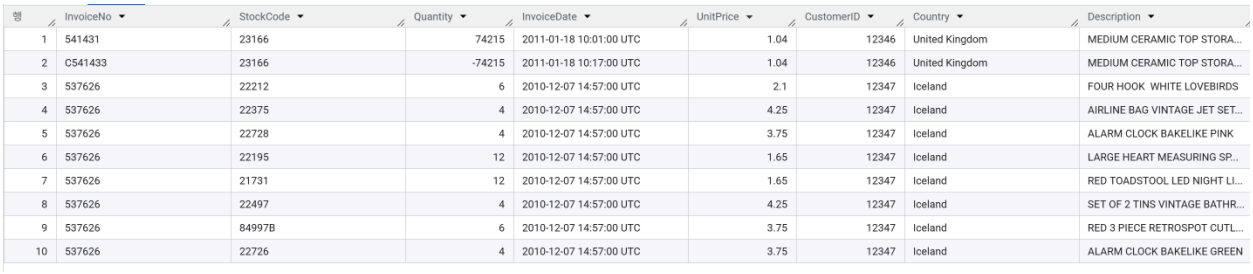
11-2. 데이터 불러오기

테이블에 있는 10개의 행만 출력

```
select *
 from symmetric-ace-465202-c6.modulabs_project.data
 LIMIT 10;
```

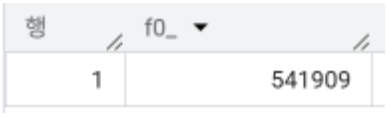


행	InvoiceNo	StockCode	Quantity	InvoiceDate	UnitPrice	CustomerID	Country	Description
1	541431	23166	74215	2011-01-18 10:01:00 UTC	1.04	12346	United Kingdom	MEDIUM CERAMIC TOP STORA...
2	C541433	23166	-74215	2011-01-18 10:17:00 UTC	1.04	12346	United Kingdom	MEDIUM CERAMIC TOP STORA...
3	537626	22212	6	2010-12-07 14:57:00 UTC	2.1	12347	Iceland	FOUR HOOK WHITE LOVEBIRDS
4	537626	22375	4	2010-12-07 14:57:00 UTC	4.25	12347	Iceland	AIRLINE BAG VINTAGE JET SET...
5	537626	22728	4	2010-12-07 14:57:00 UTC	3.75	12347	Iceland	ALARM CLOCK BAKELIKE PINK
6	537626	22195	12	2010-12-07 14:57:00 UTC	1.65	12347	Iceland	LARGE HEART MEASURING SP...
7	537626	21731	12	2010-12-07 14:57:00 UTC	1.65	12347	Iceland	RED TOADSTOOL LED NIGHT LI...
8	537626	22497	4	2010-12-07 14:57:00 UTC	4.25	12347	Iceland	SET OF 2 TINS VINTAGE BATHR...
9	537626	84997B	6	2010-12-07 14:57:00 UTC	3.75	12347	Iceland	RED 3 PIECE RETROSPOT CUTL...
10	537626	22726	4	2010-12-07 14:57:00 UTC	3.75	12347	Iceland	ALARM CLOCK BAKELIKE GREEN

데이터 셋을 구성하는 컬럼의 갯수는 8개

전체 데이터는 몇 행으로 구성되어 있는지 확인

```
select count(*)
from symmetric-ace-465202-c6.modulabs_project.data;
```

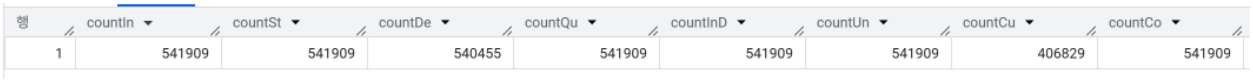


행	f0_
1	541909

모든 컬럼에 대하여 COUNT 함수를 적용

```
select
count(InvoiceNo) as countIn,
count(StockCode) as countSt,
count(Description) as countDe,
```

```
count(Quantity) as countQu,
count(InvoiceDate) as countInD,
count(UnitPrice) as countUn,
count(CustomerID) as countCu,
count(Country) as countCo
from symmetric-ace-465202-c6.modulabs_project.data;
```



행	countIn	countSt	countDe	countQu	countInD	countUn	countCu	countCo
1	541909	541909	540455	541909	541909	541909	406829	541909

누락된 값이 있는 컬럼은

Description, CustomerID

11-4. 데이터 전처리(1): 결측치 제거

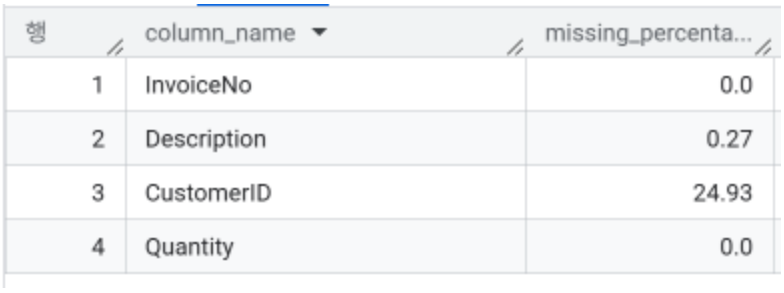
```
SELECT
    'InvoiceNo' AS column_name,
    ROUND(SUM(CASE WHEN InvoiceNo IS NULL THEN 1 ELSE 0 END) / COUNT(*
FROM project_name.modulabs_project.data
```

행	column_name	missing_percenta...
1	InvoiceNo	0.0

결측치 비율 계산 Union all

```
SELECT
    'InvoiceNo' AS column_name,
    ROUND(SUM(CASE WHEN InvoiceNo IS NULL THEN 1 ELSE 0 END) / COUNT(*
from symmetric-ace-465202-c6.modulabs_project.data
```

```
UNION ALL

SELECT
   'CustomerID' AS column_name,
   ROUND(SUM(CASE WHEN CustomerID IS NULL THEN 1 ELSE 0 END) / COUNT
FROM symmetric-ace-465202-c6.modulabs_project.data

UNION ALL

SELECT
   'Description' AS column_name,
   ROUND(SUM(CASE WHEN Description IS NULL THEN 1 ELSE 0 END) / COUNT
FROM symmetric-ace-465202-c6.modulabs_project.data

UNION ALL

SELECT
   'Quantity' AS column_name,
   ROUND(SUM(CASE WHEN Quantity IS NULL THEN 1 ELSE 0 END) / COUNT(*)
FROM symmetric-ace-465202-c6.modulabs_project.data;
```



행	column_name	missing_percenta...
1	InvoiceNo	0.0
2	Description	0.27
3	CustomerID	24.93
4	Quantity	0.0

결측치 비율 계산 case when

```
SELECT column_name, ROUND((total - column_value) / total * 100, 2)
FROM
(
   SELECT 'InvoiceNo' AS column_name, COUNT(InvoiceNo) AS column_value, C
```

```
    SELECT 'StockCode' AS column_name, COUNT(StockCode) AS column_value,
    SELECT 'Description' AS column_name, COUNT(Description) AS column_value
    SELECT 'Quantity' AS column_name, COUNT(Quantity) AS column_value, COU
    SELECT 'InvoiceDate' AS column_name, COUNT(InvoiceDate) AS column_valu
    SELECT 'UnitPrice' AS column_name, COUNT(UnitPrice) AS column_value, CO
    SELECT 'CustomerID' AS column_name, COUNT(CustomerID) AS column_valu
    SELECT 'Country' AS column_name, COUNT(Country) AS column_value, COUI
) AS column_data;
```

행	column_name	f0_
1	UnitPrice	0.0
2	CustomerID	24.93
3	Country	0.0
4	StockCode	0.0
5	InvoiceNo	0.0
6	Quantity	0.0
7	InvoiceDate	0.0
8	Description	0.27

결측치 처리 전략

CustomerID (24.93%)

CustomerID는 고객을 클러스터링할 때 필수적인 정보입니다. 그러나 결측치의 비중이 약 4분의 1이나 됩니다.

이렇게 큰 비율의 누락된 값을 다른 값으로 대체하는 것은 분석에 상당한 편향을 주고 노이즈가 될 수 있습니다. 또한 궁극적으로 여러분이 진행하는 프로젝트는 RFM 분석 기법에 따른 고객 세그먼테이션이기 때문에, 고객 식별자 데이터는 정확해야 합니다. 따라서 누락된 CustomerID가 있는 행을 제거하는 것이 가장 합리적인 접근 방법으로 보입니다.

Description (0.27%)

```
select DISTINCT Description
from symmetric-ace-465202-c6.modulabs_project.data
where StockCode = '85123A';
```

행	Description
1	WHITE HANGING HEART T-LIG...
2	?
3	wrongly marked carton 22804
4	CREAM HANGING HEART T-LIG...

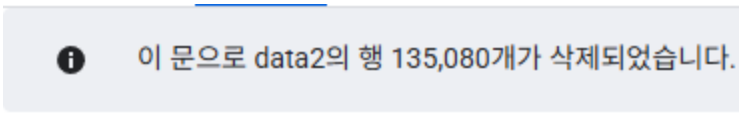
제품을 지칭하는 설명(`Description`)이 'WHITE HANGING HEART T-LIGHT HOLDER', 'CREAM HANGING HEART T-LIGHT HOLDER', 'wrongly marked carton 22804', '?'로, 4개의 설명이 모두 다릅니다.

이는 제품 설명(`Description`)이 일관적으로 기록되지 않은 오류로 해석할 수 있습니다.

일관성 결여를 고려할 때, `StockCode`를 기반으로 누락된 설명을 대체하는 것은 신뢰할 수 없을 가능성이 있습니다. 또한 누락된 비율이 0.27%로 매우 낮기 때문에, 데이터의 일관성 문제가 후속 분석 과정에 영향을 주지 않게 하기 위해 누락된 설명이 있는 행을 제거하는 것이 현명

결측치 처리

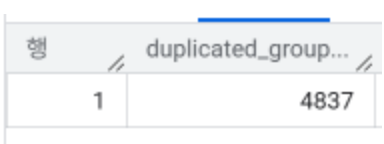
```
DELETE FROM symmetric-ace-465202-c6.modulabs_project.data
WHERE CustomerID IS NULL
  or Description IS NULL;
```



11-5. 데이터 전처리(2): 중복값 처리

중복값 확인

```
SELECT COUNT(*) AS duplicated_group_count
FROM (
    SELECT
        InvoiceNo,
        StockCode,
        Description,
        Quantity,
        InvoiceDate,
        UnitPrice,
        CustomerID,
        Country,
        COUNT(*) AS cnt
    FROM symmetric-ace-465202-c6.modulabs_project.data
    GROUP BY
        InvoiceNo,
        StockCode,
        Description,
        Quantity,
        InvoiceDate,
        UnitPrice,
        CustomerID,
        Country
    HAVING COUNT(*) > 1
) AS subquery;
```

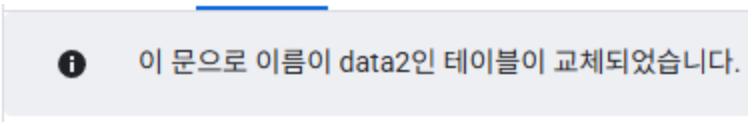


행	duplicated_group...
1	4837

중복값 처리

동일한 거래 시간을 포함한 동일한 행은 데이터 오류일 가능성이 높습니다. 이러한 중복 행을 유지하면 분석 결과에 영향을 줄 수가 있습니다. 따라서 데이터셋에서 완전히 동일한 중복 행들을 제거하도록 하겠습니다.

```
create or replace table symmetric-ace-465202-c6.modulabs_project.data AS
SELECT distinct *
from symmetric-ace-465202-c6.modulabs_project.data
```



중복값 처리 후 남은 행

```
SELECT COUNT(*) AS remaining_rows
 FROM symmetric-ace-465202-c6.project.data2;
```

행	remaining_rows
1	401604

11-6. 데이터 전처리(3): 오류값 처리

InvoiceNo 살펴보기

고유(unique)한 InvoiceNo의 개수를 출력해 보세요.

```
select count(distinct InvoiceNo) as unique_invoice_count
from symmetric-ace-465202-c6.modulabs_project.data;
```

행	unique_invoice_c...
1	22190

고유한 `InvoiceNo` 를 100개를 출력

```
select distinct InvoiceNo
from symmetric-ace-465202-c6.modulabs_project.data
LIMIT 100;
```

행	InvoiceNo
1	541431
2	C541433
3	537626
4	542237
5	549222
6	556201
7	562032
8	573511

`InvoiceNo` 가 'C'로 시작하는 행을 필터링

```
SELECT *
FROM project_name.modulabs_project.data
WHERE InvoiceNo LIKE 'C%'
LIMIT 100;
```

행	InvoiceNo	StockCode
1	C541433	23166
2	C545329	M
3	C545329	M
4	C545330	M
5	C547388	22645
6	C547388	22413
7	C547388	21914
8	C547388	37448

구매 건 상태가 `Canceled` 인 데이터의 비율(%)

```
select
  round(sum(CASE WHEN InvoiceNo LIKE 'C%' then 1 else 0 end)/count(*) * 100,
  canceled_rate_pct
  from symmetric-ace-465202-c6.modulabs_project.data;
```

행	canceled_rate_pct
1	2.2

`StockCode` 살펴보기

고유한 `StockCode`의 개수를 출력

```
select count(distinct StockCode) as unSt
from symmetric-ace-465202-c6.modulabs_project.data;
```

행	unSt
1	3684

StockCode 별 등장 빈도

```
SELECT StockCode, COUNT(*) AS sell_cnt
FROM project_name.modulabs_project.data
GROUP BY StockCode
ORDER BY sell_cnt DESC
LIMIT 10;
```

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

StockCode 의 문자열 내 숫자의 길이

```
WITH UniqueStockCodes AS (
  SELECT DISTINCT StockCode
  FROM project_name.modulabs_project.data
)
SELECT
  LENGTH(StockCode) - LENGTH(REGEXP_REPLACE(StockCode, r'[0-9]', '')) AS
  COUNT(*) AS stock_cnt
FROM UniqueStockCodes
GROUP BY number_count
ORDER BY stock_cnt DESC;
```

행	number_count	stock_cnt
1	5	3676
2	0	7
3	1	1

숫자가 0~1개인 값들에는 어떤 코드들이 들어가 있는지를 확인

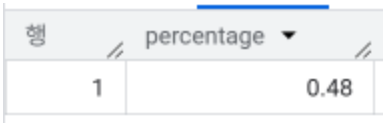
```
SELECT DISTINCT StockCode, number_count
FROM (
  SELECT StockCode,
    LENGTH(StockCode) - LENGTH(REGEXP_REPLACE(StockCode, r'[0-9]', '')) AS
  FROM project_name.modulabs_project.data
)
WHERE number_count <= 1;
```

행	StockCode
1	POST
2	M
3	C2
4	D
5	BANK CHARGES
6	PADS
7	DOT
8	CRUK

데이터 수는 전체 데이터 수 대비 몇 퍼센트

```
select
    round(
        (select count(*)
```

```
      from symmetric-ace-465202-c6.modulabs_project.data
      where LENGTH(StockCode) - LENGTH(REGEXP_REPLACE(StockCode, r'[0-
  )
  / count(*) *100, 2
)as percentage
from symmetric-ace-465202-c6.modulabs_project.data
```

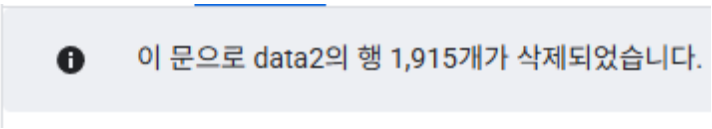


행	percentage
1	0.48

이 코드들은 'BANK CHARGES, POST' 등 제품과 관련되지 않은 거래 기록으로 보입니다. 진행하고 있는 프로젝트의 목표는 고객들의 '제품 구매'에 기반하여 세그먼테이션을 하는 것이므로, 이런 `StockCode` 가 포함된 기록은 데이터셋에서 제외하도록 하겠습니다.

제품과 관련되지 않은 거래 기록을 제거

```
DELETE FROM project_name.modulabs_project.data
WHERE StockCode IN (
  SELECT DISTINCT StockCode
  FROM (
    SELECT StockCode
    FROM symmetric-ace-465202-c6.modulabs_project.data
    where LENGTH(StockCode) - LENGTH(REGEXP_REPLACE(StockCode, r'[0-9]'
));
```



Description 살펴보기

고유한 Description 별 출현 빈도

```
SELECT Description, COUNT(*) AS description_cnt
FROM project_name.modulabs_project.data
ORDER BY description_cnt DESC
LIMIT 30;
```

행	Description	description_cnt
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 D...	1224
8	LUNCH BAG BLACK SKULL.	1099

대소문자가 혼합된 Description이 있는지 확인

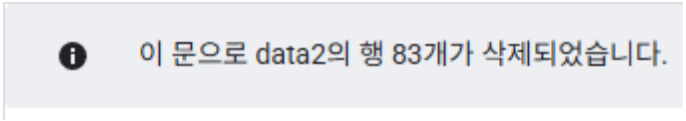
```
SELECT DISTINCT Description
FROM project_name.modulabs_project.data
WHERE REGEXP_CONTAINS(Description, r'[a-z]');
```

행	Description
1	BAG 250g SWIRLY MARBLES
2	3 TRADITIONAl BISCUIT CUTTE...
3	BAG 125g SWIRLY MARBLES
4	POLYESTER FILLER PAD 30CMx...
5	BAG 500g SWIRLY MARBLES
6	POLYESTER FILLER PAD 45x45...
7	POLYESTER FILLER PAD 40x40...
8	ESSENTIAL BALM 3.5g TIN IN E...

출력 결과를 보면 사이즈(cm)나 무게(g) 등의 단위를 나타내는 설명이 포함되어 있고, 'Next Day Carriage'나 'High Resolution Image'와 같은 일부 항목들처럼 실제 제품에 대한 Description이 아닌 것도 있는 것을 알 수 있습니다.

Next Day Carriage'와 'High Resolution Image'와 같은 서비스 관련 정보를 포함하는 행들을 제거

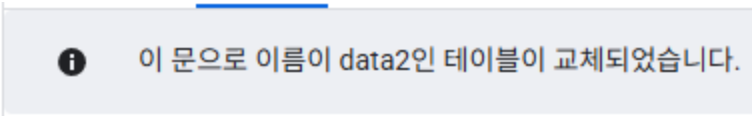
```
DELETE
 FROM symmetric-ace-465202-c6.modulabs_project.data
 WHERE
 Description IN('High Resolution Image', 'Next Day Carriage');
```



대소문자를 혼합하고 있는 데이터를 대문자로 표준화

```
CREATE OR REPLACE TABLE project_name.modulabs_project.data AS
SELECT
  * EXCEPT (Description),
```

```
  upper(Description) AS Description
FROM project_name.modulabs_project.data;
```



UnitPrice 살펴보기

UnitPrice의 최솟값, 최댓값, 평균

```
select
 MIN(UnitPrice) as min_price,
 MAX(UnitPrice) as max_price,
 AVG(UnitPrice) as avg_price
 from symmetric-ace-465202-c6.project.data2;
```

행	min_price	max_price	avg_price
1	0.0	649.5	2.904956757406...

단가가 0원인 거래의 개수, 구매 수량(Quantity)의 최솟값, 최댓값, 평균

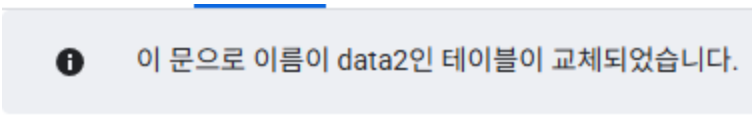
```
SELECT
  COUNT(*) AS cnt_quantity,
  MIN(Quantity) AS min_quantity,
  MAX(Quantity) AS max_quantity,
  AVG(Quantity) AS avg_quantity
FROM project_name.modulabs_project.data
WHERE UnitPrice = 0;
```

행	cnt_quantity	min_quantity	max_quantity	avg_quantity
1	33	1	12540	420.515151515

UnitPrice가 0인 행의 수는 33개로 비교적 적음

데이터의 수가 적은 걸 보니 무료 제품이라기보다 데이터 오류일 가능성이 더 높을 것 같습니다. 그래서 이 데이터(`UnitPrice = 0`)를 제거하고 일관된 데이터셋을 유지

```
CREATE OR REPLACE TABLE project_name.modulabs_project.data AS
SELECT *
FROM project_name.modulabs_project.data
WHERE UnitPrice != 0;
```



11-7. RFM 스코어

`InvoiceDate`를 '2010-12-01 08:26:00'와 같은 'YYYY-MM-DD HH:MM:SS' 형태에서 'YYYY-MM-DD' 형태로 날짜에 해당하는 부분만 남겨놓고 싶습니다. 이를 위해 DATE 함수를 활용하여 `InvoiceDate` 컬럼을 연월일 자료형으로 변경

```
SELECT
  InvoiceNo,
  CustomerID,
  DATE(InvoiceDate) AS InvoiceDate_only_date
FROM
  symmetric-ace-465202-c6.modulabs_project.data;
```

행	InvoiceNo	CustomerID	InvoiceDate_only
1	541431	12346	2011-01-18
2	C541433	12346	2011-01-18
3	537626	12347	2010-12-07
4	537626	12347	2010-12-07
5	537626	12347	2010-12-07
6	537626	12347	2010-12-07
7	537626	12347	2010-12-07
8	537626	12347	2010-12-07

가장 최근 구매 일자를 MAX()

```
SELECT
  MAX(DATE(InvoiceDate)) AS most_recent_date
FROM symmetric-ace-465202-c6.modulabs_project.data;
```

행	most_recent_date
1	2011-12-09

유저 별로 가장 큰 InvoiceDay를 찾아서 가장 최근 구매일로 저장

```
SELECT
   CustomerID,
   MAX(DATE(InvoiceDate)) AS InvoiceDay
FROM symmetric-ace-465202-c6.modulabs_project.data
group by CustomerID;
```

행	CustomerID	InvoiceDay
1	12346	2011-01-18
2	12347	2011-12-07
3	12348	2011-09-25
4	12349	2011-11-21
5	12350	2011-02-02
6	12352	2011-11-03
7	12353	2011-05-19
8	12354	2011-04-21

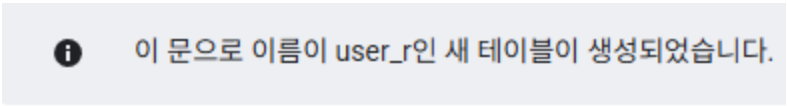
가장 최근 일자(`most_recent_date`)와 유저별 마지막 구매일(`InvoiceDay`)간의 차이를 계산

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

행	CustomerID	recency
1	12481	22
2	12580	246
3	12605	365
4	12683	4
5	12702	19
6	12929	122
7	13168	36
8	13221	240

최종 데이터 셋에 필요한 데이터들을 각각 정제해서 이어붙이도록 하겠습니다. 지금까지의 결과를 `user_r` 이라는 이름의 테이블로 저장

```
CREATE OR REPLACE TABLE symmetric-ace-465202-c6.modulabs_project.use
SELECT
  CustomerID,
  EXTRACT(DAY FROM MAX(InvoiceDay) OVER () - InvoiceDay) AS recency
FROM (
  SELECT
    CustomerID,
    MAX(DATE(InvoiceDate)) AS InvoiceDay
  FROM symmetric-ace-465202-c6.modulabs_project.data
  GROUP BY CustomerID
);
```



Frequency

1. 전체 거래 건수 계산

고객마다 고유한 InvoiceNo의 수

```
SELECT
  CustomerID,
  COUNT(DISTINCT InvoiceNo) AS purchase_cnt
FROM
  symmetric-ace-465202-c6.modulabs_project.data
GROUP BY
  CustomerID;
```

행	CustomerID	purchase_cnt
1	12346	2
2	12347	7
3	12348	4
4	12349	1
5	12350	1
6	12352	8
7	12353	1
8	12354	1

2. 구매한 아이템의 총 수량 계산

그 다음으로는 각 고객 별로 구매한 아이템의 총 수량

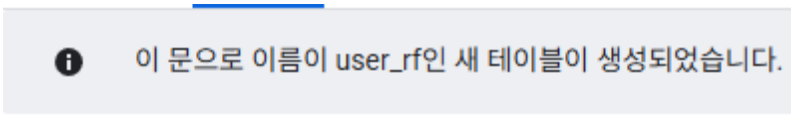
```
SELECT
  CustomerID,
  SUM(Quantity) AS item_cnt
FROM
  symmetric-ace-465202-c6.modulabs_project.data
GROUP BY
  CustomerID;
```

행	CustomerID	item_cnt
1	12346	0
2	12347	2458
3	12348	2332
4	12349	630
5	12350	196
6	12352	463
7	12353	20
8	12354	530

1. 전체 거래 건수 계산'과 '2. 구매한 아이템의 총 수량 계산'의 결과를 합쳐서 `user_rf` 라는 이름의 테이블에 저장

```
CREATE OR REPLACE TABLE project_name.modulabs_project.user_rf AS

-- (1) 전체 거래 건수 계산
WITH purchase_cnt AS (
  SELECT
  CustomerID,
  COUNT(DISTINCT InvoiceNo) AS purchase_cnt
FROM
  symmetric-ace-465202-c6.modulabs_project.data
GROUP BY
  CustomerID;
),

-- (2) 구매한 아이템 총 수량 계산
item_cnt AS (
  SELECT
  CustomerID,
  SUM(Quantity) AS item_cnt
FROM
  symmetric-ace-465202-c6.modulabs_project.data
GROUP BY
  CustomerID;
)

-- 기존의 user_r에 (1)과 (2)를 통합
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 project_name.modulabs_project.user_r AS ur
  ON pc.CustomerID = ur.CustomerID;
```



행	CustomerID	purchase_cnt	item_cnt	recency
1	12713	1	505	0
2	14569	1	79	1
3	13436	1	76	1
4	15520	1	314	1
5	13298	1	96	1
6	15471	1	256	2
7	14204	1	72	2
8	15195	1	1404	2

Monetary

Monetary를 계산하는 단계에서는 고객이 지불한 총 금액에 초점을 맞춥니다.

1. 고객별 총 지출액 계산

고객별 총 지출액을 계산해 보세요. 소수점 첫째 자리에서 반올림

```
SELECT
  CustomerID,
  ROUND(SUM(UnitPrice * Quantity), 1) AS user_total
FROM
  symmetric-ace-465202-c6.modulabs_project.data
GROUP BY
  CustomerID;
```

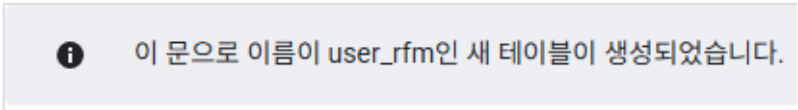
행	CustomerID	user_total
1	12346	0.0
2	12347	4310.0
3	12348	1437.2
4	12349	1457.5
5	12350	294.4
6	12352	1265.4
7	12353	89.0
8	12354	1079.4

2. 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 저장

```
CREATE OR REPLACE TABLE symmetric-ace-465202-c6.modulabs_project.user_
SELECT
  rf.CustomerID AS CustomerID,
  rf.purchase_cnt,
  rf.item_cnt,
  rf.recency,
  ut.user_total,
  ROUND(ut.user_total / rf.purchase_cnt, 1) AS user_average
FROM symmetric-ace-465202-c6.modulabs_project.user_rf rf
LEFT JOIN (
  -- 고객별 총 지출액 계산
  SELECT
    CustomerID,
    SUM(UnitPrice * Quantity) AS user_total
  FROM
    symmetric-ace-465202-c6.modulabs_project.data
  GROUP BY
    CustomerID
```

```
) ut
ON rf.CustomerID = ut.CustomerID;
```



RFM 통합 테이블 출력하기

```
select *
from symmetric-ace-465202-c6.modulabs_project.user_rfm;
```

행	CustomerID	purchase_cnt	item_cnt	recency	user_total	user_average
1	12713	1	505	0	794.55	794.5
2	13436	1	76	1	196.8900000000...	196.9
3	13298	1	96	1	360.0	360.0
4	15520	1	314	1	343.4999999999...	343.5
5	14569	1	79	1	227.3899999999...	227.4
6	14204	1	72	2	150.6099999999...	150.6
7	15195	1	1404	2	3861.0	3861.0
8	15471	1	256	2	454.4800000000...	454.5

11-8. 추가 Feature 추출

1. 구매하는 제품의 다양성

우선 1) 고객 별로 구매한 상품들의 고유한 수를 계산합니다. 높은 숫자가 나오는 것은 해당 고객이 다양한 제품들을 구매한다는 의미이며, 낮은 값이 나오는 경우 소수의 제품들만 구매한다는 것을 의미합니다.

이후 2) `user_rfm` 테이블과 결과를 합치고, 이를 3) `user_data` 라는 이름의 테이블에 저장하겠습니다.

```
CREATE OR REPLACE TABLE project_name.modulabs_project.user_data AS
WITH unique_products AS (
  SELECT
    CustomerID,
    COUNT(DISTINCT StockCode) AS unique_products
  FROM project_name.modulabs_project.data
  GROUP BY CustomerID
)
SELECT ur.*, up.* EXCEPT (CustomerID)
FROM project_name.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	17923	1	50	282	208.0	208.0	1
2	15753	1	144	304	79.0	79.0	1
3	18113	1	72	368	76.0	76.0	1
4	13307	1	4	120	15.0	15.0	1
5	17925	1	72	372	244.0	244.0	1
6	13017	1	48	7	204.0	204.0	1
7	16323	1	50	196	208.0	208.0	1
8	14679	1	-1	371	-3.0	-3.0	1
9	16738	1	3	297	4.0	4.0	1
10	12814	1	48	101	86.0	86.0	1
11	16737	1	288	53	418.0	418.0	1

2. 평균 구매 주기

이 단계에서는 고객들의 쇼핑 패턴을 이해하는 것을 목표로 합니다. 그 중에서도 고객 별 재방문 주기를 살펴볼 것입니다. 고객들의 구매와 구매 사이의 기간이 평균적으로 몇 일인지를 보여주는 평균 일수를 계산하면, 고객이 다음 구매를 언제할지 예측하는 데에도 큰 도움이 됩니다. 평균 구매 소요 일수를 계산하고, 그 결과를 `user_data` 에 통합

```
CREATE OR REPLACE TABLE project_name.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(int
  FROM (
```

```
    -- (1) 구매와 구매 사이에 소요된 일수
    SELECT
      CustomerID,
      DATE_DIFF(InvoiceDate, LAG(InvoiceDate) OVER (PARTITION BY CustomerID
    FROM
      project_name.modulabs_project.data
    WHERE CustomerID IS NOT NULL
  )
  GROUP BY CustomerID
)

SELECT u.*, pi.* EXCEPT (CustomerID)
FROM project_name.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	average_interval
1	14432	6	2013	9	2248.0	375.0	256	0.2
2	12428	11	3477	25	6366.0	579.0	256	0.87
3	13268	14	3525	17	3106.0	222.0	256	0.56
4	16738	1	3	297	4.0	4.0	1	0.0
5	17925	1	72	372	244.0	244.0	1	0.0
6	17715	1	384	200	326.0	326.0	1	0.0
7	13120	1	12	238	31.0	31.0	1	0.0
8	15195	1	1404	2	3861.0	3861.0	1	0.0
9	13135	1	4300	196	3096.0	3096.0	1	0.0
10	13302	1	5	155	64.0	64.0	1	0.0
11	14705	1	100	198	179.0	179.0	1	0.0

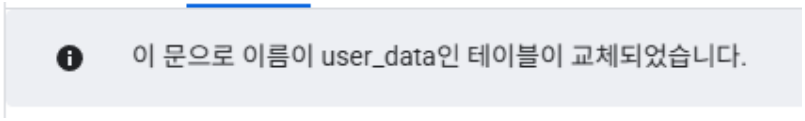
3. 구매 취소 경향성

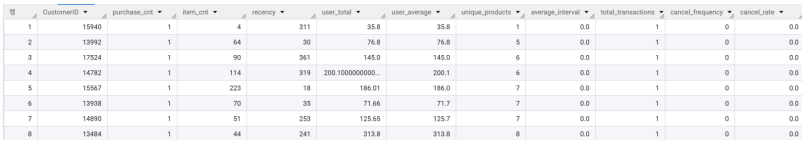
2. 취소 비율(cancel_rate)

취소 빈도와 취소 비율을 계산하고 그 결과를 `user_data` 에 통합해 줍시다. 취소 비율은 소수점 두 번째 자리까지 구해 주세요.

```
CREATE OR REPLACE TABLE symmetric-ace-465202-c6.modulabs_project.user_

WITH TransactionInfo AS (
  SELECT
```

```
    CustomerID,
    COUNT(DISTINCT InvoiceNo) AS total_transactions,
    COUNT(DISTINCT IF(STARTS_WITH(InvoiceNo, 'C'), InvoiceNo, NULL)) AS ca
  FROM symmetric-ace-465202-c6.modulabs_project.data
  WHERE CustomerID IS NOT NULL
  GROUP BY CustomerID
)

SELECT
  u.*,
  t.* EXCEPT(CustomerID),
  ROUND(t.cancel_frequency / t.total_transactions, 2) AS cancel_rate
FROM
  symmetric-ace-465202-c6.modulabs_project.user_data AS u
LEFT JOIN
  TransactionInfo AS t
ON
  u.CustomerID = t.CustomerID;
```





행	CustomerID	purchase_cnt	item_cnt	recency	user_total	user_average	unique_products	average_interval	total_transactions	cancel_frequency	cancel_rate
1	15940	1	4	311	35.8	35.8	1	0.0	1	0	0.0
2	13992	1	64	30	76.8	76.8	5	0.0	1	0	0.0
3	17524	1	90	361	145.0	145.0	6	0.0	1	0	0.0
4	14782	1	114	319	200.1000000000...	200.1	6	0.0	1	0	0.0
5	15567	1	223	18	186.01	186.0	7	0.0	1	0	0.0
6	13938	1	70	35	71.66	71.7	7	0.0	1	0	0.0
7	14890	1	51	253	125.65	125.7	7	0.0	1	0	0.0
8	13484	1	44	241	313.8	313.8	8	0.0	1	0	0.0