하는 다양한 방법

[ 스프레드시트 ]

## [ 쿼리를 날린 후, 결과 저장 - Google 스프레드시트 ]

The screenshot shows the BigQuery web interface. On the left, there's a sidebar with sections like '쿼리 기록', '저장된 쿼리', '작업 기록', '전송', '예약된 쿼리', 'BI Engine', and '리소스'. Under '리소스', it lists 'noted-tide-156308' and 'bigquery-public-data'. The main area displays a query in the '쿼리 편집기' tab:

```
1 SELECT DATE(start_time) as date, count(trip_id) as count
2 FROM `bigquery-public-data.austin_bikeshare.bikeshare_trips`
3 GROUP BY date
```

Below the query are buttons for '실행', '쿼리 저장', '보기 저장', '쿼리 예약', and '더보기'. The '쿼리 결과' tab is selected, showing a table with columns '행', 'date', and 'count'. The data is as follows:

행	date	count
1	2014-10-26	698
2	2014-10-27	494
3	2017-01-29	465
4	2014-10-28	482
5	2014-10-29	409
6	2014-10-30	412
7	2014-10-31	538
8	2015-10-01	576
9	2015-10-02	1375
10	2015-10-03	1856
11	2015-10-04	1642
12	2015-10-05	643
13	2015-10-06	589
14	2015-10-07	473
15	2015-10-08	650

To the right of the table, there are options for saving the results: '결과 저장' (CSV, JSON, BigQuery Table, Google Sheets), '데이터 스튜디오에서 살펴보기', and a '더보기' button. A yellow box highlights the 'Google Sheets' option.

# 시각화 하는 다양한 방법

[ 스프레드시트 ]

## [ 스프레드시트를 열고 (date 기준으로 정렬한 후) 삽입 - 차트 ]

The screenshot shows a Google Sheets interface with a context menu open over a selected cell. The menu items are:

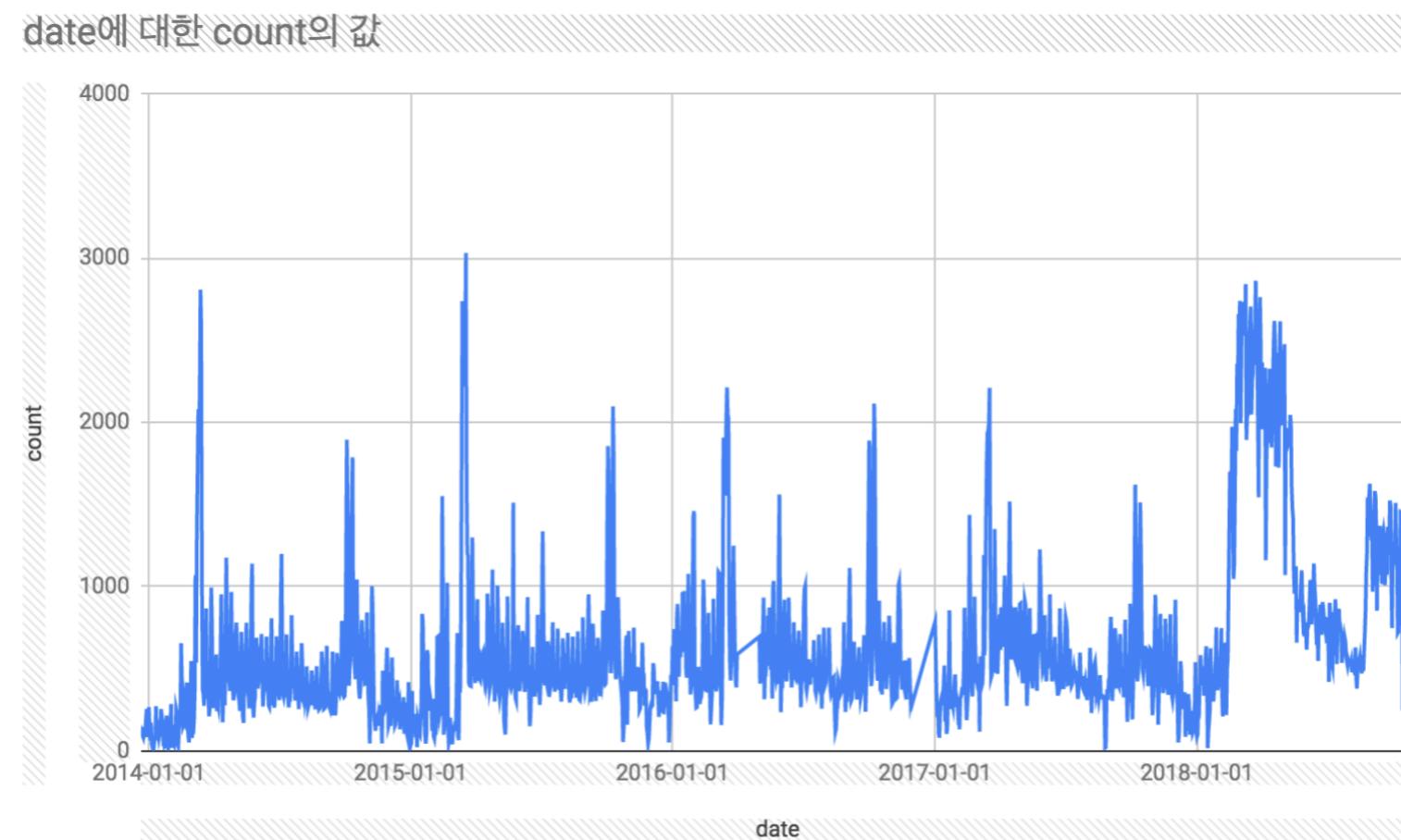
- 위로 1715행 삽입
- 아래로 1715행 삽입
- 왼쪽으로 2열 삽입
- 오른쪽에 2열 삽입
- 셀을 삽입하고 기존 셀을 아래로 이동
- 셀을 삽입하고 기존 셀을 오른쪽으로 이동
- 차트** (highlighted with an orange box)
- 이미지
- 그림...
- 설문지...
- 함수
- 링크... ⌘K
- 체크박스
- 댓글 ⌘+옵션+M
- 메모 Shift+F2
- 새 시트 Shift+F11

The main spreadsheet view shows a table with columns 'date' and 'count'. The date column contains dates from 2014-10-26 to 2015-10-17. The count column contains values from 1 to 1715. The cell containing '2014-10-27' is currently selected.

# 시각화 하는 다양한 방법

[ 스프레드시트 ]

[ 스프레드시트를 열고 (date 기준으로 정렬한 후) 삽입 - 차트 ]



# 시각화 하는 다양한 방법

[ Tableau ]

[ Tableau에서 Google BigQuery 클릭한 후, 브라우저에서 확인 클릭 ]

The screenshot shows the Tableau desktop application interface. On the left, the '연결' (Connections) pane lists various data sources, with 'Google BigQuery' selected. Below it, the '저장된 데이터 원본' (Saved Data Sources) pane shows 'Sample - APAC Superstore', 'Sample - Superstore', and 'World Indicators'. In the center, the '열기' (Open) pane displays three sample visualizations: 'Superstore' (a treemap), 'Regional' (a choropleth map of the US), and 'World Indicators' (a stacked bar chart). On the right, the '더 알아보기' (Learn More) pane provides links to education, resources, and forums. A separate browser window on the right shows a detailed player profile for Cristiano Ronaldo from the FIFA 19 database.

연결

파일로

- Microsoft Excel
- 텍스트 파일
- JSON 파일
- PDF 파일
- 공간 파일
- 통계 파일
- 자세히...

서버로

- Tableau Server
- MySQL
- Oracle
- Amazon Redshift
- Google BigQuery
- 자세히... >

저장된 데이터 원본

- Sample - APAC Superstore
- Sample - Superstore
- World Indicators

샘플 통합 문서

- Superstore
- Regional
- World Indicators

추가 샘플

더 알아보기

통합 문서 열기

▶ 교육

- 시작하기
- 데이터에 연결
- 시각적 분석
- Tableau 이해
- 더 많은 교육 동영상...

리소스

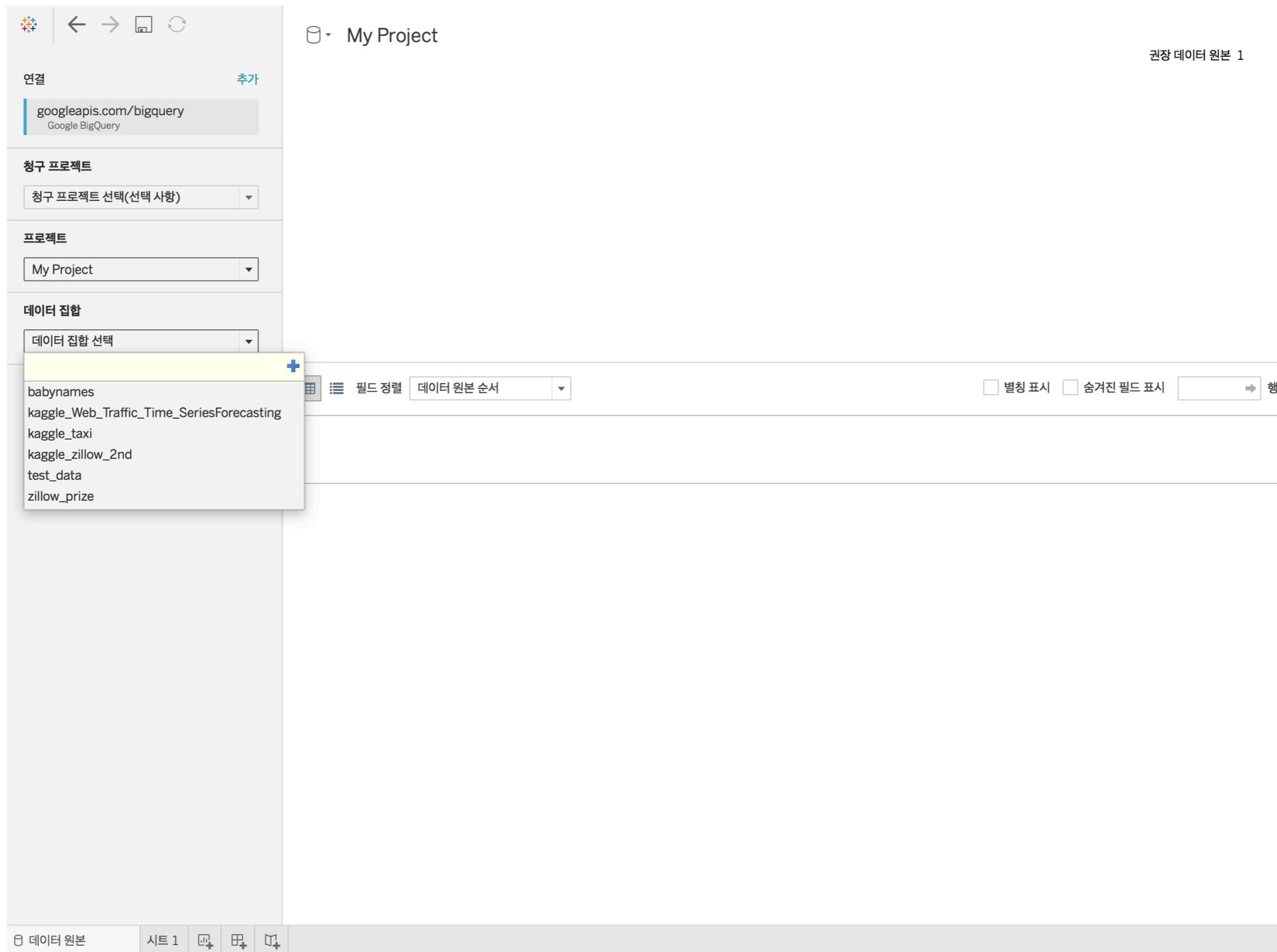
- Tableau Prep 받기
- 블로그 - 3 data governance considerations for deploying NLP in the enterprise
- Tableau Conference - 지금 등록
- 포럼

금주의 비주얼리제이션

Top FIFA 19 Players Ranked →

A detailed player profile for Cristiano Ronaldo from the FIFA 19 database. The profile includes his overall rating of 94, height of 6'2, weight of 77kg, and a circular radar chart showing his attributes across various football skills like Acceleration, Strength, Vision, Dribbling, Sprint Speed, Finishing, Heading Accuracy, Balance, and Shooting. His overall rating is highlighted in red at the top of the profile.

### [ 프로젝트, 데이터셋 선택 ]



# 시각화 하는 다양한 방법

[ Tableau ]

[ 데이터셋 선택한 후, 테이블 선택 또는 SQL 쿼리를 날릴 수 있음 ]

The screenshot shows the 'Data Source' configuration window in Tableau. On the left, there's a sidebar with sections for '연결' (Connection), '청구 프로젝트' (Billing Project), '프로젝트' (Project), '데이터 집합' (Data Source), and '테이블' (Tables). The 'Tables' section lists three tables: 'temp\_view', 'test\_data', 'train\_data', and a highlighted option '새 사용자 지정 SQL' (New Custom SQL) which is circled in orange. At the bottom of this sidebar is a checkbox for '레거시 SQL 사용' (Use Legacy SQL). The main panel on the right shows a preview area with a placeholder message '여기로 테이블 끌기' (Drag table here) and various filtering and sorting options at the top.

# 시각화 하는 다양한 방법

[ Tableau ]

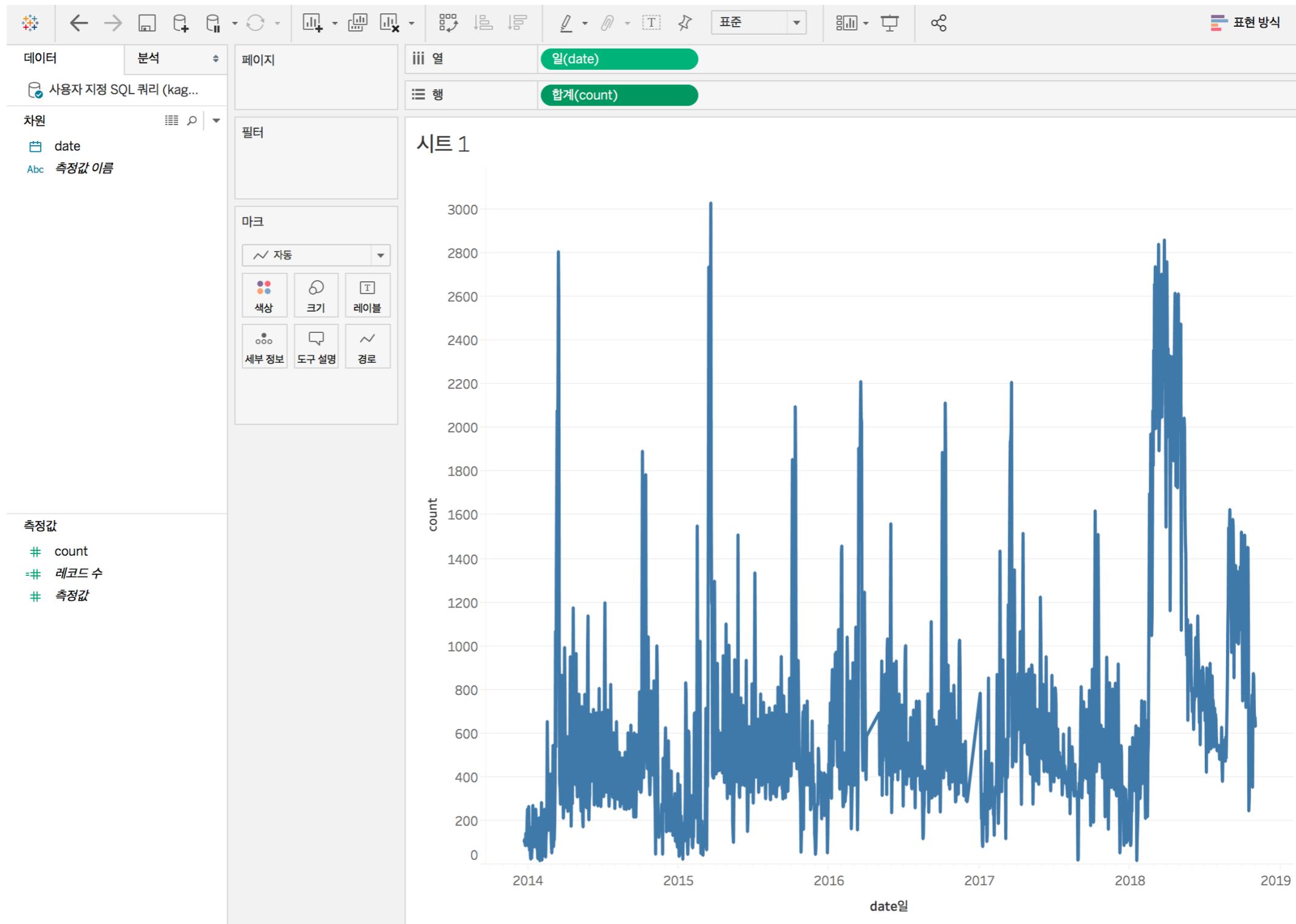
[ 데이터를 선택한 후, 시트1을 눌러 이동 ]

The screenshot shows the Tableau desktop application interface. The top navigation bar includes icons for file operations, search, and various visualization types like maps and charts. The left sidebar contains a '데이터' (Data) section with a connection to '사용자 지정 SQL 쿼리 (kag...)', a '차원' (Dimensions) section with 'date', and a '측정값' (Measures) section with 'count', '레코드 수' (Record Count), and '측정값' (Measure Value). The main workspace is titled '시트 1' (Sheet 1) and features three 'Drag and Drop' areas labeled '여기에 필드 놓기' (Drop field here) for rows, columns, and filters. A '마크' (Marks) shelf on the left provides options for different mark types: 색상 (Color), 크기 (Size), 텍스트 (Text), 세부 정보 (Detail), and 도구 설명 (Tool Tip).

# 시각화 하는 다양한 방법

[ Tableau ]

[ 클릭 3번만에 시각화한 자료 ]



이번 발표가 Tableau가 주제는 아니지만

추천 자료

- 김민지님의 [Youtube](#)
- Tableau 컨퍼런스 [Youtube](#)
- Youtube에 “tableau webinar”로 검색하면 다양한 자료가 존재

# 시각화 하는 다양한 방법

[ Python ]

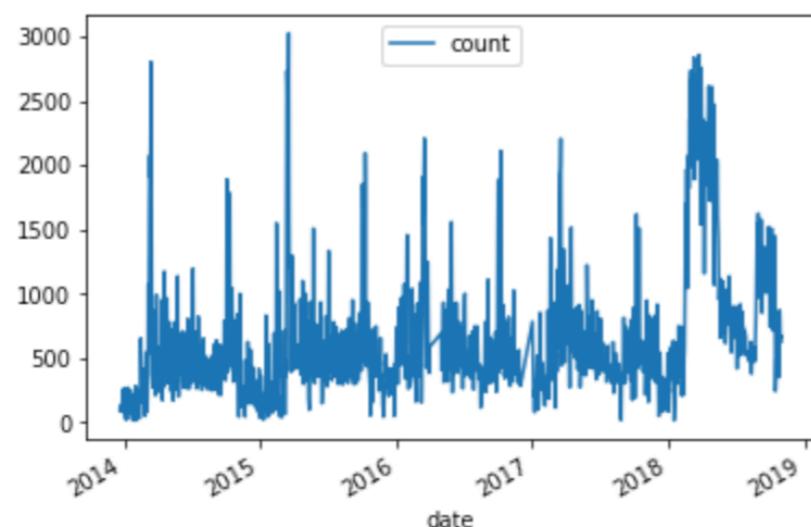
[ pd.read\_gbq 또는 pandas\_gbq 사용해서 데이터 추출 ]  
private\_key에 json key 설정 또는 GOOGLE\_APPLICATION\_CREDENTIALS 등록  
참고 링크(인증 시작하기)

```
In [1]: import pandas as pd
import pandas_gbq
import matplotlib.pyplot as plt
```

```
In [2]: query = """
SELECT DATE(start_time) as date, count(trip_id) as count
FROM `bigquery-public-data.austin_bikeshare.bikeshare_trips`
GROUP BY date
ORDER BY date
"""
```

```
In [3]: df = pd.read_gbq(query=query, dialect='standard', private_key='<your json key>')
```

```
In [4]: df.set_index('date').plot();
```



## Python에서 데이터 시각화하는 다양한 방법

<https://zzsza.github.io/development/2018/08/24/data-visualization-in-python/>

### pandas-gbq에서 인증(Authentication) 설정하기 :

GOOGLE\_APPLICATION\_CREDENTIALS 등록이 귀찮을 경우, 브라우저 로그인을 통해 권한 획득 가능

<https://zzsza.github.io/gcp/2019/03/17/pandas-gbq-auth/>

# 시각화 하는 다양한 방법

[ 데이터 스튜디오 ]

## [ 데이터 스튜디오에서 살펴보기 클릭 ]

The screenshot shows the BigQuery web interface. On the left, there's a sidebar with sections like '쿼리 기록', '저장된 쿼리', '작업 기록', '전송', '예약된 쿼리', 'BI Engine', and '리소스'. Under '리소스', two datasets are listed: 'noted-tide-156308' and 'bigquery-public-data'. The main area displays a query in the '쿼리 편집기' tab:

```
1 SELECT DATE(start_time) as date, count(trip_id) as count
2 FROM `bigquery-public-data.austin_bikeshare.bikeshare_trips`
3 GROUP BY date
```

Below the query, there are several buttons: '실행', '쿼리 저장', '보기 저장', '쿼리 예약', and '더보기'. A dropdown menu labeled '결과 저장' is open, showing options: '데이터 스튜디오에서 살펴보기' (highlighted with an orange box), 'CSV(Google 드라이브)', 'JSON(Google 드라이브)', 'BigQuery 테이블', and 'Google 스프레드시트'. The '데이터 스튜디오에서 살펴보기' option is described as saving the results to Google Drive for Data Studio. To the right of the dropdown, a preview table shows data from the query:

행	date	count
1	2014-10-26	698
2	2014-10-27	494
3	2017-01-29	465
4	2014-10-28	482
5	2014-10-29	409
6	2014-10-30	412
7	2014-10-31	538
8	2015-10-01	576
9	2015-10-02	1375
10	2015-10-03	1856
11	2015-10-04	1642
12	2015-10-05	643
13	2015-10-06	589
14	2015-10-07	473
15	2015-10-08	650

# 실습 (2)

# 실습 (2)

데이터 설명 : bikeshare trip data  
**bikeshare\_trips Table** ( 지금은 이거만 사용해요 )

bikeshare\_trips

퀵 맵 테이블 쿼리 테이블 복사 테이블 삭제 내보내기 ▾

행	trip_id	subscriber_type	bikeid	start_time	start_station_id	start_station_name	end_st
1	9900285987	24-Hour Kiosk (Austin B-cycle)	446	2014-10-26 15:12:00 UTC	2712	Toomey Rd @ South Lamar	
2	9900285989	24-Hour Kiosk (Austin B-cycle)	203	2014-10-26 15:12:00 UTC	2712	Toomey Rd @ South Lamar	
3	9900285991	24-Hour Kiosk (Austin B-cycle)	101	2014-10-26 15:12:00 UTC	2712	Toomey Rd @ South Lamar	
4	9900286140	24-Hour Kiosk (Austin B-cycle)	242	2014-10-26 18:12:00 UTC	2541	State Capitol @ 14th & Colorado	
5	9900286143	24-Hour Kiosk (Austin B-cycle)	924	2014-10-26 18:12:00 UTC	2541	State Capitol @ 14th & Colorado	
6	9900286214	Annual Membership (Austin B-cycle)	24	2014-10-26 20:12:00 UTC	2712	Toomey Rd @ South Lamar	
7	9900286338	24-Hour Kiosk (Austin B-cycle)	101	2014-10-27 10:12:00 UTC	2712	Toomey Rd @ South Lamar	
8	13575843	Walk Up	302	2017-01-29 16:42:52 UTC	3464	Pease Park	
9	9900286942	24-Hour Kiosk (Austin B-cycle)	660	2014-10-28 14:12:00 UTC	2712	Toomey Rd @ South Lamar	
10	9900287148	24-Hour Kiosk (Austin B-cycle)	428	2014-10-28 19:12:00 UTC	2712	Toomey Rd @ South Lamar	

## 실습 (2)

---

#1. 몇 시에 trip 시작하는 사용자가 많을까요?

#1-1. hour만 추출해서 COUNT하는 쿼리

#1-2. datetime에서 시간까지만 자르고 COUNT하는 쿼리

#1-3. 2017년 1월 1일부터 시간대별 추세를 구하는 쿼리

#2. 무슨 요일에 제일 사용자가 많을까요?

#2-1. 요일별 시간대 패턴이 다를까요?

#2-2. 요일마다 제일 count가 많은 시간대는 어떻게 구해야 할까요?

#3. 제일 많이 운행한 bikeid는 무엇일까요?

#4. 제일 오래 여행한 여행의 trip\_id는?

#5. 제일 오래 여행한 여행은 몇시간 여행했나요?

## 실습 (2)

---

#1. 몇 시에 trip 시작하는 사용자가 많을까요?

#1-1. hour만 추출해서 COUNT하는 쿼리

#1-2. datetime에서 시간까지만 자르고 COUNT하는 쿼리

#1-3. 2017년 1월 1일부터 시간대별 추세를 구하는 쿼리

```
SELECT start_hour, count(trip_id) as count
FROM (
    SELECT EXTRACT(hour FROM start_time) as start_hour, *
    FROM `bigquery-public-data.austin_bikeshare.bikeshare_trips`
)
group by start_hour
order by count desc
```

문제 출제 의도 : EXTRACT(hour FROM ~) 를 사용해 hour를 추출

## 실습 (2)

---

#1. 몇 시에 trip 시작하는 사용자가 많을까요?

#1-1. hour만 추출해서 COUNT하는 쿼리

**#1-2. datetime에서 시간까지만 자르고 COUNT하는 쿼리**

#1-3. 2017년 1월 1일부터 시간대별 추세를 구하는 쿼리

```
SELECT start_hour, count(trip_id) as count
```

```
FROM (
```

```
    SELECT DATETIME_TRUNC(DATETIME(start_time), hour) as start_hour, *
```

```
    FROM `bigquery-public-data.austin_bikeshare.bikeshare_trips`
```

```
)
```

```
group by start_hour
```

```
order by count desc
```

문제 출제 의도 : DATETIME\_TRUNC(Column, hour) 를 사용해 hour를 추출

1-1과 차이는 Datetime까지 추가된 것

1-2는 일자별 시간대별 추세를 볼 때 사용하고 1-1은 단순 시간대의 값을 알 수 있음

## 실습 (2)

---

#1. 몇 시에 trip 시작하는 사용자가 많을까요?

#1-1. hour만 추출해서 COUNT하는 쿼리

#1-2. datetime에서 시간까지만 자르고 COUNT하는 쿼리

#1-3. 2017년 1월 1일부터 시간대별 추세를 구하는 쿼리

```
SELECT start_hour, count(trip_id) as count
```

```
FROM (
```

```
    SELECT DATETIME_TRUNC(DATETIME(start_time), hour) as start_hour, *
```

```
    FROM `bigquery-public-data.austin_bikeshare.bikeshare_trips`
```

```
    where date(start_time) >= '2017-01-01'
```

```
)
```

```
group by start_hour
```

```
order by start_hour
```

문제 출제 의도 : WHERE 조건을 걸어본다

## 실습 (2)

---

#2. 무슨 요일에 제일 사용자가 많을까요?

#2-1. 요일별 시간대 패턴이 다를까요?

#2-2. 요일마다 제일 count가 많은 시간대는 어떻게 구해야 할까요?

```
SELECT weekday, count(trip_id) as count
FROM (
    SELECT DATETIME_TRUNC(DATETIME(start_time), hour) as start_hour,
    FORMAT_DATETIME("%u", DATETIME(start_time)) AS weekday,
    *
    FROM `bigquery-public-data.austin_bikeshare.bikeshare_trips`
)
group by weekday
order by count desc
```

문제 출제 의도 : FORMAT\_DATETIME("%u" Column)으로曜일을 추출

## 실습 (2)

---

#2. 무슨 요일에 제일 사용자가 많을까요?

**#2-1. 요일별 시간대 패턴이 다를까요?**

#2-2. 요일마다 제일 count가 많은 시간대는 어떻게 구해야 할까요?

```
SELECT weekday, start_hour, count(trip_id) as count
FROM (
    SELECT EXTRACT(hour from start_time) as start_hour,
    FORMAT_DATETIME("%u", DATETIME(start_time)) AS weekday,
    *
    FROM `bigquery-public-data.austin_bikeshare.bikeshare_trips`
)
group by weekday, start_hour
order by count desc
```

문제 출제 의도 : 앞의 쿼리에 start\_hour만 추가. 질문이 추상적으로 주어질 때 어떻게 쿼리를 날리는지 생각해볼 수 있도록 출제함

## 실습 (2)

---

#3. 제일 많이 운행한 bikeid는 무엇일까요?

```
SELECT bikeid, count(trip_id) as count
FROM `bigquery-public-data.austin_bikeshare.bikeshare_trips`
GROUP BY bikeid
ORDER BY 2 DESC
LIMIT 1
```

문제 출제 의도 : ORDER BY => 2번째 컬럼인 count라는 것을 알려드리기 위해  
출제

## 실습 (2)

---

#4. 제일 오래 여행한 여행의 trip\_id는?

```
SELECT MAX(duration_minutes) AS max_duration_minutes  
FROM `bigquery-public-data.austin_bikeshare.bikeshare_trips`
```

# 21296

```
SELECT *  
FROM `bigquery-public-data.austin_bikeshare.bikeshare_trips`  
where duration_minutes = 21296
```

문제 출제 의도 : 오래 여행한이란 조건의 값을 뽑기 위해 어떻게 해야하는지  
고민을 위함

+ 이런 경우엔 뒤에 나오는 Analytics Function이 유용

## 실습 (2)

---

#4. 제일 오래 여행한 여행의 trip\_id는? (다른 풀이)

```
SELECT *
FROM `bigquery-public-data.austin_bikeshare.bikeshare_trips`
ORDER BY duration_minutes DESC
LIMIT 1
```

문제 출제 의도 : 그냥 ORDER BY를 잘 쓰면 한방에 될 수도 있다는 점을 전달하기 위해

## 실습 (2)

---

#5. 제일 오래 여행한 여행은 몇시간 여행했나요? : FLOAT -> INT

```
SELECT *, CAST((duration_minutes/60/24) AS INT64) as duration_hour  
FROM `bigquery-public-data.austin_bikeshare.bikeshare_trips`  
ORDER BY duration_minutes DESC  
LIMIT 1
```

문제 출제 의도 : INT64로 변환하면 소수점이 사라지는 것을 알려주기 위해

## 실습 (2)

---

#5. 제일 오래 여행한 여행은 몇시간 여행했나요? : ROUND

```
SELECT *, ROUND((duration_minutes/60/24)) as duration_hour  
FROM `bigquery-public-data.austin_bikeshare.bikeshare_trips`  
ORDER BY duration_minutes DESC  
LIMIT 1
```

문제 출제 의도 : ROUND로 반올림이 가능한 것을 알려드리기 위해

# BigQuery 심화 지식

# 자료형

---

## 데이터 유형, 자료형(Type)

알아야 하는 이유

1. "1"과 1은 다른
2. "2019-05-15"와 DATE "2019-05-15"와 다른
3. 배열(ARRAY)과 구조체(STRUCT)를 잘 이해하면 사실상 BigQuery 거의 마스터

# 자료형

---

# 숫자형  
INT64 : 정수  
ex) 1

# 부동 소수점  
FLOAT64 : 소수점  
ex) 3.14

# 부울 유형  
BOOL : 참, 거짓  
ex) TRUE, FALSE

# 문자열  
STRING

# 지리  
GEOGRAPHY

# 날짜  
DATE

# 날짜/시간  
DATETIME  
  
# 시간  
TIME : 시간

# 타임스탬프  
TIMESTAMP : 마이크로초 단위 정밀도를 갖는 절대 시점

# 배열  
ARRAY : 뒤에서 계속

# 구조체  
STRUCT : 뒤에서 계속

# 자료형

[ 배열(ARRAY) ]

---

배열은 하나의 변수에 여러 값들을 저장하는 자료형

다양한 개발 언어에서 배열이 존재하며, 하나의 값에 여러 값을 넣을 수 있는 장점

문법 :

1. 괄호를 사용

ex) [1, 2, 3]

2. 특정 데이터 유형을 선언할 경우 꺽쇠 괄호 사용

ex) ARRAY<INT64>[1, 2, 4]

SELECT

[1, 2, 3] as numbers,

ARRAY<INT64>[1,2,4,9,10] as numbers2

**GENERATE\_DATE\_ARRAY()** : Date 값 집합을 생성할 수 있음

```
SELECT  
  GENERATE_DATE_ARRAY('2017-11-21', '2017-12-31', INTERVAL 1 WEEK)  
  AS date_array
```

```
+-----+  
| date_array |  
+-----+  
| [2017-11-21, 2017-11-28, 2017-12-05, 2017-12-12, 2017-12-19, 2017-12-26] |  
+-----+
```

### ARRAY\_LENGTH() : ARRAY 안에 있는 요소 count

```
SELECT some_numbers,
       ARRAY_LENGTH(some_numbers) AS len
  FROM (SELECT [0, 1, 1, 2, 3, 5] AS some_numbers)
```

some_numbers	len
[0, 1, 1, 2, 3, 5]	6

## UNNEST + WITH OFFSET : 오프셋(순서) 나옴

```
SELECT *
FROM UNNEST(['foo', 'bar', 'baz', 'qux', 'corge', 'garply', 'waldo', 'fred'])
  AS element
WITH OFFSET AS offset
ORDER BY offset
```

element	offset
foo	0
bar	1
baz	2
qux	3
corge	4
garply	5
waldo	6
fred	7

**ARRAY[OFFSET(N)]** : ARRAY에서 N번째 요소(0부터 시작)

SELECT

```
["apple", "pear", "orange"] AS fruit,  
["apple", "pear", "hi"][OFFSET(0)] AS one
```

**ARRAY[ORDINAL(N)]** : ARRAY에서 N번째 요소(1부터 시작)

SELECT

```
["apple", "pear", "orange"] AS fruit,  
["apple", "pear", "hi"][ORDINAL(2)] AS one
```

---

WITH sequences AS

```
(SELECT [0, 1, 1, 2, 3, 5] AS some_numbers  
UNION ALL SELECT [2, 4, 8, 16, 32] AS some_numbers  
UNION ALL SELECT [5, 10] AS some_numbers)
```

SELECT some\_numbers,

```
some_numbers[OFFSET(1)] AS offset_1,  
some_numbers[ORDINAL(1)] AS ordinal_1
```

FROM sequences;

[ 그 외 함수들은 아래 링크 참고! ]

# 자료형

[ 배열(ARRAY)의 함수 ]

**UNNEST()** : ARRAY 안에 있는 요소를 펼치고 싶은 경우

```
SELECT *
FROM UNNEST(['foo', 'bar', 'baz', 'qux', 'corge', 'garply', 'waldo', 'fred'])
AS element
```

비교를 위해 아래 쿼리를 실행해보세요!

```
SELECT ['foo', 'bar', 'baz', 'qux', 'corge', 'garply', 'waldo', 'fred']
```

Row	element
1	foo
2	bar
3	baz
4	qux
5	corge
6	garply
7	waldo
8	fred

Row	element
1	foo
	bar
	baz
	qux
	corge
	garply
	waldo
	fred

# 자료형

[ 배열(ARRAY)의 함수 ]

**UNNEST()** : ARRAY 안에 있는 요소를 펼치고 싶은 경우

```
SELECT *
FROM UNNEST(['foo', 'bar', 'baz', 'qux', 'corge', 'garply', 'waldo', 'fred'])
AS element
```

비교를 위해 아래 쿼리를 실행해보세요!

```
SELECT ['foo', 'bar', 'baz', 'qux', 'corge', 'garply', 'waldo', 'fred']
```

Row	element
1	foo
2	bar
3	baz
4	qux
5	corge
6	garply
7	waldo
8	fred

좌측은 각 행마다 값이 있는 형태,  
우측은 하나의 행에 여러 값이 있는 형태

Row	element
1	foo
	bar
	baz
	qux
	corge
	garply
	waldo
	fred

**STRUCT()** : 순서가 있고 이름으로(선택) 필드에 접근할 수 있는 컨테이너  
Container of ordered fields each with a type (required) and field name (optional)

ARRAY와 자주 쓰임

꺾쇠 괄호를 사용해 정의

STRUCT<INT64, STRING> : 필드 이름이 없는 경우

STRUCT<id INT64, name STRING> : 필드 이름이 있는 경우

ARRAY는 이름으로 접근하지 않고 OFFSET(순서)로 접근

ARRAY[offset(0)]

(Python의 List와 유사)

**STRUCT()** : 순서가 있고 이름으로(선택) 필드에 접근할 수 있는 컨테이너

STRUCT<INT64, STRING> : 필드 이름이 없는 경우

STRUCT<id INT64, name STRING> : 필드 이름이 있는 경우

WITH locations AS

(SELECT STRUCT("Seattle" AS city, "Washington" AS state) AS location

UNION ALL

SELECT STRUCT("Phoenix" AS city, "Arizona" AS state) AS location)

SELECT l.location.\*

FROM locations as l

**STRUCT()** : 순서가 있고 이름으로(선택) 필드에 접근할 수 있는 컨테이너

STRUCT<INT64, STRING> : 필드 이름이 없는 경우

STRUCT<id INT64, name STRING> : 필드 이름이 있는 경우

```
SELECT *, struct_value
FROM UNNEST(ARRAY<STRUCT<x INT64, y STRING>>[(1, 'foo'), (3, 'bar')]) AS struct_value;
```

STRUCT<INT64>

이름이 없는 INT64 정수 필드가 하나 있는 단순 STRUCT

STRUCT<x STRUCT<y INT64,z INT64>>

안에 x라는 이름의 중첩된 STRUCT가 있는 STRUCT  
STRUCT x에는 y와 z, 두 개의 필드가 있고, 두 필드 모두 INT64

STRUCT<inner\_array ARRAY<INT64>>

64비트 정수 요소를 갖는 inner\_array라는 ARRAY가 포함된 STRUCT

# 자료형 바꾸기 [ CAST ]

---

**CAST(column AS type)** : column을 Type으로 변경

# WITH

---

아까 봤던 쿼리

```
SELECT some_numbers,  
       ARRAY_LENGTH(some_numbers) AS len  
  FROM (SELECT [0, 1, 1, 2, 3, 5] AS some_numbers)
```

# WITH

---

아까 봤던 쿼리  
쿼리의 가독성 증가  
numbers를 여러번 사용할 수도 있음

```
SELECT some_numbers,  
       ARRAY_LENGTH(some_numbers) AS len  
  FROM (SELECT [0, 1, 1, 2, 3, 5] AS some_numbers)
```

```
WITH numbers AS (SELECT [0, 1, 1, 2, 3, 5] AS some_numbers)  
SELECT some_numbers,  
       ARRAY_LENGTH(some_numbers) AS len  
  FROM numbers
```

# VIEW

---

VIEW는 SQL 쿼리로 정의하는 가상 테이블

자주 사용하는 쿼리 구문 템플릿이 있으면 VIEW로 저장해 활용 가능

속도가 느린 편

# VIEW

## [ 보기(View) 저장 ]

The screenshot shows the BigQuery web interface. On the left, there's a sidebar with links like 'BigQuery', '기능 및 정보', '단축키', '새 쿼리 작성', '쿼리 기록', '저장된 쿼리', '작업 기록', '전송', '예약된 쿼리', 'BI Engine', and '리소스'. Below '리소스' is a search bar with '표 및 데이터세트 검색' and a dropdown menu '데이터 추가 ▾'. The main area is titled '쿼리 편집기' and contains the following SQL code:

```
1 WITH numbers AS (SELECT [0, 1, 1, 2, 3, 5] AS some_numbers)
2
3 SELECT some_numbers,
4       ARRAY_LENGTH(some_numbers) AS len
5 FROM numbers
```

Below the code are several buttons: '실행', '쿼리 저장', '보기 저장' (which is highlighted with an orange box), '쿼리 예약', and '더보기'. A green checkmark indicates '실행 시 이 쿼리가 OB를 처리합니다.' To the right of the buttons is a section titled '쿼리 결과' with tabs for '결과' (selected), 'JSON', and '실행 세부정보'. The results table shows the following data:

행	some_numbers	len
1	0	6
	1	
	1	
	2	
	3	
	5	

<https://cloud.google.com/bigquery/docs/views-intro>

# VIEW

## [ 보기(View) 저장 ]

BigQuery 기능 및 정보 단축키 + 새 쿼리 작성

쿼리 기록  
저장된 쿼리  
작업 기록  
전송  
예약된 쿼리  
BI Engine  
리소스 표 및 데이터세트 검색

보기 편집기

```
1 WITH numbers AS (SELECT [0, 1, 1, 2, 3, 5] AS some_numbers)
2
3 SELECT some_numbers,
```

편집기 숨기기 전체 화면

보기 저장

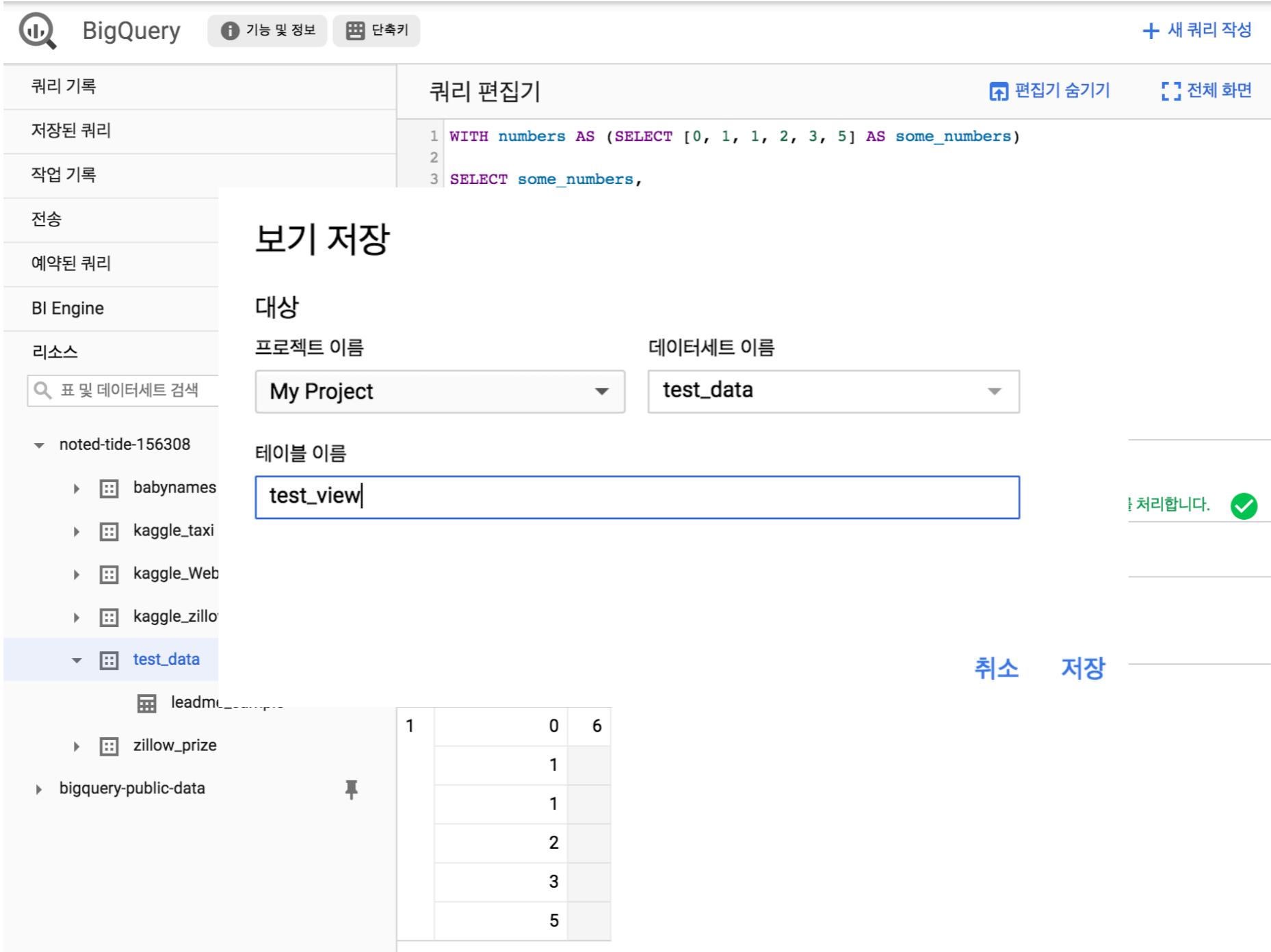
대상

프로젝트 이름: My Project      데이터세트 이름: test\_data

테이블 이름: test\_view 처리합니다. ✓

취소      저장

1 0 6  
1  
1  
2  
3  
5



<https://cloud.google.com/bigquery/docs/views-intro>

# VIEW

## [ View 생성 ]

The screenshot shows the Google BigQuery web interface. On the left, the sidebar navigation includes '쿼리 기록', '저장된 쿼리' (selected), '작업 기록', '전송', '예약된 쿼리', 'BI Engine', and '리소스'. Below '리소스' is a search bar for '표 및 데이터세트 검색'. The main area displays a query titled 'noted-tide-156308:test\_data.test\_view' with the following SQL code:

```
1 WITH numbers AS (SELECT [0, 1, 1, 2, 3, 5] AS some_numbers)
2
3 SELECT some_numbers,
4       ARRAY_LENGTH(some_numbers) AS len
5 FROM numbers
```

Below the code are execution controls: '실행' (Execute), '쿼리 저장' (Save Query), '보기 저장' (Save View), '쿼리 예약' (Schedule Query), and '더보기' (More). A note says '실행 시 이 쿼리가 0B를 처리합니다.' with a checkmark. The '쿼리 결과' (Query Results) section shows the results of the executed query:

쿼리 완료(0.9초 경과, 0B 처리됨)

작업 정보    [결과](#)    [JSON](#)    [실행 세부정보](#)

행	some_numbers	len
1	0	6
	1	
	1	
	2	
	3	
	5	

<https://cloud.google.com/bigquery/docs/views-intro>

# VIEW

[ View 세부정보에서 쿼리 확인 가능, 쿼리 수정도 가능 ]

The screenshot shows the Google Cloud BigQuery interface for a dataset named 'noted-tide-156308'. A specific view, 'test\_view', is selected and highlighted with a blue background. The top navigation bar includes buttons for '실행' (Execute), '쿼리 저장장' (Query Storage), '보기 저장장' (View Storage), '쿼리 예약' (Query Reservation), and '더보기' (More). A note indicates that the query will process 0B when executed. Below the navigation, the view name 'test\_view' is displayed along with tabs for '쿼리 보기' (Query View), '보기 복사' (View Copy), '보기 삭제' (View Delete), and '내보내기' (Export). The '세부정보' (Details) tab is selected and highlighted with an orange box. The '설명' (Description) field is empty, and the '라벨' (Label) field is also empty. The '정보 보기' (Information View) section provides metadata: ID is 'noted-tide-156308:test\_data.test\_view', creation time is '2019. 5. 17. 오전 1:18:17', and last modified time is '2019. 5. 17. 오전 1:18:17'. It also states that it is not materialized and does not use inline SQL. The '쿼리' (Query) section contains the following SQL code:

```
1 WITH numbers AS (SELECT [0, 1, 1, 2, 3, 5] AS some_numbers)
2
3 SELECT some_numbers,
4       ARRAY_LENGTH(some_numbers) AS len
5 FROM numbers
```

<https://cloud.google.com/bigquery/docs/views-intro>

# Analytic Function(Window Function)

---

## [ Analytic Function란? ]

Aggregation Function(집계 함수)는 행 그룹에 대해 단일 집계값을 반환  
(행 그룹 : Group by 한 컬럼)

반면 Analytic Function은 각 행마다 단일 값을 반환  
(각 행마다 값 반환)

### 종류

1. 탐색 함수 : LEAD, LAG, FIRST\_VALUE, LAST\_VALUE
2. 번호 지정 함수 : RANK, DENSE\_RANK, PERCENT\_RANK, CUME\_DIST, NTILE
3. 집계 분석 함수 : 집계 함수들, AVG, COUNT, SUM, MAX, MIN ...

# Analytic Function(Window Function)

---

데이터 분석하다 궁금해지는 것들

"A라는 유저의 전 주문 수량은 얼마나 될까?"

"B라는 유저가 물건 구입하기 전에 몇번이나 상세보기를 했을까?"

"고객의 구매 시점에 총 누적 구매 횟수는?"

"C라는 유저의 이전/다음 앱 접속시간이 어떻게 될까?"

"랭킹, 누적합을 알고싶다"



이런 경우 모두 **Analytic Function**을 사용하면 삶이 윤택

# Analytic Function(Window Function) [ LEAD ]

---

LEAD : 후속 행의 값 반환

user_id	visit_month
1004	1
1004	3
1004	7
1004	8
2112	3
2112	6
2112	7
3912	4

# Analytic Function(Window Function) [ LEAD ]

---

[ next\_visit\_month를 구하고 싶다면? ]

user_id	visit_month	next_visit_month
1004	1	
1004	3	
1004	7	
1004	8	
2112	3	
2112	6	
2112	7	
3912	4	

# Analytic Function(Window Function) [ LEAD ]

[ 예상 Output ]

user_id	visit_month	next_visit_month
1004	1	3
1004	3	7
	7	8
	8	null
2112	3	6
	6	7
	7	null
3912	4	null

# Analytic Function(Window Function) [ LEAD ]

---

[ Query ]

```
SELECT
    user_id,
    visit_month, [ 다음 값 ]
    LEAD(visit_month) OVER (PARTITION BY user_id) AS next_visit_month
FROM `table`
```

[ user\_id로 파티션 ]

# Analytic Function(Window Function) [ LEAD ]

user_id	visit_month
1004	1
1004	3
1004	7
1004	8
2112	3
2112	6
2112	7
3912	4

[ Partition by user\_id ]

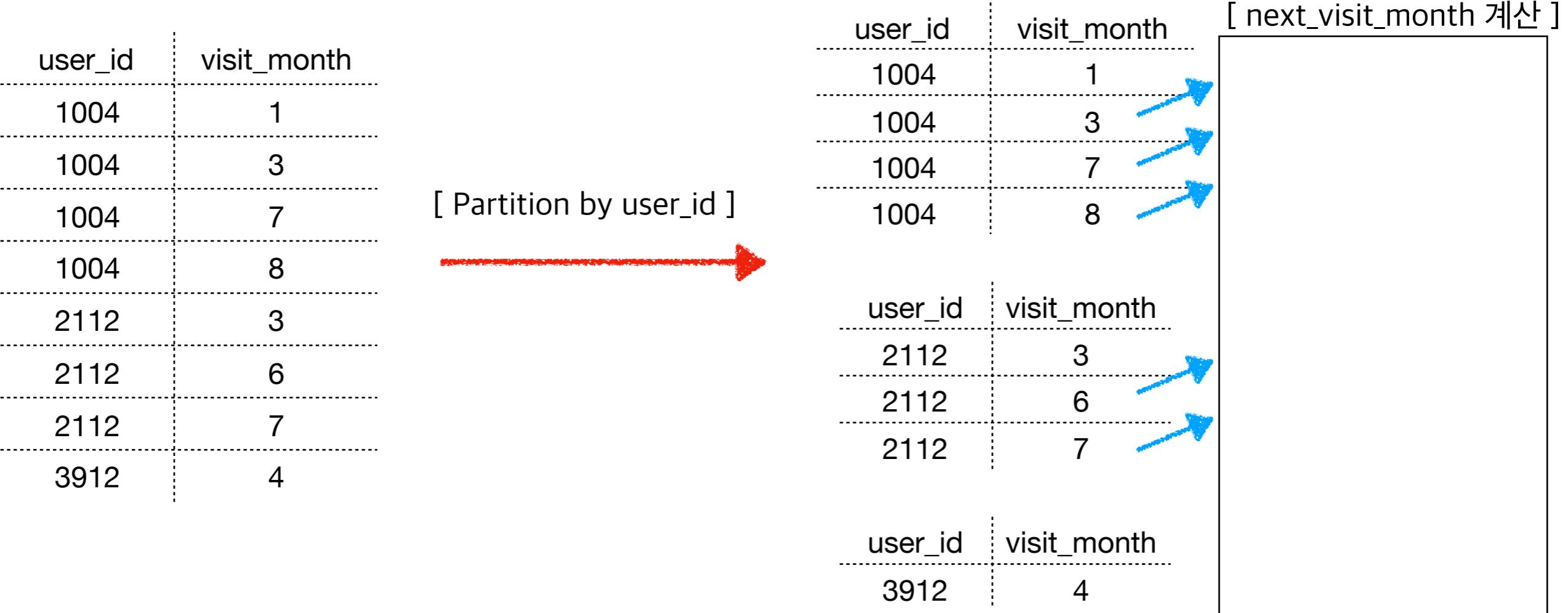


user_id	visit_month
1004	1
1004	3
1004	7
1004	8

user_id	visit_month
2112	3
2112	6
2112	7

user_id	visit_month
3912	4

# Analytic Function(Window Function) [ LEAD ]



# Analytic Function(Window Function) [ LEAD ]

user_id	visit_month
1004	1
1004	3
1004	7
1004	8
2112	3
2112	6
2112	7
3912	4

[ Partition by user\_id ]

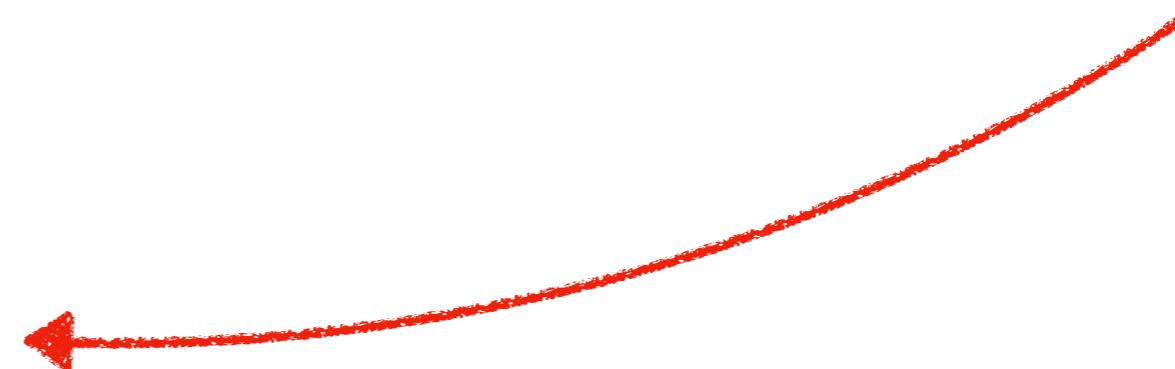


user_id	visit_month
1004	1
1004	3
1004	7
1004	8
2112	3
2112	6
2112	7
3912	4

[ next\_visit\_month 계산 ]

user_id	visit_month
1004	1
1004	3
1004	7
1004	8
2112	3
2112	6
2112	7
3912	4

user_id	visit_month	next_visit_month
1004	1	3
1004	3	7
1004	7	8
1004	8	null
2112	3	6
2112	6	7
2112	7	null
3912	4	null



# Analytic Function(Window Function) [ LEAD ]

---

[ next\_two\_visit\_month를 알고 싶다면? ]

user_id	visit_month	next_visit_month	next_two_visit_month
1004	1	3	7
1004	3	7	8
1004	7	8	null
1004	8	null	null
2112	3	6	7
2112	6	7	null
2112	7	null	null
3912	4	null	null

# Analytic Function(Window Function) [ LEAD ]

---

[ Query ]

```
SELECT
    user_id,
    visit_month,
    LEAD(visit_month, 2) OVER (PARTITION BY user_id) AS next_two_visit_month
FROM `table` [ 2번째 뒤 값 ]
```

# Analytic Function(Window Function) [ LAG ]

---

LAG : 전 행의 값 반환

user_id	visit_month
1004	1
1004	3
1004	7
1004	8
2112	3
2112	6
2112	7
3912	4

# Analytic Function(Window Function) [ LAG ]

---

[ before\_visit\_month를 구하고 싶다면? ]

user_id	visit_month	before_visit_month
1004	1	
1004	3	
1004	7	
1004	8	
2112	3	
2112	6	
2112	7	
3912	4	

# Analytic Function(Window Function) [ LAG ]

[ before\_visit\_month를 구하고 싶다면? ]

user_id	visit_month	before_visit_month
1004	1	null
1004	3	1
	7	3
	8	7
2112	3	null
2112	6	3
	7	6
3912	4	null

# Analytic Function(Window Function) [ LAG ]

---

[ Query ]

```
SELECT
    user_id,
    visit_month, [ 이전 값 ]
    LAG(visit_month) OVER (PARTITION BY user_id) AS before_visit_month
FROM `table`                                [ user_id로 파티션 ]
```

# Analytic Function(Window Function) [ RANK ]

---

# 분기별, 팀별, 쿼리 수 랭킹 1위

```
WITH temp AS (
    SELECT '1분기' AS quarter, '데이터팀' as team, 'kyle' as user_name, 10 as query_count
    UNION ALL
    SELECT '2분기' AS quarter, '데이터팀' as team, 'kyle' as user_name, 30 as query_count
    UNION ALL
    SELECT '1분기' AS quarter, '머신러닝팀' as team, 'matthew' as user_name, 1 as query_count
    UNION ALL
    SELECT '2분기' AS quarter, '머신러닝팀' as team, 'matthew' as user_name, 40 as query_count
    UNION ALL
    SELECT '1분기' AS quarter, '데이터팀' as team, 'stark' as user_name, 50 as query_count
    UNION ALL
    SELECT '1분기' AS quarter, '마케팅팀' as team, 'thor' as user_name, 30 as query_count
    UNION ALL
    SELECT '2분기' AS quarter, '데이터팀' as team, 'thor' as user_name, 90 as query_count
)
```

```
SELECT *
FROM temp
```

quarter	team	user_name	query_count
1분기	데이터팀	kyle	10
2분기	데이터팀	kyle	30
1분기	머신러닝팀	matthew	1
2분기	머신러닝팀	matthew	40
1분기	데이터팀	stark	50
1분기	마케팅팀	thor	30
2분기	데이터팀	thor	90

# Analytic Function(Window Function) [ RANK ]

---

[ 쿼터별 팀별 쿼리를 많이 날린 사람은? ]

[ 예상 Output ]

quarter	team	user_name	query_count	quarter_rank
1분기	데이터팀	stark	50	1
1분기	데이터팀	kyle	10	2
1분기	마케팅팀	thor	30	1
1분기	머신러닝팀	matthew	1	1
2분기	데이터팀	thor	90	1
2분기	데이터팀	kyle	30	2
2분기	머신러닝팀	matthew	40	1

# Analytic Function(Window Function) [ RANK ]

---

[ 쿼터별 팀별 쿼리를 많이 날린 사람은? ]

# 분기별, 팀별, 쿼리 수 랭킹 1위

WITH temp AS (

```
SELECT '1분기' AS quarter, '데이터팀' as team, 'kyle' as user_name, 10 as query_count  
UNION ALL
```

```
SELECT '2분기' AS quarter, '데이터팀' as team, 'kyle' as user_name, 30 as query_count  
UNION ALL
```

```
SELECT '1분기' AS quarter, '머신러닝팀' as team, 'matthew' as user_name, 1 as query_count  
UNION ALL
```

```
SELECT '2분기' AS quarter, '머신러닝팀' as team, 'matthew' as user_name, 40 as query_count  
UNION ALL
```

```
SELECT '1분기' AS quarter, '데이터팀' as team, 'stark' as user_name, 50 as query_count  
UNION ALL
```

```
SELECT '1분기' AS quarter, '마케팅팀' as team, 'thor' as user_name, 30 as query_count  
UNION ALL
```

```
SELECT '2분기' AS quarter, '데이터팀' as team, 'thor' as user_name, 90 as query_count  
)
```

SELECT

```
quarter, team, user_name, query_count,  
RANK() OVER (PARTITION BY quarter, team ORDER BY query_count DESC) as quarter_rank  
FROM temp
```

quarter	team	user_name	query_count	quarter_rank
1분기	데이터팀	stark	50	1
1분기	데이터팀	kyle	10	2
1분기	마케팅팀	thor	30	1
1분기	머신러닝팀	matthew	1	1
2분기	데이터팀	thor	90	1
2분기	데이터팀	kyle	30	2
2분기	머신러닝팀	matthew	40	1

# Analytic Function(Window Function) [ RANK ]

---

[ 기간 상관없이 쿼리 랭킹은? ]

[ 예상 Output ]

quarter	team	user_name	query_count	quarter_rank	total_rank
2분기	데이터팀	thor	90	1	1
1분기	데이터팀	stark	50	1	2
2분기	머신러닝팀	matthew	40	1	3
1분기	마케팅팀	thor	30	1	4
2분기	데이터팀	kyle	30	2	4
1분기	데이터팀	kyle	10	2	6
1분기	머신러닝팀	matthew	1	1	7

# Analytic Function(Window Function) [ RANK ]

---

[ 기간 상관없이 쿼리 랭킹은? ]

# 분기별, 팀별, 쿼리 수 랭킹 1위

WITH temp AS (

SELECT '1분기' AS quarter, '데이터팀' as team, 'kyle' as user\_name, 10 as

UNION ALL

SELECT '2분기' AS quarter, '데이터팀' as team, 'kyle' as user\_name, 30 as

UNION ALL

SELECT '1분기' AS quarter, '머신러닝팀' as team, 'matthew' as user\_name, 1 as query\_count

UNION ALL

SELECT '2분기' AS quarter, '머신러닝팀' as team, 'matthew' as user\_name, 40 as query\_count

UNION ALL

SELECT '1분기' AS quarter, '데이터팀' as team, 'stark' as user\_name, 50 as query\_count

UNION ALL

SELECT '1분기' AS quarter, '마케팅팀' as team, 'thor' as user\_name, 30 as query\_count

UNION ALL

SELECT '2분기' AS quarter, '데이터팀' as team, 'thor' as user\_name, 90 as query\_count

)

SELECT

quarter, team, user\_name, query\_count,

RANK() OVER (PARTITION BY quarter, team ORDER BY query\_count DESC) as quarter\_rank,

RANK() OVER (ORDER BY query\_count DESC) as total\_rank

FROM temp

quarter	team	user_name	query_count	quarter_rank	total_rank
2분기	데이터팀	thor	90	1	1
1분기	데이터팀	stark	50	1	2
2분기	머신러닝팀	matthew	40	1	3
1분기	마케팅팀	thor	30	1	4
2분기	데이터팀	kyle	30	2	4
1분기	데이터팀	kyle	10	2	6
1분기	머신러닝팀	matthew	1	1	7

# Analytic Function(Window Function) [ ROW\_NUMBER ]

---

ROW\_NUMBER() : 행의 순서를 출력

RANK와 차이는?

# Analytic Function(Window Function) [ ROW\_NUMBER ]

---

ROW\_NUMBER() : 행의 순서를 출력

RANK와 차이는?

```
# 분기별, 팀별, 쿼리 수 랭킹 1위
WITH temp AS (
    SELECT '1분기' AS quarter, '데이터팀' as team, 'kyle' as user_name, 10 as query_count
    UNION ALL
    SELECT '2분기' AS quarter, '데이터팀' as team, 'kyle' as user_name, 30 as query_count
    UNION ALL
    SELECT '1분기' AS quarter, '머신러닝팀' as team, 'matthew' as user_name, 1 as query_count
    UNION ALL
    SELECT '2분기' AS quarter, '머신러닝팀' as team, 'matthew' as user_name, 40 as query_count
    UNION ALL
    SELECT '1분기' AS quarter, '데이터팀' as team, 'stark' as user_name, 50 as query_count
    UNION ALL
    SELECT '1분기' AS quarter, '마케팅팀' as team, 'thor' as user_name, 30 as query_count
    UNION ALL
    SELECT '2분기' AS quarter, '데이터팀' as team, 'thor' as user_name, 90 as query_count
)
SELECT
    quarter, team, user_name, query_count,
    RANK() OVER (ORDER BY query_count DESC) as total_rank,
    ROW_NUMBER() OVER (ORDER BY query_count DESC) as row_number
FROM temp
```

# Analytic Function(Window Function) [ ROW\_NUMBER ]

---

RANK는 같을 경우 “공동 4위” 이런 식으로 표현  
ROW\_NUMBER는 그냥 행 순서

quarter	team	user_name	query_count	total_rank	row_number
2분기	데이터팀	thor	90	1	1
1분기	데이터팀	stark	50	2	2
2분기	머신러닝팀	matthew	40	3	3
2분기	데이터팀	kyle	30	4	4
1분기	마케팅팀	thor	30	4	5
1분기	데이터팀	kyle	10	6	6
1분기	머신러닝팀	matthew	1	7	7

# Analytic Function(Window Function) [ Boundary ]

---

Analytics Function에서 Window의 boundary를 지정하고 싶을 경우

**UNBOUNDED PRECEDING , UNBOUNDED FOLLOWING, CURRENT ROW** 등을 사용

# Analytic Function(Window Function) [ Boundary ]

datetime별 moving\_average(이동 평균)을 구하고 싶은 경우  
(행의 앞의 1시간 전과 1시간 후의 사용해 평균)

datetime	demand	moving_average
2019-05-15 14:00:00	13	15
2019-05-15 15:00:00	16	16
2019-05-15 16:00:00	20	20
2019-05-15 17:00:00	25	29
2019-05-15 18:00:00	41	32
2019-05-15 19:00:00	31	34
2019-05-15 20:00:00	29	30

# Analytic Function(Window Function) [ Boundary ]

---

[ Sample Query ]

```
SELECT
  x,
  AVG(x) OVER (ORDER BY x ROWS BETWEEN 1 PRECEDING AND CURRENT ROW) AS avg_current,
  AVG(x) OVER (ORDER BY x ROWS BETWEEN 1 PRECEDING AND 1 FOLLOWING) AS avg_following
FROM UNNEST([0, 2, 4, 4, 5]) AS x
```

Row	x	avg_current	avg_following
1	0	0.0	1.0
2	2	1.0	2.0
3	4	3.0	3.333333333333335
4	4	4.0	4.33333333333333
5	5	4.5	4.5

# Analytic Function(Window Function) [ 정리 ]

## [ Sample Query ]

```
SELECT
```

```
*  
SUM(amount) OVER () as amount_total,  
SUM(amount) OVER (order by order_id rows between unbounded preceding and current row) as running_sum,  
SUM(amount) OVER (partition by customer_id order by datetime rows between unbounded preceding and current row) as  
running_sum_by_customer,  
avg(amount) over (order by datetime rows between 5 preceding and current row) as trailing_avg
```

```
FROM orders
```

```
ORDER BY 1
```

## [ 원본 데이터 ]

	order_id	customer_id	state	datetime	amount
1	1	A	CA	2017-01-01 00:00:00.000000	200
2	2	B	CA	2017-01-05 00:00:00.000000	250
3	3	C	NY	2017-01-12 00:00:00.000000	200
4	4	A	CA	2017-02-04 00:00:00.000000	400
5	5	D	CA	2017-02-05 00:00:00.000000	250
6	5	D	CA	2017-02-05 12:00:00.000000	300
7	6	C	NY	2017-02-19 00:00:00.000000	300
8	7	A	CA	2017-03-01 00:00:00.000000	150
9	8	E	NY	2017-03-05 00:00:00.000000	500
10	9	F	CA	2017-03-09 00:00:00.000000	250
11	10	B	CA	2017-03-21 00:00:00.000000	600

# Analytic Function(Window Function) [ 정리 ]

## [ Sample Query ]

SELECT

```
*  
SUM(amount) OVER () as amount_total,  
SUM(amount) OVER (order by order_id rows between unbounded preceding and current row) as running_sum,  
SUM(amount) OVER (partition by customer_id order by datetime rows between unbounded preceding and current row) as  
running_sum_by_customer,  
avg(amount) over (order by datetime rows between 5 preceding and current row) as trailing_avg
```

FROM orders

ORDER BY 1

## [ Output ]

	order_id	customer_id	state	datetime	amount	amount_total	running_sum	running_sum_by_customer	trailing_avg
1	1	A	CA	2017-01-01 00:00:00.000000	200	3400	200	200	200
2	2	B	CA	2017-01-05 00:00:00.000000	250	3400	450	250	225
3	3	C	NY	2017-01-12 00:00:00.000000	200	3400	650	200	216
4	4	A	CA	2017-02-04 00:00:00.000000	400	3400	1050	600	262
5	5	D	CA	2017-02-05 00:00:00.000000	250	3400	1300	250	260
6	5	D	CA	2017-02-05 12:00:00.000000	300	3400	1600	550	266
7	6	C	NY	2017-02-19 00:00:00.000000	300	3400	1900	500	283
8	7	A	CA	2017-03-01 00:00:00.000000	150	3400	2050	750	266
9	8	E	NY	2017-03-05 00:00:00.000000	500	3400	2550	500	316
10	9	F	CA	2017-03-09 00:00:00.000000	250	3400	2800	250	291
11	10	B	CA	2017-03-21 00:00:00.000000	600	3400	3400	850	350

# JSON\_EXTRACT

---

가끔 BigQuery 한 Value에 JSON 파일을 넣는 경우가 있음

아래 쿼리로 필요한 JSON 추출

```
SELECT JSON_EXTRACT(json_text, ".$class['students']") AS student_names
FROM UNNEST([
    '{"class" : {"students" : [{"name" : "Jane"}]}},
    '{"class" : {"students" : []}}',
    '{"class" : {"students" : [{"name" : "John"}, {"name": "Jamie"}]}}'
]) AS json_text;
```

student_names
[{"name":"Jane"}]
[]
[{"name":"John"}, {"name": "Jamie"}]

# JSON\_EXTRACT\_SCALAR

---

가끔 BigQuery 한 Value에 JSON 파일을 넣는 경우가 있음

아래 쿼리로 필요한 값만 추출

```
SELECT JSON_EXTRACT_SCALAR('{"a.b": {"c": "world"} }', "$['a.b'].c") as hello;
```

```
+-----+  
| hello |  
+-----+  
| world |  
+-----+
```

# UDF

---

UDF : User Defined Function, 사용자가 정의한 함수

## 사용하는 이유

- #1. BigQuery에서 없는 함수를 사용하고 싶은 경우
- #2. 쿼리로 처리하는데 반복해서 사용할 경우
- #3. 외부 라이브러리를 활용해야 할 경우

## 2가지 언어로 만들 수 있음

- #1. SQL
- #2. 자바스크립트(JavaScript)

단, VIEW에서 사용 불가능

# UDF

[ SQL ]

---

[ 정의 ]

[ 함수 이름 ]

[ 인자 ]

```
CREATE TEMP FUNCTION add_four_and_divide(x INT64, y INT64) AS (  
    (x + 4) / y  
);
```

```
SELECT val, add_four_and_divide(val, 2) AS result  
FROM (  
    SELECT 1 as val  
    UNION ALL  
    SELECT 3 as val  
    UNION ALL  
    SELECT 4 as val  
    UNION ALL  
    SELECT 5 as val  
)
```

[ 정의 ]

[ 함수 이름 ]

[ 인자 ]

```
CREATE TEMP FUNCTION customGreeting(a STRING)
```

```
RETURNS STRING
```

```
LANGUAGE js AS """
```

```
var d = new Date();
if (d.getHours() < 12) {
    return 'Good Morning, ' + a + '!';
} else {
    return 'Good Evening, ' + a + '!';
}
""";
```

[ 연산, 자바스크립트 ]

```
SELECT customGreeting(names) as everyone
FROM UNNEST(["Hannah", "Max", "Jakob"]) AS names;
```

```
CREATE TEMP FUNCTION myFunc(a FLOAT64, b STRING)
```

```
RETURNS STRING
```

```
LANGUAGE js AS
```

```
"""
```

```
// Assumes 'doInterestingStuff' is defined in one of the library files.
```

```
return doInterestingStuff(a, b);
```

```
""" [ 외부 라이브러리 사용 가능, Google Storage에 업로드해서 사용 ]
```

```
OPTIONS (
```

```
library="gs://my-bucket/path/to/lib1.js",
```

```
library=["gs://my-bucket/path/to/lib2.js", "gs://my-bucket/path/to/lib3.js"]
```

```
);
```

```
SELECT myFunc(3.14, 'foo');
```

# UDF

---

자주 사용하는 UDF는 Github Repo에 따로 관리하는 것도 좋음

(아쉽게도 **Default UDF 사용** 이런 기능이 없습니다...)

# PARTITION

---

BigQuery는 Query에서 탐색하는 데이터의 수가 많을수록 비용이 부과됨  
따라서 파티션을 설정해 쿼리 성능도 높이고 비용도 제어함

#1. Table 이름에 날짜 Suffix를 추가

Ex) Firebase App Data

`dataset.events\_20190517` => `dataset.events\_{date}`

#2. Table에서 TIMESTAMP 또는 DATE 열을 파티션으로 추가

- 웹 UI에선 불가능하고 CLI나 CREATE TABLE 쿼리로 생성

# PARTITION

---

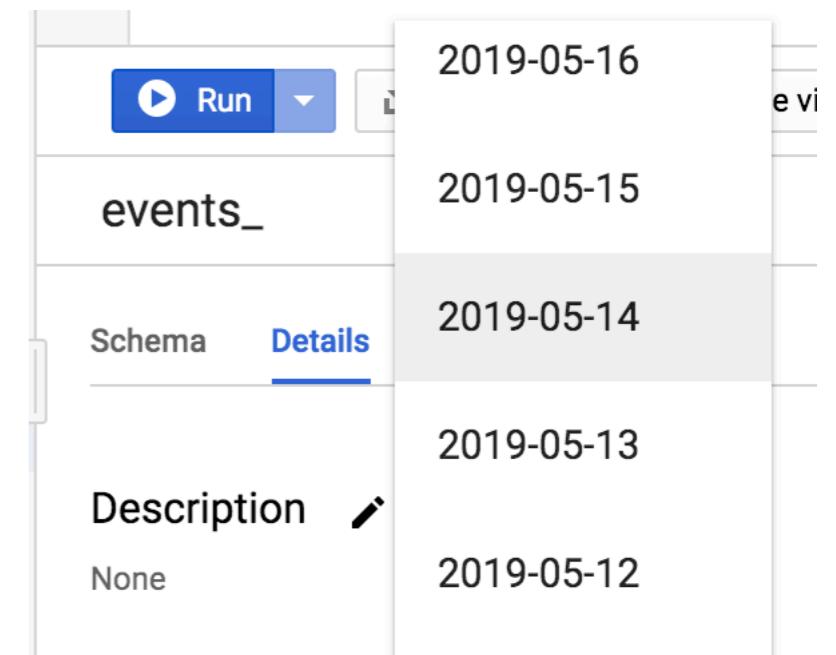
#1. Table 이름에 날짜 Suffix를 추가할 경우 Query : \_TABLE\_SUFFIX 사용

Ex) Firebase App Data

`dataset.events\_20190517` => `dataset.events\_{date}`

Table 생성할 때 suffix을 20190517 이런 형태로 저장

```
SELECT
  COUNT(user_id) AS count
FROM
  `firebase_data.events_*`
WHERE
  _TABLE_SUFFIX BETWEEN '20190515' AND '20190519'
```



# PARTITION

---

## #2. Table에서 TIMESTAMP 또는 DATE 열을 파티션으로 추가

- 웹 UI에선 불가능하고 CLIP나 CREATE TABLE 쿼리로 생성

```
CREATE TABLE mydataset.newtable (transaction_id INT64, transaction_date DATE)
```

```
PARTITION BY transaction_date
```

```
OPTIONS(
```

```
partition_expiration_days=3,
```

```
description="a table partitioned by transaction_date"
```

```
)
```

### Table info

Table ID	bigquery-public-data:wise_all_sky_data_release.all_wise
Table size	1.34 TB
Number of rows	747,634,026
Created	Apr 19, 2019, 2:44:42 AM
Table expiration	Never
Last modified	Apr 19, 2019, 2:44:42 AM
Data location	US
Table type	Partitioned
Partitioned by	Day
Partitioned on field	single_date
Partition filter	Not required
Clustered by	spt_ind, htm20

# CLI에서

```
bq --location=[LOCATION] query --destination_table [PROJECT_ID]:[DATASET].[TABLE] --time_partitioning_field [COLUMN] --use_legacy_sql=false '[QUERY]'
```

# PARTITION

---

#2. Table에서 TIMESTAMP 또는 DATE 열을 파티션으로 추가할 경우 Query  
: **\_PARTITIONTIME**, **\_PARTITIONDATE** 사용

# TIMESTAMP 사용시

```
SELECT  
  [COLUMN]  
FROM  
  [DATASET].[TABLE2]  
WHERE  
  _PARTITIONTIME BETWEEN TIMESTAMP('2017-01-01') AND TIMESTAMP('2017-03-01')
```

# DATE 사용시

```
SELECT  
  [COLUMN]  
FROM  
  [DATASET].[TABLE2]  
WHERE  
  _PARTITIONDATE BETWEEN '2017-01-01' AND '2017-03-01'
```

# PARTITION

---

#2. Table에서 TIMESTAMP 또는 DATE 열을 파티션으로 추가할 경우 Query  
: \_PARTITIONTIME, \_PARTITIONDATE 사용

```
SELECT
    _PARTITIONTIME as pt,
    FORMAT_TIMESTAMP("%Y%m%d", _PARTITIONTIME) as partition_id
FROM `PROJECT_ID.DATASET_ID.TABLE_ID`
GROUP BY _PARTITIONTIME
ORDER BY _PARTITIONTIME
```

# 실습 (3)

# 실습 (3)

데이터 설명 : bikeshare trip data  
**bikeshare\_trips Table** ( 지금은 이거만 사용해요 )

bikeshare\_trips

퀵 맵 테이블 쿼리 테이블 복사 테이블 삭제 내보내기 ▾

행	trip_id	subscriber_type	bikeid	start_time	start_station_id	start_station_name	end_st
1	9900285987	24-Hour Kiosk (Austin B-cycle)	446	2014-10-26 15:12:00 UTC	2712	Toomey Rd @ South Lamar	
2	9900285989	24-Hour Kiosk (Austin B-cycle)	203	2014-10-26 15:12:00 UTC	2712	Toomey Rd @ South Lamar	
3	9900285991	24-Hour Kiosk (Austin B-cycle)	101	2014-10-26 15:12:00 UTC	2712	Toomey Rd @ South Lamar	
4	9900286140	24-Hour Kiosk (Austin B-cycle)	242	2014-10-26 18:12:00 UTC	2541	State Capitol @ 14th & Colorado	
5	9900286143	24-Hour Kiosk (Austin B-cycle)	924	2014-10-26 18:12:00 UTC	2541	State Capitol @ 14th & Colorado	
6	9900286214	Annual Membership (Austin B-cycle)	24	2014-10-26 20:12:00 UTC	2712	Toomey Rd @ South Lamar	
7	9900286338	24-Hour Kiosk (Austin B-cycle)	101	2014-10-27 10:12:00 UTC	2712	Toomey Rd @ South Lamar	
8	13575843	Walk Up	302	2017-01-29 16:42:52 UTC	3464	Pease Park	
9	9900286942	24-Hour Kiosk (Austin B-cycle)	660	2014-10-28 14:12:00 UTC	2712	Toomey Rd @ South Lamar	
10	9900287148	24-Hour Kiosk (Austin B-cycle)	428	2014-10-28 19:12:00 UTC	2712	Toomey Rd @ South Lamar	

## 실습 (3)

---

#1. trip별 운행 시간의 total을 Analytics Function으로 표현

#2. bike별 운행 시간 total ( Analytics Function )

#3. bike별 운행 횟수, 평균까지 추가 ( Analytics Function )

#4. start\_station - end\_station별 total, 횟수, 평균 ( Analytics Function )

#5. start\_station\_id, bikeid별 Duration\_minutes 많은 순

#5-1. MAX값만 가져오기

#6. start\_station\_id별 전후 duration\_minutes

## 실습 (3)

---

#1. trip별 운행 시간의 total을 Analytics Function으로 표현

```
SELECT *,  
       SUM(duration_minutes) OVER (PARTITION BY trip_id ORDER BY start_time  
running_total  
FROM `bigquery-public-data.austin_bikeshare.bikeshare_trips`
```

문제 출제 의도 : Analytics Function 맛보기,  
원본 데이터에 있는 duration\_minutes 컬럼과 내 답이 맞는지 확인

## 실습 (3)

---

#2. bike별 운행 시간 total ( Analytics Function 사용해서! )

```
SELECT distinct bikeid,  
    SUM(duration_minutes) OVER (PARTITION BY bikeid) AS bike_running_total  
FROM `bigquery-public-data.austin_bikeshare.bikeshare_trips`  
where bikeid is not null  
order by 1 desc
```

문제 출제 의도 : Null값 처리(bikeid is not null), distinct를 맨 앞에 두면 중복을 제거한 후 나타남(distinct 빼고 해보세요!)

## 실습 (3)

---

#2. bike별 운행 시간 total ( Analytics Function 사용해서! )

```
SELECT distinct bikeid,  
    SUM(duration_minutes) OVER (PARTITION BY bikeid) AS bike_running_total  
FROM `bigquery-public-data.austin_bikeshare.bikeshare_trips`  
where bikeid is not null  
order by 1 desc
```

# 동일한 결과가 나오는 쿼리

```
SELECT bikeid, sum(duration_minutes) as bike_running_total  
FROM `bigquery-public-data.austin_bikeshare.bikeshare_trips`  
where bikeid is not null  
GROUP BY bikeid  
order by 1 desc
```

## 실습 (3)

---

#3. bike별 운행 횟수, 평균까지 추가 ( Analytics Function )

```
SELECT distinct bikeid,
    SUM(duration_minutes) OVER (PARTITION BY bikeid ORDER BY start_time) AS
bike_running_total,
    COUNT(duration_minutes) OVER (PARTITION BY bikeid ORDER BY start_time) AS
bike_running_count,
    AVG(duration_minutes) OVER (PARTITION BY bikeid ORDER BY start_time) AS
bike_running_avg
FROM `bigquery-public-data.austin_bikeshare.bikeshare_trips`  
where bikeid is not null
```

## 실습 (3)

---

#4. start\_station - end\_station별 total, 횟수, 평균 ( Analytics Function )

```
SELECT distinct start_station_id, end_station_id,
    SUM(duration_minutes) OVER (PARTITION BY start_station_id, end_station_id) AS
start_end_running_total,
    COUNT(duration_minutes) OVER (PARTITION BY start_station_id, end_station_id)
AS start_end_running_count,
    AVG(duration_minutes) OVER (PARTITION BY start_station_id, end_station_id) AS
start_end_running_avg
FROM `bigquery-public-data.austin_bikeshare.bikeshare_trips`  
where start_station_id is not null and end_station_id is not null
```

## 실습 (3)

---

#5. start\_station\_id, bikeid별 Duration\_minutes 많은 순

```
SELECT start_station_id, bikeid, duration_minutes,  
       ROW_NUMBER() OVER (PARTITION BY start_station_id, bikeid ORDER BY  
duration_minutes DESC) AS row_number  
FROM `bigquery-public-data.austin_bikeshare.bikeshare_trips`  
where start_station_id is not null and bikeid is not null
```

## 실습 (3)

---

#5-1. MAX값만 가져오기

```
SELECT *
FROM (
    SELECT start_station_id, bikeid, duration_minutes,
           ROW_NUMBER() OVER (PARTITION BY start_station_id, bikeid ORDER BY
duration_minutes DESC) AS row_number
    FROM `bigquery-public-data.austin_bikeshare.bikeshare_trips`  

    where start_station_id is not null and bikeid is not null
)
WHERE row_number=1
```

## 실습 (3)

---

#6. start\_station\_id별 전후 duration\_minutes

```
SELECT start_station_id, start_time, duration_minutes,
       LAG(duration_minutes, 1) OVER (PARTITION BY start_station_id ORDER BY
start_time) AS lag,
       LEAD(duration_minutes, 1) OVER (PARTITION BY start_station_id ORDER BY
start_time) AS lead
FROM `bigquery-public-data.austin_bikeshare.bikeshare_trips`  
where start_station_id is not null and bikeid is not null
```

# 스케줄 쿼리

# 스케줄 쿼리

---

특정 쿼리를 매일 특정 시간에 돌려야할 경우  
(예시 : Daily 매출을 쿼리로 계산하고 모니터링해야하는 경우)

## #1. 개발자

- 인스턴스를 하나 생성하고 crontab 등록
- AWS Lambda + CloudWatch Event
- (제 경우) Airflow 인스턴스 띄우고 관리

## #2. 기획자 또는 마케터라면?

# 스케줄 쿼리

---

특정 쿼리를 매일 특정 시간에 돌려야할 경우  
(예시 : Daily 매출을 쿼리로 계산하고 모니터링해야하는 경우)

## #1. 개발자

- 인스턴스를 하나 생성하고 crontab 등록
- AWS Lambda + CloudWatch Event
- (제 경우) Airflow 인스턴스 띄우고 관리

## #2. 기획자 또는 마케터라면?

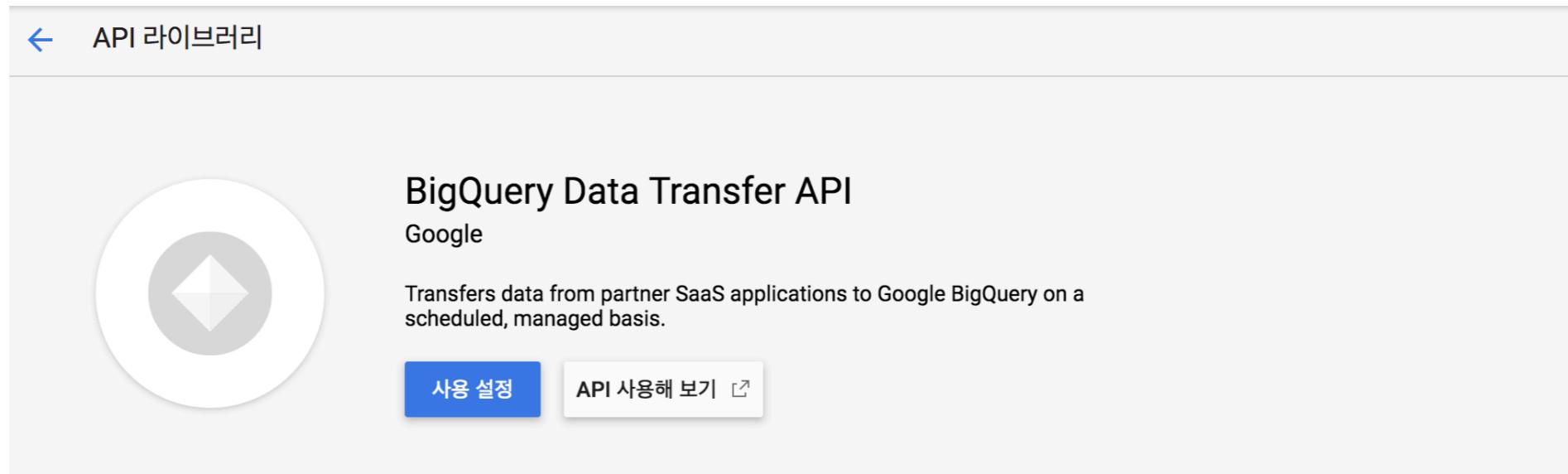
**간단히 스케줄 쿼리 사용!**

(TMI : 현재 베타)

# 스케줄 쿼리

## BigQuery Data Transfer Service 사용 설정

<https://console.cloud.google.com/apis/library/bigquerydatatransfer.googleapis.com>



### 유형

[API 및 서비스](#)

### 최종 업데이트

19. 2. 20. 오전 7:19

### 카테고리

[빅데이터](#)

[Google Cloud API](#)

### 서비스 이름

bigquerydatatransfer.googleapis.com

### 개요

Schedule queries or transfer external data from SaaS applications to Google BigQuery on a regular basis.

### Google 정보

Google's mission is to organize the world's information and make it universally accessible and useful. Through products and platforms like Search, Maps, Gmail, Android, Google Play, Chrome and YouTube, Google plays a meaningful role in the daily lives of billions of people.

### 가격

AdWords - Customer ID	DoubleClick Campaign Manager - Advertiser ID	\$ 2.50
	DoubleClick for Publishers - Network ID	가격/month
	Google Play - Package Name	

참고: API를 호출하는 데 사용한 인프라에서도 추가 요금이 부과될 수 있습니다. . USD 외의 통화로 지불하면 [Cloud Platform SKU](#)에 해당 통화로 표기된 가격이 적용됩니다. 최근 가격 정보는 [GCP 가격 목록](#)을 참조하세요.

# 스케줄 쿼리

---

## 권한 설정

- 예약된 쿼리는 [BigQuery Data Transfer Service](#)의 기능을 사용합니다. [BigQuery Data Transfer Service 사용 설정](#)에 필요한 모든 작업을 완료했는지 확인합니다.
- 다음과 같은 필수 권한이 있는지 확인합니다.
  - BigQuery**: 예약된 전송을 만들기 위한 `bigquery.transfers.update` 권한. `bigquery.admin` 사전 정의된 프로젝트 수준 IAM 역할에는 `bigquery.transfers.update` 권한이 포함됩니다. BigQuery의 IAM 역할에 대한 자세한 내용은 [액세스 제어](#)를 참조하세요.
  - `bigquery.cloud.google.com`에서 브라우저의 팝업을 허용하여 권한 창을 볼 수 있도록 합니다. 예약된 쿼리를 관리하는 BigQuery Data Transfer Service 권한을 허용해야 합니다.

# 스케줄 쿼리

## 권한 설정

- Role에 추가해서 관리하는 것 추천
- Roles - CREATE ROLE

The screenshot shows the Google Cloud IAM & Admin interface. On the left, a sidebar lists various IAM management options. The '역할' (Role) option is highlighted with a blue background. The main panel is titled '역할 만들기' (Create Role). It contains fields for '제목 \*' (Title) with the value '스케줄쿼리', '설명' (Description) with the value '생성일: 2019-05-18', 'ID \*' (ID) with the value '472', and a dropdown for '역할 실행 단계' (Execution Step) set to '알파'. A large orange box highlights the 'ADD PERMISSIONS' button. Below it, a section titled '할당된 권한 없음' (No assigned permissions) shows a table with columns for '권한 ↑' (Permissions), '상태' (Status), and a note '표시할 행이 없습니다.' (No rows to display). At the bottom, there are '만들기' (Create) and '취소' (Cancel) buttons.

<https://cloud.google.com/bigquery/docs/scheduling-queries>

# 스케줄 쿼리

## 권한 설정

- Role에 추가해서 관리하는 것 추천
- Roles - CREATE ROLE

## 권한 추가

The screenshot shows the 'bigquery.transfer' role configuration in the Google Cloud IAM & Admin API Permissions page. The role has three permissions assigned:

권한 ↑	상태
<input checked="" type="checkbox"/> bigquery.transfers.get	테스트 중 ⓘ
<input checked="" type="checkbox"/> bigquery.transfers.update	테스트 중 ⓘ

At the bottom right, there are two buttons: '취소' and '추가'. The '추가' button is highlighted with a yellow border.

<https://cloud.google.com/bigquery/docs/scheduling-queries>

# 스케줄 쿼리

## 권한 추가

- IAM - 유저 체크 후 수정

The screenshot shows the Google Cloud IAM & Admin interface. On the left, a sidebar lists various administrative tasks: IAM 및 관리자, IAM, ID 및 조직, 조직 정책, 할당량, 서비스 계정, 라벨, 설정, 개인정보 보호 및 보안, 암호화 키, IAP(Identity-Aware Proxy), 역할, and 감사 로그. The 'IAM' item in the sidebar is highlighted with a blue background. The main content area is titled "'My Project' 프로젝트의 권한" and displays a table of users with permissions. The table has columns: 유형 (Type), 구성원 (User), 이름 (Name), 역할 (Role), and 상속 (Inheritance). A user named 'snugyun01@gmail.com' is listed, with a checkmark in the '유형' column and a checkmark in the '소유자' (Owner) column. A yellow box highlights the edit icon (pencil) in the '상속' column for this user. The table also includes a filter section at the top right labeled 'snug' and '테이블 필터링'.

유형	구성원 ↑	이름	역할	상속
<input checked="" type="checkbox"/>		snugyun01@gmail.com	변성윤	

<https://cloud.google.com/bigquery/docs/scheduling-queries>

# 스케줄 쿼리

권한 추가

- 역할 추가 후 저장
  - BigQuery 관리자(Admin)도 설정해줘야 사용 가능 => 관점에 따라 이슈 존재할 수 있음  
권한 수정

## 구성원

snugyun01@gmail.com

## 프로젝트

## My Project

역할

소유자

모든 리소스에 대한 전체 권한입니다.

역할

## 스케줄쿼리

생성일: 2019-05-18

역할

BigQuery 관리자

모든 데이터세트 및 데이터세트 콘텐츠에 대한 전체  
액세스 권한입니다.

## +

### 다른 역할 추가

저작

취소

<https://cloud.google.com/bigquery/docs/scheduling-queries>

# 스케줄 쿼리

BigQuery 기능 및 정보 단축키 새 쿼리 작성

저장되지 않은 쿼리 수정됨 편집기 숨기기 전체 화면

쿼리 기록 저장된 쿼리 작업 기록 전송 예약된 쿼리 BI Engine

리소스 + 데이터 추가

austin

bigquery-public-data

- austin\_311
- austin\_bikeshare
  - bikeshare\_stations
  - bikeshare\_trips
- austin\_crime
- austin\_incidents
- austin\_waste

실행 쿼리 저장 보기 저장 쿼리 예약 더보기 실행 시 이 쿼리가 16.4MB를 처리합니다. ✓

쿼리 결과 결과 저장

쿼리 완료(0.0초 경과, 캐시됨)

작업 정보 결과 JSON

행	date	count
1	2013-12-21	103
2	2013-12-22	117
3	2013-12-23	96
4	2013-12-24	85
5	2013-12-25	145
6	2013-12-26	100
7	2013-12-27	115
8	2013-12-28	194
9	2013-12-29	255
10	2013-12-30	84

예약된 쿼리 업데이트 더보기 새로 예약된 쿼리 만들기

<https://cloud.google.com/bigquery/docs/scheduling-queries>

# 스케줄 쿼리

## 새로 예약된 쿼리

세부정보 및 일정

예약된 쿼리 이름

dau\_daily

### 일정 옵션

지금 시작  시작 시간 예약

반복 빈도

매주

시작일 및 실행 시간

19. 5. 20. 오전 9:00 KST

반복 요일

요일 선택

⚠️ 이 일정은 Mon May 20 2019부터 단위로 실행됩니다.

## 쿼리 결과의 대상 위치

i 예약된 쿼리 옵션을 저장하려면 대상 테이블이 필요합니다.

프로젝트 이름

My Project

데이터세트 이름

test\_data

테이블 이름

dau\_output

대상 테이블 쓰기 환경설정

테이블에 추가  
 테이블 덮어쓰기

### 알림 옵션

이메일 알림 전송 ?

<https://cloud.google.com/bigquery/docs/scheduling-queries>

# 스케줄 쿼리

## 새로 예약된 쿼리

세부정보 및 일정

예약된 쿼리 이름

dau\_daily

일정 옵션

지금 시작  시작 시간 예약

반복 빈도

매주

반복 요일

요일 선택

시작일 및 실행 시간

19. 5. 20. 오전 9:00 KST ▾

날짜

19. 5. 20.

시간

오전 9:00:00

⚠️ 이 일정은 Mon May 20 2019부터 단위로

쿼리 결과의 대상 위치

ℹ️ 예약된 쿼리 옵션을 저장하려면 대상 테이블을 선택하세요.

프로젝트 이름

My Project

테이블 이름

dau\_output

대상 테이블 쓰기 환경설정

테이블에 추가

테이블 덮어쓰기

2019년 5월						
일	월	화	수	목	금	토
28	29	30	1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	31	1
2	3	4	5	6	7	8

시간대

대한민국

날짜 설정

취소

알림 옵션

이메일 알림 전송 ?

<https://cloud.google.com/bigquery/docs/scheduling-queries>

# 스케줄 쿼리

---

# Sample Query

```
SELECT @run_time AS time,  
       title,  
       author,  
       text  
  FROM `bigquery-public-data.hacker_news.stories`  
 LIMIT  
      1000
```

<https://cloud.google.com/bigquery/docs/scheduling-queries>

# 스케줄 쿼리

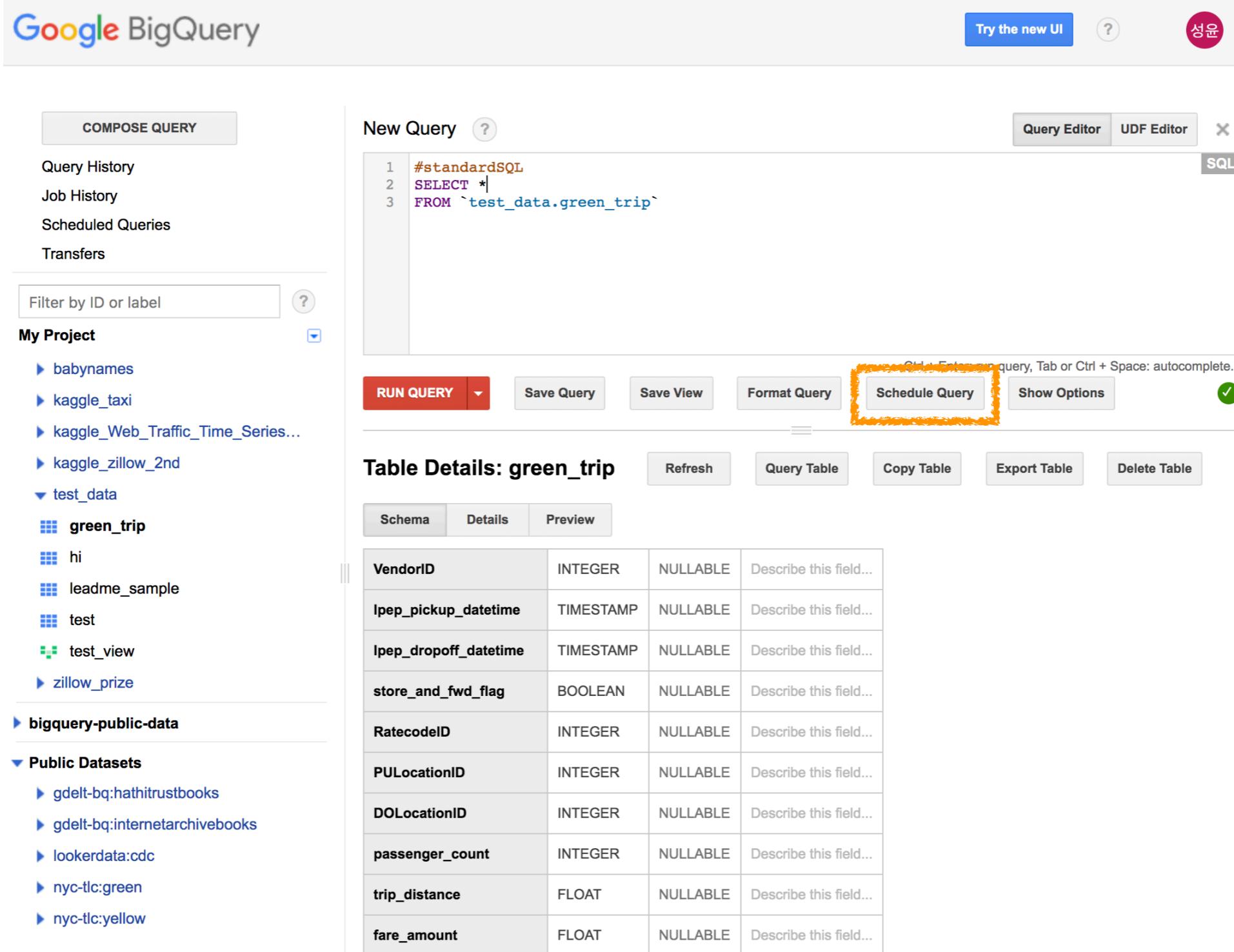
---

쿼리 결과의 대상 위치 테이블에  
`mytable_{run_time|"%Y%m%d"}` 이런 형식을 사용해 저장할 수 있음

run_time(UTC)	템플릿 매개변수	출력 대상 테이블 이름
2018-02-15 00:00:00	<code>mytable</code>	mytable
2018-02-15 00:00:00	<code>mytable_{run_time "%Y%m%d"}</code>	mytable_20180215
2018-02-15 00:00:00	<code>mytable_{+25h{run_time "%Y%m%d"}}</code>	mytable_20180216
2018-02-15 00:00:00	<code>mytable_{-1h{run_time "%Y%m%d"}}</code>	mytable_20180214
2018-02-15 00:00:00	<code>mytable_{+1.5h{run_time "%Y%m%d;%H"}}</code> 또는 <code>mytable_{+90m{run_time "%Y%m%d;%H"}}</code>	mytable_2018021501
2018-02-15 00:00:00	<code>{run_time+97s "%Y%m%d"}_mytable_{run_time "%H%M%S"}</code>	20180215_mytable_000137

# 스케줄 쿼리

만약 안되면 과거 UI에서 해보시는 것 추천!  
<https://bigquery.cloud.google.com>



The screenshot shows the Google BigQuery web interface. On the left, there's a sidebar with links like 'COMPOSE QUERY', 'Query History', 'Job History', 'Scheduled Queries', and 'Transfers'. Below that is a 'My Project' section listing datasets: 'babynames', 'kaggle\_taxi', 'kaggle\_Web\_Traffic\_Time\_Series...', 'kaggle\_zillow\_2nd', 'test\_data' (which contains 'green\_trip', 'hi', 'leadme\_sample', 'test', 'test\_view', and 'zillow\_prize'), 'bigquery-public-data', and 'Public Datasets' (listing 'gdelt-bq:hathitrustbooks', 'gdelt-bq:internetarchivebooks', 'lookerdata:cdc', 'nyc-tlc:green', and 'nyc-tlc:yellow'). The main area is titled 'New Query' and contains a code editor with the following SQL:

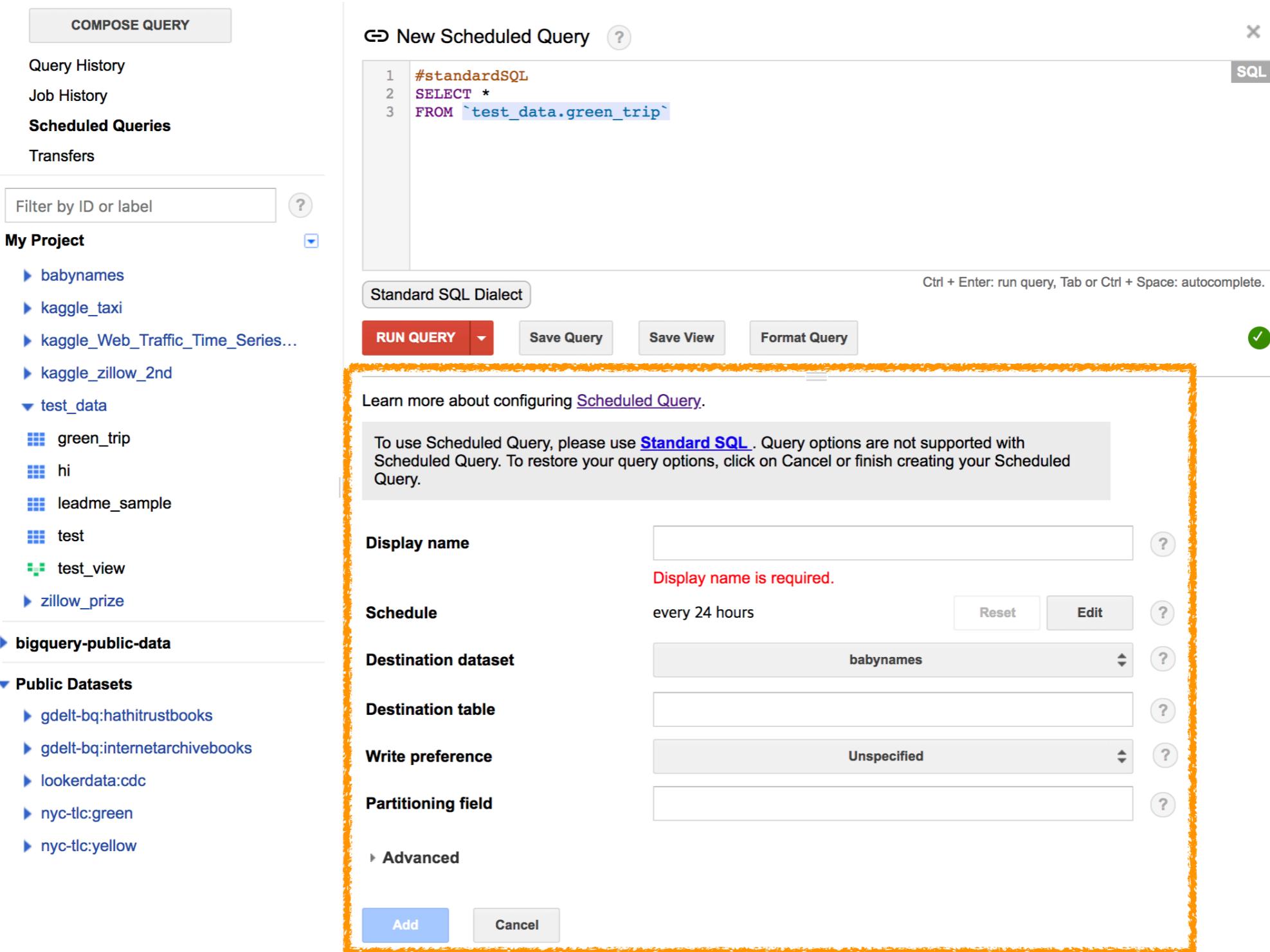
```
1 #standardSQL
2 SELECT *
3 FROM `test_data.green_trip`
```

Below the code editor are buttons for 'RUN QUERY', 'Save Query', 'Save View', 'Format Query', 'Schedule Query' (which is highlighted with a yellow box), and 'Show Options'. A status message says 'Ctrl + Enter: run query, Tab or Ctrl + Space: autocomplete.' To the right of the code editor is a 'Table Details' section for the 'green\_trip' table, showing its schema:

VendorID	INTEGER	NULLABLE	Describe this field...
Ipep_pickup_datetime	TIMESTAMP	NULLABLE	Describe this field...
Ipep_dropoff_datetime	TIMESTAMP	NULLABLE	Describe this field...
store_and_fwd_flag	BOOLEAN	NULLABLE	Describe this field...
RatecodeID	INTEGER	NULLABLE	Describe this field...
PULocationID	INTEGER	NULLABLE	Describe this field...
DOLocationID	INTEGER	NULLABLE	Describe this field...
passenger_count	INTEGER	NULLABLE	Describe this field...
trip_distance	FLOAT	NULLABLE	Describe this field...
fare_amount	FLOAT	NULLABLE	Describe this field...

# 스케줄 쿼리

만약 안되면 과거 UI에서 해보시는 것 추천!  
<https://bigquery.cloud.google.com>



COMPOSE QUERY

New Scheduled Query

```
1 #standardSQL
2 SELECT *
3 FROM `test_data.green_trip`
```

Standard SQL Dialect

RUN QUERY Save Query Save View Format Query

Ctrl + Enter: run query, Tab or Ctrl + Space: autocomplete.

Learn more about configuring [Scheduled Query](#).

To use Scheduled Query, please use [Standard SQL](#). Query options are not supported with Scheduled Query. To restore your query options, click on Cancel or finish creating your Scheduled Query.

Display name

Display name is required.

Schedule

every 24 hours

Destination dataset

babynames

Destination table

Write preference

Unspecified

Partitioning field

Advanced

Add Cancel

# Table 생성하기 (데이터 불러오기)

# Table 생성하기

## [ Table 생성하는 방법 ]

- 빈 테이블
- Google Cloud Storage
- 업로드
- 구글 드라이브
- Google Cloud Bigtable

테이블 만들기

소스

다음 항목으로 테이블 만들기:

빈 테이블

빈 테이블

Google Cloud Storage

업로드

드라이브

Google Cloud Bigtable

대상

프로젝트 이름

My Project

세트 이름

test\_data

테이블 유형

기본 테이블

테이블 이름

문자, 숫자, 밑줄이 허용됩니다.

스키마

텍스트로 편집

+ 필드 추가

# Table 생성하기

---

[ 로컬 업로드 ]

우리가 생각하는 방식으로 업로드  
단, 로컬 업로드(웹)는 10MB로 제한

테이블 만들기

소스

다음 항목으로 테이블 만들기:

업로드 ▾

파일 선택: ?

green\_tripdata\_2018-01.csv

파일 형식:

탐색 CSV ▾

로컬 업로드는 10MB로 제한됩니다. 더 큰 파일은 Google Cloud Storage를  
사용하세요. 자세히 알아보기

# Table 생성하기

---

## [ CLI 업로드 ]

웹을 통하지 않고 CLI bq 명령어로 Table 생성 가능

```
bq --location=US load --autodetect --source_format=CSV test_data.building building_0316.csv
```

bq 명령어를 사용하기 위해선 gcloud 설치 필수

```
byeon@byeon_Macbook ~ ~/Downloads ➔ bq --location=US load --autodetect --source_format=CSV test_data.green_trip green_tripdata_2018-01.csv  
Upload complete.  
Waiting on bqjob_r450ebafc1f564d87_0000016acb991908_1 ... (20s) Current status: DONE
```

# Table 생성하기

[ Google Cloud Storage ]

제일 많이 사용하는 방법

아래 이미지는 zzsza\_data 버켓에 green\_tripdata\_2018-01.csv 저장

**gs://zzsza\_data/green\_tripdata\_2018-01.csv**

The screenshot shows the Google Cloud Storage console interface. On the left is a sidebar with icons for Storage, Browser, Transfer, Transfer Analytics, and Settings. The main area shows the 'zzsza\_data' bucket details. At the top, there are back, edit, and new bucket buttons. Below that, tabs for General, Labels, and Bucket IAM are visible, with 'General' selected. A toolbar includes File Upload, Folder Upload, Create Folder, Manage Locations, and Delete. A search bar contains '프리픽스로 필터링...'. The bucket list table shows two files: 'green\_tripdata\_2018-01.csv' (67.64MB, text/csv, Regional storage class) and 'kaggle.json' (65B, application/json, Regional storage class). Each file row includes checkboxes, download icons, and detailed metadata columns.

선택	이름	크기	유형	저장소 클래스	최종 수정 시간	공개 액세스	암호화	보존 만료 날짜	보존 조치	More
<input type="checkbox"/>	green_tripdata_2018-01.csv	67.64MB	text/csv	Regional	19. 5. 19., 오전 12시 40분 4초 UTC+9	공개 아님	Google 관리 키	-	없음	<input type="button" value="⋮"/>
<input type="checkbox"/>	kaggle.json	65B	application/json	Regional	19. 1. 26., 오후 9시 28분 34초 UTC+9	공개 아님	Google 관리 키	-	없음	<input type="button" value="⋮"/>

# Table 생성하기

## [ Google Cloud Storage ]

테이블 만들기

소스

다음 항목으로 테이블 만들기:

GCS 버킷에서 파일 선택: [?](#)

[ GS URL ]

[ 파일 형식 ]

Google Cloud Storage ▾

zzsza\_data/green\_tripdata\_2018-01.csv

찾아보기

파일 형식:  
CSV ▾

대상

프로젝트 이름

My Project ▾

데이터세트 이름

test\_data ▾

테이블 유형 [?](#)

기본 테이블 ▾

테이블 이름

green\_trip\_201801

스키마

자동 감지

스키마 및 입력 매개변수

[ 스키마 자동감지(체크 안하고 지정해줘도 됨) ]

ⓘ 스키마가 자동으로 생성됩니다.

파티션 및 클러스터 설정

분할: [?](#)

파티션 없음 ▾

[ 파티션 설정, 시간을 파티션 기준으로 설정 가능 ]

# Table 생성하기

## [ Google Cloud Storage ]

### - 고급 옵션

고급 옵션 ^

쓰기 환경설정

비어 있으면 쓰기

허용되는 오류 개수: ?  
0

알 수 없는 값: ?  
 알 수 없는 값 무시

필드 구분 기호: ?  
쉼표

건너뛸 헤더 행: ?  
0

Quoted newlines ?  
 따옴표 안에 줄바꿈 허용

Jagged rows ?  
 불균일 행 허용

암호화

데이터가 자동으로 암호화됩니다. 암호화 키 관리 솔루션을 선택하세요.

Google 관리 키  
구성이 필요하지 않습니다.

고객 관리 키  
Google Cloud Key Management Service를 통해 관리합니다.

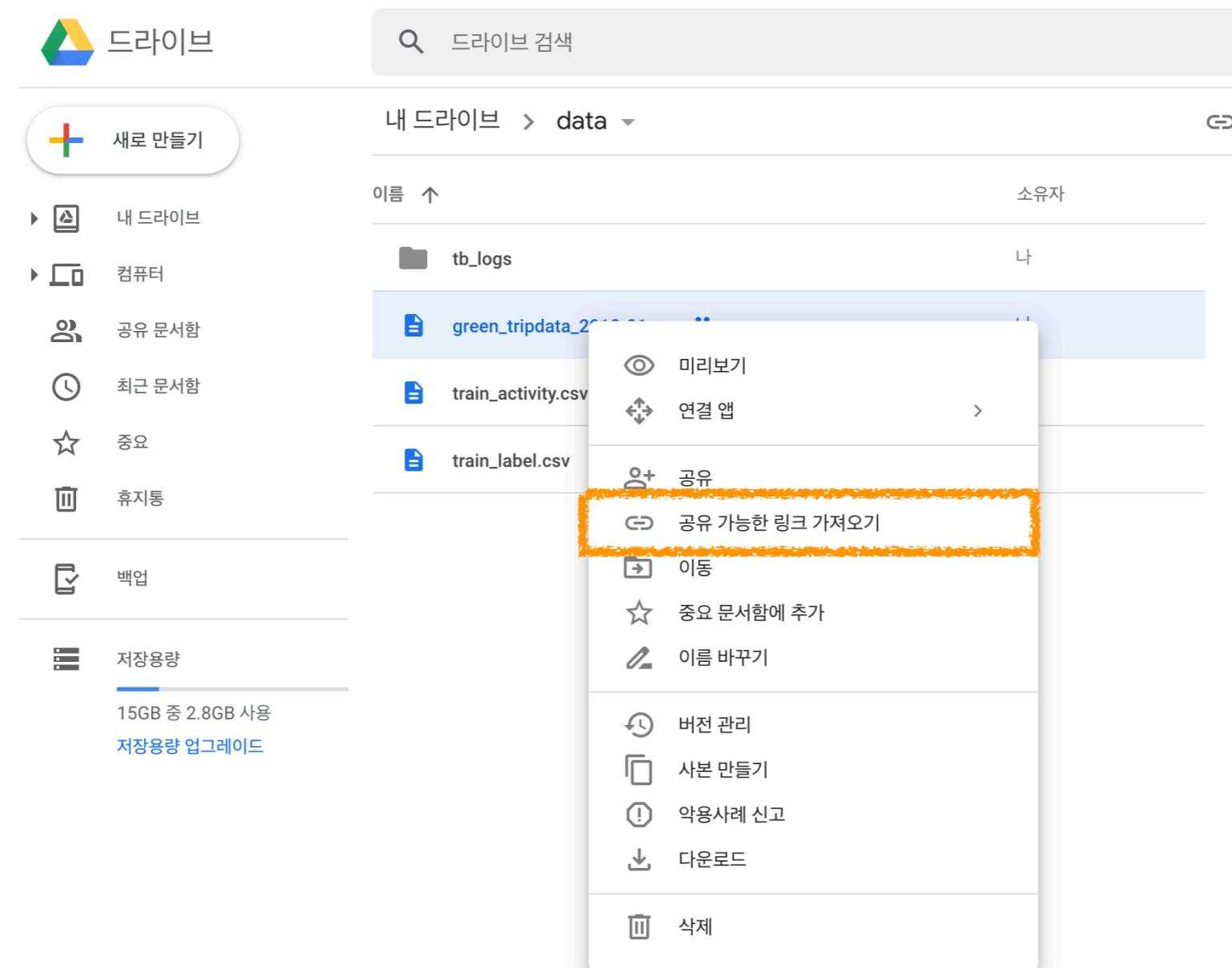
# Table 생성하기

## [ Google Drive, 스프레드시트 ]

[https://drive.google.com/open?id=\[file\\_id\]](https://drive.google.com/open?id=[file_id])

[https://docs.google.com/spreadsheets/d/\[file\\_id\]](https://docs.google.com/spreadsheets/d/[file_id])(Google 스프레드시트)

Google 드라이브 - 파일 마우스 오른쪽 버튼 클릭, 공유 가능한 링크 가져오기



# 쿼리 결과 다운로드하기 (데이터 추출하기)

# 쿼리 결과 저장하기

[ 쿼리 날린 후, 결과 저장 ]

#1. 16,000행 이내일 경우 ⇒ CSV(로컬 파일), Google 스프레드시트

#2. 1GB 이하일 경우(행 상관없음) ⇒ CSV(Google 드라이브), JSON(Google 드라이브)

#3. 1GB 초과 또는 16,000행 초과 ⇒ BigQuery 테이블로 저장한 후 ⇒ Google Storage 저장

#4. 파이썬을 사용할 수 있을 경우 ⇒ pd.read\_gbq로 쿼리 결과 메모리에 저장

쿼리 편집기

```
1 SELECT DATE(start_time) as date,
2 count(trip_id) as count
3 FROM `bigquery-public-data.austin_bikeshare.bikeshare_trips`
4 GROUP BY date
5 ORDER BY date
```

▶ 실행    ⌂ 쿼리 저장    ⏷ 보기 저장    ⏲ 쿼리 예약    ⚙ 더보기

쿼리 결과    ⌂ 결과 저장    ⚒ 데이터 스튜디오에서 살펴보기

쿼리 완료(0.1초 경과, 캐시됨)

작업 정보	결과	JSON
행	date	count
1	2013-12-21	103
2	2013-12-22	117
3	2013-12-23	96
4	2013-12-24	85
5	2013-12-25	145
6	2013-12-26	100
7	2013-12-27	115
8	2013-12-28	194

CSV(Google 드라이브)  
최대 1GB의 결과를 Google 드라이브에 저장합니다.

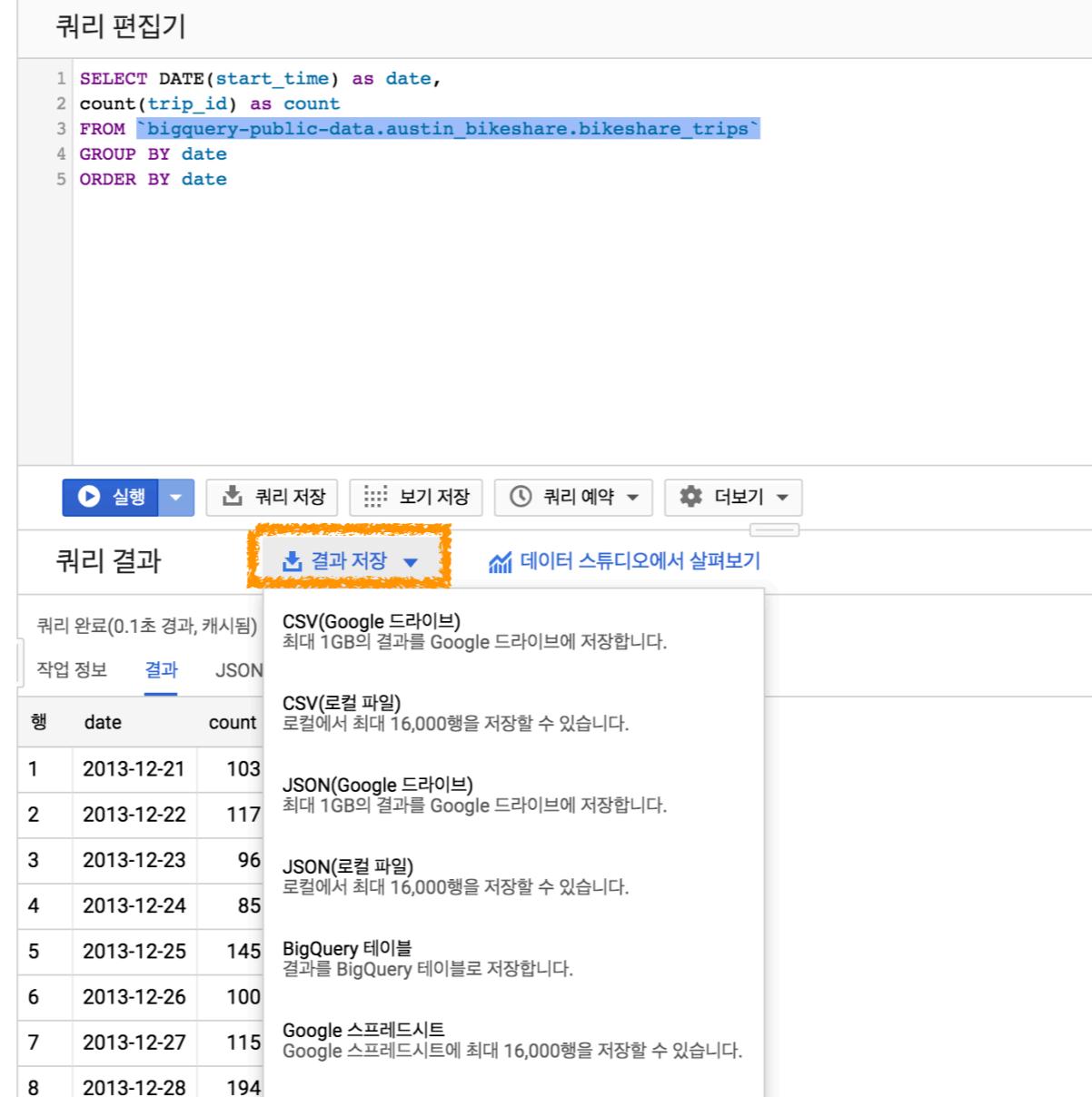
CSV(로컬 파일)  
로컬에서 최대 16,000행을 저장할 수 있습니다.

JSON(Google 드라이브)  
최대 1GB의 결과를 Google 드라이브에 저장합니다.

JSON(로컬 파일)  
로컬에서 최대 16,000행을 저장할 수 있습니다.

BigQuery 테이블  
결과를 BigQuery 테이블로 저장합니다.

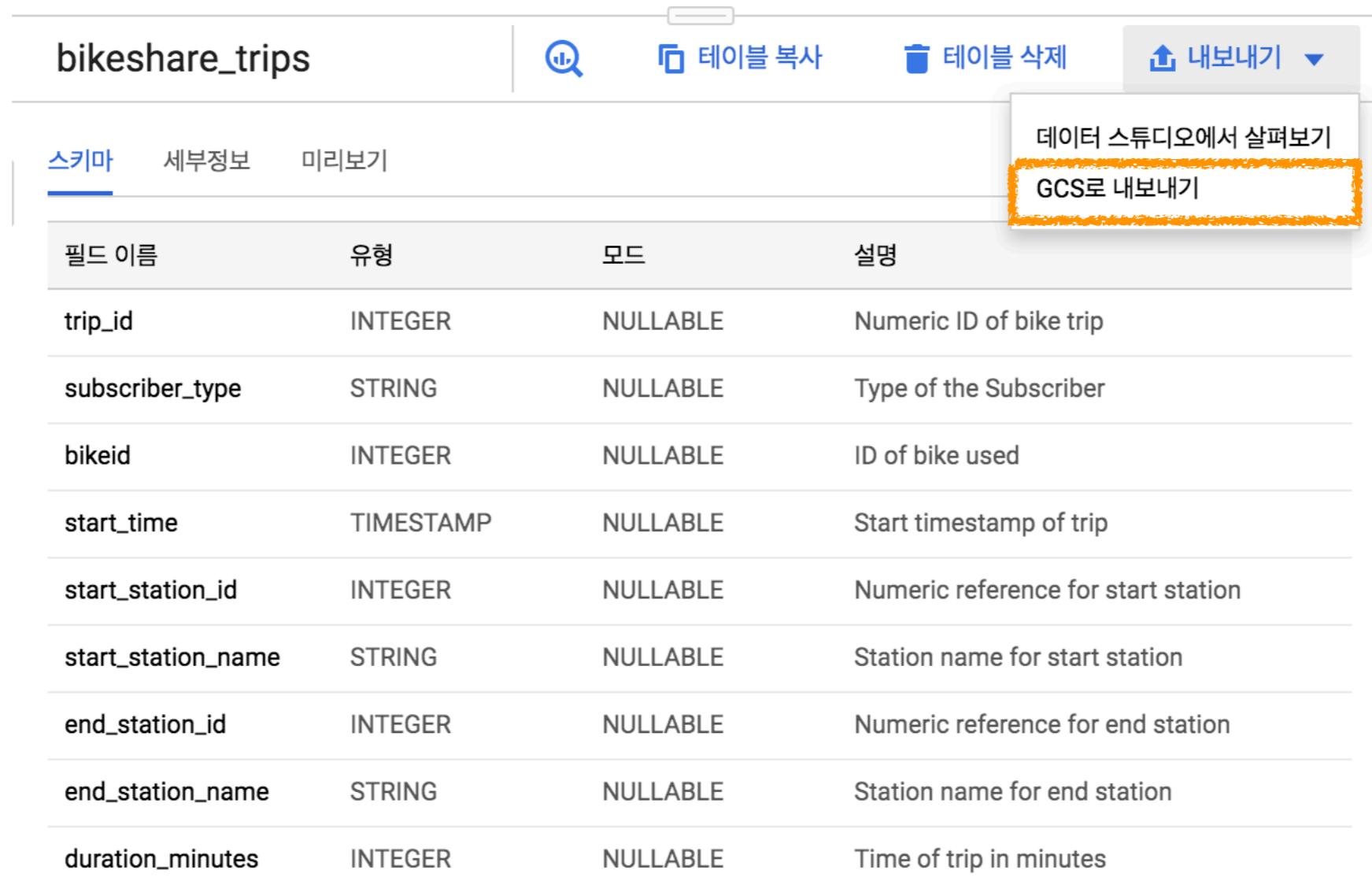
Google 스프레드시트  
Google 스프레드시트에 최대 16,000행을 저장할 수 있습니다.



# 쿼리 결과 저장하기

[ 쿼리 날린 후, 결과 저장 ]

#3. 1GB 초과 또는 16,000행 초과 => BigQuery 테이블로 저장한 후 ⇒ Google Storage 저장



The screenshot shows the BigQuery console interface for the 'bikeshare\_trips' table. At the top, there are tabs for '스키마' (Schema), '세부정보' (Details), and '미리보기' (Preview). On the right, there are buttons for '테이블 복사' (Copy Table), '테이블 삭제' (Delete Table), and '내보내기' (Export). A dropdown menu from the '내보내기' button is open, showing options: '데이터 스튜디오에서 살펴보기' (View in Data Studio) and 'GCS로 내보내기' (Export to GCS), with the second option being highlighted by an orange box.

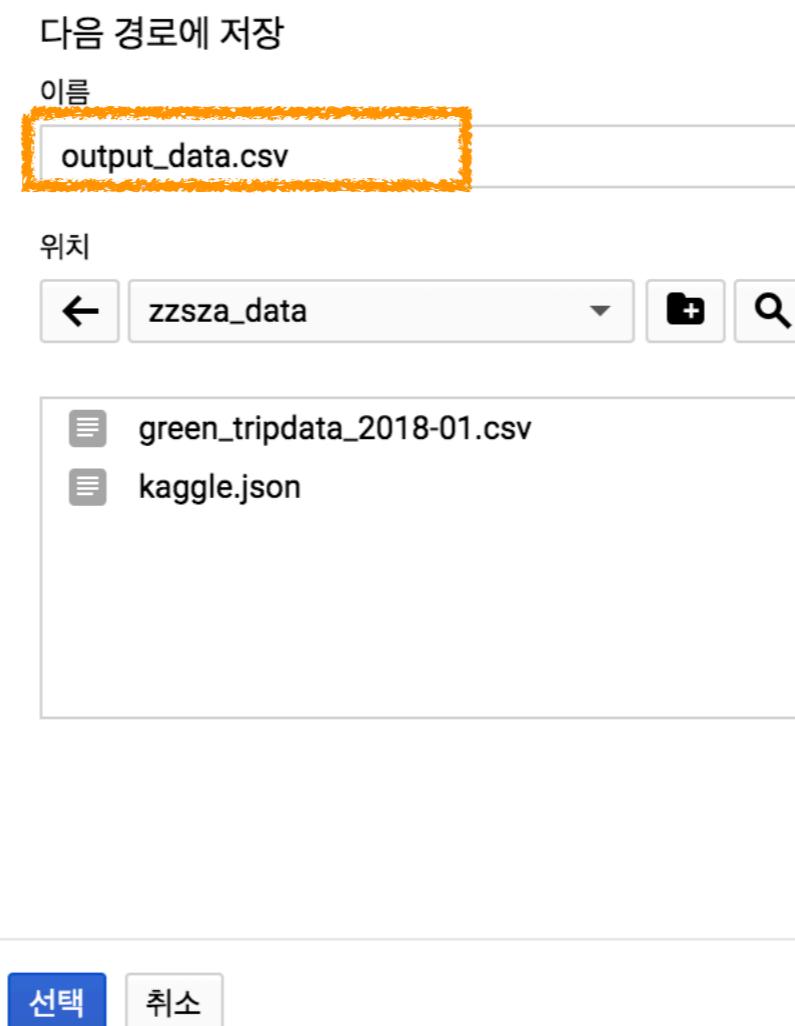
필드 이름	유형	모드	설명
trip_id	INTEGER	NULLABLE	Numeric ID of bike trip
subscriber_type	STRING	NULLABLE	Type of the Subscriber
bikeid	INTEGER	NULLABLE	ID of bike used
start_time	TIMESTAMP	NULLABLE	Start timestamp of trip
start_station_id	INTEGER	NULLABLE	Numeric reference for start station
start_station_name	STRING	NULLABLE	Station name for start station
end_station_id	INTEGER	NULLABLE	Numeric reference for end station
end_station_name	STRING	NULLABLE	Station name for end station
duration_minutes	INTEGER	NULLABLE	Time of trip in minutes

# 쿼리 결과 저장하기

[ 쿼리 날린 후, 결과 저장 ]

#3. 1GB 초과 또는 16,000행 초과 => BigQuery 테이블로 저장한 후 ⇒ Google Storage 저장

저장 후, <https://console.cloud.google.com/storage/browser> 로 이동해서 파일 확인



# 쿼리 결과 저장하기

[ 쿼리 날린 후, 결과 저장 ]

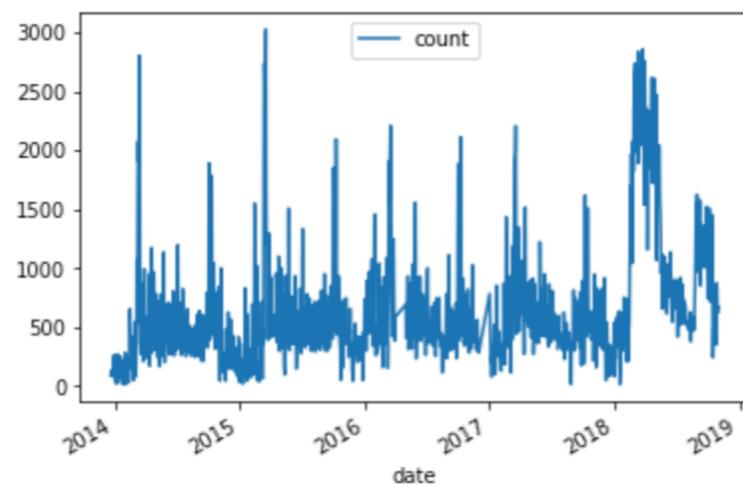
#4. 파이썬을 사용할 수 있을 경우 => pd.read\_gbq로 쿼리 결과 메모리에 저장

```
In [1]: import pandas as pd
import pandas_gbq
import matplotlib.pyplot as plt

In [2]: query = """
SELECT DATE(start_time) as date, count(trip_id) as count
FROM `bigquery-public-data.austin_bikeshare.bikeshare_trips`
GROUP BY date
ORDER BY date
"""

In [3]: df = pd.read_gbq(query=query, dialect='standard', private_key='<your json key>')

In [4]: df.set_index('date').plot();
```



# DML Query

# DML Query

---

[ 대부분 사용하지 않을 가능성이 큰 기능 ]

BigQuery Table에 데이터를 업데이트, 삽입, 삭제할 수 있는 기능

**INSERT**(추가), **UPDATE**(업데이트), **DELETE**(삭제)

# DML Query

---

[ 대부분 사용하지 않을 가능성이 큰 기능 ]

BigQuery Table에 데이터를 업데이트, 삽입, 삭제할 수 있는 기능

**INSERT(추가), UPDATE(업데이트), DELETE(삭제)**

데이터 조작 언어(DML) 문

DML 문에는 다음 한도가 적용됩니다.

- 테이블당 일일 **INSERT, UPDATE, DELETE, MERGE** 문의 최대 총 수 – 1,000개



MERGE 문은 여러 개의 INSERT, UPDATE, DELETE 절을 포함한 경우에도 DML 문 1개로 계산됩니다.

# DML Query

[ INSERT ]

## #1. 명시적 값을 지정해 INSERT

[ Destination Table 이름 ]

[ Destination Table 컬럼 ]

```
INSERT dataset.NewArrivals (product, quantity, warehouse)
VALUES('top load washer', 100, 'warehouse #1'),
      ('dryer', 200, 'warehouse #2'),
      ('oven', 300, 'warehouse #3')
```

[ INSERT 할 데이터 ]

## #2. SELECT 결과를 INSERT

[ Destination Table 이름 ]

[ Destination Table 컬럼 ]

```
INSERT dataset.Warehouse (warehouse, state)
```

```
SELECT *
FROM UNNEST([('warehouse #1', 'WA'),
              ('warehouse #2', 'CA'),
              ('warehouse #3', 'WA')])
```

[ INSERT 할 데이터 ]

## #3. 기존 테이블에 데이터가 없을 경우만 INSERT

```
INSERT `existing.table` (date, hour, gu, demand)
```

```
SELECT b.date, b.hour, b.gu, b.demand  
FROM `temp.table` AS b
```

```
WHERE NOT EXISTS (  
    SELECT 1 FROM `existing.table` AS a  
    WHERE a.date = b.date AND a.hour = b.hour AND a.gu = b.gu)
```

[ 기존에 있는지 없는지 체크하는 쿼리 ]

# DML Query

[ UPDATE ]

---

## #1. 특정 조건을 만족하는 값을 UPDATE

[ Destination Table 이름 ]

UPDATE dataset.Inventory

SET quantity = quantity - 10 [ UPDATE를 이렇게 해라 ]

WHERE product like '%washer%'

[ 특정 조건 ]

## #2. 기존 Table에 특정 조건이 만족할 경우에 UPDATE

```
UPDATE `existing.table` AS a
```

```
SET a.demand = b.demand [ UPDATE를 이렇게 해라 ]
```

```
FROM (
```

```
    SELECT * FROM `temp.table` ) AS b
```

```
WHERE a.date = b.date AND a.hour = b.hour AND a.gu = b.gu
```

[ 특정 조건 ]

## #1. Table의 모든 행 DELETE

[ Source Table 이름 ]

```
DELETE dataset.DetailedInventory  
WHERE true [ 모두 다 ]
```

## #2. Table의 특정 조건을 만족하는 행 DELETE

[ Source Table 이름 ]

```
DELETE dataset.Inventory  
WHERE quantity = 0
```

[ 특정 조건 ]

# Firebase Log

# Firebase

[ Firebase 앱 데이터의 생김새 ]

event_dim.date	event_dim.name	event_dim.params.key	event_dim.params.value.string_value
20170914	Save_Name	name	firebase
		firebase_event_origin	app
20170914	Save_Name	name	bigquery
		firebase_event_origin	app
20170914	Save_Name	firebase_event_origin	app
		name	firebase
20170914	Save_Name	name	seongyun
		firebase_event_origin	app



이쪽이 처음 볼 때 이해가 잘 안됨

여기는 Table 형태라 이해가 쉬운 편

# Firebase

---

[ 단순 Event Count는 쉽게 가능 ]  
event\_dim.name만 SELECT하면 탐색하는 데이터가 적음

event_dim.date	event_dim.name
20170914	Save_Name

```
SELECT  
    COUNT(DISTINCT user_id) AS save_name_unique  
FROM  
    `YOUR_TABLE.events_*`  
WHERE  
    event_name = 'Save_Name'  
    -- 원하는 기간으로 변경  
    AND _TABLE_SUFFIX BETWEEN '20170914' AND '20170931'
```

# Firebase

[ 하지만 사람들의 궁금증은? ]

어떤 이름을 얼마나 저장했을까? ⇒ 파라미터까지 봐야 함

event_dim.date	event_dim.name	event_dim.params.key	event_dim.params.value.string_value
20170914	Save_Name	name	firebase
		firebase_event_origin	app
20170914	Save_Name	name	bigquery
		firebase_event_origin	app
20170914	Save_Name	firebase_event_origin	app
		name	firebase
20170914	Save_Name	name	seongyun
		firebase_event_origin	app

# Firebase

---

[ Firebase 데이터 생김새(Field name은 다를 수 있음) : 디테일은 아래 링크 참고 ]

Field name	Type	Mode				
<b>user_dim</b>	RECORD	NULLABLE	<b>user_dim.geo_info.continent</b>	STRING	NULLABLE	
<b>user_dim.user_id</b>	STRING	NULLABLE	<b>user_dim.geo_info.country</b>	STRING	NULLABLE	
<b>user_dim.first_open_timestamp_micros</b>	INTEGER	NULLABLE	<b>user_dim.geo_info.region</b>	STRING	NULLABLE	
<b>user_dim.user_properties</b>	RECORD	REPEATED	<b>user_dim.geo_info.city</b>	STRING	NULLABLE	
<b>user_dim.user_properties.key</b>	STRING	NULLABLE	<b>user_dim.app_info</b>	RECORD	NULLABLE	
<b>user_dim.user_properties.value</b>	RECORD	NULLABLE	<b>user_dim.app_info.app_version</b>	STRING	NULLABLE	
<b>user_dim.user_properties.value.value</b>	RECORD	NULLABLE	<b>user_dim.app_info.app_instance_id</b>	STRING	NULLABLE	
<b>user_dim.user_properties.value.value.string_value</b>	STRING	NULLABLE	<b>user_dim.app_info.app_store</b>	STRING	NULLABLE	
<b>user_dim.user_properties.value.value.int_value</b>	INTEGER	NULLABLE	<b>user_dim.app_info.app_platform</b>	STRING	NULLABLE	
<b>user_dim.user_properties.value.value.float_value</b>	FLOAT	NULLABLE	<b>event_dim</b>	RECORD	REPEATED	
<b>user_dim.user_properties.value.value.double_value</b>	FLOAT	NULLABLE	<b>event_dim.name</b>	STRING	NULLABLE	
<b>user_dim.user_properties.value.set_timestamp_usec</b>	INTEGER	NULLABLE	<b>event_dim.params</b>	RECORD	REPEATED	
<b>user_dim.user_properties.value.index</b>	INTEGER	NULLABLE	<b>event_dim.params.key</b>	STRING	NULLABLE	
<b>user_dim.device_info</b>	RECORD	NULLABLE	<b>event_dim.params.value</b>	RECORD	NULLABLE	
<b>user_dim.device_info.device_category</b>	STRING	NULLABLE	<b>event_dim.params.value.string_value</b>	STRING	NULLABLE	
<b>user_dim.device_info.mobile_brand_name</b>	STRING	NULLABLE	<b>event_dim.params.value.int_value</b>	INTEGER	NULLABLE	
<b>user_dim.device_info.mobile_model_name</b>	STRING	NULLABLE	<b>event_dim.params.value.float_value</b>	FLOAT	NULLABLE	
<b>user_dim.device_info.mobile_marketing_name</b>	STRING	NULLABLE	<b>event_dim.params.value.double_value</b>	FLOAT	NULLABLE	
<b>user_dim.device_info.device_model</b>	STRING	NULLABLE	<b>event_dim.timestamp_micros</b>	INTEGER	NULLABLE	
			<b>event_dim.previous_timestamp_micros</b>	INTEGER	NULLABLE	
			<b>event_dim.date</b>	STRING	NULLABLE	

# Firebase

[ Firebase 데이터 생김새(Field name은 다를 수 있음) : 디테일은 아래 링크 참고 ]

Row	user_dim.user_id	user_dim.first_open_timestamp_micros	user_dim.user_properties.key	user_dim.user_properties.value.string_value	user_dim.user_properties.value.int_value	user_dim.user_properties.value.float_value	user_dim.user_properties.value.double_value	user_dim.user_properties.value.set_timestamp_usec
1	null	1465314875300038	api_version	1.2	null	null	null	1465319592400059
			user_id	null	null	null	null	1465317326700059
			powers	0	null	null	null	1465310459800059
			elite_powers	0	null	null	null	1465310459800059
			chips	2386	null	null	null	1465310459800059
			first_open_time	null	1463565600000	null	null	1465314875300059
			is_signed_in	true	null	null	null	1465319592400059
			coins	571731	null	null	null	1465310459800059
			platform	android	null	null	null	1465319591300059
			language	en_US	null	null	null	1465310459800059
			keys	0	null	null	null	1465310459800059
			xp	51292	null	null	null	1465310459800059

( 직전 Event timestamp 찍어줌 )

event_dim.name	event_dim.params.key	event_dim.params.value.string_value	event_dim.params.value.int_value	event_dim.params.value.float_value	event_dim.params.value.double_value	event_dim.timestamp_micros	event_dim.previous_timestamp_micros	event_dim.date
session_start	firebase_event_origin	auto	null	null	null	1465339417400059	1465311874900059	20160607
initialized_rh_api	firebase_event_origin	app	null	null	null	1465339527300159	1465261213800159	20160607
server_error_single	firebase_event_origin	app	null	null	null	1465340038200259	null	20160607
	code	null	2	null	null			
server_error_single	firebase_event_origin	app	null	null	null	1465340062500359	1465340038200359	20160607
	code	null	2	null	null			
server_error_single	firebase_event_origin	app	null	null	null	1465340082500459	1465340062500459	20160607
	code	null	2	null	null			
server_error_single	firebase_event_origin	app	null	null	null	1465340105800559	1465340082500559	20160607
	code	null	2	null	null			
server_error_single	firebase_event_origin	app	null	null	null	1465340130700659	1465340105800659	20160607
	code	null	2	null	null			
server_error_fatal	firebase_event_origin	app	null	null	null	1465340151400759	1465311549100759	20160607
	code	null	2	null	null			
facebook_login_dialog_accepted	firebase_event_origin	app	null	null	null	1465341193100859	1465331163000859	20160607
	source	login_screen_facebook_button	null	null	null			
facebook_account_linked	firebase_event_origin	app	null	null	null	1465341342000959	1465331453800959	20160607
	source	login_screen_facebook_button	null	null	null			
	success	true	null	null	null			

# Firebase

[ 개발자가 정의하지 않아도 자동으로 발생하는 Firebase Event : 디테일은 링크 참고 ]

표시한건 제가 주로 보던 이벤트(애플리케이션마다 자주 볼 이벤트가 다름)

<b>app_remove</b>	애플리케이션 패키지가 삭제되거나 Android 기기에서 '제거'될 때  이 이벤트는 일일 기기 제거 수 및 일일 사용자 제거 수 측정항목과는 다르며 두 측정항목 모두 <a href="#">Google Play Developer Console</a> 에서 보고됩니다. <code>app_remove</code> 이벤트는 애플리케이션 패키지 삭제를 집계하며, 이 집계는 설치 소스에 관계없이 보고서에서 지정한 기간에 따라 달라집니다. 일일 기기 제거 수 및 일일 사용자 제거 수 측정항목은 애플리케이션 패키지가 Google Play에서 설치된 경우에만 애플리케이션 패키지 삭제를 집계하며 하루에 한번 보고됩니다.	<b>in_app_purchase</b>	사용자가 iTunes의 App Store 또는 Google Play에서 최초 구독을 비롯해 인앱 구매를 완료한 때. 제품 ID, 제품 이름, 통화, 수량이 매개변수로 전달됩니다.  Android 앱용 <code>in_app_purchase</code> 데이터를 보려면 <a href="#">Firebase를 Google Play에 연결</a> 해야 합니다.  이 이벤트는 Firebase SDK가 포함된 앱 버전에서만 트리거됩니다. 참고: 유료 앱 구매 수익, 구독 수익(Android만 해당), 환불은 자동으로 추적되지 않습니다. 보고된 수익이 Google Play Developer Console에 표시되는 값과 다를 수도 있습니다. 무효 또는 샌드박스(테스트)로 표시된 이벤트는 무시됩니다. iOS 이벤트만 샌드박스로 신고됩니다. Google Play 결제 테스트에 대해 <a href="#">자세히 알아보세요</a> .
<b>app_update</b>	앱이 새 버전으로 업데이트되고 다시 실행될 때. 이전 앱 버전 ID가 매개변수로 전달됩니다.  이 이벤트는 <a href="#">Google Play Developer Console</a> 에서 보고하는 일일 기기 업그레이드 수 측정항목과 다른 개념입니다. 업그레이드는 애플리케이션 바이너리의 업데이트를 의미하지만, <code>app_update</code> 이벤트는 업그레이드된 앱이 이후에 실행될 때 발생합니다.	<code>notification_dismiss</code>	FCM에서 보낸 알림을 사용자가 닫을 때. Android 앱 전용
<b>dynamic_link_app_open</b>	사용자가 동적 링크를 통해 앱을 다시 열 때	<code>notification_foreground</code>	앱이 포그라운드 상태인 경우 FCM에서 보낸 알림이 수신될 때
<b>dynamic_link_app_update</b>	앱이 새 버전으로 업데이트되고 동적 링크를 통해 열 때. Android 앱 전용	<b>notification_open</b>	FCM에서 보낸 알림을 사용자가 열 때
<b>dynamic_link_first_open</b>	사용자가 동적 링크를 통해 처음으로 앱을 열 때.	<code>notification_receive</code>	앱이 백그라운드 상태인 경우 FCM에서 보낸 알림이 기기에 수신될 때. Android 앱 전용
<b>first_open</b>	앱 설치 또는 재설치 후 사용자가 처음으로 앱을 실행한 시점입니다.  이 이벤트는 사용자가 기기에 앱을 다운로드할 때가 아니라 앱을 처음으로 사용할 때 발생합니다. 순수 다운로드 횟수는 <a href="#">Google Play Developer Console</a> 또는 <a href="#">iTunesConnect</a> 에서 확인하세요.	<b>os_update</b>	기기 운영체제가 새 버전으로 업데이트되었을 때. 이전 운영체제 버전 ID가 매개변수로 전달됩니다.
		<b>screen_view</b>	화면 전환이 발생하고 다음 기준 중 하나가 충족될 때입니다. <ul style="list-style-type: none"><li>• 이전에 설정된 화면이 없음</li><li>• 새 화면 이름이 이전 화면 이름과 다름</li><li>• 새 screen-class 이름이 이전 screen-class 이름과 다름</li><li>• 새 화면 ID가 이전 화면 ID와 다름</li></ul>
		<b>session_start</b>	사용자가 <a href="#">세션 제한 시간</a> 보다 길게 비활성 상태였다가 <a href="#">최소 세션 시간</a> 보다 오래 앱을 사용할 때
		<b>user_engagement</b>	앱이 포그라운드 상태일 때 주기적으로 발생합니다.

# Firebase

[ 개발자가 정의하지 않아도 자동으로 발생하는 Firebase Event : 디테일은 링크 참고 ]

이 이벤트들은 정말 많이 찍힘. 전처리할 때는 제외해서 보기도 함

app_remove	애플리케이션 패키지가 삭제되거나 Android 기기에서 '제거'될 때  이 이벤트는 일일 기기 제거 수 및 일일 사용자 제거 수 측정항목과는 다르며 두 측정항목 모두 <a href="#">Google Play Developer Console</a> 에서 보고됩니다. <code>app_remove</code> 이벤트는 애플리케이션 패키지 삭제를 집계하며, 이 집계는 설치 소스에 관계없이 보고서에서 지정한 기간에 따라 달라집니다. 일일 기기 제거 수 및 일일 사용자 제거 수 측정항목은 애플리케이션 패키지가 Google Play에서 설치된 경우에만 애플리케이션 패키지 삭제를 집계하며 하루에 한번 보고됩니다.	in_app_purchase	사용자가 iTunes의 App Store 또는 Google Play에서 최초 구독을 비롯해 인앱 구매를 완료한 때. 제품 ID, 제품 이름, 통화, 수량이 매개변수로 전달됩니다.  Android 앱용 <code>in_app_purchase</code> 데이터를 보려면 <a href="#">Firebase를 Google Play에 연결</a> 해야 합니다.  이 이벤트는 Firebase SDK가 포함된 앱 버전에서만 트리거됩니다. 참고: 유료 앱 구매 수익, 구독 수익(Android만 해당), 환불은 자동으로 추적되지 않습니다. 보고된 수익이 Google Play Developer Console에 표시되는 값과 다를 수도 있습니다. 무효 또는 샌드박스(테스트)로 표시된 이벤트는 무시됩니다. iOS 이벤트만 샌드박스로 신고됩니다. Google Play 결제 테스트에 대해 <a href="#">자세히 알아보세요</a> .
app_update	앱이 새 버전으로 업데이트되고 다시 실행될 때. 이전 앱 버전 ID가 매개변수로 전달됩니다.  이 이벤트는 <a href="#">Google Play Developer Console</a> 에서 보고하는 일일 기기 업그레이드 수 측정항목과 다른 개념입니다. 업그레이드는 애플리케이션 바이너리의 업데이트를 의미하지만, <code>app_update</code> 이벤트는 업그레이드된 앱이 이후에 실행될 때 발생합니다.	notification_dismiss	FCM에서 보낸 알림을 사용자가 닫을 때. Android 앱 전용
dynamic_link_app_open	사용자가 동적 링크를 통해 앱을 다시 열 때	notification_foreground	앱이 포그라운드 상태인 경우 FCM에서 보낸 알림이 수신될 때
dynamic_link_app_update	앱이 새 버전으로 업데이트되고 동적 링크를 통해 열 때. Android 앱 전용	notification_open	FCM에서 보낸 알림을 사용자가 열 때
dynamic_link_first_open	사용자가 동적 링크를 통해 처음으로 앱을 열 때.	notification_receive	앱이 백그라운드 상태인 경우 FCM에서 보낸 알림이 기기에 수신될 때. Android 앱 전용
first_open	앱 설치 또는 재설치 후 사용자가 처음으로 앱을 실행한 시점입니다.  이 이벤트는 사용자가 기기에 앱을 다운로드할 때가 아니라 앱을 처음으로 사용할 때 발생합니다. 순수 다운로드 횟수는 Google Play Developer Console 또는 iTunesConnect에서 확인하세요.	os_update	기기 운영체제가 새 버전으로 업데이트되었을 때. 이전 운영체제 버전 ID가 매개변수로 전달됩니다.
		screen_view	화면 전환이 발생하고 다음 기준 중 하나가 충족될 때입니다. <ul style="list-style-type: none"><li>• 이전에 설정된 화면이 없음</li><li>• 새 화면 이름이 이전 화면 이름과 다름</li><li>• 새 screen-class 이름이 이전 screen-class 이름과 다름</li><li>• 새 화면 ID가 이전 화면 ID와 다름</li></ul>
		session_start	사용자가 <a href="#">세션 제한 시간</a> 보다 길게 비활성 상태였다가 <a href="#">최소 세션 시간</a> 보다 오래 앱을 사용할 때
		user_engagement	앱이 포그라운드 상태일 때 주기적으로 발생합니다.

# Firebase

## [ 게임 round 완료한 이벤트 ]

### [ Input(원본 데이터) ]

event_dim.name	event_dim.params.key	event_dim.params.value.string_value	event_dim.params.value.int_value	event_dim.params.value.float_value	event_dim.params.value.double_value	event_dim.timestamp_micros	user_dim.first_open_timestamp_micros	user_dim.user_properties.key	user_dim.user_properties.value.value.string_value
round_completed	round	null	0	null	null	1465325302500059	1465336143650000	api_version	1.2
	social_bot	false		null	null			user_id	null
	type_of_game	solo		null	null			powers	0
	rematch	false		null	null			elite_powers	0
	firebase_event_origin	app		null	null			chips	39
	bingos_claimed	null	1	null	null			first_open_time	null
	power_ups_used	null	0	null	null			is_signed_in	true
	number_cards	null	2	null	null			coins	6620
	xp_awarded	null	25	null	null			language	en_US
	coins_daubed	null	8	null	null			platform	android
	originator	true		null	null			keys	2
	score	null	9500	null	null			xp	740
	won	false		null	null				
	bot	false		null	null				
	fb_friends	false		null	null				
	squares_daubed	null	20	null	null				
	coins_awarded	null	165	null	null				

### [ 우리가 예상하는 Output ]

Row	user_id	name	timestamp_micros	api_version	round	type_of_game
1	null	round_completed	1465331929100038	1.2	0	solo
2	null	round_completed	1465339643400059	1.2	0	solo
3	null	round_completed	1465324548080009	1.2	0	solo
4	null	round_completed	1465277842100054	1.2	0	solo
5	null	round_completed	1465294904000084	1.2	0	solo
6	null	round_completed	1465294234400127	1.2	0	solo
7	null	round_completed	1465294104300045	1.2	1	social
8	null	round_completed	1465262850900051	1.2	2	social
9	null	round_completed	1465273915500056	1.2	0	solo
10	null	round_completed	1465306842000034	1.2	1	social

# Firebase

---

[ 샘플 쿼리 #1 : 나중에 사용하기 쉽도록 UDF로 제공 ]

```
# UDF for event parameters
CREATE TEMP FUNCTION paramValueByKey(k STRING, params ARRAY<STRUCT<key STRING, value STRUCT<string_value
STRING, int_value INT64, float_value FLOAT64, double_value FLOAT64 >>>) AS (
  (SELECT x.value FROM UNNEST(params) x WHERE x.key=k)
);

# UDF for user properties
CREATE TEMP FUNCTION propertyValueByKey(k STRING, properties ARRAY<STRUCT<key STRING, value
STRUCT<string_value STRING, int_value INT64, float_value FLOAT64, double_value FLOAT64, set_timestamp_micros INT64
>>>) AS (
  (SELECT x.value FROM UNNEST(properties) x WHERE x.key=k)
);

# Query : 'api_version', 'round', 'type_of_game'
SELECT
  user_id,
  event_name,
  event_timestamp,
  propertyValueByKey('api_version', user_properties).string_value AS api_version,
  paramValueByKey('round', event_params).string_value AS round,
  paramValueByKey('type_of_game', event_params).string_value as type_of_game
from `dataset.table`
LIMIT 10
```

# Firebase

---

[ 샘플 쿼리 #2 : 서브 쿼리 사용 ]  
앞 쿼리와 동일한 결과

```
SELECT
    user_id,
    event_name,
    event_timestamp,
    (SELECT properties.value.string_value FROM UNNEST(user_properties) AS properties where properties.key='api_version')
AS api_version,
    (SELECT params.value.string_value FROM UNNEST(event_params) AS params WHERE params.key='round') AS round,
    (SELECT params.value.string_value FROM UNNEST(event_params) AS params WHERE params.key='type_of_game') AS type_of_game
from `dataset.table`
LIMIT 10
```

# Firebase

[ CROSS JOIN UNNEST 시각화 ]

<table><thead><tr><th>crew</th></tr></thead><tbody><tr><td>Kaylee</td></tr><tr><td>Zoe</td></tr><tr><td>Jayne</td></tr></tbody></table>	crew	Kaylee	Zoe	Jayne	ship_name	wingspan	crew_member
crew							
Kaylee							
Zoe							
Jayne							
<table><thead><tr><th>crew</th></tr></thead><tbody><tr><td>Kaylee</td></tr><tr><td>Zoe</td></tr><tr><td>Jayne</td></tr></tbody></table>	crew	Kaylee	Zoe	Jayne	Serenity	32.4	Kaylee
crew							
Kaylee							
Zoe							
Jayne							
<table><thead><tr><th>crew</th></tr></thead><tbody><tr><td>Kaylee</td></tr><tr><td>Zoe</td></tr><tr><td>Jayne</td></tr></tbody></table>	crew	Kaylee	Zoe	Jayne	Serenity	32.4	Zoe
crew							
Kaylee							
Zoe							
Jayne							
<table><thead><tr><th>crew</th></tr></thead><tbody><tr><td>Kaylee</td></tr><tr><td>Zoe</td></tr><tr><td>Jayne</td></tr></tbody></table>	crew	Kaylee	Zoe	Jayne	Serenity	32.4	Jayne
crew							
Kaylee							
Zoe							
Jayne							

```
SELECT * FROM `starships` CROSS JOIN UNNEST (crew) AS crew_member
```

<https://medium.com/firebase-developers/using-the-unnest-function-in-bigquery-to-analyze-event-parameters-in-analytics-fb828f890b42>

# Firebase

---

[ UNNEST 시각화 ]

event_name “level_complete_ quickplay”	event_params key: “value” value:(...) key: “board” value:(...) key: “fire_” value:(...)	event_timestamp 153137344448100
--	--	------------------------------------

<https://medium.com/firebase-developers/using-the-unnest-function-in-bigquery-to-analyze-event-parameters-in-analytics-fb828f890b42>

# Firebase

---

[ 여러분 어려우시죠? ]

Firebase 데이터에 바로 쿼리를 날리는 것은 아래와 같은 이슈

- #1. **UNNEST**, 서브 쿼리 등을 이해해야 해서 초심자에겐 **어려울 수 있음**
- #2. Firebase에서 파라미터을 포함한 쿼리를 날리는 경우 **비용이 큼**

# Firebase

---

[ 여러분 어려우시죠? ]

Firebase 데이터에 바로 쿼리를 날리는 것은 아래와 같은 이슈

- #1. **UNNEST**, 서브 쿼리 등을 이해해야 해서 초심자에겐 **어려울 수 있음**
- #2. Firebase에서 파라미터를 포함한 쿼리를 날리는 경우 **비용이 큼**

**데이터 담당자가 Flatten해서 매일 저장하고(=ARRAY 개념없이 Table 형태로 저장)**

다른 분들은 저장한 테이블만 사용하면 해결!

(비용과 난이도 등에서 이점! 관리의 cost가 있지만.. ~~한명만 고생하면~~)

# Data Pipeline

# 파이프라인 예시

---

[ 파이프라인은 회사마다 다르게 구성 가능 ]

하지만 Firebase 앱 데이터는 꼭 가공해서 사용하는 것을 추천(비용 이슈)

<https://www.slideshare.net/zzsza/little-big-data-1>

## 바닥부터 시작하는 데이터 인프라

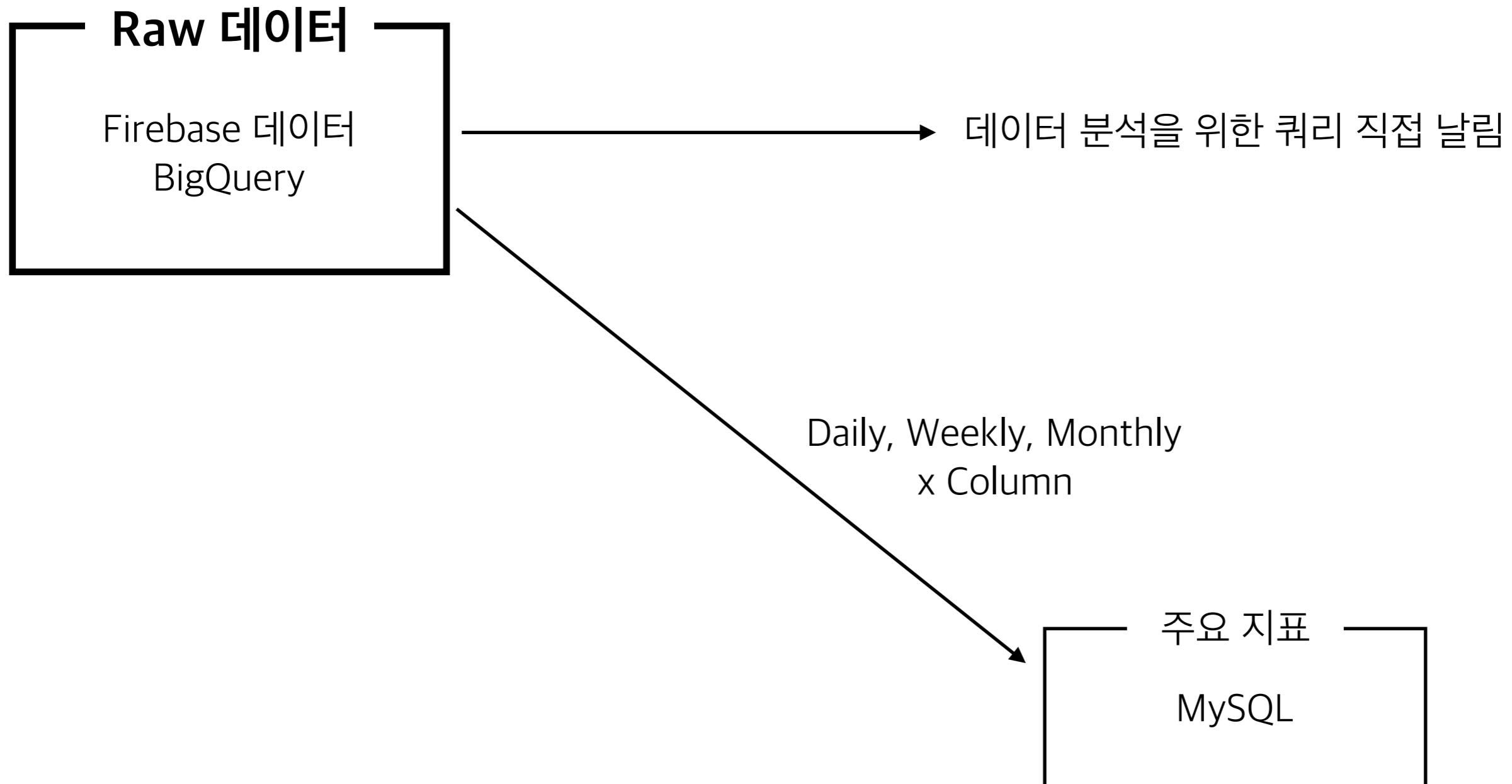
- 변성윤 -



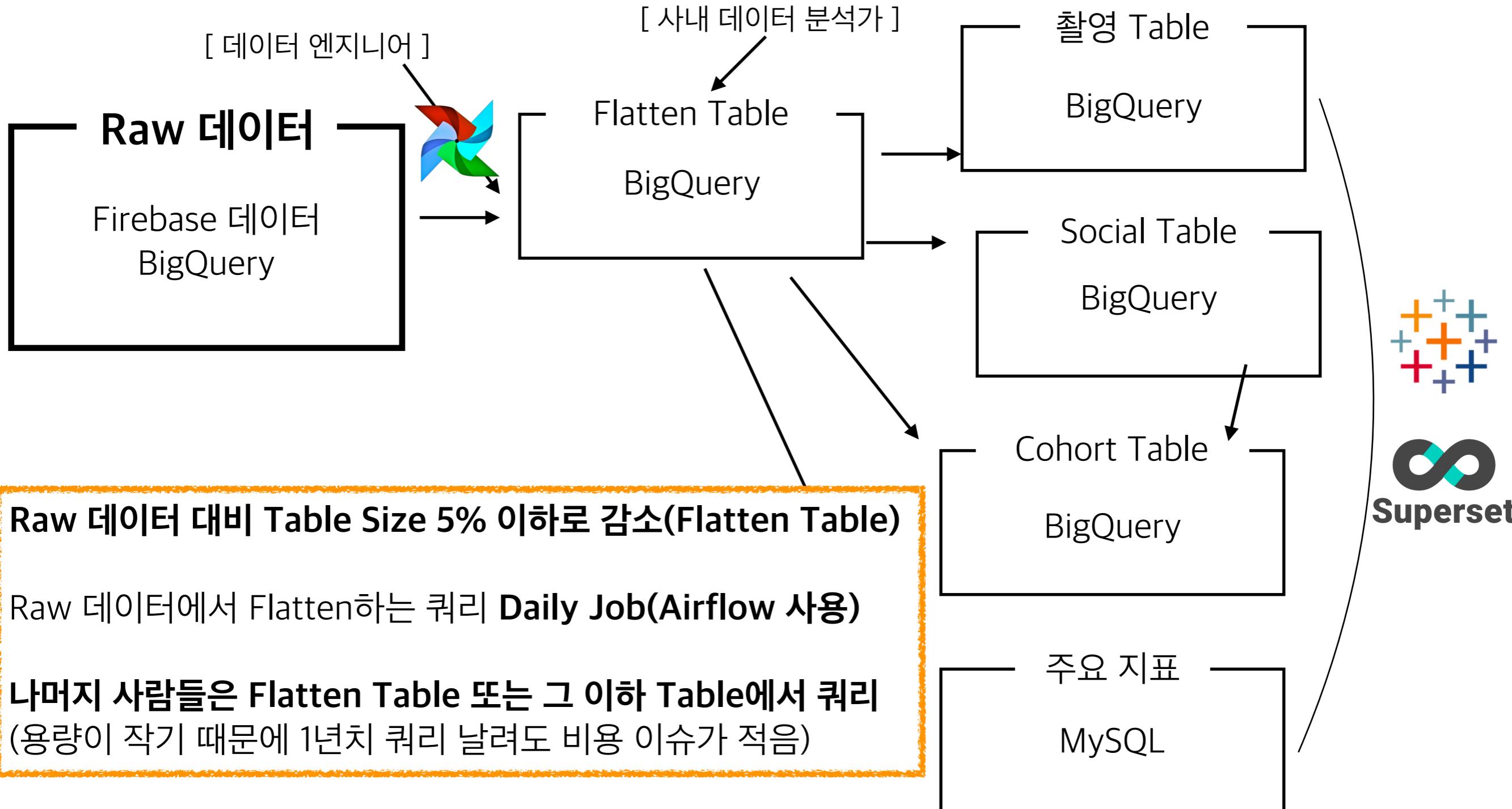
# 레트리카 입사 당시 데이터 파이프라인

---

(쿼리 비용으로 월 5,000달러 나온 적 있음)

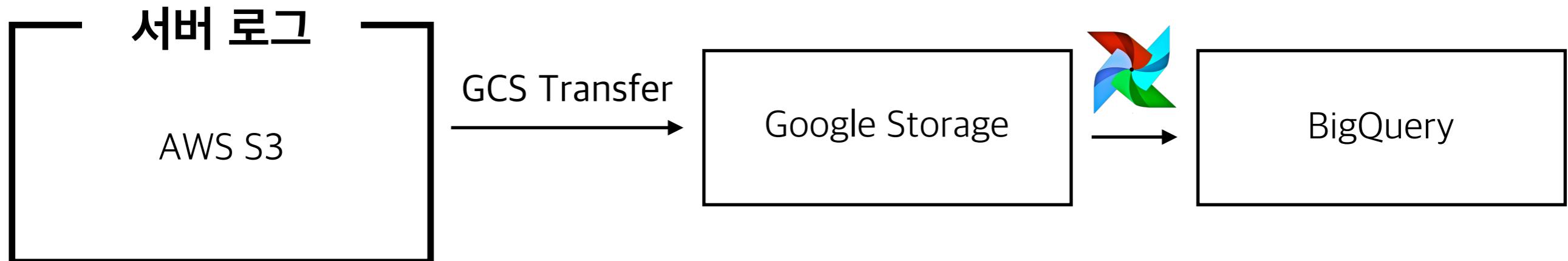


# 레트리카에서 개선한 데이터 파이프라인



# 파이프라인 예시(1)

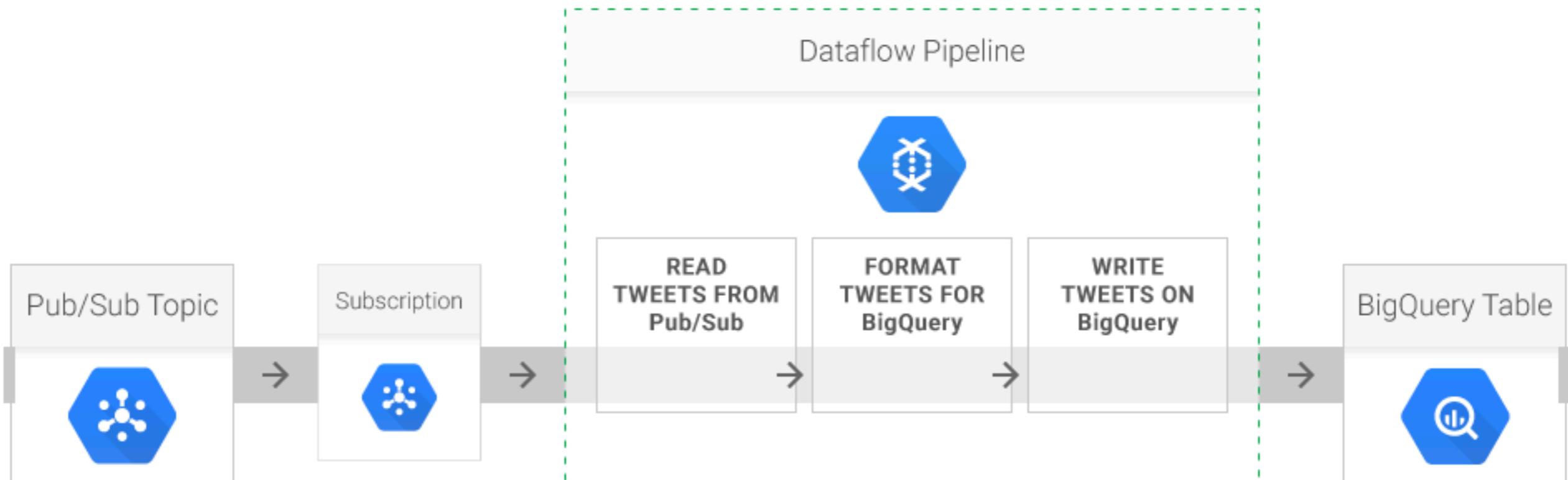
---



(너무 용량이 많으면 Athena에서 쿼리 날린 후 저장)

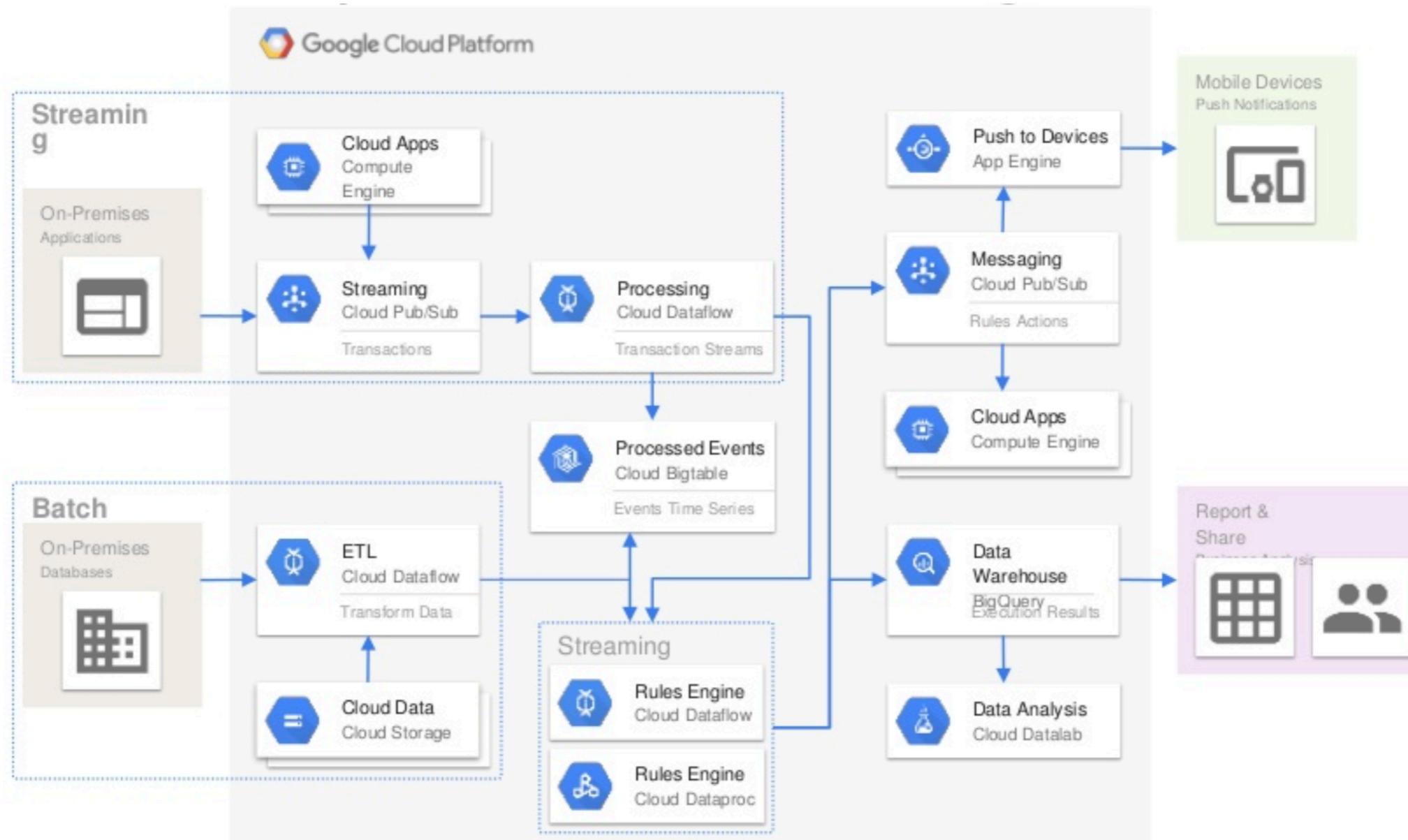
# 파이프라인 예시(2)

[ Pub/Sub + Dataflow ]



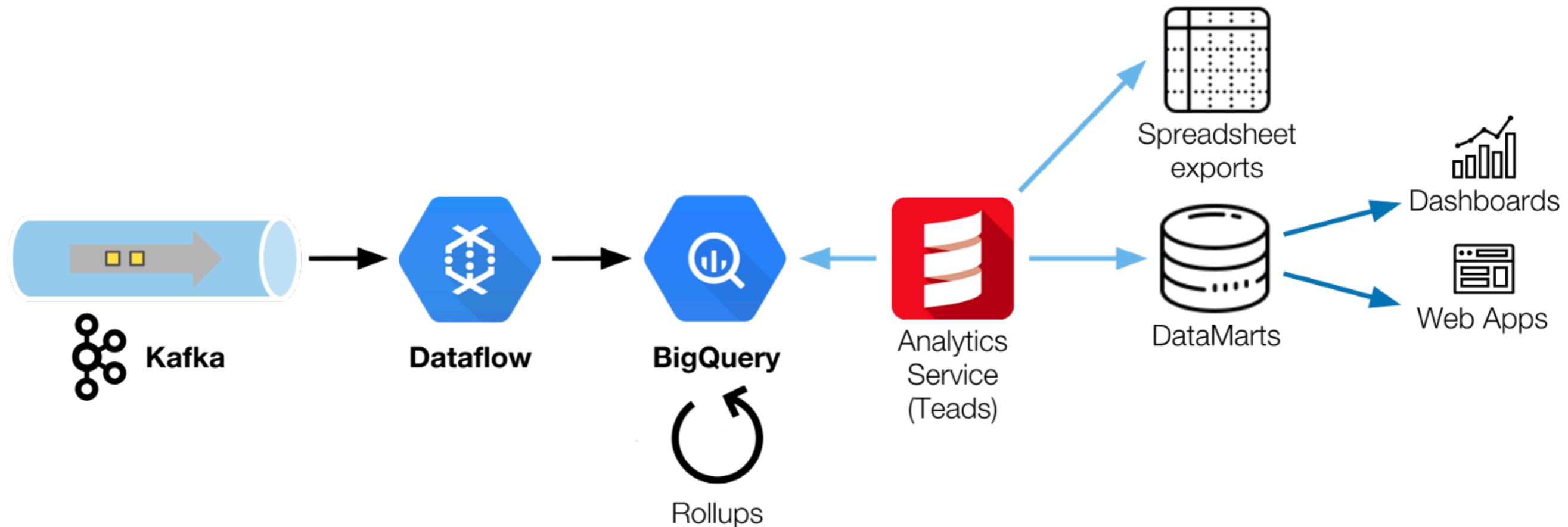
# 파이프라인 예시(3)

[ Streaming + Batch ]



# 파이프라인 예시(4)

[ Kafka + Dataflow ]



# 파이프라인 예시(5)

---

[ Airflow ]



# 정리

---

핵심은 분석할 데이터를 모두 BigQuery에 모아두는 것

처음부터 화려한 ETL을 구성할 필요 없고,

현재 리소스에서 간단히 할 수 있는 것부터 만들고  
하나씩 개선!

(+ Firebase 이벤트 로그 정리)

(+ 데이터 스키마 정리)

(+ 데이터 기반으로 대화하는 문화)

# Python에서 BigQuery 사용하기

# Python에서 BigQuery 사용하기

---

## [ Service Key 사용하는 방법 ]

```
import pandas as pd

query = "SELECT * FROM `dataset.table`"
df = pd.read_gbq(query=query, project_id="project_id",
                  dialect='standard', private_key='<your json key path>')
```

# Python에서 BigQuery 사용하기

---

[ 서비스키 없이 USER\_ACCOUNT로 인증해서 사용 ]

장점 : Service Key 발급해서 권한주는 일을 GCP IAM으로 관리 가능

```
pip3 install pydata_google_auth
```

```
import pandas as pd  
import pydata_google_auth
```

```
SCOPES = [  
    'https://www.googleapis.com/auth/cloud-platform',  
    'https://www.googleapis.com/auth/drive',  
    'https://www.googleapis.com/auth/bigquery'  
]
```

```
credentials = pydata_google_auth.get_user_credentials(SCOPES, auth_local_webserver=True)
```

```
query = "SELECT * FROM `dataset.table`"  
df = pd.read_gbq(query=query, project_id="project_id", credentials=credentials)
```

# Python에서 BigQuery 사용하기

---

[ (개인적인) 머신러닝 Flow ]

→ ex: 과거 이용 횟수, mean encoding 등

[ EDA(데이터 탐색) ] ⇒ [ BigQuery에서 만들기 쉬운 Feature ] ⇒ [ pd.read\_gbq로 데이터 추출 ]

⇒ [ Python에서 만들기 쉬운 Feature ] ⇒ [ 모델링 ]

→ ex: one-hot encoding

#1. BigQuery의 전처리 퍼포먼스 > Python 전처리 퍼포먼스

#2. BigQuery에서 처리하기 어려운 것들은 Python에서!

# 데이터셋 공유하기

# 데이터셋 공유하기

[ 데이터셋을 공유하고 싶은 경우 ]

BigQuery 기능 및 정보 단축키 새 쿼리 작성

쿼리 기록 저장되지 않은 쿼리 수정됨 편집기 숨기기 전체 화면

저장된 쿼리

작업 기록

전송

예약된 쿼리

BI Engine

리소스 + 데이터 추가

표 및 데이터세트 검색

noted-tide-156308

- babynames
- kaggle\_taxi
- kaggle\_Web\_Traffic\_Time\_Seri...
- kaggle\_zillow\_2nd**
- test\_data aleprice
- zillow\_prize

bigquery-public-data

설명 라벨

없음 없음

데이터세트 정보

데이터세트 ID	noted-tide-156308:kaggle_zillow_2nd
생성 시간	2018. 2. 2. 오후 5:42:48
기본 표 만료	사용 안함
최종 수정 시간	2018. 2. 2. 오후 5:42:48
데이터 위치	US

+ 테이블 만들기 데이터세트 공유 데이터세트 삭제

실행 쿼리 저장 보기 저장 쿼리 예약 더보기

데이터세트 공유



# 데이터셋 공유하기

[ 데이터 뷰어로 설정하면 데이터셋 아래 모든 테이블이 보임 ]  
다른 데이터셋은 여전히 안보임

## 데이터세트 권한

데이터세트에 대한 액세스 권한을 부여하려면 구성원을 추가하고 IAM(ID 및 액세스 관리) 역할을 할당하여 액세스 수준을 지정하세요. 여러 역할을 지정할 수 있습니다.

더 이상 콘솔에서 ACL을 설정하여 액세스 권한을 관리할 수 없습니다. IAM과 ACL이 어떻게 관련되어 있는지 알아보려면 [문서](#)를 참조하세요.

[데이터세트 권한](#)     [승인된 보기](#)

The screenshot shows the 'Dataset Permissions' page for a dataset named 'BigQuery 데이터 뷰어'. A dropdown menu is open under the 'Role' column for the user 'kyle@socar.kr', showing the following options:

- 선택됨
- BigQuery 데이터 뷰어
- 소유자
- 편집자
- 뷰어
- BigQuery 관리자
- BigQuery 데이터 소유자
- BigQuery 데이터 편집자
- BigQuery 메타데이터 뷰어
- BigQuery 사용자
- 보안 검토자

The 'BigQuery 데이터 뷰어' role is highlighted with a yellow border. Below the dropdown, there are several other roles listed with their descriptions:

- BigQuery 데이터 뷰어(구성원 3명)
- BigQuery 데이터 소유자(구성원 4명)
- BigQuery 데이터 편집자(구성원 2명)
- BigQuery 사용자(구성원 1명)

At the bottom of the page are two buttons: '완료' (Finish) and '취소' (Cancel).

# 정리

# 더 알면 좋을 내용들(키워드 위주)

---

## BigQuery ML

- SQL로 머신러닝을 할 수 있음
- 현재(19.05.15 기준) Linear regression, Logistic Regression, Kmeans
- <https://cloud.google.com/bigquery/docs/bigqueryml?hl=ko>

## BigQuery GIS

- 좌표 데이터를 처리할 때 유용
- 공공 데이터에서 geojson, shp 파일 등을 구해서 테이블에 넣고 좌표 <=> 주소 변환 가능
- <https://cloud.google.com/bigquery/docs/gis?hl=ko>

## BigQuery Data Transfer

- Google Ad Manager, YouTube 데이터를 정기적으로 로드
- <https://cloud.google.com/bigquery/docs/dts?hl=ko>

## Optimizing query performance

- 쿼리를 어떻게 짜느냐에 따라 데이터가 나오는 속도가 결정
- <https://cloud.google.com/bigquery/docs/best-practices-performance-overview>

# 더 알면 좋을 내용들(키워드 위주)

---

## Google Cloud Platform Dataflow

- 데이터 전처리시 유용. Spark도 있지만 Dataflow도 편함
- <https://cloud.google.com/dataflow/?hl=ko>

## BigQuery Quota

- 할당량 및 한도 체크!
- <https://cloud.google.com/bigquery/quotas?hl=ko>

## Stackdriver를 사용한 BigQuery 모니터링

- 사내에서 누가 BigQuery를 많이 사용하고 있는지 확인!
- <https://cloud.google.com/bigquery/docs/monitoring?hl=ko>

## BigQuery Access Control

- Role에 따라 어떤 Permission인지 확인! (IAM과도 연관)
- <https://cloud.google.com/bigquery/docs/access-control>

## 릴리즈 노트

- BigQuery 업데이트가 매우 빠르게 진행되고 있음
- <https://cloud.google.com/bigquery/docs/release-notes>

## Airflow

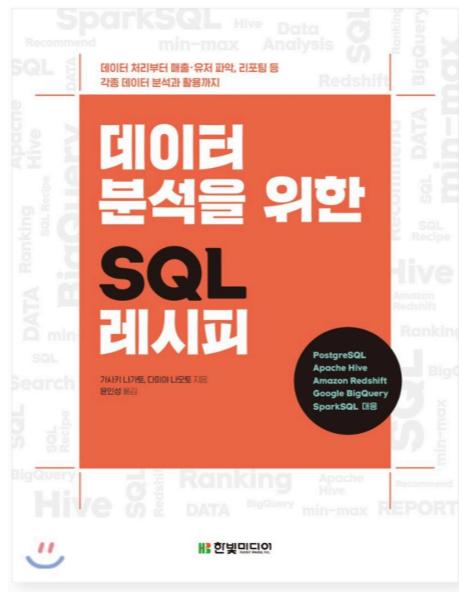
- 스케줄링 돌릴 때 매우 좋음! GCP랑 호환이 매우 좋음

# 추천 도서

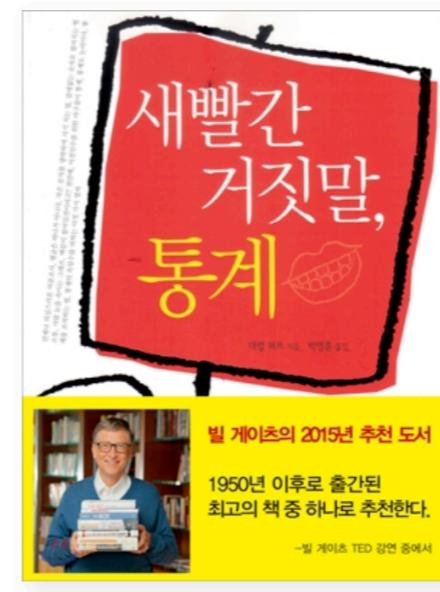
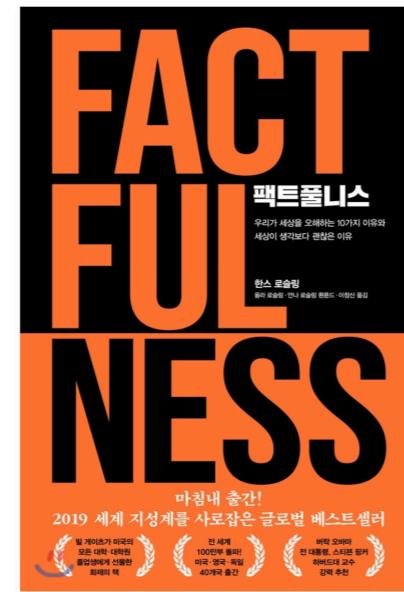
[ SQL 입문 도서 ]



[ SQL 심화 도서 ]



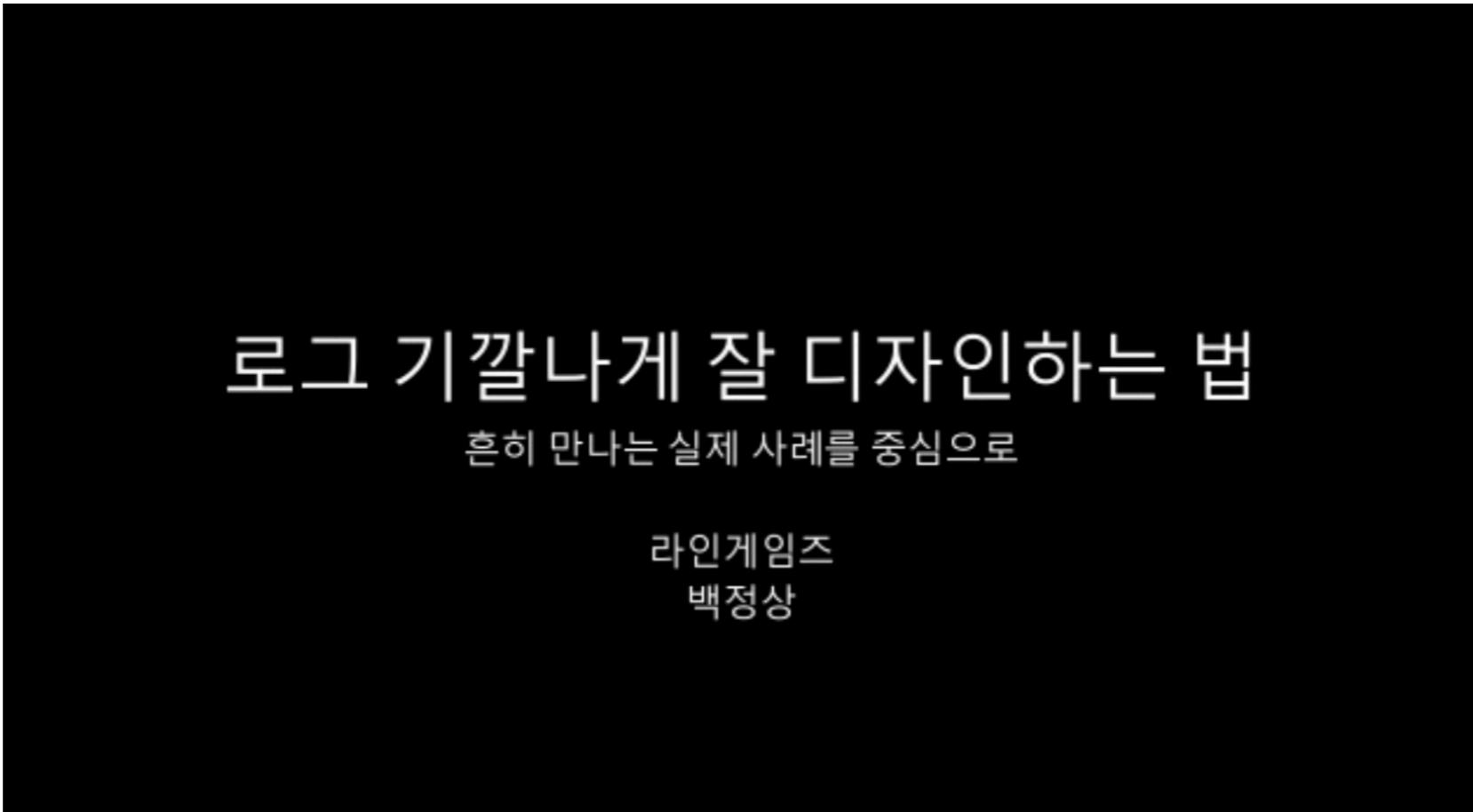
난이도 있음, 하지만 이 책에 있는 내용 마스터하면  
이제 당당히 나 SQL 잘해<- 라고 하셔도 됩니다!  
BigQuery도 있음



데이터 분석을  
더 논리적으로 잘하기 위해  
읽으면 좋은 책들

# 추천 자료

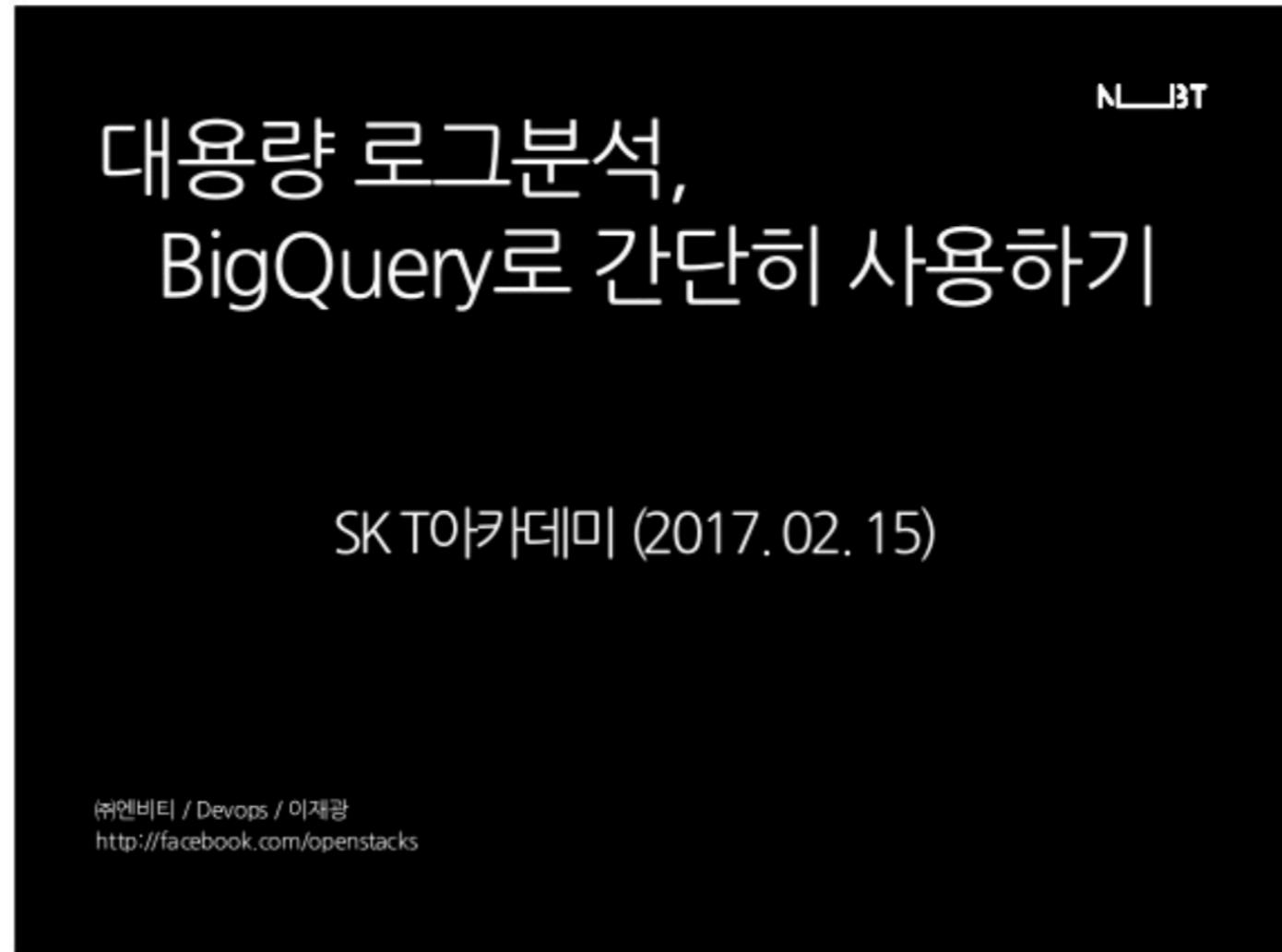
---



백정상님 - 로그 기깔나게 잘 디자인하는 법  
(<https://www.slideshare.net/jeongsangbaek/ss-80795259>)

# 추천 자료

---



이재광님 - 대용량 로그분석, BigQuery로 간단히 사용하기  
[\(<https://www.slideshare.net/openstacks/bigquery-20170215-t>\)](https://www.slideshare.net/openstacks/bigquery-20170215-t)

# 정리

## [ Query 총 정리 ]

```
query_statement:  
  [ WITH with_query_name AS ( query_expr ) [ , ... ] ]  
  
query_expr:  
  { select | ( query_expr ) | query_expr set_op query_expr }  
  [ ORDER BY expression [ { ASC | DESC } ] [ , ... ] ]  
  [ LIMIT count [ OFFSET skip_rows ] ]  
  
select:  
  SELECT [ { ALL | DISTINCT } ]  
    { [ expression. ]* [ EXCEPT ( column_name [ , ... ] ) ]  
      [ REPLACE ( expression [ AS ] column_name [ , ... ] ) ]  
      | expression [ [ AS ] alias ] } [ , ... ]  
    [ FROM from_item [ , ... ] ]  
    [ WHERE bool_expression ]  
    [ GROUP BY { expression [ , ... ] | ROLLUP ( expression [ , ... ] ) } ]  
    [ HAVING bool_expression ]  
    [ WINDOW window_name AS ( window_definition ) [ , ... ] ]  
  
set_op:  
  UNION { ALL | DISTINCT } | INTERSECT DISTINCT | EXCEPT DISTINCT  
  
from_item: {  
  table_name [ [ AS ] alias ] [ FOR SYSTEM TIME AS OF timestamp_expression ] |  
  join |  
  ( query_expr ) [ [ AS ] alias ] |  
  field_path |  
  { UNNEST( array_expression ) | UNNEST( array_path ) | array_path }  
  [ [ AS ] alias ] [ WITH OFFSET [ [ AS ] alias ] ] |  
  with_query_name [ [ AS ] alias ]  
}  
  
join:  
  from_item [ join_type ] JOIN from_item  
  [ { ON bool_expression | USING ( join_column [ , ... ] ) } ]  
  
join_type:  
  { INNER | CROSS | FULL [OUTER] | LEFT [OUTER] | RIGHT [OUTER] }
```

- 대괄호 '[']'는 절(선택사항)을 의미합니다.
- 괄호 '(' ')'는 리터럴 괄호를 의미합니다.
- 세로 막대 '|'는 논리 OR을 의미합니다.
- 중괄호 '{}'는 옵션 조합을 묶는데 사용됩니다.
- 꺎쇠 괄호 '[ ... ]' 안에 있는 쉼표 다음에 오는 줄임표는 앞의 항목이 쉼표로 구분된 목록으로 반복될 수 있음을 나타냅니다.

# 정리

---

[ 여러 분들이 이런 생각을 하시면 성공한 발표 ]

- #1. 이제 우리 데이터 어떻게 생겼는지 확인해야지라고 생각
- #2. Firebase 데이터를 주기적으로 Flatten해야지
- #3. 자주 쓰는 쿼리는 Notion에 공유해야지
- #4. 데이터 엔지니어링쪽을 개선해야지
- #5. 평소 일하며 궁금한 내용을 쿼리로 직접 짜봐야지
- #6. 생소해서 어렵지만 그래도 엄청 어렵진 않은 것 같다
- #7. 데이터 기반으로 이야기하는 문화 만들어야지

# 정리

---

SQL 자체는 난이도가 쉬운 편입니다  
많이 쿼리 날리시면 실력이 상승합니다

중요한 것은

- 데이터로 대화하는 문화
- 데이터를 편하게 볼 수 있는 데이터 인프라

# 정리

---

감사합니다!

# 정리

---

쏘카 / VCNC에서 인재 언제나 채용합니다!  
(데이터 분석, 데이터 엔지니어, 머신러닝 엔지니어, 클라이언트 개발자, 서버 개발자 등 모두!)

상담 원하시면 [kyle@socar.kr](mailto:kyle@socar.kr)로 메일을!



☰☰☰

★ Between

# Thanks to

---

발표 자료 피드백 주신

WireBarley 허진한님  
쏘카 데이터그룹 이민우님