고객을 세그먼테이션하자 [프로젝트]

Course	
Status	Not started

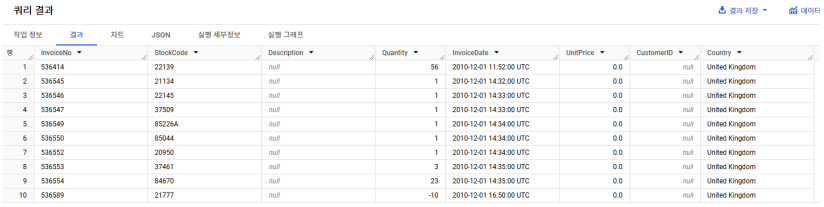
11-2. 데이터 불러오기

데이터 살펴보기

- **테이블에 있는 10개의 행만 출력하기**

```
SELECT *
FROM `studied-source-439405-c0.modulabs_project.data`
LIMIT 10;
```

쿼리 결과



행	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country
1	536414	22139	null	56	2010-12-01 11:52:00 UTC	0.0	null	United Kingdom
2	536545	21134	null	1	2010-12-01 14:32:00 UTC	0.0	null	United Kingdom
3	536546	22145	null	1	2010-12-01 14:33:00 UTC	0.0	null	United Kingdom
4	536547	37509	null	1	2010-12-01 14:33:00 UTC	0.0	null	United Kingdom
5	536549	85226A	null	1	2010-12-01 14:34:00 UTC	0.0	null	United Kingdom
6	536550	85044	null	1	2010-12-01 14:34:00 UTC	0.0	null	United Kingdom
7	536552	20950	null	1	2010-12-01 14:34:00 UTC	0.0	null	United Kingdom
8	536553	37461	null	3	2010-12-01 14:35:00 UTC	0.0	null	United Kingdom
9	536554	84670	null	23	2010-12-01 14:35:00 UTC	0.0	null	United Kingdom
10	536589	21777	null	-10	2010-12-01 16:50:00 UTC	0.0	null	United Kingdom

- **전체 데이터는 몇 행으로 구성되어 있는지 확인하기**

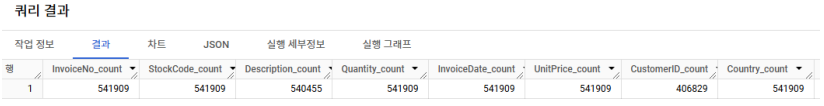
```
SELECT COUNT(*)
FROM `studied-source-439405-c0.modulabs_project.data'
```

행	f0_
1	541909

데이터 수 세기

- **COUNT 함수를 사용해서, 각 컬럼별 데이터 포인트의 수를 세어 보기**

```
SELECT
  COUNT(InvoiceNo) AS InvoiceNo_count,
  COUNT(StockCode) AS StockCode_count,
  COUNT(Description) AS Description_count,
  COUNT(Quantity) AS Quantity_count,
  COUNT(InvoiceDate) AS InvoiceDate_count,
  COUNT(UnitPrice) AS UnitPrice_count,
  COUNT(CustomerID) AS CustomerID_count,
  COUNT(Country) AS Country_count
FROM `studied-source-439405-c0.modulabs_project.data`;
```



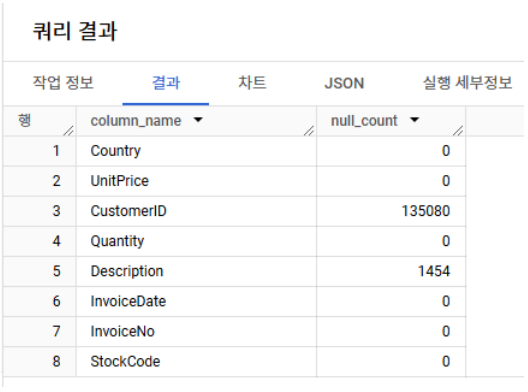
행	InvoiceNo_count	StockCode_count	Description_count	Quantity_count	InvoiceDate_count	UnitPrice_count	CustomerID_count	Country_count
1	541909	541909	540455	541909	541909	541909	406829	541909

11-4. 데이터 전처리 방법(1): 결측치 제거

컬럼 별 누락된 값의 비율 계산

- **각 컬럼 별 누락된 값의 비율을 계산**
  - **각 컬럼에 대해서 누락 값을 계산한 후, 계산된 누락 값을 UNION ALL을 통해 합치기**

```
SELECT 'InvoiceNo' AS column_name, COUNT(*) - COUNT(InvoiceNo) AS null_count
FROM `studied-source-439405-c0.modulabs_project.data`
UNION ALL
SELECT 'StockCode' AS column_name, COUNT(*) - COUNT(StockCode) AS null_count
FROM `studied-source-439405-c0.modulabs_project.data`
UNION ALL
SELECT 'Description' AS column_name, COUNT(*) - COUNT(Description) AS null_count
FROM `studied-source-439405-c0.modulabs_project.data`
UNION ALL
SELECT 'Quantity' AS column_name, COUNT(*) - COUNT(Quantity) AS null_count
FROM `studied-source-439405-c0.modulabs_project.data`
UNION ALL
SELECT 'InvoiceDate' AS column_name, COUNT(*) - COUNT(InvoiceDate) AS null_count
FROM `studied-source-439405-c0.modulabs_project.data`
UNION ALL
SELECT 'UnitPrice' AS column_name, COUNT(*) - COUNT(UnitPrice) AS null_count
FROM `studied-source-439405-c0.modulabs_project.data`
UNION ALL
SELECT 'CustomerID' AS column_name, COUNT(*) - COUNT(CustomerID) AS null_count
FROM `studied-source-439405-c0.modulabs_project.data`
UNION ALL
SELECT 'Country' AS column_name, COUNT(*) - COUNT(Country) AS null_count
FROM `studied-source-439405-c0.modulabs_project.data`;
```



쿼리 결과

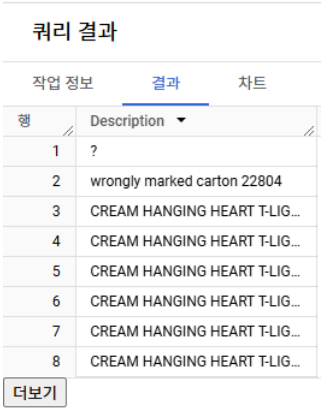
작업 정보 | 결과 | 차트 | JSON | 실행 세부정보

행	column_name	null_count
1	Country	0
2	UnitPrice	0
3	CustomerID	135080
4	Quantity	0
5	Description	1454
6	InvoiceDate	0
7	InvoiceNo	0
8	StockCode	0

결측치 처리 전략

- `StockCode = '85123A'`의 `Description`을 추출하는 쿼리문을 작성하기

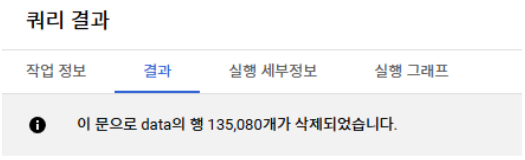
```
SELECT Description
FROM `studied-source-439405-c0.modulabs_project.data`
WHERE StockCode = '85123A';
```

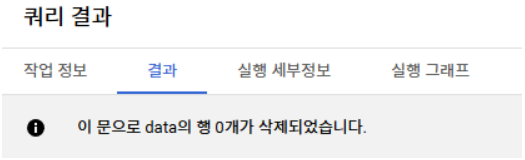


결측치 처리

- **DELETE 구문을 사용하며, WHERE 절을 통해 데이터를 제거할 조건을 제시**

```
DELETE FROM `studied-source-439405-c0.modulabs_project.data`
WHERE CustomerID IS NULL;

DELETE FROM `studied-source-439405-c0.modulabs_project.data`
WHERE Description is NULL;
```





11-5. 데이터 전처리(2): 중복값 처리

중복값 확인

- **중복된 행의 수를 세어보기**
  - **8개의 컬럼에 그룹 함수를 적용한 후, COUNT가 1보다 큰 데이터를 세어보기**

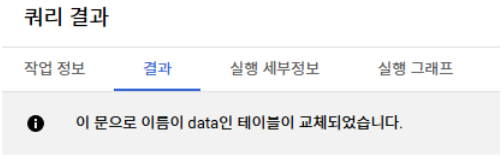
```
SELECT COUNT(*) AS duplicate_count
FROM (
    SELECT
        InvoiceNo,
        StockCode,
        Description,
        Quantity,
        InvoiceDate,
        UnitPrice,
        CustomerID,
        Country
    FROM
        `studied-source-439405-c0.modulabs_project.data`
```

```
    GROUP BY
        InvoiceNo,
        StockCode,
        Description,
        Quantity,
        InvoiceDate,
        UnitPrice,
        CustomerID,
        Country
    HAVING COUNT(*) > 1
) AS subquery;
```

중복값 처리

- **중복값을 제거하는 쿼리문 작성하기**
  - **CREATE OR REPLACE TABLE 구문을 활용하여 모든 컬럼(*)을 DISTINCT 한 데이터로 업데이트**

```
CREATE OR REPLACE TABLE `studied-source-439405-c0.modulabs_project.data` AS
SELECT DISTINCT *
FROM `studied-source-439405-c0.modulabs_project.data`;
```

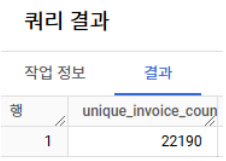


11-6. 데이터 전처리(3): 오류값 처리

`InvoiceNo` 살펴보기

- **고유(unique)한 `InvoiceNo`의 개수를 출력하기**

```
SELECT COUNT(DISTINCT InvoiceNo) AS unique_invoice_count
FROM `studied-source-439405-c0.modulabs_project.data`;
```



- **고유한 `InvoiceNo`를 앞에서부터 100개를 출력하기**

```
SELECT DISTINCT InvoiceNo
FROM `studied-source-439405-c0.modulabs_project.data`
```

```
ORDER BY InvoiceNo
LIMIT 100;
```

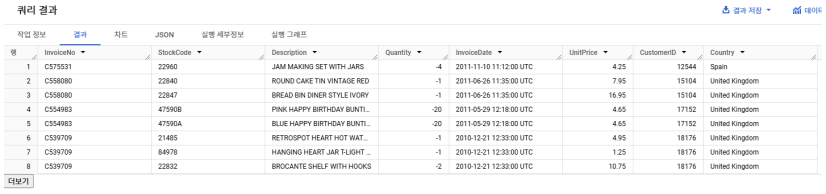
쿼리 결과

작업 정보 | 결과 | 차트

행	InvoiceNo
1	536365
2	536365
3	536365
4	536365
5	536365
6	536365
7	536365
8	536366
9	536366
10	536367
11	536367
12	536367
13	536367
14	536367
15	536367
16	536367
17	536367
18	536367
19	536367

- **`InvoiceNo`가 'C'로 시작하는 행을 필터링 할 수 있는 쿼리문을 작성하기 (100행까지만 출력)**

```
SELECT *
FROM `studied-source-439405-c0.modulabs_project.data`
WHERE InvoiceNo LIKE 'C%'
LIMIT 100;
```



쿼리 결과

작업 정보 | 결과 | 차트 | JSON | 실행 세부정보 | 실행 그래프

행	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country
1	C575531	22960	JAM MAKING SET WITH JARS	-4	2011-11-10 11:12:00 UTC	4.25	12544	Spain
2	C558080	22840	ROUND CAKE TIN VINTAGE RED	-1	2011-06-26 11:35:00 UTC	7.95	15104	United Kingdom
3	C558080	22847	BREAD BIN DINER STYLE IVORY	-1	2011-06-26 11:35:00 UTC	16.95	15104	United Kingdom
4	C554983	47590B	PINK HAPPY BIRTHDAY BUNTI...	-20	2011-05-29 12:18:00 UTC	4.65	17152	United Kingdom
5	C554983	47590A	BLUE HAPPY BIRTHDAY BUNTI...	-20	2011-05-29 12:18:00 UTC	4.65	17152	United Kingdom
6	C539709	21485	RETROSPOT HEART HOT WAT...	-1	2010-12-21 12:33:00 UTC	4.95	18176	United Kingdom
7	C539709	84978	HANGING HEART JAR T-LIGHT ...	-1	2010-12-21 12:33:00 UTC	1.25	18176	United Kingdom
8	C539709	22832	BROCANTE SHELF WITH HOOKS	-2	2010-12-21 12:33:00 UTC	10.75	18176	United Kingdom

더보기

- **구매 건 상태가 `Canceled` 인 데이터의 비율(%) - 소수점 첫번째 자리까지**

```
SELECT
  ROUND(SUM(CASE WHEN InvoiceNo LIKE 'C%' THEN 1 ELSE 0 END) / COUNT(*) * 100, 1) AS canceled_percen
FROM
  `studied-source-439405-c0.modulabs_project.data`;
```

쿼리 결과

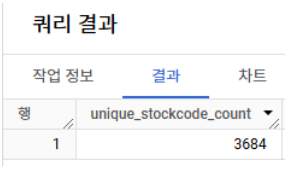
작업 정보 | 결과

행	canceled_percentage
1	2.2

`StockCode` 살펴보기

- **고유한 `StockCode`의 개수를 출력하기**

```
SELECT COUNT(DISTINCT StockCode) AS unique_stockcode_count
FROM `studied-source-439405-c0.modulabs_project.data`;
```



- **어떤 제품이 가장 많이 판매되었는지 보기 위하여 `StockCode` 별 등장 빈도를 출력하기**
  - 상위 10개의 제품들을 출력하기

```
SELECT StockCode, COUNT(*) AS sell_cnt
FROM `studied-source-439405-c0.modulabs_project.data`
GROUP BY StockCode
ORDER BY sell_cnt DESC
LIMIT 10;
```

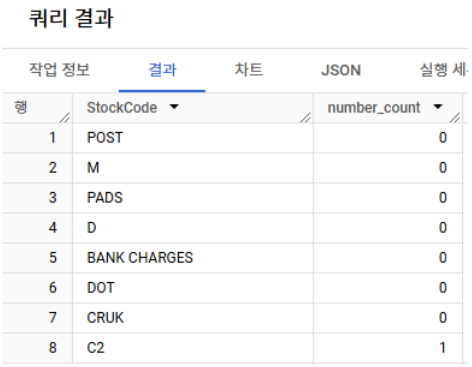
쿼리 결과

작업 정보 | 결과 | 차트 | JSON | 실행 세

행	StockCode	sell_cnt
1	85123A	2065
2	22423	1894
3	85099B	1659
4	47566	1409
5	84879	1405
6	20725	1346
7	22720	1224
8	POST	1196
9	22197	1110
10	23203	1108

- **`StockCode`의 컬럼에 있던 값 중에서 숫자를 제외한 문자만 남기고 문자가 몇 자리 수 인지 세고**
  - **숫자가 0~1개인 값**들에는 어떤 코드들이 들어가 있는지 출력하기

```
SELECT DISTINCT StockCode, number_count
FROM (
  SELECT StockCode,
    LENGTH(StockCode) - LENGTH(REGEXP_REPLACE(StockCode, r'[0-9]', '')) AS number_count
  FROM `studied-source-439405-c0.modulabs_project.data`
)
WHERE number_count <= 1;
```

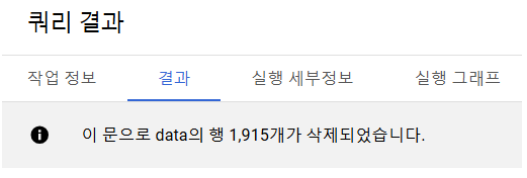


- `StockCode`의 컬럼에 있던 값 중에서 숫자를 제외한 문자만 남기고 문자가 몇 자리 수 인지 세고
  - **숫자가 0~1개인 값들을 가지고 있는 데이터 수는 전체 데이터 수 대비 몇 퍼센트**인지 구하기 (소수점 두 번째 자리까지)

```
SELECT
  ROUND((SUM(CASE WHEN number_count <= 1 THEN 1 ELSE 0 END) / COUNT(*)) * 100, 2) AS percentage_with
FROM (
  SELECT StockCode,
    LENGTH(StockCode) - LENGTH(REGEXP_REPLACE(StockCode, r'[0-9]', '')) AS number_count
  FROM `studied-source-439405-c0.modulabs_project.data`
);
```

- **제품과 관련되지 않은 거래 기록을 제거하기**

```
DELETE FROM `studied-source-439405-c0.modulabs_project.data`
WHERE StockCode IN (
    SELECT DISTINCT StockCode
    FROM (
        SELECT StockCode,
               LENGTH(StockCode) - LENGTH(REGEXP_REPLACE(StockCode, r'[0-9]', '')) AS number_count
        FROM `studied-source-439405-c0.modulabs_project.data`
    )
    WHERE number_count < 5  -- 숫자가 5개 미만인 경우를 제품과 관련되지 않은 것으로 간주
);
```



`Description` 살펴보기

- **고유한 Description 별 출현 빈도를 계산하고 상위 30개를 출력하기**

```
SELECT Description, COUNT(*) AS frequency
FROM `studied-source-439405-c0.modulabs_project.data`
GROUP BY Description
ORDER BY frequency DESC
LIMIT 30;
```

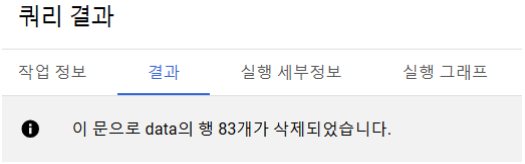
쿼리 결과

작업 정보 | 결과 | 차트 | JSON | 실행

행	Description	frequency
1	WHITE HANGING HEART T-LIG...	2058
2	REGENCY CAKESTAND 3 TIER	1894
3	JUMBO BAG RED RETROSPOT	1659
4	PARTY BUNTING	1409
5	ASSORTED COLOUR BIRD ORN...	1405
6	LUNCH BAG RED RETROSPOT	1345
7	SET OF 3 CAKE TINS PANTRY ...	1224
8	POSTAGE	1196
9	LUNCH BAG BLACK SKULL.	1099
10	PACK OF 72 RETROSPOT CAKE...	1062
11	SPOTTY BUNTING	1026

- **서비스 관련 정보를 포함하는 행들을 제거하기**

```
DELETE FROM `studied-source-439405-c0.modulabs_project.data`
WHERE UPPER(Description) IN ('NEXT DAY CARRIAGE', 'HIGH RESOLUTION IMAGE');
```



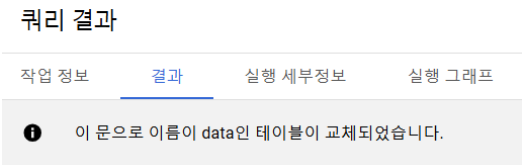
쿼리 결과

작업 정보 | 결과 | 실행 세부정보 | 실행 그래프

이 문으로 data의 행 83개가 삭제되었습니다.

- **대소문자를 혼합하고 있는 데이터를 대문자로 표준화 하기**

```
CREATE OR REPLACE TABLE `studied-source-439405-c0.modulabs_project.data` AS
SELECT
  * EXCEPT (Description),
  UPPER(Description) AS Description
FROM `studied-source-439405-c0.modulabs_project.data`;
```



쿼리 결과

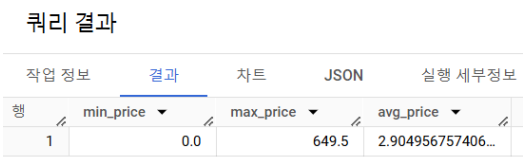
작업 정보 | 결과 | 실행 세부정보 | 실행 그래프

이 문으로 이름이 data인 테이블이 교체되었습니다.

`UnitPrice` 살펴보기

- `UnitPrice`**의 최솟값, 최댓값, 평균을 구하기**

```
SELECT
  MIN(UnitPrice) AS min_price,
  MAX(UnitPrice) AS max_price,
  AVG(UnitPrice) AS avg_price
FROM `studied-source-439405-c0.modulabs_project.data`;
```

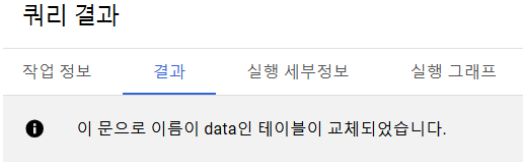


- **단가가 0원인 거래의 개수, 구매 수량(`Quantity`)의 최솟값, 최댓값, 평균 구하기**

```
SELECT
  COUNT(*) AS zero_price_count,
  MIN(Quantity) AS min_quantity,
  MAX(Quantity) AS max_quantity,
  AVG(Quantity) AS avg_quantity
FROM `studied-source-439405-c0.modulabs_project.data`
WHERE UnitPrice = 0;
```

- **`UnitPrice = 0`를 제거하고 일관된 데이터셋을 유지하기**

```
CREATE OR REPLACE TABLE `studied-source-439405-c0.modulabs_project.data` AS
SELECT *
FROM `studied-source-439405-c0.modulabs_project.data`
WHERE UnitPrice != 0;
```

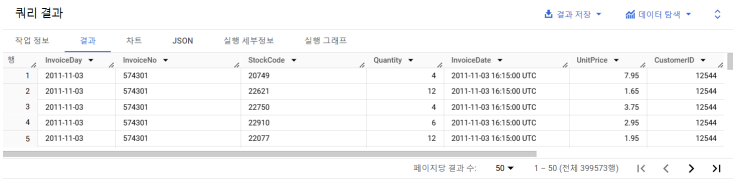


11-7. RFM 스코어

Recency

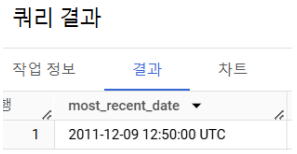
- **`InvoiceDate` 컬럼을 연월일 자료형으로 변경하기**

```
SELECT DATE(InvoiceDate) AS InvoiceDay, *
FROM `studied-source-439405-c0.modulabs_project.data`;
```



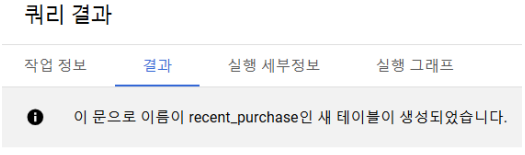
- **가장 최근 구매 일자를 MAX() 함수로 찾아보기**

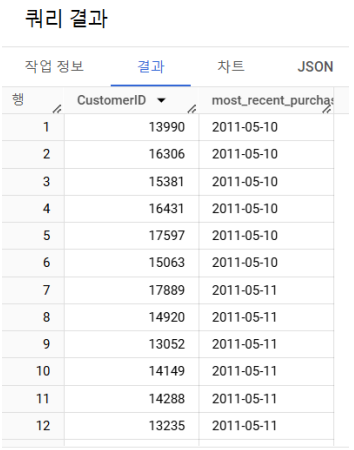
```
SELECT MAX(InvoiceDate) AS most_recent_date,
FROM `studied-source-439405-c0.modulabs_project.data`;
```



- **유저 별로 가장 큰 InvoiceDay를 찾아서 가장 최근 구매일로 저장하기**

```
CREATE OR REPLACE TABLE `studied-source-439405-c0.modulabs_project.recent_purchase` AS
SELECT
    CustomerID,
    MAX(DATE(InvoiceDate)) AS most_recent_purchase
FROM `studied-source-439405-c0.modulabs_project.data`
GROUP BY CustomerID;

SELECT *
FROM `studied-source-439405-c0.modulabs_project.recent_purchase`;
```

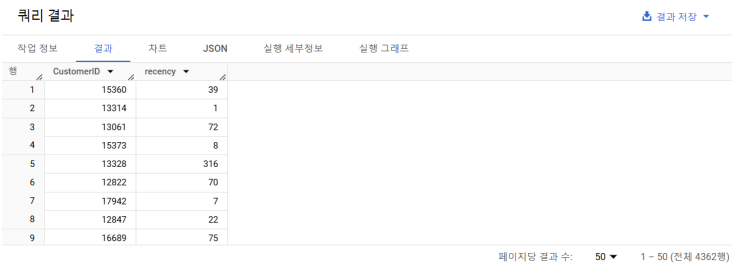




- **가장 최근 일자(`most_recent_date`)와 유저별 마지막 구매일(`InvoiceDay`)간의 차이를 계산하기**

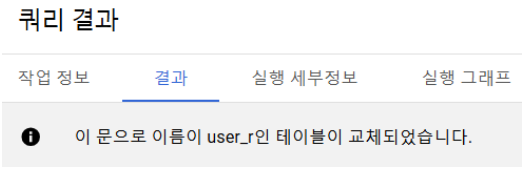
```
SELECT
  CustomerID,
```

```
  EXTRACT(DAY FROM MAX(InvoiceDay) OVER () - InvoiceDay) AS recency
FROM (
  SELECT
    CustomerID,
    MAX(DATE(InvoiceDate)) AS InvoiceDay
  FROM studied-source-439405-c0.modulabs_project.data
  GROUP BY CustomerID
);
```



- **최종 데이터 셋에 필요한 데이터들을 각각 정제해서 이어 붙이고 지금까지의 결과를 `user_r` 이라는 이름의 테이블로 저장하기**

```
CREATE OR REPLACE TABLE `studied-source-439405-c0.modulabs_project.user_r` AS
SELECT
    CustomerID,
    EXTRACT(DAY FROM MAX(InvoiceDay) OVER () - InvoiceDay) AS recency
FROM (
    SELECT
        CustomerID,
        MAX(DATE(InvoiceDate)) AS InvoiceDay
    FROM `studied-source-439405-c0.modulabs_project.data`
    GROUP BY CustomerID
);

CREATE OR REPLACE TABLE `studied-source-439405-c0.modulabs_project.user_r` AS
SELECT
  CustomerID,
  EXTRACT(DAY FROM MAX(InvoiceDay) OVER () - InvoiceDay) AS recency
FROM (
  SELECT
    CustomerID,
    MAX(DATE(InvoiceDate)) AS InvoiceDay
  FROM `studied-source-439405-c0.modulabs_project.data`
  GROUP BY CustomerID
);
```

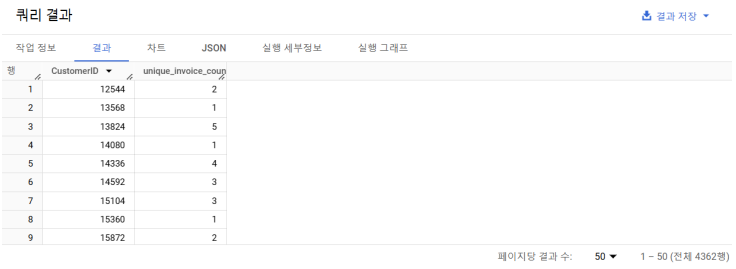


Frequency

- **고객마다 고유한 InvoiceNo의 수를 세어보기**

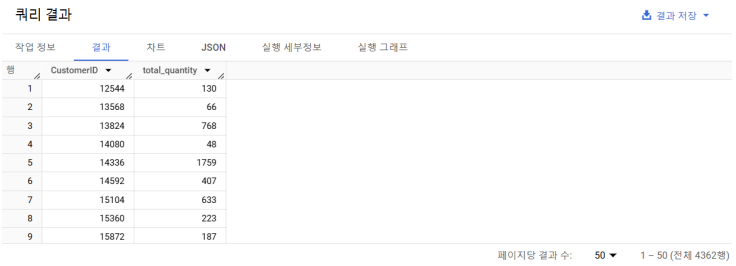
```
SELECT
    CustomerID,
    COUNT(DISTINCT InvoiceNo) AS unique_invoice_count
```

```
FROM `studied-source-439405-c0.modulabs_project.data`
GROUP BY CustomerID;
```



- **각 고객 별로 구매한 아이템의 총 수량 더하기**

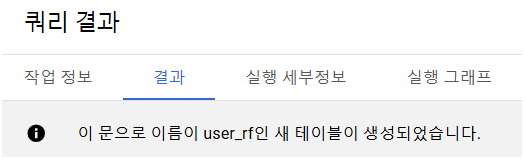
```
SELECT
    CustomerID,
    SUM(Quantity) AS total_quantity
FROM `studied-source-439405-c0.modulabs_project.data`
GROUP BY CustomerID;
```



- **전체 거래 건수 계산와 구매한 아이템의 총 수량 계산의 결과를 합쳐서 `user_rf` 라는 이름의 테이블에 저장하기**

```
CREATE OR REPLACE TABLE `studied-source-439405-c0.modulabs_project.user_rf` AS

-- (1) 전체 거래 건수 계산
WITH purchase_cnt AS (
  SELECT
    CustomerID,
    COUNT(DISTINCT InvoiceNo) AS purchase_cnt  -- 고객별 고유한 거래 건수 계산
  FROM `studied-source-439405-c0.modulabs_project.data`
  GROUP BY CustomerID
),

-- (2) 구매한 아이템 총 수량 계산
item_cnt AS (
  SELECT
    CustomerID,
    SUM(Quantity) AS item_cnt  -- 고객별 총 구매한 아이템 수량 계산
  FROM `studied-source-439405-c0.modulabs_project.data`
  GROUP BY CustomerID
)

-- 기존 user_r 테이블에 (1)과 (2)를 통합하여 최종 결과 생성
SELECT
  pc.CustomerID,
  pc.purchase_cnt,
```

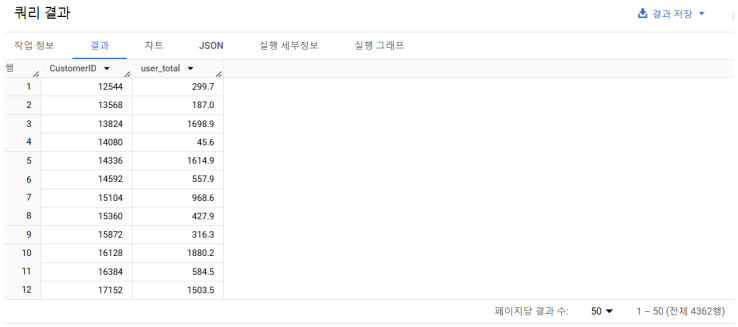
```
  ic.item_cnt,
  ur.recency,
FROM purchase_cnt AS pc
JOIN item_cnt AS ic
  ON pc.CustomerID = ic.CustomerID
JOIN `studied-source-439405-c0.modulabs_project.user_r` AS ur
  ON pc.CustomerID = ur.CustomerID;


-- CustomerID: 고객 고유 식별자.
-- purchase_cnt: 고객이 진행한 고유한 거래 건수 (중복되지 않은 InvoiceNo 기준).
-- item_cnt: 고객이 구매한 모든 아이템의 총 수량.
-- most_recent_purchase: 고객의 가장 최근 구매일.
-- total_purchases: 고객의 전체 구매 횟수.
-- total_spent: 고객이 지출한 총 금액.
```



Monetary

- **고객별 총 지출액 계산 (소수점 첫째 자리에서 반올림)**

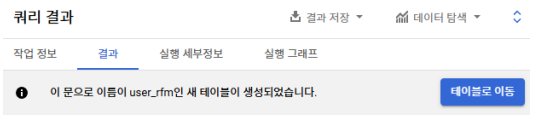
```
SELECT
  CustomerID,
  ROUND(SUM(UnitPrice * Quantity), 1) AS user_total  -- 소수점 첫째 자리에서 반올림한 총 지출액
FROM `studied-source-439405-c0.modulabs_project.data`
GROUP BY CustomerID;
```



- **고객별 평균 거래 금액 계산**
  - **고객별 평균 거래 금액을 구하기 위해 1) `data` 테이블을 `user_rf` 테이블과 조인(LEFT JOIN) 한 후, 2) `purchase_cnt` 로 나누어서 3) `user_rfm` 테이블로 저장하기**

```
CREATE OR REPLACE TABLE `studied-source-439405-c0.modulabs_project.user_rfm` AS

-- (1) user_rf 테이블과 data 테이블을 조인하여 총 지출 금액과 평균 거래 금액 계산
SELECT
  rf.CustomerID AS CustomerID,              -- 고객 ID
  rf.purchase_cnt,                          -- 총 거래 횟수
  rf.item_cnt,                              -- 총 구매 아이템 수
  rf.recency,                               -- 최근 거래 후 경과 일수
  ut.user_total,                            -- 총 지출 금액
  ROUND(ut.user_total / rf.purchase_cnt, 1) AS user_average  -- 평균 거래 금액 (소수점 첫째 자리에서 반올림)
```

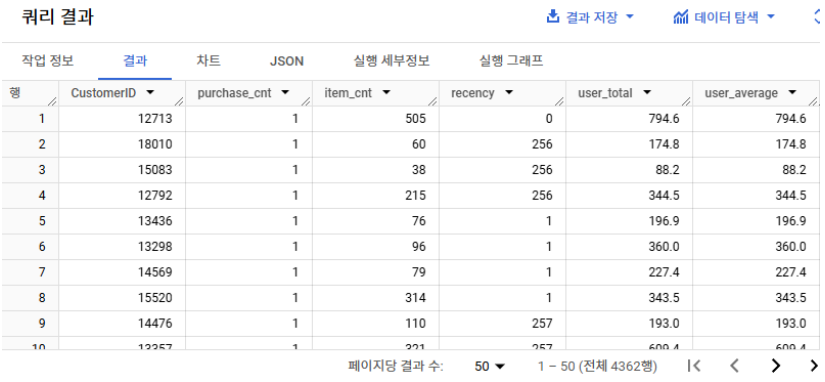
```
FROM `studied-source-439405-c0.modulabs_project.user_rf` rf
LEFT JOIN (
  -- 고객별 총 지출액 계산 (user_total)
  SELECT
    CustomerID,
    ROUND(SUM(UnitPrice * Quantity), 1) AS user_total  -- 소수점 첫째 자리에서 반올림한 총 지출 금액
  FROM `studied-source-439405-c0.modulabs_project.data`
  GROUP BY CustomerID
) ut
ON rf.CustomerID = ut.CustomerID;
```



RFM 통합 테이블 출력하기

- **최종 user_rfm 테이블을 출력하기**

```
SELECT * FROM studied-source-439405-c0.modulabs_project.user_rfm
```



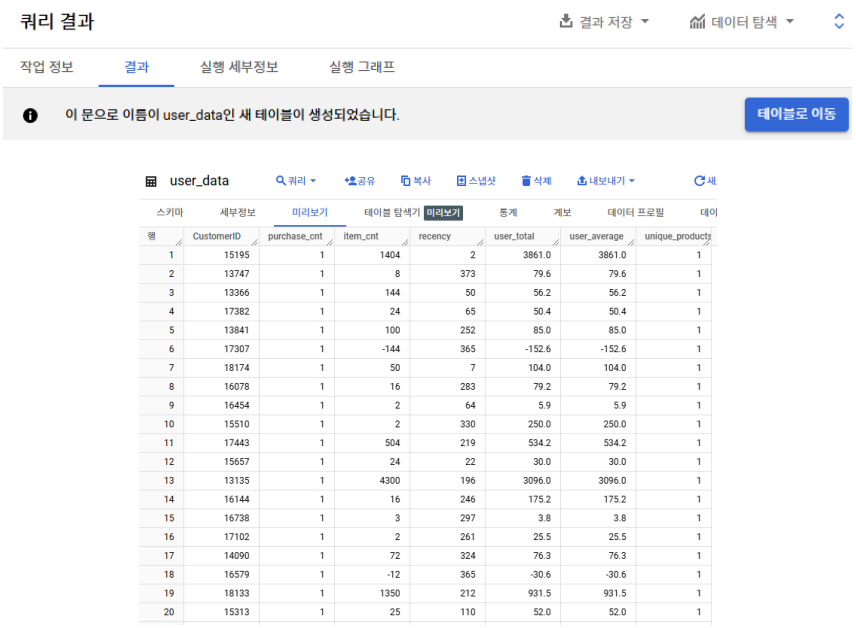
11-8. 추가 Feature 추출

1. 구매하는 제품의 다양성

- **1) 고객 별로 구매한 상품들의 고유한 수를 계산하기**
  2)
  `user_rfm` **테이블과 결과를 합치기**
  3)
  `user_data` **라는 이름의 테이블에 저장하기**

```
CREATE OR REPLACE TABLE studied-source-439405-c0.modulabs_project.user_data AS
WITH unique_products AS (
  SELECT
    CustomerID,
    COUNT(DISTINCT StockCode) AS unique_products
  FROM studied-source-439405-c0.modulabs_project.data
  GROUP BY CustomerID
)
SELECT ur.*, up.* EXCEPT (CustomerID)
FROM studied-source-439405-c0.modulabs_project.user_rfm AS ur
```

```
JOIN unique_products AS up
ON ur.CustomerID = up.CustomerID;
```



행	CustomerID	purchase_cnt	item_cnt	recency	user_total	user_average	unique_products
1	15195	1	1404	2	3861.0	3861.0	1
2	13747	1	8	373	79.6	79.6	1
3	13366	1	144	50	56.2	56.2	1
4	17382	1	24	65	50.4	50.4	1
5	13841	1	100	252	85.0	85.0	1
6	17307	1	-144	365	-152.6	-152.6	1
7	18174	1	50	7	104.0	104.0	1
8	16078	1	16	283	79.2	79.2	1
9	16454	1	2	64	5.9	5.9	1
10	15510	1	2	330	250.0	250.0	1
11	17443	1	504	219	534.2	534.2	1
12	15657	1	24	22	30.0	30.0	1
13	13135	1	4300	196	3096.0	3096.0	1
14	16144	1	16	246	175.2	175.2	1
15	16738	1	3	297	3.8	3.8	1
16	17102	1	2	261	25.5	25.5	1
17	14090	1	72	324	76.3	76.3	1
18	16579	1	-12	365	-30.6	-30.6	1
19	18133	1	1350	212	931.5	931.5	1
20	15313	1	25	110	52.0	52.0	1

2. 평균 구매 주기

- **고객들의 쇼핑 패턴을 이해하는 것을 목표 (고객 별 재방문 주기 살펴보기)**
  - **균 구매 소요 일수를 계산하고, 그 결과를 `user_data` 에 통합**

```
CREATE OR REPLACE TABLE studied-source-439405-c0.modulabs_project.user_data AS
WITH purchase_intervals AS (
  -- (2) 고객 별 구매와 구매 사이의 평균 소요 일수
  SELECT
    CustomerID,
    CASE WHEN ROUND(AVG(interval_), 2) IS NULL THEN 0 ELSE ROUND(AVG(interval_), 2) END AS average_inte
  FROM (
    -- (1) 구매와 구매 사이에 소요된 일수
    SELECT
      CustomerID,
      DATE_DIFF(InvoiceDate, LAG(InvoiceDate) OVER (PARTITION BY CustomerID ORDER BY InvoiceDate), DAY)
    FROM
      studied-source-439405-c0.modulabs_project.data
    WHERE CustomerID IS NOT NULL
  )
  GROUP BY CustomerID
)

SELECT u.*, pi.* EXCEPT (CustomerID)
FROM studied-source-439405-c0.modulabs_project.user_data AS u
LEFT JOIN purchase_intervals AS pi
ON u.CustomerID = pi.CustomerID;
```

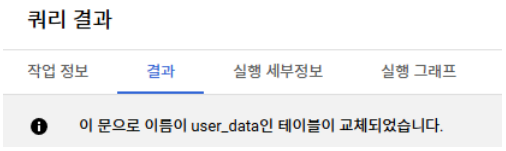
행	CustomerID	purchase_cnt	item_cnt	recency	user_total	user_average	unique_products
1	15195	1	1404	2	3861.0	3861.0	1
2	13747	1	8	373	79.6	79.6	1
3	13366	1	144	50	56.2	56.2	1
4	17382	1	24	65	50.4	50.4	1
5	13841	1	100	252	85.0	85.0	1
6	17307	1	-144	365	-152.6	-152.6	1
7	18174	1	50	7	104.0	104.0	1
8	16078	1	16	283	79.2	79.2	1
9	16454	1	2	64	5.9	5.9	1
10	15510	1	2	330	250.0	250.0	1
11	17443	1	504	219	534.2	534.2	1
12	15657	1	24	22	30.0	30.0	1
13	13135	1	4300	196	3096.0	3096.0	1
14	16144	1	16	246	175.2	175.2	1
15	16738	1	3	297	3.8	3.8	1
16	17102	1	2	261	25.5	25.5	1
17	14090	1	72	324	76.3	76.3	1
18	16579	1	-12	365	-30.6	-30.6	1
19	18133	1	1350	212	931.5	931.5	1
20	15313	1	25	110	52.0	52.0	1

3. 구매 취소 경향성

- **고객의 취소 패턴 파악하기**
  1) 취소 빈도(cancel_frequency) : 고객 별로 취소한 거래의 총 횟수
  2) 취소 비율(cancel_rate) : 각 고객이 한 모든 거래 중에서 취소를 한 거래의 비율
  - **취소 빈도와 취소 비율을 계산하고 그 결과를 `user_data`에 통합하기**
    (취소 비율은 소수점 두번째 자리)

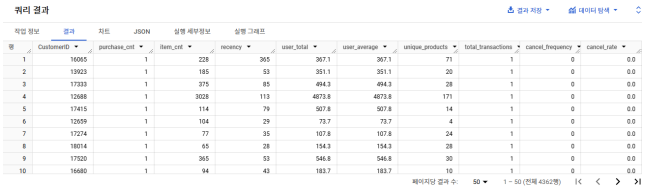
```
CREATE OR REPLACE TABLE `studied-source-439405-c0.modulabs_project.user_data` AS

WITH TransactionInfo AS (
  -- (1) 중복 제거를 위한 서브쿼리
  WITH sup AS (
    SELECT DISTINCT InvoiceNo, CustomerID
    FROM `studied-source-439405-c0.modulabs_project.data`
  )
  -- (2) 고객별 전체 거래 수와 취소 빈도 계산
  SELECT
    CustomerID,
    COUNT(DISTINCT InvoiceNo) AS total_transactions,  -- 고객의 전체 거래 수
    SUM(CASE WHEN InvoiceNo LIKE 'C%' THEN 1 ELSE 0 END) AS cancel_frequency  -- 취소된 거래 수
  FROM sup
  GROUP BY CustomerID
)

-- (3) 기존 user_data 테이블의 모든 컬럼을 명시적으로 지정하여 결합
SELECT
  u.CustomerID,
  u.purchase_cnt,
  u.item_cnt,
  u.recency,
  u.user_total,
  u.user_average,
  u.unique_products,
  t.cancel_frequency,
  ROUND(t.cancel_frequency / NULLIF(t.total_transactions, 0), 2) AS cancel_rate  -- 취소 비율 계산 (소수점
```

```
FROM `studied-source-439405-c0.modulabs_project.user_data` AS u
LEFT JOIN TransactionInfo AS t
ON u.CustomerID = t.CustomerID;
```



- **다양한 컬럼들을 활용하여 고객의 구매 패턴과 선호도를 보다 심층적으로 이해할 수 있도록 최종적으로 `user_data`를 출력하기**

```
select * from studied-source-439405-c0.modulabs_project.user_data
```



회고

SQL 쿼리 학습 회고

SQL에 대해 어느 정도 기초적인 지식은 있었지만, 이번 프로젝트에서 쿼리를 실제 분석에 활용해본 경험은 처음이었습니다. **고객별 취소 패턴 분석**과 같은 세부 분석을 진행하며, SQL이 단순한 데이터 조회를 넘어 복잡한 데이터를 정제하는 작업에 강력한 도구라는 것을 느꼈습니다.

SQL을 활용하여 **데이터를 정제하고 필요한 정보를 추출하는 과정**은 기대 이상으로 흥미로웠습니다. 특히 **CTE**와 **JOIN**을 사용해 데이터를 결합하고 중복 컬럼을 처리하는 과정에서 많은 배움을 얻었습니다. 그동안 단순히 공부했던 SQL이 실제로 어떻게 분석에 활용되는지에 대한 이해도가 높아졌습니다.

어려움과 해결 과정

사실 데이터를 다루며 여러 문제에 부딪혔습니다. 오류 메시지를 이해하고 문제의 원인을 파악하는 데 시간이 걸렸고, 특히 중복된 컬럼 처리나 조건부 집계에서 예상치 못한 부분이 자주 발생했습니다. 그러나 문제를 하나하나 해결하는 과정에서 SQL 쿼리에 대한 이해가 깊어졌고, 데이터 사이언티스트로서 논리적 사고와 꼼꼼함이 필요하다는 것을 실감했습니다.

느낀 점과 앞으로의 목표

이번 프로젝트를 통해 SQL의 중요성과 강력함을 다시금 느꼈고, 특히 대용량 데이터를 다룰 때의 효율성을 깨달았습니다. 앞으로 SQL을 더욱 깊이 있게 학습해 데이터 분석의 효율성을 높이고, 실무에서 문제를 해결하는 능력을 길러야겠다고 다짐했습니다.