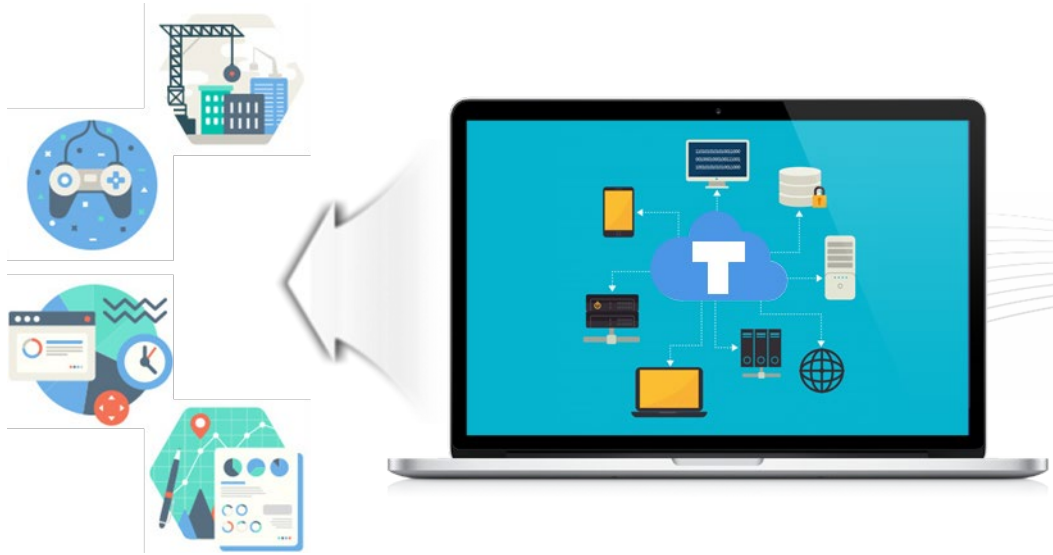


(이론) GCP-3차수

Ⅰ SQL 기초

Ⅱ BigQuery

Ⅲ BigQuery 최적화

SQL 이란?

- SQL(Structured Query Language)은 관계형 데이터베이스 관리 시스템(RDBMS)에서 데이터를 관리하고 조작하기 위한 표준화된 프로그래밍 언어
- SQL을 사용하면 데이터베이스에서 데이터를 저장, 검색, 수정, 삭제할 수 있으며, 데이터베이스 구조를 정의하고 관리할 수 있음

SQL 주요 특징	
비절차적 언어	• 원하는 결과를 명시하면 시스템이 실행 방법을 결정
표준화된 문법	• ANSI SQL 표준을 따름
다양한 데이터베이스 시스템 지원	• MySQL, PostgreSQL, Oracle, SQL Server 등
강력한 데이터 조작 기능	• 복잡한 쿼리와 대용량 데이터 처리 가능

[표 1] sql 주요 특징

SQL의 주요 구성 요소	
DDL **(Data Definition Language)**	데이터베이스 구조 정의 • CREATE: 데이터베이스 객체 생성 • ALTER: 객체 구조 변경 • DROP: 객체 삭제 • TRUNCATE: 테이블 데이터 전체 삭제
DML **(Data Manipulation Language)**	데이터 컨트롤 • SELECT: 데이터 조회 • INSERT: 새 데이터 삽입 • UPDATE: 기존 데이터 수정 • DELETE: 데이터 삭제
DCL **(Data Control Language)**	데이터 접근 권한 관리 • GRANT: 사용자에게 권한 부여 • REVOKE: 사용자의 권한 철회

[표 2] sql 주요 특징

SQL 기본 명령어	
SELECT	• 데이터 조회 • SELECT column1, column2 FROM table_name WHERE condition;
INSERT	• 새 데이터 삽입 • INSERT INTO table_name (column1, column2) VALUES (value1, value2);
UPDATE	• 기존 데이터 수정 • UPDATE table_name SET column1 = value1 WHERE condition;
DELETE	• 데이터 삭제 • DELETE FROM table_name WHERE condition;
CREATE TABLE	• 새 테이블 생성 • CREATE TABLE table_name (column1 datatype, column2 datatype, ...);
ALTER TABLE	• 테이블 구조 변경 • ALTER TABLE table_name ADD column_name datatype;
DROP TABLE	• 테이블 삭제 • DROP TABLE table_name;

[표 3] sql 기본 명령어

SQL 데이터 타입	
숫자형	• INT: 정수 • FLOAT: 부동 소수점 • DECIMAL: 고정 소수점
문자형	• CHAR: 고정 길이 문자열 • VARCHAR: 가변 길이 문자열 • TEXT: 대용량 텍스트
날짜/시간형	• DATE: 날짜 • TIME: 시간 • DATETIME: 날짜와 시간 • TIMESTAMP: 타임스탬프
기타	• BOOLEAN: 참/거짓 • BLOB: 이진 대용량 객체

[표 4] sql 기본 명령어

I SQL 기초 - 제약 조건

SQL 제약 조건	
PRIMARY KEY	• 고유 식별자
FOREIGN KEY	• 다른 테이블과의 관계 정의
UNIQUE	• 중복 값 방지
NOT NULL	• NULL 값 방지
CHECK	• 특정 조건 충족 확인
DEFAULT	• 기본값 설정

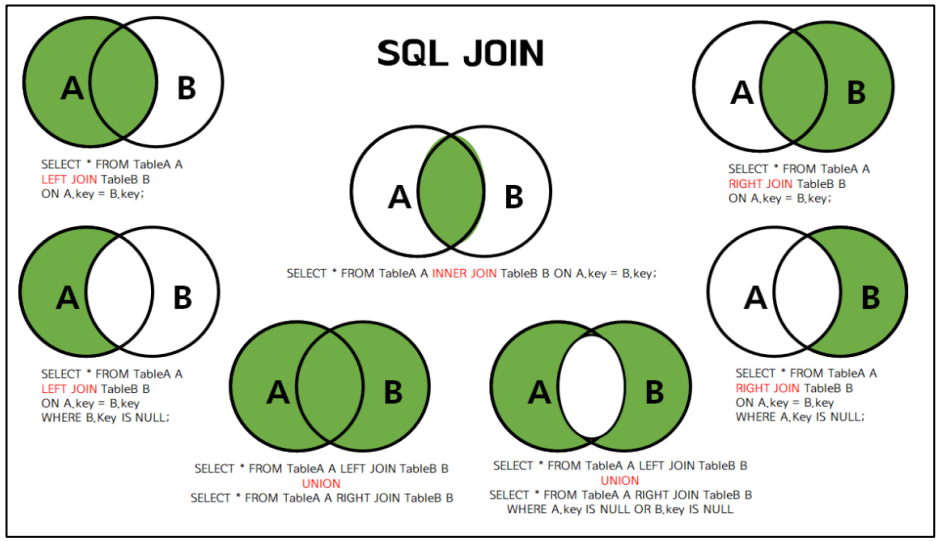
[표 5] sql 제약 조건

```sql
CREATE TABLE orders (
  id INT PRIMARY KEY,
  user_id INT,
  total DECIMAL(10, 2) NOT NULL,
  order_date DATE DEFAULT CURRENT_DATE,
  FOREIGN KEY (user_id) REFERENCES users(id),
  CHECK (total > 0)
);
```

[그림 1] sql 제약 조건 예시

SQL JOIN	
INNER JOIN	• 두 테이블에서 일치하는 행만 반환
LEFT JOIN	• 왼쪽 테이블의 모든 행과 오른쪽 테이블의 일치하는 행 반환
RIGHT JOIN	• 오른쪽 테이블의 모든 행과 왼쪽 테이블의 일치하는 행 반환
FULL OUTER JOIN	• 두 테이블의 모든 데이터를 포함하여 결합, 양쪽 테이블 모두에 데이터가 없는 경우에는 NULL 값을 채워서 결과를 반환

[표 6] sql join



[그림 2] sql join 집합 구조

함수(Function)란?

- 보통 값을 계산하고, 결과 변환을 위해 함수 사용
- 대부분 구성이 프로시저(Procedure)와 유사하지만 IN 파라미터만 사용
- 반드시 반환될 값의 데이터 타입을 RETURN 문에 선언
- 또한 PL/SQL 블록 내에서 RETURN 문을 통해서 반드시 값을 반환

SQL 함수	
집계 함수	• COUNT(), SUM(), AVG(), MAX(), MIN()
문자열 함수	• CONCAT(), SUBSTRING(), UPPER(), LOWER()
날짜 함수	• DATE(), YEAR(), MONTH(), DAY()
수학 함수	• ROUND(), ABS(), CEILING(), FLOOR()

[표 7] sql 함수

```
CREATE OR REPLACE FUNCTION func_name                -- Function(함수) 선언
(var_nm1 IN VARCHAR2, var_nm2 IN NUMBER, ...)       -- 매개 변수 선언

    RETURN datatype                                 -- 리턴 타입 선언

IS

BEGIN                                               -- 실행부

    RETURN res;                                     -- 리턴 값 명시

END;
```

[그림 3] 함수 생성 구조

```
CREATE OR REPLACE FUNCTION fn_info
(name IN VARCHAR2, age IN NUMBER)
RETURN VARCHAR2

IS

name_result VARCHAR2(300);   --OR name_result TABLE.COLUMN%TYPE;
age_result NUMBER;

BEGIN
    SELECT name, age
           INTO name_result, age_result
           FROM DUAL;

    RETURN '이름 : ' || name_result || ', 나이 : ' ||  age_result;

END;
```

[그림 4] 함수 생성 예제

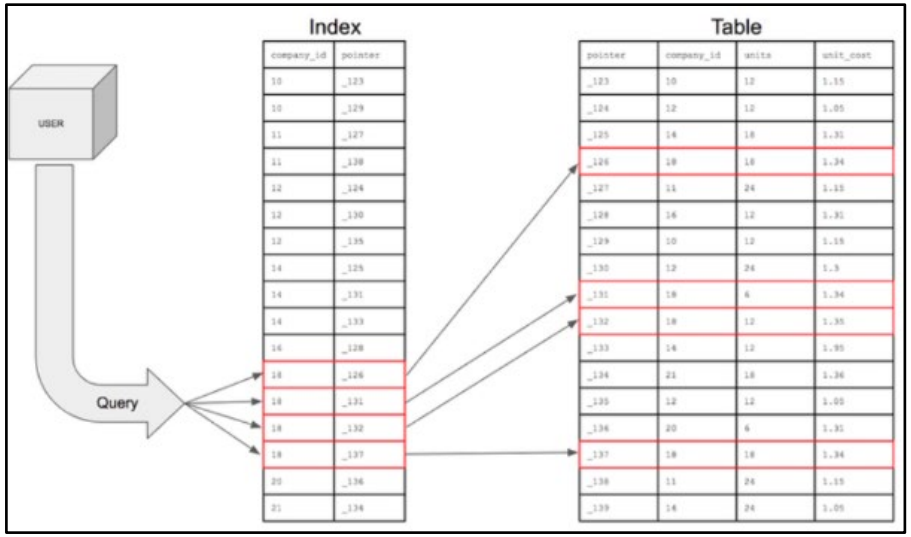
인덱스(Index)란?

데이터베이스에서 데이터를 더 빠르게 검색할 수 있게 해주는 자료구조

- 데이터들을 어떤 값들을 기준으로 데이터 블록을 나열한 **목차**
- 일반적으로 클러스터형 인덱스(Clustered Index)와 보조형 인덱스(Secondary Index) 가 존재함

인덱스의 장단점 및 필요한 경우	
장점	• 검색(Select) 하는 속도가 매우 빨라짐(항상 그런 것은 아님) • 쿼리의 부하가 줄어들어 시스템의 성능이 향상됨
단점	• 데이터베이스 크기의 10% 정도의 추가적인 공간이 필요함 • 데이터의 변경(Insert, Update, Delete) 작업이 자주 일어나는 경우 오히려 성능이 나빠질 수도 있음
인덱스를 사용하면 좋은 경우	• 해당 컬럼이 Where 조건에 자주 사용되는 경우 • 해당 컬럼이 Join의 조건이 되는 경우 • 해당 테이블의 데이터의 변경(Insert, Update, Delete) 작업이 자주 일어나지 않는 경우

[표 8] sql 함수



[그림 5] 인덱스 구조

뷰(View)란?

하나 이상의 기본 테이블이나 다른 뷰를 이용하여 생성되는 가상 테이블

- 기본 테이블은 디스크에 공간이 할당되어 데이터를 저장함
- 데이터 딕셔너리 (Data Dictionary) 테이블에 뷰에 대한 정의 (SQL 문)만 저장되어 디스크 저장 공간 할당이 이루어지지 않음
- 전체 데이터 중에서 일부만 접근할 수 있도록 함
- 뷰에 대한 수정 결과는 뷰를 정의한 기본 테이블에 적용됨
- 뷰를 정의한 기본 테이블에서 정의된 무결성 제약조건은 그대로 유지됨

뷰의 필요성 및 장단점	
필요성	• 사용자마다 특정 객체만 조회할 수 있도록 할 필요가 있음 (모든 직원에 대한 정보를 모든 사원이 볼 수 있도록 하면 안 됨) • 복잡한 질의문을 단순화할 수 있음 • 데이터의 중복성을 최소화할 수 있음
장점	• 논리적 독립성을 제공함 • 데이터의 접근 제어 (보안) • 사용자의 테이터 관리 단순화 • 여러 사용자의 다양한 데이터 요구 지원
단점	• 뷰의 정의 변경 불가 • 삽입, 삭제, 갱신 연산에 제한이 있음

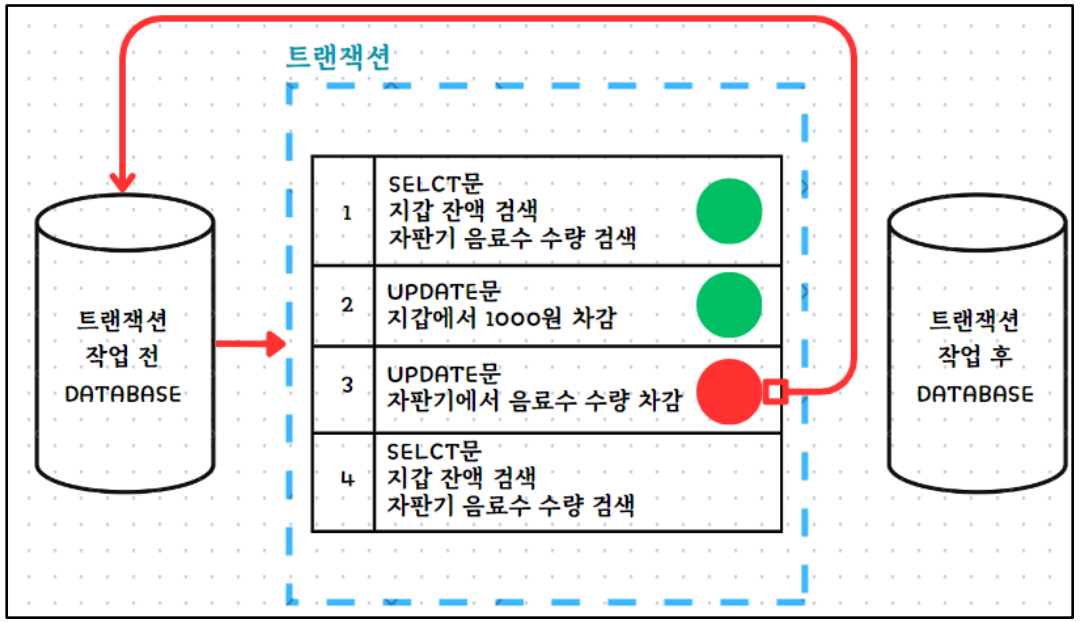
[표 9] 뷰 필요성 및 장단점

트랜잭션(Transaction)란?

- 트랜잭션이란 데이터베이스 내에서 하나의 그룹으로 처리되어야 하는 명령문들을 모아 놓은 논리적인 작업 단위
- 즉, 데이터베이스의 상태를 변화시키기 위해 수행하는 작업의 단위

트랜잭션 필요성	
필요성	• 트랜잭션은 여러 단계의 처리를 하나의 처리처럼 다루는 기능으로 여러 개의 명령어의 집합이 정상적으로 처리되면 정상 종료 • 하나의 명령어라도 잘못되면 전체 취소 • 데이터의 일관성을 유지하면서 안정적으로 데이터를 복구하기 위함

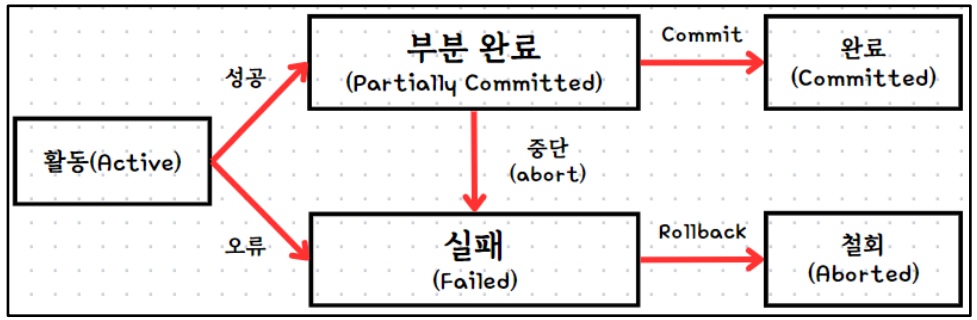
[표 10] 트랜잭션 필요성



[그림 5] 트랜잭션의 예

트랜잭션의 특징	
원자성(Atomicity)	• 트랜잭션과 관련된 작업들이 모두 수행되던지 아니면 모두 실행이 안 되도록 보장해야 하며, • 중간상태는 있을 수 없음(All or Nothing)
일관성(Consistency)	• 트랜잭션이 실행을 성공적으로 완료하면, 언제나 일관성이 있는 데이터베이스 상태로 유지되어야 함
독립성(Isolation)	• 하나의 트랜잭션이 데이터를 갱신하는 동안 이 트랜잭션이 완료되기 전에 갱신 중인 데이터에 다른 트랜잭션이 접근하지 못해야 함
지속성(Durability)	• 트랜잭션이 완료된 이후의 상태는 데이터베이스에 영구적으로 반영되어야 하며, • 비록 시스템이 실패하여도 그 상태가 일관되게 유지되어야 함

[표 11] 트랜잭션의 특징

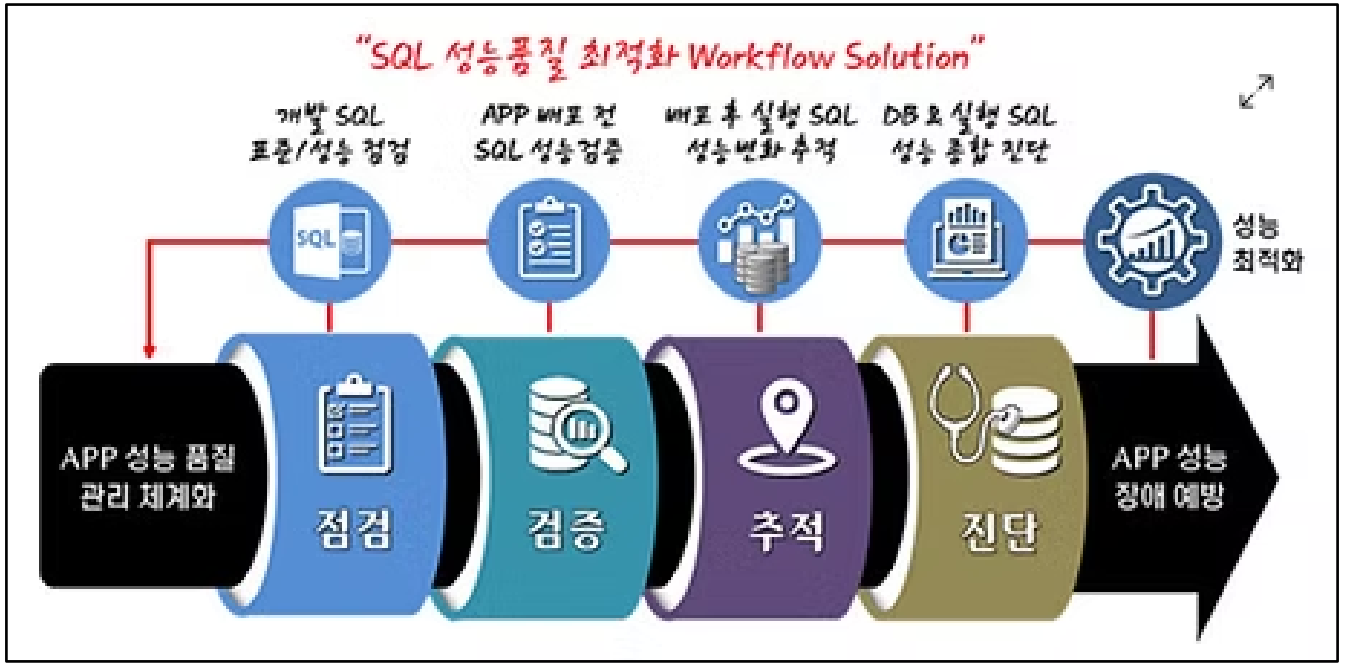


[그림 6] 트랜잭션 작동 원리

- 활동(Active) : 트랜잭션이 실행 중인 상태
- 실패(Failed) : 트랜잭션 실행에 오류가 발생하여 중단된 상태
- 철회(Aborted) : 트랜잭션이 비정상적으로 종료되어 롤백 연산을 수행한 상태
- 부분 완료(Partially Committed) : 트랜잭션의 마지막 연산까지 실행했지만, 커밋 연산이 실행되기 직전의 상태
- 완료(Committed) : 트랜잭션이 성공적으로 종료되어 커밋 연산을 실행한 후의 상태

SQL 성능 최적화 방법

- 적절한 인덱스 사용
- 불필요한 조인 피하기
- 서브쿼리 대신 조인 사용
- LIKE 연산자 사용 시 와일드카드 위치 주의
- 실행 계획 분석



[그림 7] SQL 성능 최적화 방안

BigQuery

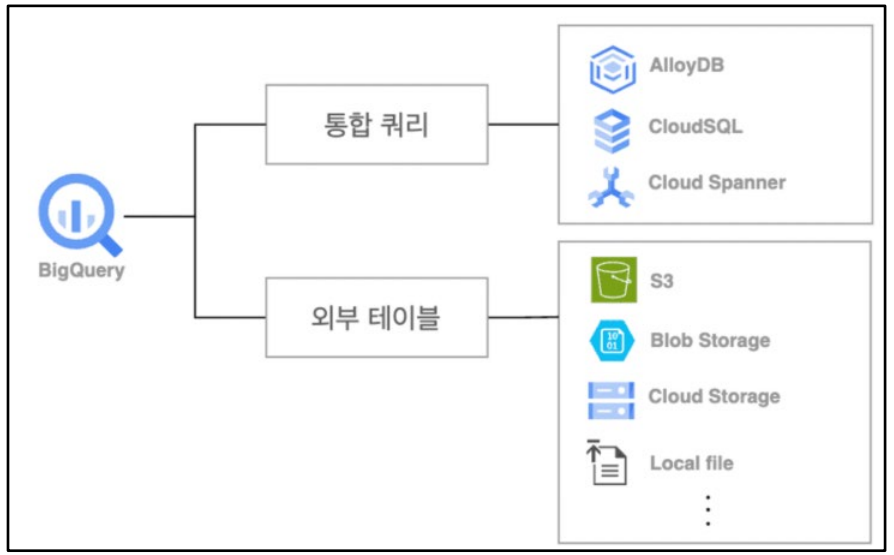
- Google Cloud에서 제공하는 Full Managed Data WareHouse 및 분석 플랫폼

BigQuery의 역사

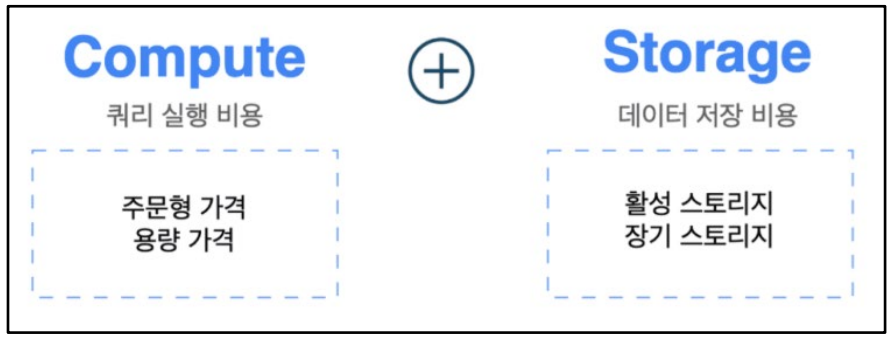
- Google은 2010년에 자체적으로 대용량 데이터 처리 요구사항을 처리하기 위한 도구로 BigQuery를 도입
- Google은 여기서 습득한 대규모 데이터셋의 관리와 분석에 대한 노하우를 활용하여 BigQuery를 자사 상품으로 내놓게 됨
- 2011년에 Google 은 BigQuery를 공개 베타로 출시하여 모든 규모의 조직이 자체 데이터 분석 요구에 BigQuery의 기능을 활용할 수 있게 함

BigQuery 가 유용한 이유

- BigQuery 자체가 이름에서 알 수 있듯이 방대한 양의 데이터를 관리하고 분석하는데 직면하는 도전을 해결하기 위해 개발된 도구
- BigQuery는 초기 투자 비용 부담 및 확장 프로세스를 간소화하고 비용 효율적으로 데이터 분석을 모든 기업이 쉽게 활용할 수 있도록 하기 위해 개발됨



[그림 8] BigQuery 지원 기능 예



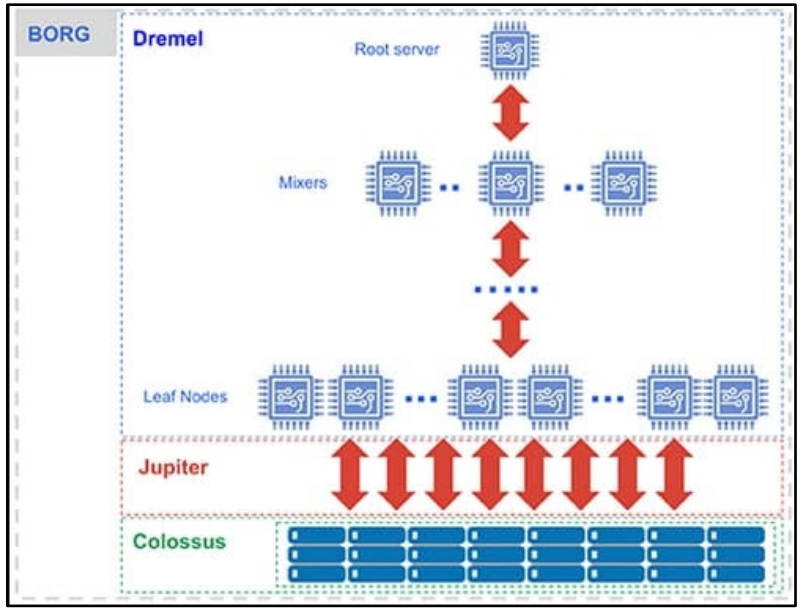
[그림 9] BigQuery 주요 비용

BigQuery 아키텍처

- Compute 하는 부분과 Storage 부분이 분리되어 있는 것이 핵심임
- 또한 분산 처리가 매우 뛰어남

항목	내용
Dremel (Compute)	• 방대한 분산 노드들에서 SQL 쿼리를 실행
Colossus (Storage)	• 데이터를 저장하고 실시간 처리를 할 수 있는 구글의 차세대 파일 시스템 • GCS(Google Cloud Storage)도 이렇게 구성되어 있음
Jupiter (Network)	• Compute 와 Storage 사이의 통신 담당
Borg (Orchestration)	• 이 모든 분산 노드들을 조율 및 운영, 쿠버네티스의 전신

[표 12] BigQuery 구성 항목



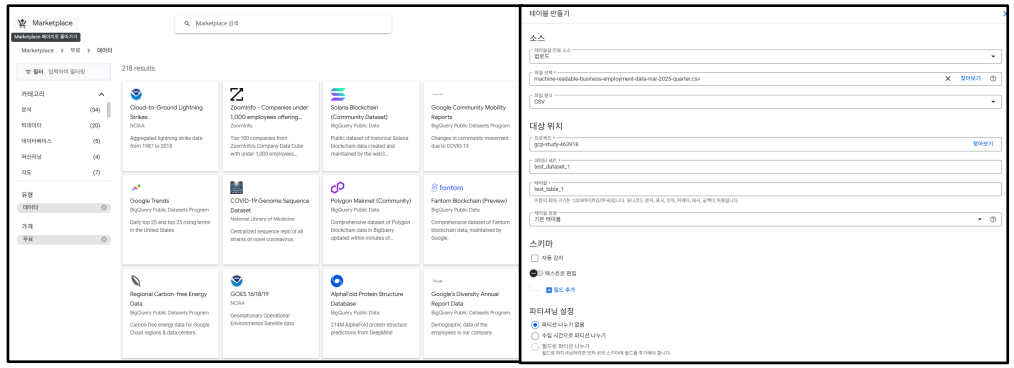
[그림 10] BigQuery 아키텍처

항목	내용
Scalability (확장성)	• BigQuery는 대용량 데이터셋을 쉽게 처리할 수 있도록 설계됨 • 데이터 크기에 관계없이 PB규모의 데이터를 처리하기 위해 수평적으로 확장할 수 있으며, 데이터 크기에 상관없이 높은 성능과 최소한의 쿼리 실행 시간을 보장함 • 이러한 확장성은 기업이 인프라 제한에 대해 걱정하지 않아도 대량의 데이터를 처리하고 분석할 수 있도록 함
Serverless Architecture	• BigQuery는 서버리스 모델에서 작동하여 밑에단 인프라 관리를 추상화함 • 서버를 프로비저닝하거나 관리할 필요가 없기 때문에 조직은 하드웨어 구성이나 소프트웨어 업데이트에 대해 걱정할 필요 없이 데이터 분석에 집중할 수 있음 • 서버리스 접근 방식은 BigQuery가 워크로드에 따라 자동으로 리소스를 조정하여 확장성을 보장함
Cost-Effective	• 사용한 저장 및 계산 리소스에 대해서만 가격을 지불함 • 사전 비용이나 장기적인 약정이 없으며, 저장 비용은 매우 강력한 경쟁력임
Fast Query Execution	• Google의 고급 인프라와 분산 컴퓨팅 기능을 활용하여 빠른 쿼리 실행을 제공함 • 병렬 처리와 열 지향 저장 방식을 사용하여 데이터의 효율적인 스캔과 필터링이 가능함 • 이러한 속도와 성능은 복잡한 분석 쿼리에 대해 빠른 응답 시간을 보장하여 생산성과 의사 결정을 개선함
Integration with Google Cloud Ecosystem	• Google Cloud 생태계의 다른 서비스와 완전히 통합 관리됨 • 예를 들어 Google Data Studio, Cloud Machine Learning Engine, Google Cloud Storage 등임 • 이러한 통합은 일과된 데이터 분석 워크플로우를 가능하게 하며 시각화 및 머신러닝을 위한 추가 도구와 서비스를 활용할 수 있음

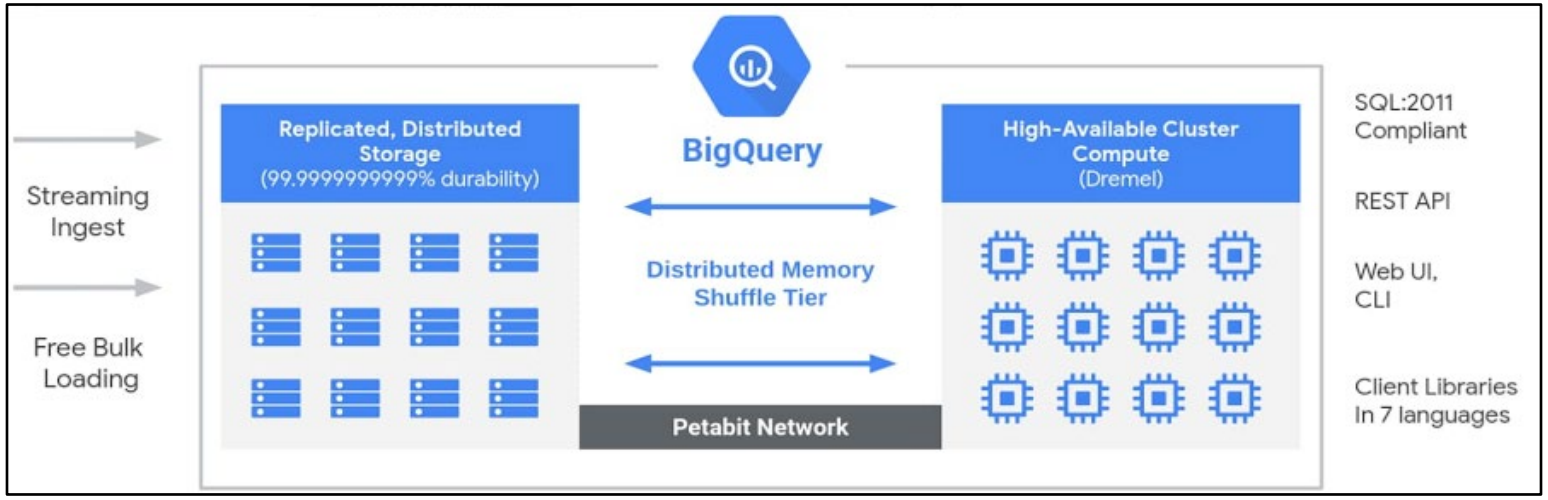
[표 13] BigQuery 장점

BigQuery에 데이터를 연결하는 방법

- Google에서 제공하는 공개 데이터(Google Merchandise Store) 연결
- CSV 데이터 BigQuery에 직접 업로드
- Etc(Streaming, Bulk Loading, API, …)



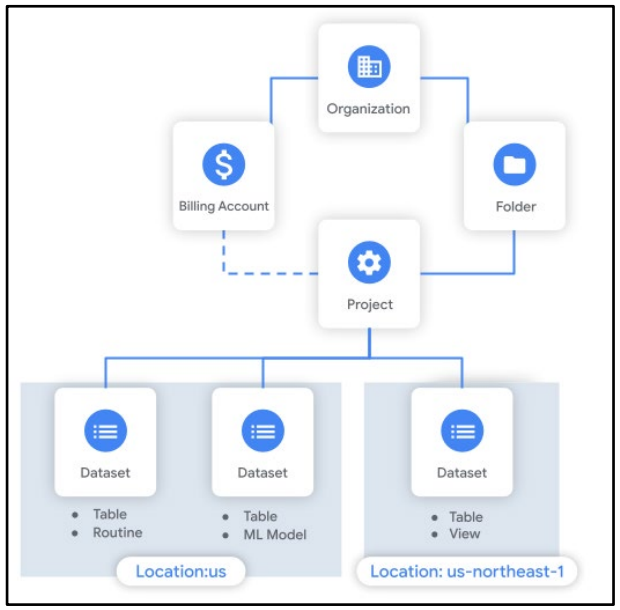
[그림 11] Google Marketplace & CSV



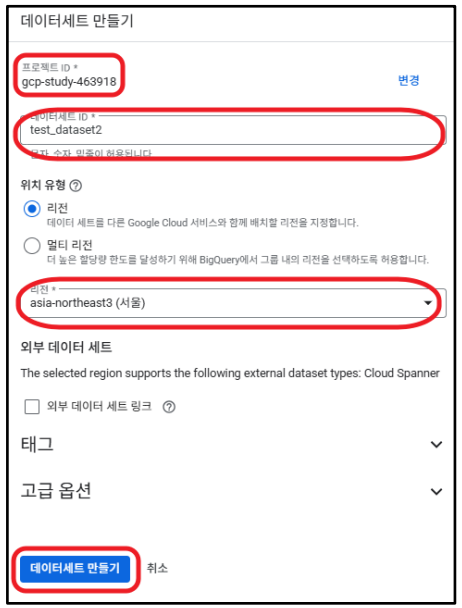
[그림 12] etc

빅쿼리의 데이터 구조: project > dataset > table

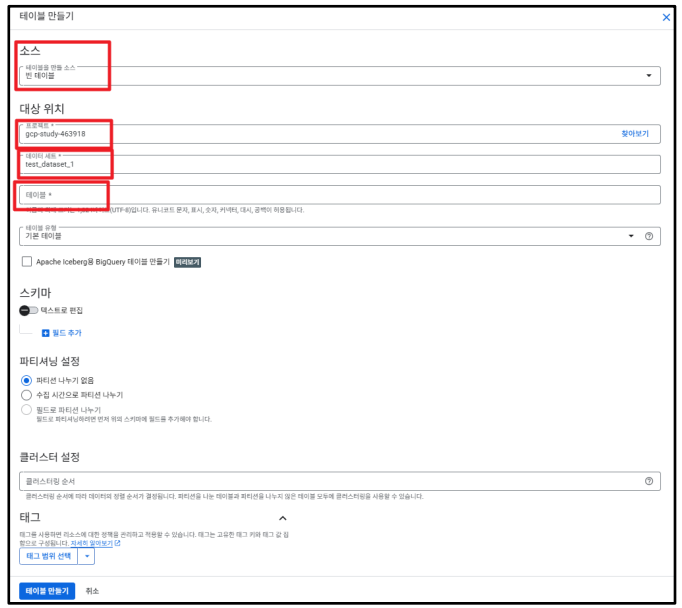
- 프로젝트: GCP 자원을 조직하는 단위. 여러 개의 데이터셋을 가지고 있음.
- 데이터셋: 여러 개의 테이블을 가지고 있는 테이블의 집합
- 테이블: 데이터가 저장되는 테이블



[그림 13] 프로젝트/데이터세트/테이블 관계



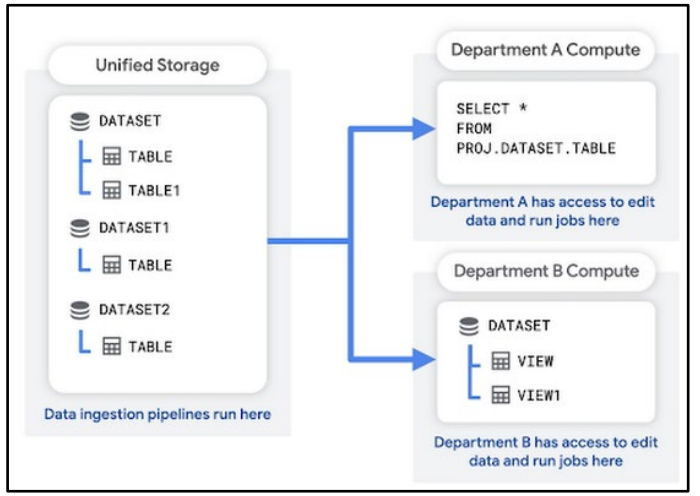
[그림 14] 데이터세트 만들기 설정



[그림 15] 테이블 만들기 설정

중앙 데이터 레이크, 부서 데이터 마트

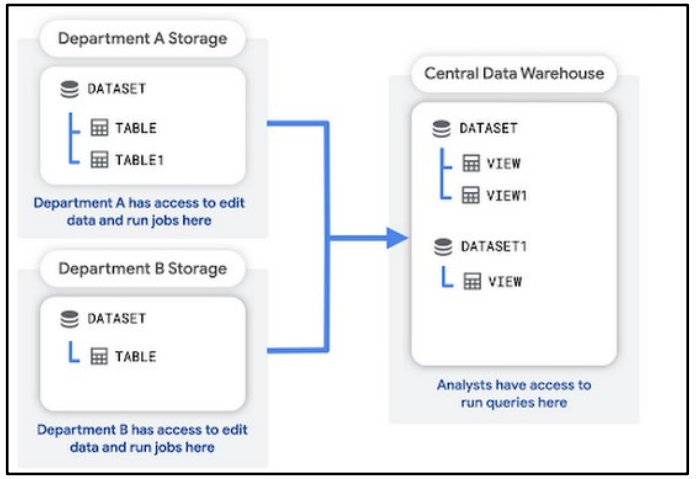
- 이 조직은 원시 데이터를 저장하기 위해 통합된 스토리지 프로젝트로 구성
- 조직 내 부서는 분석을 위해 부서 자체의 데이터 마트 프로젝트로 구성



[그림 16] 중앙 데이터 레이크, 부서 데이터 마트

부서 데이터 레이크, 중앙 데이터 웨어하우스

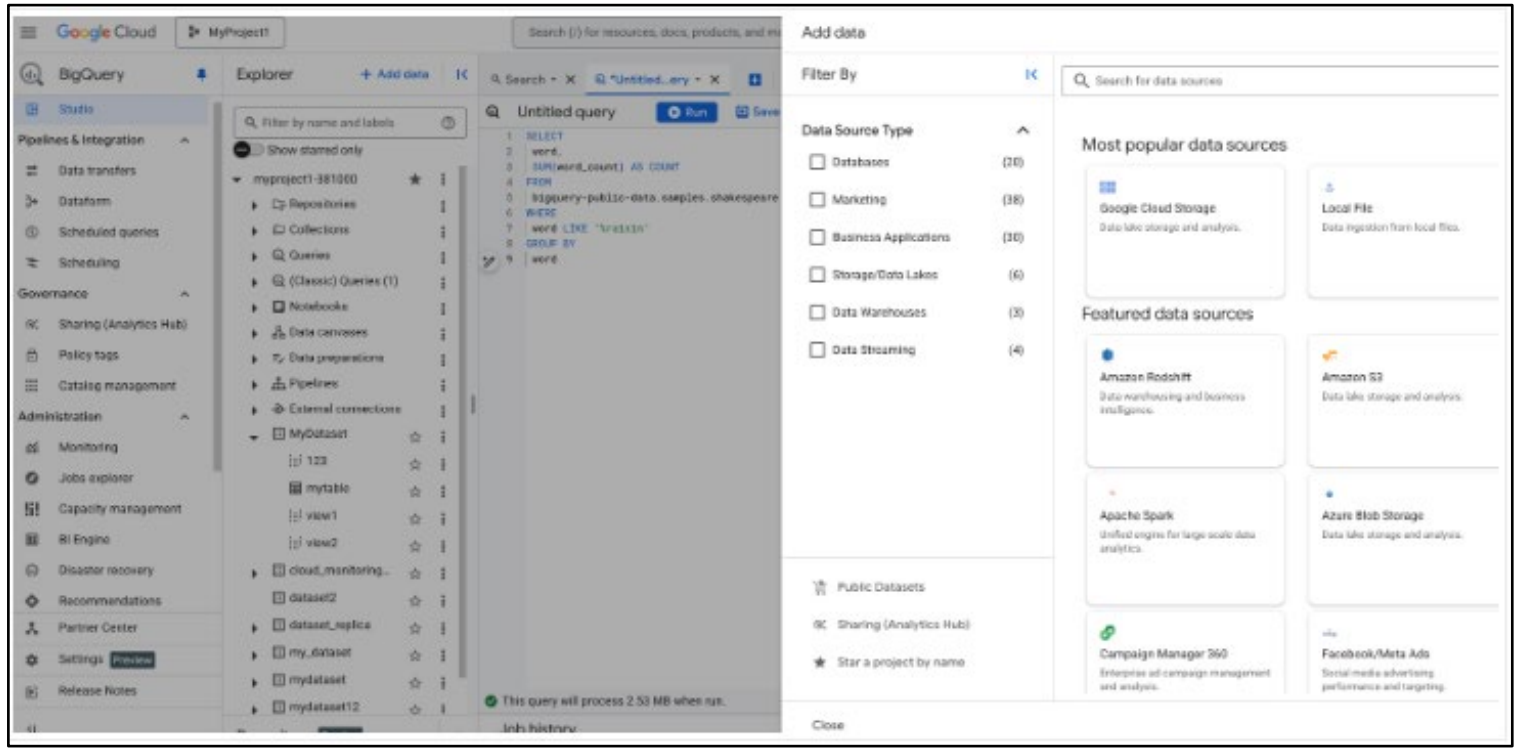
- 각 부서는 부서 원시 데이터를 저장하기 위해 자체 스토리지 프로젝트를 만들고 관리
- 조직은 그런 후 분석을 위해 중앙 데이터 웨어하우스 프로젝트로 구성



[그림 17] 부서 데이터 레이크, 중앙 데이터 웨어하우스

BigQuery 페이지 주요 섹션

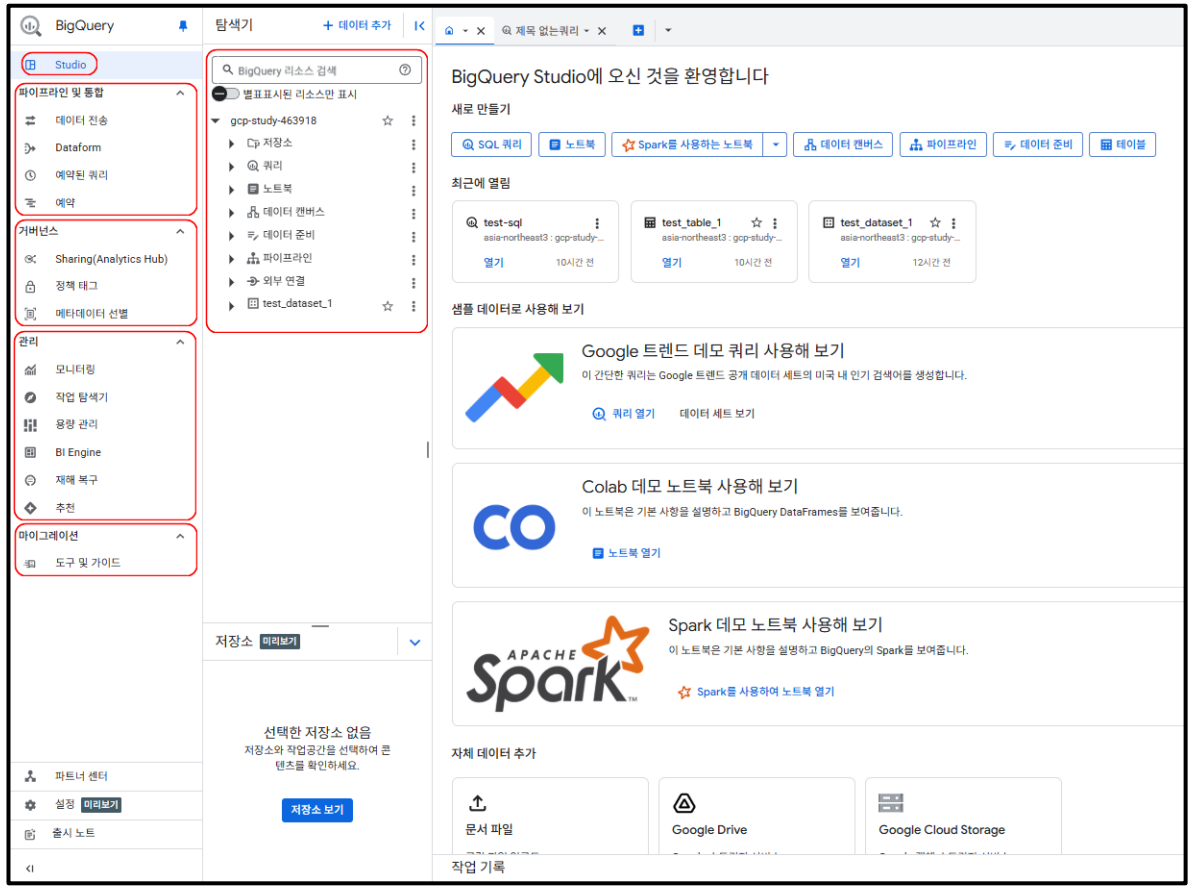
- BigQuery 탐색 메뉴
- **탐색기** 창
- 세부정보 창
- **추가** 대화상자



[그림 18] BigQuery 페이지 주요 세션

Studio

- 데이터 세트, 테이블, 기타 BigQuery 리소스를 표시
- 일반적인 BigQuery 태스크를 수행
- 쿼리 및 Colab Enterprise 노트북을 생성, 실행, 저장, 공유
- 테이블, 뷰, 루틴, 기타 BigQuery 리소스를 사용하여 작업
- BigQuery 작업 기록을 확인



[그림 19] BigQuery 주요 메뉴

Ⅱ BigQuery - 탐색 메뉴

파이프라인 및 통합

- **데이터 전송**: BigQuery Data Transfer Service에 액세스하여 데이터 전송을 만들고 구성
- **파이프라인 (Dataform)**: Google Cloud 프로젝트용으로 생성된 Dataform 저장소 목록을 표시
- **예약된 쿼리**: 예약된 쿼리를 표시
- **예약**(프리뷰): Google Cloud 프로젝트의 워크플로 및 일정 목록을 제공

관리

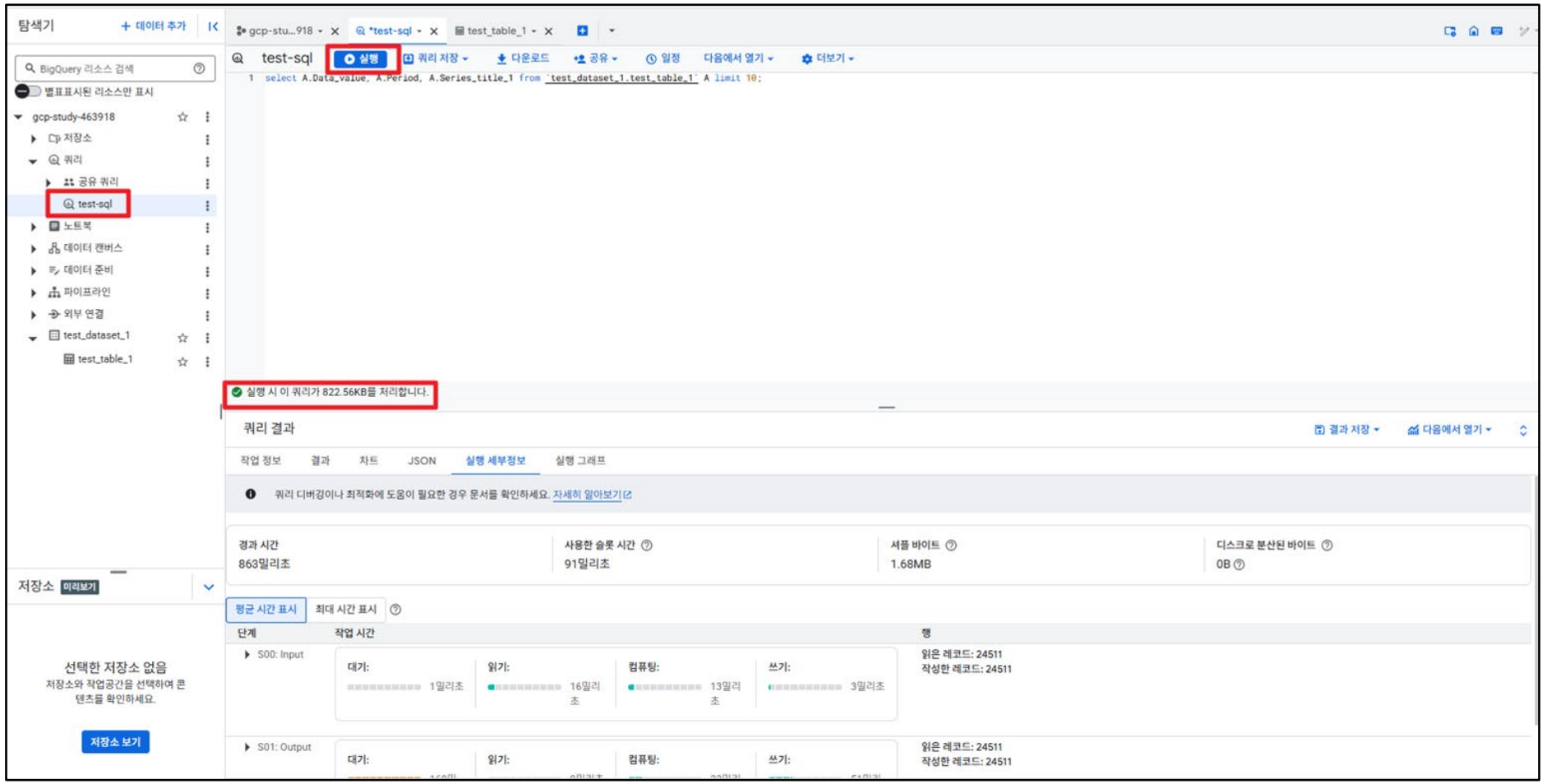
- **모니터링**: 조직 전반의 운영 상태와 리소스 사용률을 모니터링
- **작업 탐색기**: 조직 전반의 작업 활동을 모니터링하는 데 도움
- **용량 관리**: 슬롯 약정, 예약, 예약 할당을 표시
- **BI Engine**: BigQuery BI Engine을 사용하여 쿼리 최적화를 위한 예약 용량을 관리할 수 있음
- **재해 복구**: 페일오버 예약 및 페일오버 데이터 세트를 표시
- **권장사항**: 조직 또는 프로젝트의 권장사항 목록을 표시

거버넌스

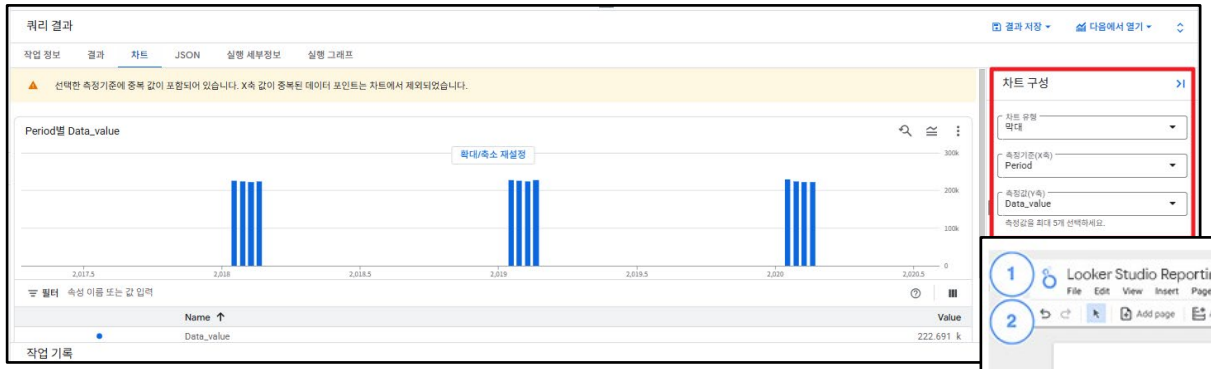
- **Analytics Hub**: Google Cloud 프로젝트에서 액세스할 수 있는 모든 데이터 교환을 표시
- **정책 태그**: 정책 태그의 계층 구조 그룹을 만드는 데 사용할 수 있는 분류 목록을 표시

추가 리소스

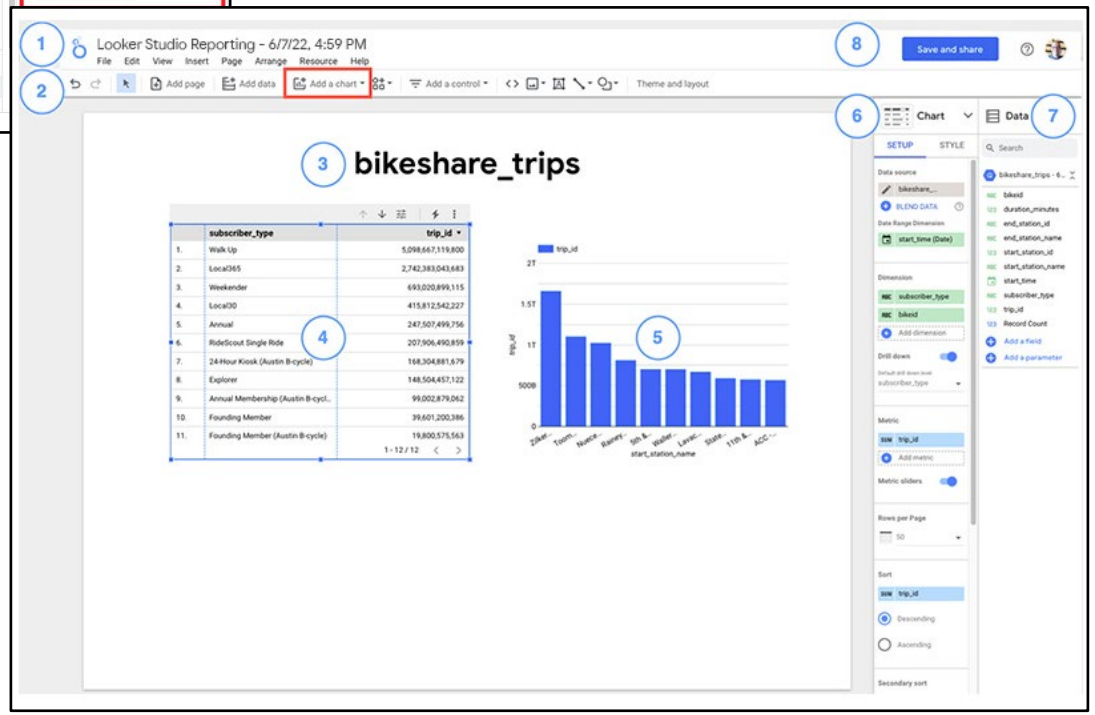
- **파트너 센터**: 워크플로를 가속화하는 파트너의 도구와 서비스를 제공
- **설정**(프리뷰): BigQuery Studio에서 세션을 시작할 때 적용되는 기본 설정을 지정할 수 있음. 일부 설정은 프로젝트 또는 조직에서 상속되지만 **설정** 페이지에서 재정의할 수 있음.
- **출시 노트**: BigQuery의 최신 제품 업데이트 및 공지사항이 포함됨



[그림 20] 쿼리 에디터 및 사용의 예



[그림 21] 시각화 : 기본차트



[그림 22] 시각화 : Looker Studio

BigQuery 성능 최적화 이론

- 효율적인 쿼리 작성
- 비용 절감 전략

최적화가 중요한 이유

- 쿼리 비용은 스캔 한 데이터양에 따라 청구됨
- 잘못된 쿼리는 느린 응답 시간과 높은 비용 발생
- 최적화는 속도 향상 + 비용 절감 모두 가능

SQL 최적화 핵심 전략
• 필요한 컬럼만 SELECT
• 파티션 & 클러스터 테이블 사용
• WITH 문과 서브쿼리 최소화
• 필터 조건(Predicate)을 최대한 앞에
• 조인 순서와 조건 조정
• 자료형 정제 및 테이블 구조 개선

전략 1 - 필요한 컬럼만 SELECT

- 불필요한 데이터 스캔 방지 → 비용 절감

```
-- ❌ 비효율
SELECT * FROM dataset.table;

-- ✅ 효율적
SELECT user_id, session_id FROM dataset.table;
```

전략 2 - 파티셔닝 사용

- 시간 또는 범위 기반 파티션 설정 권장
- 쿼리 할 파티션 범위 지정 필수

```
SELECT *
FROM dataset.table
WHERE _PARTITIONTIME = DATE('2024-01-01')
```

전략 3 - 클러스터링 사용

- 필터/조인에 자주 사용하는 컬럼으로 클러스터링
- 예: user_id, country 등
- 관련 데이터 블록만 조회 → 빠르고 저렴

```
CREATE TABLE dataset.table (
  ...
)
PARTITION BY DATE(timestamp)
CLUSTER BY user_id;
```

전략 4 - 서브쿼리와 WITH 최소화

- 중첩된 쿼리 반복 실행 → 성능 저하
- 가능하면 JOIN 또는 TEMP TABLE 활용

```
-- WITH 쿼리 중복보다
CREATE TEMP TABLE temp AS (...);
SELECT ... FROM temp;
```

전략 5 - 조인 최적화

- 조인 대상 테이블 크기 최소화
- 조인 조건에 인덱싱 효과 있는 컬럼 사용
- 필요한 경우 JOIN EACH → JOIN 변경

```
SELECT ...
FROM big_table a
JOIN small_table b
ON a.key = b.key
```

전략 6 - 자료형 및 구조 개선

- INT → STRING 잘못 사용 시 비용 증가
- 중첩 REPEATED/STRUCT 타입 적절히 정규화
- 스키마 정리 주기적 필요

전략 7 - 쿼리 플래너와 EXPLAIN 사용

- 쿼리 실행 전에 EXPLAIN 사용
- 각 단계의 작업량 확인
- 느린 단계 위치 파악 가능

```
EXPLAIN SELECT ... FROM ...
```

전략 8 - 비용 분석 도구 활용

- BigQuery 콘솔: 쿼리 수행 전 데이터 스캔 양 표시
- INFORMATION_SCHEMA.JOBS_BY_* 뷰 활용

```
SELECT total_bytes_processed, query
FROM region-us.INFORMATION_SCHEMA.JOBS_BY_USER
ORDER BY start_time DESC
```

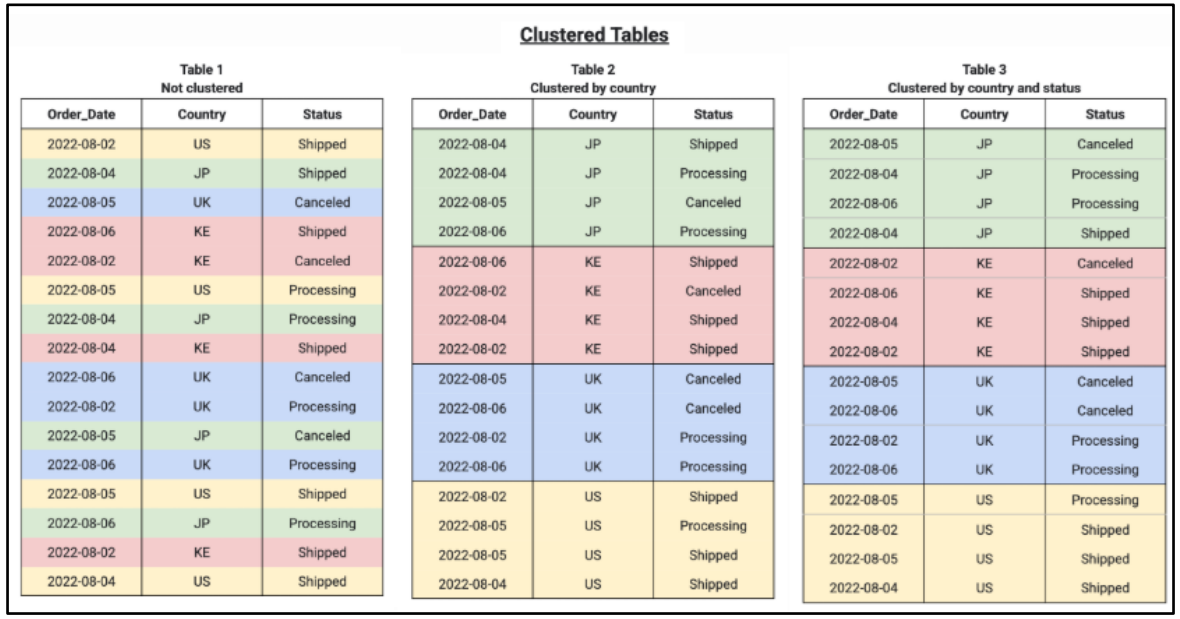
BigQuery 최적화 - Clustered Table

클러스터링 된 테이블

- 클러스터링 된 열을 사용하여 사용자 정의 열 정렬 순서가 있는 테이블
- 클러스터링 된 테이블을 사용하면 쿼리 성능을 높이고 쿼리 비용을 줄일 수 있음
- 클러스터링 된 열은 **클러스터링 된 열의 값을 기준으로 스토리지 블록을 정렬하는 사용자 정의 테이블 속성**임
- 스토리지 블록 크기는 테이블 크기에 따라 조정됨
- 클러스터링 된 테이블에서는 해당 테이블을 수정하는 각 작업과 관련하여 정렬 속성이 유지됨
- 클러스터링 된 열을 기준으로 필터링 또는 집계하는 쿼리는 전체 테이블 또는 테이블 파티션 대신 클러스터링 된 열을 기준으로 관련 블록만 스캔함
- 그 결과 BigQuery가 쿼리에 의해 처리되는 바이트나 쿼리 비용을 정확히 예측하지 못할 수도 있지만, 실행 시 총 바이트 수를 줄이려고 시도함
- 여러 열을 사용하여 테이블을 클러스터링 할 경우 열 순서는 BigQuery가 데이터를 정렬하여 스토리지 블록으로 그룹화할 때 우선 적용되는 열을 결정함

클러스터링을 사용해야 하는 경우

- **파티션을 나누지 않은 테이블이 64MB보다 클 경우** 클러스터링 하면 이점이 있을 수 있음
- 마찬가지로 **64MB보다 큰 테이블 파티션도 클러스터링 시 이점을 누릴 수 있음**
- 더 작은 테이블 또는 파티션도 클러스터링 할 수 있지만 일반적으로 성능 개선이 미미함
- 쿼리에서 특정 열을 기준으로 필터링하는 경우가 많다면 쿼리에서 필터와 일치하는 블록만 스캔하므로 클러스터링이 쿼리 속도를 높임
- 쿼리의 필터링 기준 열에 고유 값이 많다면(카디널리티가 높음) 클러스터링은 입력 데이터를 가져올 위치에 대한 상세 메타데이터를 BigQuery에 제공하여 쿼리 속도를 높임
- 클러스터링을 사용하면 테이블의 기본 스토리지 블록의 크기를 테이블 크기에 따라 적응적으로 조절할 수 있음



Clustered Tables

Table 1
Not clustered

Order_Date	Country	Status
2022-08-02	US	Shipped
2022-08-04	JP	Shipped
2022-08-05	UK	Canceled
2022-08-06	KE	Shipped
2022-08-02	KE	Canceled
2022-08-05	US	Processing
2022-08-04	JP	Processing
2022-08-04	KE	Shipped
2022-08-06	UK	Canceled
2022-08-02	UK	Processing
2022-08-05	JP	Canceled
2022-08-06	UK	Processing
2022-08-05	US	Shipped
2022-08-06	JP	Processing
2022-08-02	KE	Shipped
2022-08-04	US	Shipped

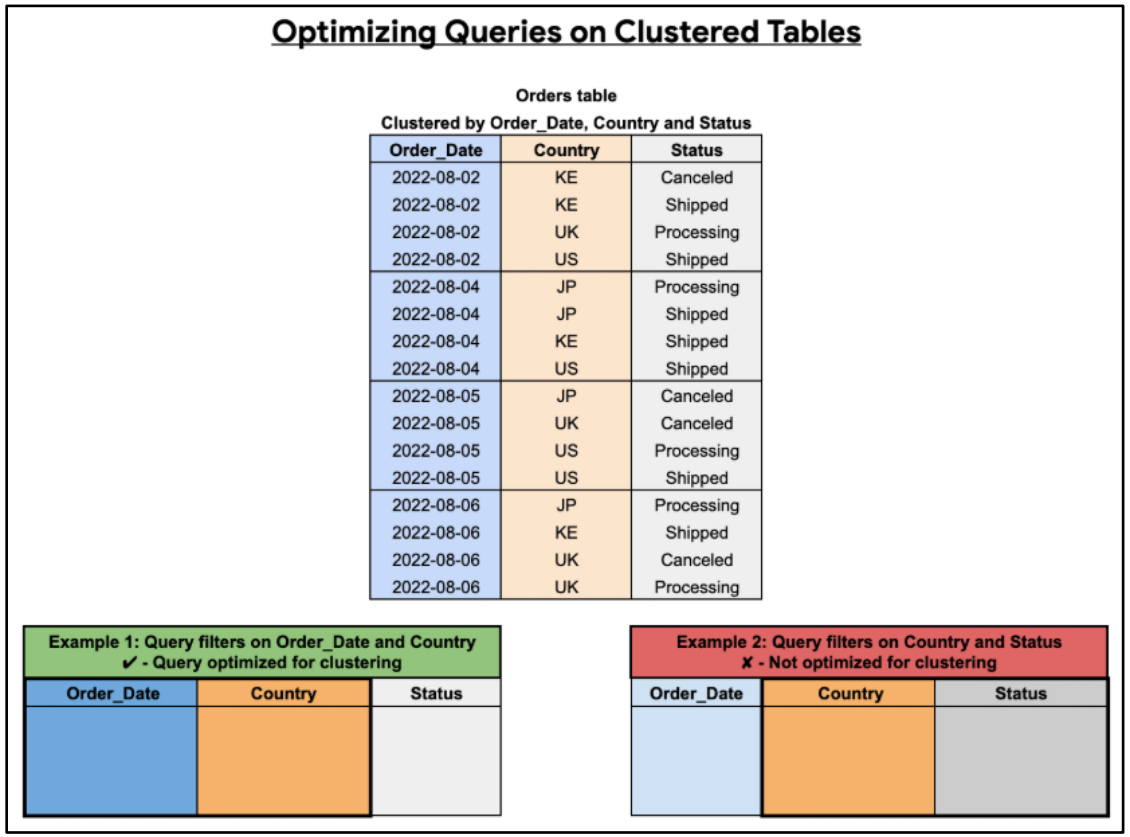
Table 2
Clustered by country

Order_Date	Country	Status
2022-08-04	JP	Shipped
2022-08-04	JP	Processing
2022-08-05	JP	Canceled
2022-08-06	JP	Processing
2022-08-06	KE	Shipped
2022-08-02	KE	Canceled
2022-08-04	KE	Shipped
2022-08-02	KE	Shipped
2022-08-05	UK	Canceled
2022-08-06	UK	Canceled
2022-08-02	UK	Processing
2022-08-06	UK	Processing
2022-08-02	US	Shipped
2022-08-05	US	Processing
2022-08-05	US	Shipped
2022-08-04	US	Shipped

Table 3
Clustered by country and status

Order_Date	Country	Status
2022-08-05	JP	Canceled
2022-08-04	JP	Processing
2022-08-06	JP	Processing
2022-08-04	JP	Shipped
2022-08-02	KE	Canceled
2022-08-06	KE	Shipped
2022-08-04	KE	Shipped
2022-08-02	KE	Shipped
2022-08-05	UK	Canceled
2022-08-06	UK	Canceled
2022-08-02	UK	Processing
2022-08-06	UK	Processing
2022-08-05	US	Processing
2022-08-02	US	Shipped
2022-08-05	US	Shipped
2022-08-04	US	Shipped

[그림 23] Clusterd Table의 예



Optimizing Queries on Clustered Tables

Orders table
Clustered by Order_Date, Country and Status

Order_Date	Country	Status
2022-08-02	KE	Canceled
2022-08-02	KE	Shipped
2022-08-02	UK	Processing
2022-08-02	US	Shipped
2022-08-04	JP	Processing
2022-08-04	JP	Shipped
2022-08-04	KE	Shipped
2022-08-04	US	Shipped
2022-08-05	JP	Canceled
2022-08-05	UK	Canceled
2022-08-05	US	Processing
2022-08-05	US	Shipped
2022-08-06	JP	Processing
2022-08-06	KE	Shipped
2022-08-06	UK	Canceled
2022-08-06	UK	Processing

Example 1: Query filters on Order_Date and Country
✔ - Query optimized for clustering

Order_Date	Country	Status

Example 2: Query filters on Country and Status
✘ - Not optimized for clustering

Order_Date	Country	Status

[그림 23] Clusterd Table 최적화

파티션을 나눈 테이블

- **파티션이라는 세그먼트로 분할**되어 데이터를 보다 쉽게 관리하고 쿼리 할 수 있게 해줌
- 큰 테이블을 작은 파티션으로 나누면 쿼리 성능을 높일 수 있으며 쿼리에서 읽는 바이트 수를 줄여 비용을 제어할 수 있음
- 테이블을 세분화하는데 사용되는 파티션 열을 지정하여 테이블을 파티션으로 나눔
- 쿼리에서 파티션 나누기 열의 값에 대해 한정 필터가 사용되는 경우 BigQuery가 필터와 일치하는 파티션을 스캔하고 남은 파티션을 건너뛸 수 있으며, 이 프로세스를 프루닝이라 함
- **파티션을 나눈 테이블에서 데이터는 데이터의 한 파티션을 포함하는 각 물리적 블록에 저장됨**
- 파티션을 나눈 각 테이블은 테이블을 수정하는 모든 작업에서 정렬 속성에 대한 다양한 메타데이터를 유지함
- 메타데이터를 통해 BigQuery가 쿼리를 실행하기 전에 쿼리 비용을 더 정확히 예측할 수 있음

파티션 테이블을 사용해야 하는 경우

- 테이블의 일부만 스캔하여 쿼리 성능을 개선
- 테이블 작업이 표준 테이블 할당량을 초과하며 테이블 작업 범위를 특정 파티션 열 값으로 제한하여 더 높은 파티션을 나눈 테이블 할당량을 허용할 수 있음
- 파티션이 나뉜 테이블에서 쿼리가 실행되기 전에 쿼리 비용 추정치를 제공하며, 파티션을 나눈 테이블을 프루닝 후 쿼리 테스트 실행을 수행하여 쿼리 비용을 추정하여 쿼리 비용에 대한 추정치를 계산함

Clustered and Partitioned Tables

Orders table
Not Clustered; Not partitioned

Order_Date	Country	Status
2022-08-02	US	Shipped
2022-08-04	JP	Shipped
2022-08-05	UK	Canceled
2022-08-06	KE	Shipped
2022-08-02	KE	Canceled
2022-08-05	US	Processing
2022-08-04	JP	Processing
2022-08-04	KE	Shipped
2022-08-06	UK	Canceled
2022-08-02	UK	Processing
2022-08-05	JP	Canceled
2022-08-06	UK	Processing
2022-08-05	US	Shipped
2022-08-06	JP	Processing
2022-08-02	KE	Shipped
2022-08-04	US	Shipped

Orders table
Clustered by Country; Not partitioned

Order_Date	Country	Status
2022-08-04	JP	Shipped
2022-08-04	JP	Processing
2022-08-05	JP	Canceled
2022-08-06	JP	Processing
2022-08-06	KE	Shipped
2022-08-02	KE	Canceled
2022-08-04	KE	Shipped
2022-08-02	KE	Shipped
2022-08-05	UK	Processing
2022-08-06	UK	Canceled
2022-08-02	UK	Canceled
2022-08-06	UK	Processing
2022-08-02	US	Shipped
2022-08-05	US	Processing
2022-08-05	US	Shipped
2022-08-04	US	Shipped

Orders table
Clustered by Country; Partitioned by Order_Date (Daily)

	Order_Date	Country	Status
Partition: 2022-08-02	2022-08-02	KE	Shipped
	2022-08-02	KE	Canceled
Clusters: Country	2022-08-02	UK	Processing
	2022-08-02	US	Shipped

	Order_Date	Country	Status
Partition: 2022-08-04	2022-08-04	JP	Shipped
	2022-08-04	JP	Processing
Cluster: Country	2022-08-04	KE	Shipped
	2022-08-04	US	Shipped

	Order_Date	Country	Status
Partition: 2022-08-05	2022-08-05	JP	Canceled
	2022-08-05	UK	Canceled
Cluster: Country	2022-08-05	US	Shipped
	2022-08-05	US	Processing

	Order_Date	Country	Status
Partition: 2022-08-06	2022-08-06	JP	Processing
	2022-08-06	KE	Shipped
Cluster: Country	2022-08-06	UK	Canceled
	2022-08-06	UK	Processing

[그림 24] Clusterd and Partitioned Table 최적화 구조

BigQuery 최적화 정리

- 전략) SELECT 컬럼 최소화
  - 비용 절감, 속도 향상
- 전략) 파티션 필터 사용
  - 전체 테이블 스캔 방지
- 전략) 클러스터 컬럼 지정
  - 검색 최적화
- 전략) 조인 최적화
  - 불필요한 비용 차단
- 전략) 쿼리 분석 도구 사용
  - 병목 지점 파악