

# 260102/sql/Main\_Quest\_1

## 11-2. 데이터 불러오기

### ▼ 데이터 살펴보기

- 테이블에 있는 10개의 행만 출력하기

```
SELECT *
FROM modulabs_project.data
LIMIT 10;
```

쿼리 결과								
작업 정보		결과	시작화	JSON	실행 세부정보	실행 그래프		
행	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country
1	S36365	85123A	WHITE HANGING HEART T-LIG...	6	2010-12-01 08:26:00 UTC	2.55	17850	United Kingdom
2	S36365	71053	WHITE METAL LANTERN	6	2010-12-01 08:26:00 UTC	3.39	17850	United Kingdom
3	S36365	844068	CREAM CUPID HEARTS COAT H...	8	2010-12-01 08:26:00 UTC	2.75	17850	United Kingdom
4	S36365	840296	KNITTED UNION FLAG HOT WA...	6	2010-12-01 08:26:00 UTC	3.39	17850	United Kingdom
5	S36365	840296	RED WOOLLY HOTTIE WHITE H...	6	2010-12-01 08:26:00 UTC	3.39	17850	United Kingdom
6	S36365	22752	SET 7 BABUSHKA NESTING BO...	2	2010-12-01 08:26:00 UTC	7.65	17850	United Kingdom
7	S36365	21730	GLASS STAR FROSTED TLIGHT...	6	2010-12-01 08:26:00 UTC	4.25	17850	United Kingdom
8	S36366	22633	HAND WARMER UNION JACK	6	2010-12-01 08:26:00 UTC	1.85	17850	United Kingdom
9	S36366	22632	HAND WARMER RED POLKA DOT	6	2010-12-01 08:26:00 UTC	1.85	17850	United Kingdom
10	S36367	84879	ASSORTED COLOUR BIRD ORN...	32	2010-12-01 08:34:00 UTC	1.69	13047	United Kingdom

- 전체 데이터는 몇 행으로 구성되어 있는지 확인하기

```
SELECT *
FROM modulabs_project.data;
```

쿼리 결과								
작업 정보		결과	시작화	JSON	실행 세부정보	실행 그래프		
행	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country
1	S36365	85123A	WHITE HANGING HEART T-LIG...	6	2010-12-01 08:26:00 UTC	2.55	17850	United Kingdom
2	S36365	71053	WHITE METAL LANTERN	6	2010-12-01 08:26:00 UTC	3.39	17850	United Kingdom
3	S36365	844068	CREAM CUPID HEARTS COAT H...	8	2010-12-01 08:26:00 UTC	2.75	17850	United Kingdom
4	S36365	840296	KNITTED UNION FLAG HOT WA...	6	2010-12-01 08:26:00 UTC	3.39	17850	United Kingdom
5	S36365	840296	RED WOOLLY HOTTIE WHITE H...	6	2010-12-01 08:26:00 UTC	3.39	17850	United Kingdom
6	S36365	22752	SET 7 BABUSHKA NESTING BO...	2	2010-12-01 08:26:00 UTC	7.65	17850	United Kingdom
7	S36365	21730	GLASS STAR FROSTED TLIGHT...	6	2010-12-01 08:26:00 UTC	4.25	17850	United Kingdom
8	S36366	22633	HAND WARMER UNION JACK	6	2010-12-01 08:26:00 UTC	1.85	17850	United Kingdom
9	S36366	22632	HAND WARMER RED POLKA DOT	6	2010-12-01 08:26:00 UTC	1.85	17850	United Kingdom
10	S36367	84879	ASSORTED COLOUR BIRD ORN...	32	2010-12-01 08:34:00 UTC	1.69	13047	United Kingdom
11	S36367	22745	POPPY'S PLAYHOUSE BEDROOM	6	2010-12-01 08:34:00 UTC	2.1	13047	United Kingdom
12	S36367	22748	POPPY'S PLAYHOUSE KITCHEN	6	2010-12-01 08:34:00 UTC	2.1	13047	United Kingdom
13	S36367	22749	FELTRAFT PRINCESS CHARLO...	8	2010-12-01 08:34:00 UTC	3.75	13047	United Kingdom
14	S36367	22310	IVORY KNITTED MUG COSY	6	2010-12-01 08:34:00 UTC	1.65	13047	United Kingdom

## 데이터 수 세기

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

```
SELECT
COUNT (InvoiceNo) AS count_InvoiceNo,
COUNT (StockCode) AS count_StockCode,
COUNT (Description) AS count_Description,
COUNT (Quantity) AS count_Quantity,
COUNT (InvoiceDate) AS count_InvoiceDate,
COUNT (UnitPrice) AS count_UnitPrice,
COUNT (CustomerID) AS count_CustomerID,
COUNT (Country) AS count_Country,
FROM modulabs_project.data;
```

쿼리 결과								
작업 정보		결과	시작화	JSON	실행 세부정보	실행 그래프		
행	count_InvoiceNo	count_StockCode	count_Description	count_Quantity	count_InvoiceDate	count_UnitPrice	count_CustomerID	count_Country
1	541909	541909	540455	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

```
-- 각 행별 결측치 유무 확인
SELECT
    COUNTIF(InvoiceNo IS NULL) AS missing_InvoiceNo,
    COUNTIF(StockCode IS NULL) AS missing_StockCode,
    COUNTIF(Description IS NULL) AS missing_Description,
    COUNTIF(Quantity IS NULL) AS missing_Quantity,
    COUNTIF(InvoiceDate IS NULL) AS missing_InvoiceDate,
    COUNTIF(UnitPrice IS NULL) AS missing_UnitPrice,
    COUNTIF(CustomerID IS NULL) AS missing_CustomerID,
    COUNTIF(Country IS NULL) AS missing_Country,
FROM modulabs_project.data;

-- 결측치 비율 계산 후 UNION ALL을 통해 결측치 합치기
SELECT 'Description',
    -- Description의 결측치 비율 백분율로 계산 (소수점 둘째 자리 반올림)
    -- SUM(CASE WHEN...)을 사용해 NULL이면 1을 더하고, 이를 전체 행 수로 나눈 뒤 100을 곱해 백분율 산출 (소수점 둘째 자리 반올림)
    ROUND(SUM(CASE WHEN Description IS NULL THEN 1 ELSE 0 END) / COUNT(*) * 100, 2) AS missing_percentage
FROM `modulabs_project.data`'

UNION ALL --위아래의 쿼리 결과 집합을 하나로 통합

SELECT 'CustomerID',
    -- CustomerID의 결측치 비율 백분율로 계산 (소수점 둘째 자리 반올림)
    ROUND(SUM(CASE WHEN CustomerID IS NULL THEN 1 ELSE 0 END) / COUNT(*) * 100, 2) AS missing_percentage
FROM `modulabs_project.data`;
```

쿼리 결과					
작업 정보	결과	시작일	JSON	실행 세부 정보	실행 그래프
[1]	결과	missing_percentages			
1	Description	0.27			
2	CustomerID	24.93			

### ▼ 결측치 처리 전략

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

```
-- 단순 추출 쿼리 (현상 파악용)
-- StockCode = '85123A'의 Description을 추출하는 쿼리문을 작성
SELECT Description
FROM modulabs_project.data
WHERE StockCode = '85123A';
ORDER BY record_count ASC; -- 개수가 적은 순서대로 정렬

-----
-- [추가수행] 데이터 카운트 기반 분석 쿼리 (데이터 정제 및 의사결정용)
-- 필요한 이유:
-- 1. 데이터 무결성 확인: 동일한 StockCode에 여러 Description이 섞여 있는지(데이터 오염) 한눈에 파악 가능
-- 2. 대표성 판단: 가장 많이 사용된 데이터를 대표값으로 채택 등
-- 3. 이상치 탐지: 개수가 현저히 적은(예: 1건) Description은 오타나 일시적 오류일 가능성 높음
SELECT
    Description,
    COUNT(*) AS record_count
```

```

FROM `modulabs_project.data`
WHERE StockCode = '85123A'
GROUP BY Description -- 개수를 세기 위해 그룹화 추가
ORDER BY record_count ASC; -- 개수가 적은 순서대로 정렬

```

행	Description
1	wrongly marked carton 22804
2	WHITE HANGING HEART T-LIGHT HOLDER
3	WHITE HANGING HEART T-LIGHT HOLDER
4	WHITE HANGING HEART T-LIGHT HOLDER
5	WHITE HANGING HEART T-LIGHT HOLDER
6	WHITE HANGING HEART T-LIGHT HOLDER
7	WHITE HANGING HEART T-LIGHT HOLDER
8	WHITE HANGING HEART T-LIGHT HOLDER
9	WHITE HANGING HEART T-LIGHT HOLDER
10	WHITE HANGING HEART T-LIGHT HOLDER

  

작업 정보	Description	record_count
1	wrongly marked carton 22804	1
2	wrongly marked carton 22804	1
3	CREAM HANGING HEART T-LIGHT HOLDER	9
4	WHITE HANGING HEART T-LIGHT HOLDER	2302

## ▼ 결측치 처리

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

```

-- 테이블에서 결측치가 포함된 행 삭제
DELETE FROM `modulabs_project.data` -- 삭제 작업 수행할 테이블 지정
WHERE
CustomerID IS NULL -- 조건 1: CustomerID 컬럼에 값이 없는 경우
OR
-- 조건 1, 2 중 하나만 만족해도 삭제
Description IS NULL; -- 조건 2: Description 컬럼에 값이 없는 경우

```

## 11-5. 데이터 전처리(2): 중복값 처리

### ▼ 중복값 확인

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

```

-- 중복된 행의 갯수 구해보기
SELECT
-- 서브쿼리에서 넘어온 '중복 Group'들의 총 개수 계산
COUNT(*) AS total_duplicate_groups
FROM (
-- 데이터 그룹화 테이블의 8개 컬럼을 기준으로 삼아, 값이 모두 똑같은 행들을
-- 하나의 Group에 모으고 그 개수(cnt) 계산
SELECT
COUNT(*) AS cnt

```

```

FROM `modulabs_project.data`
GROUP BY
    InvoiceNo, StockCode, Description, Quantity,
    InvoiceDate, UnitPrice, CustomerID, Country

-- Group에 담긴 행의 개수가 2개 이상인 것들만 산출
HAVING COUNT(*) > 1
) AS sub;

```

쿼리 결과			
작업 정보	결과		
	<table border="1"> <thead> <tr> <th>total_duplicate_g...</th> <th>4837</th> </tr> </thead> </table>	total_duplicate_g...	4837
total_duplicate_g...	4837		
페이지당 결과 수: 50 ▾ 1 ~ 1 (전체 1행)  < < > >			
작업 기록			

## ▼ 중복값 처리

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

```

CREATE OR REPLACE TABLE `modulabs_project.data` AS
SELECT DISTINCT * -- 모든 컬럼(*)을 비교하여 완벽하게 일치하는 중복 행을 제거하고 1건씩만 추출
FROM `modulabs_project.data`;

```

```

-- 업데이트 결과 확인
SELECT
    COUNT(*)
FROM modulabs_project.data;

```

쿼리 결과	
작업 정보	결과
	<p>❶ 이 문으로 이름이 data인 테이블이 교체되었습니다.</p>
테이블로 이동	

쿼리 결과					
작업 정보	결과				
	<table border="1"> <thead> <tr> <th>행</th> <th>f0_</th> </tr> </thead> <tbody> <tr> <td>1</td> <td>401604</td> </tr> </tbody> </table>	행	f0_	1	401604
행	f0_				
1	401604				
페이지당 결과 수: 50 ▾ 1 ~ 1 (전체 1행)  < < > >					
작업 기록					
표시					

## 11-6. 데이터 전처리(3): 오류값 처리

### ▼ InvoiceNo 살펴보기

- 고유(unique)한 InvoiceNo의 개수를 출력하기

```
SELECT COUNT(DISTINCT InvoiceNo) -- InvoiceNo 컬럼에서 중복을 제거한 후 개수 카운트  
FROM modulabs_project.data;
```

쿼리 결과	
작업 정보	결과
행	f0_
1	22190

- 고유한 InvoiceNo를 앞에서부터 100개를 출력하기

```
SELECT DISTINCT InvoiceNo  
FROM modulabs_project.data  
LIMIT 100;
```

작업 정보	결과	시작화	JSON	실행 세부정보	실행 그레프
행	InvoiceNo				
4	542237				
5	549222				
6	556201				
7	562032				
8	573511				
9	581180				
10	539318				
11	541998				
12	548955				
13	568172				
14	577609				
15	543037				
16	544164				

- InvoiceNo가 'C'로 시작하는 행을 필터링 할 수 있는 쿼리문을 작성하기 (100행까지만 출력)

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

작업 정보	결과	시작화	JSON	실행 세부정보	실행 그레프
행	InvoiceNo	StockCode	Description	Quantity	InvoiceDate
1	C541433	Z3166	MEDIUM CERAMIC TOP STORA...	-74215	2011-01-18 10:17:00 UTC
2	C545329	M	Manual	-1	2011-03-01 15:47:00 UTC
3	C545329	M	Manual	-1	2011-03-01 15:47:00 UTC
4	C545330	M	Manual	-1	2011-03-01 15:49:00 UTC
5	C547388	23413	METAL SIGN TAKE IT OR LEAVE...	-6	2011-03-22 16:07:00 UTC
6	C547388	84050	PINK HEART SHAPE EGG PRYN...	-12	2011-03-22 16:07:00 UTC
7	C547388	23645	CERAMIC HEART FAIRY CAKE ...	-12	2011-03-22 16:07:00 UTC
8	C547388	22784	LANTERN CREAM GAZEBO	-3	2011-03-22 16:07:00 UTC

- 구매 건 상태가 Canceled인 데이터의 비율(%) - 소수점 첫번째 자리까지

```
SELECT  
ROUND(
```

```

SUM(CASE WHEN InvoiceNo LIKE 'C%' THEN 1 ELSE 0 END) -- 취소 건수
/ COUNT(*) -- 전체 건수
* 100, -- 백분율 단위로 변환
1 -- 소수점 첫째 자리까지 반올림
) AS Canceled_rate
FROM `modulabs_project.data`;

```

쿼리 결과		작업 정보	결과	시각화	JSON	실행 세부정보	실행 그레프	결과 저장	다음에서 열기	▼
행	Canceled_rate		1		2.2					

페이지당 결과 수: 50 ▾ 1 ~ 1 (전체 1행) | < < > > | 표시

## ▼ StockCode 살펴보기

- 고유한 StockCode의 개수를 출력하기

```

SELECT COUNT(DISTINCT StockCode) -- StockCode 컬럼에서 중복을 제거한 후 개수 카운트
FROM modulabs_project.data;

```

쿼리 결과		작업 정보	결과	시각화	JSON	실행 세부정보	실행 그레프	결과 저장	다음에서 열기	▼
행	f0_		1		3684					

페이지당 결과 수: 50 ▾ 1 ~ 1 (전체 1행) | < < > > | 표시

- 어떤 제품이 가장 많이 판매되었는지 보기 위하여 StockCode 별 등장 빈도를 출력하기

- 상위 10개의 제품들을 출력하기

```

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

```

쿼리 결과		작업 정보	결과	시각화	JSON	실행 세부정보	실행 그레프	결과 저장	다음에서 열기	▼
행	StockCode			sell_cnt						
1	85123A			2065						
2	22423			1894						
3	850998			1659						
4	47566			1409						
5	84879			1405						
6	20725			1346						
7	22720			1224						
8	POST			1196						
9	22197			1110						
10	23203			1108						

페이지당 결과 수: 50 ▾ 1 ~ 10 (전체 10행) | < < > > | 표시

- StockCode의 컬럼에 있던 값 중에서 숫자를 제외한 문자만 남기고 문자가 몇 자리 수 인지 세고

- 숫자가 0~1개인 값들에는 어떤 코드들이 들어가 있는지 출력하기

```

SELECT DISTINCT StockCode, number_count
FROM (
    -- 각 StockCode 내에 포함된 '숫자의 개수'를 계산
    SELECT StockCode,
        -- 전체 길이에서 숫자를 제거한 길이를 빼서 '숫자가 몇 개였는지' 계산
        LENGTH(StockCode) - LENGTH(REGEXP_REPLACE(StockCode, r'[0-9]', '')) AS number_count
    FROM modulabs_project.data
)
WHERE number_count >= 0 AND number_count <= 1;

```

쿼리 결과		결과 저장	다음에서 열기	▼	
작업 정보	결과	시각화	JSON	실행 세부정보	실행 그래프
<b>행</b> // StockCode // number_count //					
1	POST		0		
2	M		0		
3	C2		1		
4	D		0		
5	BANK CHARGES		0		
6	PADS		0		
7	DOT		0		
8	CRUK		0		

페이지당 결과 수: 50 ▾ 1 ~ 8 (전체 8행) |< < > >|

- StockCode의 컬럼에 있던 값 중에서 숫자를 제외한 문자만 남기고 문자가 몇 자리 수인지 세고
- 숫자가 0~1개인 값을 가지고 있는 데이터 수는 전체 데이터 수 대비 몇 퍼센트인지 구하기 (소수점 두 번째 자리까지)

```

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

```

쿼리 결과		결과 저장	다음에서 열기	▼	
작업 정보	결과	시각화	JSON	실행 세부정보	실행 그래프
<b>행</b> // StockCode_numb... //					
1	0.48				

페이지당 결과 수: 50 ▾ 1 ~ 1 (전체 1행) |< < > >|

- 제품과 관련되지 않은 거래 기록을 제거하기

```

DELETE FROM 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 modulabs_project.data)
WHERE number_count <= 1);
```

The screenshot shows a SQL query results page from a database interface. The query is:

```
FROM modulabs_project.data)
WHERE number_count <= 1);
```

The results table has one row with the following data:

행	결과	description	description_cnt
1	이 문으로 data의 행 1,915개가 삭제되었습니다.		

## ▼ Description 살펴보기

- 고유한 Description 별 출현 빈도를 계산하고 상위 30개를 출력하기

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

The screenshot shows a SQL query results page from a database interface. The query is:

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

The results table contains 30 rows of data, showing the most frequent product descriptions and their counts. The top few rows are:

행	Description	description_cnt
1	WHITE HANGING HEART T-LIGHT HOLDER	2058
2	REGENCY CAKESTAND 3 TIER	1894
3	JUMBO BAG RED RETROSPOT	1659
4	PARTY BUNTING	1409
5	ASSORTED COLOUR BIRD ORNAMENT	1405
6	LUNCH BAG RED RETROSPOT	1345
7	SET OF 3 CAKE TINS PANTRY DESIGN	1224
8	LUNCH BAG BLACK SKULL.	1099

- 서비스 관련 정보를 포함하는 행들을 제거하기

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

The screenshot shows a SQL query results page from a database interface. The query is:

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

The results table has one row with the following data:

행	결과	description	description_cnt
1	이 문으로 data의 행 83개가 삭제되었습니다.		

- 대소문자를 혼합하고 있는 데이터를 대문자로 표준화 하기

```

CREATE OR REPLACE TABLE modulabs_project.data AS
SELECT
    * EXCEPT (Description),-- 기존 Description은 제외하고
    UPPER(Description) AS Description -- 대문자로 변환한 Description을 새로 추가
FROM modulabs_project.data;

```

작업 정보    결과    실행 세부정보    실행 그레프

이 문으로 이릅니다 data인 테이블이 교체되었습니다.

테이블로 이동

## ▼ 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 modulabs_project.data;

```

행	min_price	max_price	avg_price
1	0.0	649.5	2.904956757406...

페이지당 결과 수: 50 1 - 1 (전체 1행) |< < > >|

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

```

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

```

행	cnt_quantity	min_quantity	max_quantity	avg_quantity
1	33	1	12540	420.5151515151515...

페이지당 결과 수: 50 1 - 1 (전체 1행) |< < > >|

- UnitPrice = 0 를 제거하고 일관된 데이터셋을 유지하기

```

CREATE OR REPLACE TABLE modulabs_project.data AS
SELECT *

```

```
FROM modulabs_project.data
WHERE UnitPrice > 0;
```

쿼리 결과

작업 정보 결과 실행 세부정보 실행 그레프

❶ 이 문으로 이름이 data인 테이블이 교체되었습니다.

테이블로 이동 표시

## 11-7. RFM 스코어

### ▼ Recency

- InvoiceDate 컬럼을 연월일 자료형으로 변경하기

```
SELECT CAST(InvoiceDate AS DATE) AS InvoiceDay, *
FROM modulabs_project.data;
```

번호	InvoiceDay	CustomerID	StockCode	Quantity	InvoiceDate	UnitPrice	CustomerID	Country	Description
1	2011-01-18	541431	23166	74215	2011-01-18 10:00:00.0	1.04	12346	United Kingdom	MEDIUM CERAMIC TOP STO...
2	2011-01-18	541433	23166	74215	2011-01-18 10:00:00.0	1.04	12346	United Kingdom	MEDIUM CERAMIC TOP STO...
3	2011-12-07	537626	22494	12	2010-12-07 14:00:00.0	1.25	12347	Iceland	EMERGENCY FIRST AID TIN
4	2011-12-07	537626	20782	6	2010-12-07 14:00:00.0	5.49	12347	Iceland	CAMOUFLAGE EAR MUFF HEA...
5	2011-12-07	537626	22865	12	2010-12-07 14:00:00.0	1.25	12347	Iceland	BLUE DRAWER KNOB ACRYLIC...
6	2010-12-07	537626	22774	12	2010-12-07 14:00:00.0	1.25	12347	Iceland	RED DRAWER KNOB ACRYLIC...

- 가장 최근 구매 일자를 MAX() 함수로 찾아보기

```
SELECT
    MAX(CAST(InvoiceDate AS DATE)) OVER() AS most_recent_date,
    CAST(InvoiceDate AS DATE) AS InvoiceDay,
    *
FROM modulabs_project.data;
```

번호	most_recent	CustomerID	StockCode	Quantity	InvoiceDate	UnitPrice	CustomerID	Country	Description
1	2011-12-09	2011-01...	541431	23166	74215	2011-01-18 10:00:00.0	1.04	12346	United Kingd...
2	2011-12-09	2011-01...	541433	23166	74215	2011-01-18 10:00:00.0	1.04	12346	United Kingd...
3	2011-12-09	2010-12...	537626	22494	12	2010-12-07 14:00:00.0	1.25	12347	Iceland
4	2011-12-09	2010-12...	537626	20782	6	2010-12-07 14:00:00.0	5.49	12347	Iceland
5	2011-12-09	2010-12...	537626	22865	12	2010-12-07 14:00:00.0	1.25	12347	Iceland
6	2011-12-09	2010-12...	537626	22774	12	2010-12-07 14:00:00.0	1.25	12347	Iceland
7	2011-12-09	2010-12...	537626	22773	12	2010-12-07 14:57:00.0	1.25	12347	Iceland
8	2011-12-09	2010-12...	537626	8499...	6	2010-12-07 14:57:00.0	9.75	12347	Iceland

- 유저 별로 가장 큰 InvoiceDay를 찾아서 가장 최근 구매일로 저장하기

```
SELECT
    CustomerID,
    MAX(CAST(InvoiceDate AS DATE)) AS InvoiceDay
FROM modulabs_project.data
GROUP BY CustomerID;
```

쿼리 결과

작업 정보 **결과** 시각화 JSON 실행 세부정보 실행 그래프

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

페이지당 결과 수: 50 1 ~ 50 (전체 4362행) |< < > >| 표시

- 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 modulabs_project.data
    GROUP BY CustomerID
);
```

쿼리 결과

작업 정보 **결과** 시각화 JSON 실행 세부정보 실행 그래프

행	CustomerID	recency
1	12561	302
2	12616	85
3	12853	107
4	13030	11
5	13239	261
6	13336	77
7	13363	17
8	13489	105
9	13508	243
10	13672	301

페이지당 결과 수: 50 1 ~ 50 (전체 4362행) |< < > >| 표시

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

```
CREATE OR REPLACE TABLE modulabs_project.user_r AS
SELECT
    CustomerID, -- 전체 데이터 마감일에서 각 고객의 마지막 구매일을 뺀 일수 계산
    EXTRACT(DAY FROM MAX(InvoiceDay) OVER () - InvoiceDay) AS recency
FROM (
    -- 고객별 가장 최근 구매 날짜(InvoiceDay)추출 서브쿼리
    SELECT
        CustomerID, MAX(DATE(InvoiceDate)) AS InvoiceDay
    FROM modulabs_project.data
    GROUP BY CustomerID
);
```

project-573d05ee-0149-4ee2-b90 / Da

user\_r      쿼리      다음에서 열기

스키마 세부정보 미리보기 테이

행	CustomerID	recency
1	14446	0
2	12423	0
3	13069	0
4	15311	0
5	15910	0
6	14397	0
7	13777	0
8	17428	0
9	16954	0
10	14441	0

## ▼ Frequency

- 고객마다 고유한 InvoiceNo의 수를 세어보기

```
SELECT
    CustomerID,
    COUNT(DISTINCT InvoiceNo) AS purchase_cnt -- 동일한 주문 번호(InvoiceNo)를 중복 없이 카운트하여 실제 주문 횟수 계산
FROM modulabs_project.data
GROUP BY CustomerID;
```

쿼리 결과      결과 저장      다음에서 열기

작업 정보      결과      시작화      JSON      실행 세부정보      실행 그래프

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

페이지당 결과 수: 50 1 - 50 (전체 4362행) |< < > >|

- 각 고객 별로 구매한 아이템의 총 수량 더하기

```
SELECT
    CustomerID,
    SUM (Quantity) AS item_cnt
FROM modulabs_project.data
GROUP BY CustomerID;
```

쿼리 결과

작업 정보 결과 시각화 JSON 실행 세부정보 실행 그래프

행	CustomerID	item_cnt
1	12346	0
2	12347	2458
3	12348	2332
4	12349	630
5	12350	196
6	12352	463

페이지당 결과 수: 50 1 – 50 (전체 4362행) |< < > >|

- 전체 거래 건수 계산과 구매한 아이템의 총 수량 계산의 결과를 합쳐서 user\_rf라는 이름의 테이블에 저장하기

```
-- 전체 거래 건수 계산과 구매한 아이템의 총 수량 계산의 결과를 합쳐서 user_rf라는 이름의 테이블에 저장하기
```

```
CREATE OR REPLACE TABLE modulabs_project.user_rf AS
-- (1) 전체 거래 건수 계산
WITH purchase_cnt AS (
    SELECT CustomerID, COUNT(DISTINCT InvoiceNo) AS purchase_cnt
    FROM modulabs_project.data
    GROUP BY CustomerID),
-- (2) 구매한 아이템 총 수량 계산
item_cnt AS (
    SELECT CustomerID, SUM(Quantity) AS item_cnt
    FROM modulabs_project.data
    GROUP BY CustomerID)
-- 기존의 user_r에 (1)과 (2)를 통합
SELECT pc.CustomerID, pc.purchase_cnt, ic.item_cnt, ur.recenty
FROM purchase_cnt AS pc
JOIN item_cnt AS ic
    ON pc.CustomerID = ic.CustomerID
JOIN modulabs_project.user_r AS ur
    ON pc.CustomerID = ur.CustomerID;
```

project-573d05ee-0149-4ee2-b90 / Datasets / modulabs\_project / Tables / user\_rf

★ user\_rf 쿼리 다음에서 열기 공유 프리뷰 통계 계보 데이터 프리뷰

행	CustomerID	purchase_cnt	item_cnt	recenty
1	12713	1	505	0
2	15520	1	314	1
3	13298	1	96	1
4	13436	1	76	1
5	14569	1	79	1
6	15195	1	1404	2
7	14204	1	72	2
8	15471	1	256	2
9	14578	1	240	3
10	16569	1	93	3
11	12478	1	233	3
12	17914	1	457	3

## ▼ Monetary

- 고객별 총 지출액 계산 (소수점 첫째 자리에서 반올림)

```

SELECT CustomerID,
       ROUND(SUM(Quantity * UnitPrice), 0) AS user_total
  FROM modulabs_project.data
 GROUP BY CustomerID;

```

쿼리 결과		결과 저장 ▾		다음에서 열기 ▾	
작업 정보		시각화	JSON	실행 세부정보	실행 그래프
행	CustomerID	user_total			
1	12346	0.0			
2	12347	4310.0			
3	12348	1437.0			
4	12349	1458.0			
5	12350	294.0			
6	12352	1265.0			
7	12353	89.0			

페이지당 결과 수: 50 ▾ 1 - 50 (전체 4362행) |< < > >|

▶▶▶

#### • 고객별 평균 거래 금액 계산

- 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 modulabs_project.user_rfm AS
SELECT
    rf.CustomerID AS CustomerID,
    rf.purchase_cnt,
    rf.item_cnt,
    rf.recency,
    ut.user_total,
    ROUND(ut.user_total / rf.purchase_cnt, 0) AS user_average -- 총 지출액/주문 횟수
  FROM modulabs_project.user_rf rf
 LEFT JOIN (
    -- 고객 별 총 지출액
    SELECT
        CustomerID,
        ROUND(SUM(Quantity * UnitPrice), 0) AS user_total
      FROM modulabs_project.data
     GROUP BY CustomerID
  ) ut
 ON rf.CustomerID = ut.CustomerID;

```

project-573d05ee-0149-4ee2-b90 / Datasets / modulabs_project / Tables / user_rfm						
스키마		세부정보		미리보기		
행	CustomerID	purchase_cnt	item_cnt	recency	user_total	user_average
1	12713	1	505	0	795.0	795.0
2	14569	1	79	1	227.0	227.0
3	13298	1	96	1	360.0	360.0
4	15520	1	314	1	344.0	344.0
5	13436	1	76	1	197.0	197.0
6	15195	1	1404	2	3861.0	3861.0
7	14204	1	72	2	151.0	151.0
8	15471	1	256	2	454.0	454.0
9	16569	1	93	3	124.0	124.0
10	15992	1	17	3	42.0	42.0
11	12650	1	250	3	242.0	242.0
12	15318	1	642	3	313.0	313.0

#### ▼ RFM 통합 테이블 출력하기

- 최종 user\_rfm 테이블을 출력하기

```
SELECT
    CustomerID,
    purchase_cnt, -- Frequency (주문 횟수)
    recency, -- Recency (마지막 구매 후 경과일)
    user_total -- Monetary (총 지출액)
FROM modulabs_project.user_rfm
ORDER BY CustomerID;
```

쿼리 결과					
작업 정보		결과	시각화	JSON	실행 세부정보
행	CustomerID	purchase_cnt	recency	user_total	
1	12346	2	325	0.0	
2	12347	7	2	4310.0	
3	12348	4	75	1437.0	
4	12349	1	18	1458.0	
5	12350	1	310	294.0	
6	12352	8	36	1265.0	
7	12353	1	204	89.0	
8	12354	1	232	1079.0	
9	12355	1	214	459.0	

페이지당 결과 수: 50 | 1 - 50 (전체 4362행) | < < > >|

## 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\_data라는 이름의 테이블에 저장하기

```
CREATE OR REPLACE TABLE modulabs_project.user_data AS
WITH unique_products AS (
    SELECT
        CustomerID,
        COUNT(DISTINCT StockCode) AS unique_products
    FROM modulabs_project.data
    GROUP BY CustomerID
)
SELECT ur.*, up.* EXCEPT (CustomerID)
FROM modulabs_project.user_rfm AS ur
JOIN unique_products AS up
ON ur.CustomerID = up.CustomerID;
```

project-573d05ee-0149-4ea2-b90 / Datasets / modulabs_project / Tables / user_data									
스키마	세부정보	미리보기		데이터 탐색기		표현식		데이터 흐름표	데이터 풀집
		CustomerID	purchase_cnt	item_cnt	recency	user_total	user_average		
		1	14351	1	12	164	51.0	51.0	1
		2	16257	1	1	176	22.0	22.0	1
		3	16138	1	-1	368	-8.0	-8.0	1
		4	12791	1	96	373	178.0	178.0	1
		5	15118	1	1440	134	245.0	245.0	1
		6	17443	1	504	219	534.0	534.0	1
		7	17763	1	12	263	15.0	15.0	1
		8	18174	1	50	7	104.0	104.0	1
		9	16344	1	18	158	101.0	101.0	1
		10	18133	1	1350	212	931.0	931.0	1
		11	16738	1	3	297	4.0	4.0	1
		12	13841	1	100	252	85.0	85.0	1

### ▼ 2. 평균 구매 주기

- 고객들의 쇼핑 패턴을 이해하는 것을 목표 (고객 별 재방문 주기 살펴보기)

- 평균 구매 소요 일수를 계산하고, 그 결과를 `user_data`에 통합

```

CREATE OR REPLACE TABLE 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_interval
    FROM (
        -- (1) 구매와 구매 사이에 소요된 일수
        SELECT
            CustomerID,
            DATE_DIFF(InvoiceDate, LAG(InvoiceDate) OVER (PARTITION BY CustomerID ORDER BY InvoiceDate), DAY) AS interval_
        FROM
            modulabs_project.data
        WHERE CustomerID IS NOT NULL
    )
    GROUP BY CustomerID
)

SELECT u.*, pi.* EXCEPT (CustomerID)
FROM 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	18174	1	50	7	104.0	104.0	1	0.0
2	14119	1	-2	354	-20.0	-20.0	1	0.0
3	18133	1	1350	212	931.0	931.0	1	0.0
4	15510	1	2	330	250.0	250.0	1	0.0
5	13185	1	12	267	71.0	71.0	1	0.0
6	16138	1	-1	368	-8.0	-8.0	1	0.0
7	17763	1	12	263	15.0	15.0	1	0.0
8	12791	1	96	373	178.0	178.0	1	0.0
9	12943	1	-1	301	-4.0	-4.0	1	0.0
10	15316	1	100	326	165.0	165.0	1	0.0
11	14090	1	72	324	76.0	76.0	1	0.0
12	15488	1	72	92	76.0	76.0	1	0.0
13	15668	1	72	217	76.0	76.0	1	0.0
14	14705	1	100	198	179.0	179.0	1	0.0
15	17715	1	384	200	326.0	326.0	1	0.0

### ▼ 3. 구매 취소 경향성

- 고객의 취소 패턴 파악하기
  - 취소 빈도(cancel\_frequency) : 고객 별로 취소한 거래의 총 횟수
  - 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 modulabs_project.user_data AS

WITH TransactionInfo AS (
    SELECT
        CustomerID,
        COUNT(DISTINCT InvoiceNo) AS total_transactions, -- 전체 거래 건수
        COUNT(DISTINCT CASE WHEN InvoiceNo LIKE 'C%' THEN InvoiceNo END) AS cancel_frequency -- 취소된 거래 건수
    FROM modulabs_project.data
    GROUP BY CustomerID
)

SELECT u.* t.* EXCEPT(CustomerID),
    ROUND(SAFE_DIVIDE(t.cancel_frequency, t.total_transactions), 2) AS cancel_rate -- 취소 비율 (취소 건수 / 전체 건수)

```

```
FROM modulabs_project.user_data AS u
LEFT JOIN TransactionInfo AS t
ON u.CustomerID = t.CustomerID;
```

스키마	세부정보	미리보기	데이터 탐색기	프리뷰	등계	개보	데이터 프로필	데이터 품질
영	CustomerID	purchase_cnt	item_cnt	recency	user_total	user_average	unique_products	average_interval
1	15488	1	72	92	76.0	76.0	1	0.0
2	14351	1	12	164	51.0	51.0	1	0.0
3	17986	1	10	56	21.0	21.0	1	0.0
4	16162	1	4	252	37.0	37.0	2	0.0
5	14371	1	34	295	101.0	101.0	4	0.0
6	16959	1	49	87	117.0	117.0	7	0.0
7	16050	1	278	173	138.0	138.0	10	0.0
8	14889	1	90	336	136.0	136.0	10	0.0
9	17496	1	79	358	271.0	271.0	10	0.0
10	13976	1	154	206	358.0	358.0	10	0.0
11	18074	1	190	373	490.0	490.0	13	0.0
12	12497	1	1231	81	953.0	953.0	13	0.0

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

```
SELECT
    CustomerID,
    recency, purchase_cnt, user_total,
    user_average, item_cnt,
    average_interval,
    total_transactions, cancel_frequency, cancel_rate
FROM modulabs_project.user_data
ORDER BY user_total DESC; -- 가장 기여도가 높은 고객부터 확인
```

작업 정보	결과	시작화면	JSON	실행 세부정보	실행 그래프
1	14646	1	73	25771.0	3819.0
2	18102	0	60	259657.0	4328.0
3	17450	8	49	189575.0	3869.0
4	14911	1	242	138748.0	532.0
5	12415	24	24	123638.0	5152.0
6	14156	9	64	113665.0	1776.0
7	17511	2	45	88138.0	1959.0
8	16664	4	30	65920.0	2197.0
9	13694	3	57	62962.0	1105.0

## 회고

"에러 나면 → 괄호의 위치, 쉼표의 유무, 중복된 컬럼명 부터 확인하자!"

### Keep (배울점 or 유지해야 할 좋은 습관)

- 단계별 테이블 생성 및 저장: `user_r`, `user_rf`, `user_rfm` 순으로 단계별 결과를 테이블로 저장, 복잡한 로직 단순화
- 에러 메시지 기반의 문제 해결: `duplicate name` 이나 `aggregate function` 에러 발생 시 원인을 분석하고 `EXCEPT`, `REPLACE` 등의 문법으로 해결함

### ⚠ Problem (초심자가 겪은 주요 실수 및 어려움)

- ★ 집계 함수와 일반 컬럼 혼용: `SELECT` 절에 `MAX()` 와 같은 집계 함수와 일반 컬럼을 동시에 쓸 때 발생하는 그룹화 오류(Grouping error) 유의해야 함
- 함수 괄호 및 인자 위치 실수 및 오탈자 등
  - `ROUND(SUM(A * B, 0))` 와 같이 중첩 함수 사용 시 괄호 위치를 잘못 지정하여 `No matching signature` 에러 발생
  - `SELECT` 목록 마지막 컬럼 뒤에 불필요한 쉼표(,)를 남겨 구문 오류가 발생함.
- 나눗셈 에러 처리: `cancel_rate` 계산 시 분모가 0이 될 수 있는 경우에 대한 예외 처리(`SAFE_DIVIDE`) 필요함, 향후 실수 없도록 할 필요 있음

### Try (향후 적용할 복습 포인트 및 개선안)

- 원도우 함수(`OVER()`) 활용 속달: 집계와 상세 내역을 동시에 보고 싶을 때는 `GROUP BY` 대신 원도우 함수를 사용하여 행의 수를 유지하는 방식을 적극 활용할 것
- 고급 컬럼 제어 문법 활용: 컬럼 중복 문제를 해결하기 위해 `EXCEPT` 로 특정 컬럼을 제외하거나, `REPLACE` 로 특정 컬럼만 덮어쓰는 방식을 습관화할 것

- 데이터 무결성 검증: `COUNT(DISTINCT InvoiceNo)` 와 같이 '고유 건수'를 세는 로직과 일반 `COUNT` 의 차이를 명확히 구분하여 데이터 왜곡 방지할 것
- 예외 케이스 고려: 나눗셈 연산 시에는 반드시 `SAFE_DIVIDE` 를 사용하고, `NULL` 값은 `CASE WHEN` 이나 `COALESCE` 로 0점 처리를 명시할 것
- 결과 조회 시 검토 우선순위 염두: 최종 결과 조회 시에는 `ORDER BY` 를 통해 검토할 필요성이 높은 (예: 지출액 상위) 데이터를 먼저 확인하는 버릇 필요