

찬규

11-2. 데이터 불러오기

데이터 살펴보기

- 테이블에 있는 10개의 행만 출력하기

```
SELECT *
FROM continual-nomad-479202-p8.modulabs_project.data
LIMIT 10
```

행	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country
1	536365	85123A	WHITE HANGING HEART TLIGHT...	6	2010-12-01 08:26:00 UTC	2.55	17850	United Kingdom
2	536365	71053	WHITE METAL LANTERN	6	2010-12-01 08:26:00 UTC	3.39	17850	United Kingdom
3	536365	84406B	CREAM CUPID HEARTS COAT H...	8	2010-12-01 08:26:00 UTC	2.75	17850	United Kingdom
4	536365	84029G	KNITTED UNION FLAG HOT WA...	6	2010-12-01 08:26:00 UTC	3.39	17850	United Kingdom
5	536365	84029E	RED WOOLLY HOTTIE WHITE H...	6	2010-12-01 08:26:00 UTC	3.39	17850	United Kingdom
6	536365	22752	SET 7 BABUSHKA NESTING BO...	2	2010-12-01 08:26:00 UTC	7.65	17850	United Kingdom
7	536365	21730	GLASS STAR FROSTED FLIGHT ...	6	2010-12-01 08:26:00 UTC	4.25	17850	United Kingdom
8	536366	22633	HAND WARMER UNION JACK	6	2010-12-01 08:26:00 UTC	1.85	17850	United Kingdom

- 전체 데이터는 몇 행으로 구성되어 있는지 확인하기

```
SELECT COUNT(*)
FROM continual-nomad-479202-p8.modulabs_project.data
```

행	10_
1	541909

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 continual-nomad-479202-p8.test.data
```

행	COUNT_InvoiceNO	COUNT_StockCode	COUNT_Description	COUNT_Quantity	COUNT_InvoiceD...	COUNT_UnitPrice	COUNT_Custome...	COUNT_Country
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, ROUND(SUM(CASE WHEN InvoiceNo IS NULL THEN 1 ELSE 0 END) / COUNT(*) * 100, 2) AS missing_percentage
FROM continual-nomad-479202-p8.test.data
UNION ALL
SELECT 'StockCode' AS column_name, ROUND(SUM(CASE WHEN StockCode IS NULL THEN 1 ELSE 0 END) / COUNT(*) * 100, 2) AS missing_percentage
FROM continual-nomad-479202-p8.test.data
UNION ALL
SELECT 'Description' AS column_name, ROUND(SUM(CASE WHEN Description IS NULL THEN 1 ELSE 0 END) / COUNT(*) * 100, 2) AS missing_percentage
FROM continual-nomad-479202-p8.test.data
UNION ALL
SELECT 'Quantity' AS column_name, ROUND(SUM(CASE WHEN Quantity IS NULL THEN 1 ELSE 0 END) / COUNT(*) * 100, 2) AS missing_percentage
FROM continual-nomad-479202-p8.test.data
UNION ALL
SELECT 'InvoiceDate' AS column_name, ROUND(SUM(CASE WHEN InvoiceDate IS NULL THEN 1 ELSE 0 END) / COUNT(*) * 100, 2) AS missing_percentage
FROM continual-nomad-479202-p8.test.data
UNION ALL
SELECT 'UnitPrice' AS column_name, ROUND(SUM(CASE WHEN UnitPrice IS NULL THEN 1 ELSE 0 END) / COUNT(*) * 100, 2) AS missing_percentage
FROM continual-nomad-479202-p8.test.data
UNION ALL
SELECT 'CustomerID' AS column_name, ROUND(SUM(CASE WHEN CustomerID IS NULL THEN 1 ELSE 0 END) / COUNT(*) * 100, 2) AS missing_percentage
FROM continual-nomad-479202-p8.test.data
UNION ALL
SELECT 'Country' AS column_name, ROUND(SUM(CASE WHEN Country IS NULL THEN 1 ELSE 0 END) / COUNT(*) * 100, 2) AS missing_percentage
FROM continual-nomad-479202-p8.test.data

-- 다른 방법
SELECT column_name, ROUND((total - column_value) / total * 100, 2)
FROM (
    SELECT 'InvoiceNo' AS column_name, COUNT(InvoiceNo) AS column_value, COUNT(*) AS total FROM continual-nomad-479202-p8.test.data UNION ALL
    SELECT 'StockCode' AS column_name, COUNT(StockCode) AS column_value, COUNT(*) AS total FROM continual-nomad-479202-p8.test.data UNION ALL
    SELECT 'Description' AS column_name, COUNT(Description) AS column_value, COUNT(*) AS total FROM continual-nomad-479202-p8.test.data UNION ALL
    SELECT 'Quantity' AS column_name, COUNT(Quantity) AS column_value, COUNT(*) AS total FROM continual-nomad-479202-p8.test.data UNION ALL
    SELECT 'InvoiceDate' AS column_name, COUNT(InvoiceDate) AS column_value, COUNT(*) AS total FROM continual-nomad-479202-p8.test.data UNION ALL
    SELECT 'UnitPrice' AS column_name, COUNT(UnitPrice) AS column_value, COUNT(*) AS total FROM continual-nomad-479202-p8.test.data UNION ALL
    SELECT 'CustomerID' AS column_name, COUNT(CustomerID) AS column_value, COUNT(*) AS total FROM continual-nomad-479202-p8.test.data UNION ALL
    SELECT 'Country' AS column_name, COUNT(Country) AS column_value, COUNT(*) AS total FROM continual-nomad-479202-p8.test.data
) AS column_data;
```

[다른 방법으로 했습니다]

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

결측치 처리 전략

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

```
SELECT Description
FROM continual-nomad-479202-p8.test.data
WHERE StockCode = '85123A'
GROUP BY Description
```

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

결측치 처리

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

```
DELETE FROM continual-nomad-479202-p8.test.data
WHERE CustomerID IS NULL OR Description IS NULL
```

쿼리 결과

작업 정보	결과	실행 세부정보	실행 그래프
●	이 문으로 data의 행 135,080개가 삭제되었습니다.		

11-5. 데이터 전처리(2): 중복값 처리

중복값 확인

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

```
WITH temp AS(
SELECT *, COUNT(*) AS cnt_row
FROM continual-nomad-479202-p8.test.data
GROUP BY InvoiceNO, StockCode, Description, QUANTITY, InvoiceDate, UnitPrice, CustomerID, Country
HAVING cnt_row > 1
)
-- 중복개수 확인
SELECT COUNT(*) FROM temp
```

행	f0_	4837
1		

중복값 처리

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

```
CREATE OR REPLACE TABLE continual-nomad-479202-p8.test.data AS
SELECT DISTINCT *
FROM continual-nomad-479202-p8.test.data
```

```
83 -- 중복값 처리
84 CREATE OR REPLACE TABLE continual-nomad-479202-p8.test.data AS
85 SELECT DISTINCT *
86 FROM continual-nomad-479202-p8.test.data
87
88 -- 중복값 처리 후 남은 데이터 행 개수 파악
89 SELECT COUNT(*)
90 FROM continual-nomad-479202-p8.test.data
91
92 -- 데이터 전처리(3) : 오류값 처리
```

쿼리 완료됨
주문형 처리 할당량 사용 중

쿼리 결과

작업 정보 결과 실행 세부정보 실행 그래프

❶ 이 문으로 이름이 data인 테이블이 교체되었습니다.

중복값 처리 후 남은 데이터 행 개수 파악

행	f0_	401604
1		

11-6. 데이터 전처리(3): 오류값 처리

InvoiceNo 살펴보기

- 고유(unique)한 InvoiceNo의 개수를 출력하기

```
SELECT COUNT(DISTINCT InvoiceNO)
FROM continual-nomad-479202-p8.test.data
```

행	f0_	22190
1		

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

```
SELECT DISTINCT InvoiceNO
FROM continual-nomad-479202-p8.test.data
LIMIT 100
```

행	InvoiceNo
1	541431
2	C541433
3	537626
4	542237
5	549222
6	556201
7	562032
8	573511
9	581180

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

```
SELECT *
FROM continual-nomad-479202-p8.test.data
WHERE InvoiceNo LIKE "C%"
LIMIT 100
```

행	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country
1	C541433	23166	MEDIUM CERAMIC TOP STORA...	-74215	2011-01-18 10:17:00 UTC	1.04	12346	United Kingdom
2	C545329	M	Manual	-1	2011-03-01 15:47:00 UTC	183.75	12352	Norway
3	C545329	M	Manual	-1	2011-03-01 15:47:00 UTC	280.05	12352	Norway
4	C545330	M	Manual	-1	2011-03-01 15:49:00 UTC	376.5	12352	Norway
5	C547388	22701	PINK DOG BOWL	-6	2011-03-22 16:07:00 UTC	2.95	12352	Norway
6	C547388	21914	BLUE HARMONICA IN BOX	-12	2011-03-22 16:07:00 UTC	1.25	12352	Norway
7	C547388	84050	PINK HEART SHAPE EGG FRYIN...	-12	2011-03-22 16:07:00 UTC	1.65	12352	Norway
8	C547388	37448	CERAMIC CAKE DESIGN SPOTT...	-12	2011-03-22 16:07:00 UTC	1.49	12352	Norway

- 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_ratio
FROM continual-nomad-479202-p8.test.data
```

행	canceled_ratio
1	2.2

StockCode 살펴보기

- 고유한 **StockCode** 의 개수를 출력하기

```
SELECT COUNT(DISTINCT StockCode)
FROM continual-nomad-479202-p8.test.data
```

행	f0_
1	3684

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

- 상위 10개의 제품들을 출력하기

```
SELECT StockCode, COUNT(*) AS sell_cnt
FROM continual-nomad-479202-p8.test.data
GROUP BY StockCode
ORDER BY sell_cnt DESC
LIMIT 10
```

쿼리 결과		결과 저장			
행	StockCode	sell_cnt	JSON	설명 세부정보	설명 그래프
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			

페이지당 결과 수: 50 ▾ 1 ~ 10 (전체 10행)

- StockCode** 의 컬럼에 있던 값 중에서 숫자를 제외한 문자만 남기고 문자가 몇 자리 수인지 세고

- 숫자가 0~1개인 값들에는 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 continual-nomad-479202-p8.test.data
)
WHERE number_count = 1 OR number_count = 0
```

행	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

- StockCode** 의 컬럼에 있던 값 중에서 숫자를 제외한 문자만 남기고 문자가 몇 자리 수인지 세고

- 숫자가 0~1개인 값들을 가지고 있는 데이터 수는 전체 데이터 수 대비 몇 퍼센트인지 구하기 (소수점 두 번째 자리까지)

```
SELECT StockCode, ROUND (COUNT(*) / (SELECT COUNT(*) FROM continual-nomad-479202-p8.test.data) * 100, 2) AS ratio
FROM continual-nomad-479202-p8.test.data
GROUP BY StockCode
HAVING StockCode IN ('POST','D','C2','M','BANK CHARGES','PADS','DOT','CRUK')
```

행	StockCode	ratio
1	POST	0.3
2	M	0.11
3	C2	0.03
4	D	0.02
5	BANK CHARGES	0.0
6	PADS	0.0
7	DOT	0.0
8	CRUK	0.0

- 제품과 관련되지 않은 거래 기록을 제거하기

```
DELETE FROM continual-nomad-479202-p8.test.data
WHERE StockCode IN (
    SELECT DISTINCT StockCode
    FROM continual-nomad-479202-p8.test.data
    WHERE StockCode IN ('POST', 'D', 'C2', 'M', 'BANK CHARGES', 'PADS', 'DOT', 'CRUK')
);
```

104 -- 제품과 관련되지 않은 거래 기록 제거하기
105 65 DELETE FROM continual-nomad-479202-p8.test.data
106 WHERE StockCode IN (
107 -> SELECT DISTINCT StockCode
108 -> FROM continual-nomad-479202-p8.test.data
109 -> WHERE StockCode IN ('POST', 'D', 'C2', 'M', 'BANK CHARGES', 'PADS', 'DOT', 'CRUK')
110);
111
112 -- Description 살펴보기
113 -- 고유한 Description 별 출현 빈도를 계산하고 상위 30개 출력
114 ✓ 쿼리 완료됨
115 주문형 처리 활성화 사용 중

쿼리 결과

작업 정보	결과	실행 세부정보	실행 그래프
❶ 이 문장으로 data의 행 1,915개가 삭제되었습니다.			

Description 살펴보기

- 고유한 Description 별 출현 빈도를 계산하고 상위 30개를 출력하기

```
SELECT Description, COUNT(*) AS description_cnt
FROM continual-nomad-479202-p8.test.data
GROUP BY Description
ORDER BY description_cnt DESC
LIMIT 30
```

작업 정보	결과	시각화	JSON	실행 세부정보	실행 그래프
행	description	description_cnt			
1	WHITE HANGING HEART TLIGHT...	2058			
2	REGENTY CAKESTAND 3 TIER	1894			
3	JUMBO BAG RED RETROSPOT	1659			
4	PARTY BUNTING	1409			
5	ASSORTED COLOUR BIRD BNIN...	1405			
6	LUNCH BAG RED RETROSPOT	1345			
7	SET OF 3 CAKE TINS PANTRY D...	1224			
8	LUNCH BAG BLACK SKULL	1099			
9	PACK OF 72 RETROSPOT CAKE ...	1062			
10	SPOTTY BUNTING	1026			
11	PAPER CHAIN KIT 50'S CHRIST...	1013			

페이지당 결과 수: 50 ▾ 1 ~ 30 (전체 30행)

- 서비스 관련 정보를 포함하는 행들을 제거하기

```
DELETE FROM continual-nomad-479202-p8.test.data
WHERE Description IN ('Next Day Carriage','High Resolution Image');
```

```
185 -- 서비스 관련 정보를 포함하는 행들을 제거하기
186 DELETE FROM continual-nomad-479202-p8.test.data
187 WHERE Description IN ('Next Day Carriage','High Resolution Image');
188
189 -- 대소문자를 혼합하고 있는 데이터를 대문자로 표준화
190 CREATE OR REPLACE TABLE continual-nomad-479202-p8.test.data AS
191 SELECT
192   * EXCEPT (Description),
193   UPPER(description) AS Description
194
195 -- 퀄리 인터넷
196
197 주문형 처리 할당량 사용 중
```

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 continual-nomad-479202-p8.test.data AS
SELECT
  * EXCEPT (Description),
  UPPER(description) AS Description
FROM continual-nomad-479202-p8.test.data
```

```
189 -- 대소문자를 혼합하고 있는 데이터를 대문자로 표준화
190 CREATE OR REPLACE TABLE continual-nomad-479202-p8.test.data AS
191 SELECT
192   * EXCEPT (Description),
193   UPPER(description) AS Description
194   FROM continual-nomad-479202-p8.test.data
195
196 -- UnitPrice 살펴보기
197 SELECT MIN(UnitPrice) AS min_price,
198   MAX(UnitPrice) AS max_price
199
200 퀄리 인터넷
201 주문형 처리 할당량 사용 중
```

쿼리 결과

작업 정보 결과 실행 세부정보 실행 그래프

ⓘ 이 문으로 이름이 data인 테이블이 교체되었습니다.

UnitPrice 살펴보기

- UnitPrice의 최솟값, 최댓값, 평균을 구하기

```
SELECT MIN(UnitPrice) AS min_price,
       MAX(UnitPrice) AS max_price,
```

```
AVG(UnitPrice) AS avg_price  
FROM continual-nomad-479202-p8.test.data
```

행	min_price	max_price	avg_price
1	0.0	649.5	2.904956757405...

- 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 continual-nomad-479202-p8.test.data  
WHERE UnitPrice = 0
```

행	cnt_quantity	min_quantity	max_quantity	avg_quantity
1	33	1	12540	420.5151515151...

- UnitPrice = 0 를 제거하고 일관된 데이터셋을 유지하기

```
DELETE FROM continual-nomad-479202-p8.test.data  
WHERE UnitPrice = 0
```

```
210 -- 단가가 0원인 데이터 제거  
211 DELETE FROM continual-nomad-479202-p8.test.data  
212 WHERE UnitPrice = 0  
213 -- 혹은  
214 CREATE OR REPLACE TABLE continual-nomad-479202-p8.test.data AS  
215 SELECT *  
216 FROM continual-nomad-479202-p8.test.data  
217 WHERE UnitPrice > 0  
218  
219 -- RFM 스코어  
220 -- Recency  
221 -- InvoiceDate 컬럼을 연월일 자료형으로 변경
```

① Syntax error: Expected end of input but got keyword SELECT at [7:1]

주문형 처리 할당량 사용 중

쿼리 결과

작업 정보 결과 실행 세부정보 실행 그레프

① 이 문으로 data의 행 33개가 삭제되었습니다.

11-7. RFM 스코어

Recency

- InvoiceDate 컬럼을 연월일 자료형으로 변경하기

```
SELECT DATE(InvoiceDate) AS InvoiceDay, *  
FROM continual-nomad-479202-p8.test.data
```

행	InvoiceDay	InvoiceNo	StockCode	Quantity	InvoiceDate	UnitPrice	CustomerID	Country	Description
1	2011-01-18	541431	23166	74215	2011-01-18 10:01:00 UTC	1.04	12346	United Kingdom	MEDIUM CERAMIC TOF
2	2011-01-18	C541433	23166	-74215	2011-01-18 10:17:00 UTC	1.04	12346	United Kingdom	MEDIUM CERAMIC TOF
3	2010-12-07	537626	22729	4	2010-12-07 14:57:00 UTC	3.75	12347	Iceland	ALARM CLOCK BAKELII
4	2010-12-07	537626	84997B	6	2010-12-07 14:57:00 UTC	3.75	12347	Iceland	RED 3 PIECE RETROSP
5	2010-12-07	537626	22771	12	2010-12-07 14:57:00 UTC	1.25	12347	Iceland	CLEAR DRAWER KNOB
6	2010-12-07	537626	84997D	6	2010-12-07 14:57:00 UTC	3.75	12347	Iceland	PINK 3 PIECE POLKAD
7	2010-12-07	537626	22725	4	2010-12-07 14:57:00 UTC	3.75	12347	Iceland	ALARM CLOCK BAKELII
8	2010-12-07	537626	85167B	30	2010-12-07 14:57:00 UTC	1.25	12347	Iceland	BLACK GRAND BAROQ
9	2010-12-07	537626	22805	12	2010-12-07 14:57:00 UTC	1.25	12347	Iceland	BLUE DRAWER KNOB A

- 가장 최근 구매 일자를 MAX() 함수로 찾아보기

```
SELECT MAX(InvoiceDate) AS most_recent_date  
FROM `continual-nomad-479202-p8.test.data`;
```

행	most_recent_date
1	2011-12-09 12:50:00 UTC

- 유저 별로 가장 큰 InvoiceDay를 찾아서 가장 최근 구매일로 저장하기

```
SELECT  
CustomerID,  
MAX(InvoiceDate) AS InvoiceDay  
FROM continual-nomad-479202-p8.test.data  
GROUP BY CustomerID  
ORDER BY CustomerID
```

행	CustomerID	InvoiceDay
1	12346	2011-01-18 10:17:00 UTC
2	12347	2011-12-07 15:52:00 UTC
3	12348	2011-09-25 13:13:00 UTC
4	12349	2011-11-21 09:51:00 UTC
5	12350	2011-02-02 16:01:00 UTC
6	12352	2011-11-03 14:37:00 UTC
7	12353	2011-05-19 17:47:00 UTC
8	12354	2011-04-21 13:11:00 UTC
9	12355	2011-05-09 13:49:00 UTC
10	12356	2011-11-17 08:40:00 UTC

- 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 continual-nomad-479202-p8.test.data
GROUP BY CustomerID
);

```

행	CustomerID	recency
1	12377	315
2	12423	0
3	12426	194
4	12646	4
5	12950	2
6	13385	329
7	13575	23
8	13626	252
9	13654	43
10	13744	95

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

```

CREATE OR REPLACE TABLE continual-nomad-479202-p8.test.user_r AS
SELECT CustomerID,
       EXTRACT(DAY FROM MAX(InvoiceDay) OVER () - InvoiceDay) AS recency
  FROM (
    SELECT CustomerID, MAX(DATE(InvoiceDate)) AS InvoiceDay
      FROM continual-nomad-479202-p8.test.data
     GROUP BY CustomerID
);

```

```

248 -- 지금까지의 결과를 user_r이라는 이름의 테이블로 저장
249 CREATE OR REPLACE TABLE continual-nomad-479202-p8.test.user_r AS
250   SELECT CustomerID,
251     EXTRACT(DAY FROM MAX(InvoiceDay) OVER () - InvoiceDay) AS recency,
252   FROM (
253     SELECT CustomerID, MAX(DATE(InvoiceDate)) AS InvoiceDay
254       FROM continual-nomad-479202-p8.test.data
255     GROUP BY CustomerID
256   );
257
258 -- Frequency
259 -- 전체 거래 건수 계산
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
559
560
561
562
563
564
565
566
567
568
569
569
570
571
572
573
574
575
576
577
578
579
579
580
581
582
583
584
585
586
587
588
589
589
590
591
592
593
594
595
596
597
598
599
599
600
601
602
603
604
605
606
607
608
609
609
610
611
612
613
614
615
616
617
618
619
619
620
621
622
623
624
625
626
627
628
629
629
630
631
632
633
634
635
636
637
638
639
639
640
641
642
643
644
645
646
647
648
649
649
650
651
652
653
654
655
656
657
658
659
659
660
661
662
663
664
665
666
667
668
669
669
670
671
672
673
674
675
676
677
678
679
679
680
681
682
683
684
685
686
687
688
689
689
690
691
692
693
694
695
696
697
698
699
699
700
701
702
703
704
705
706
707
708
709
709
710
711
712
713
714
715
716
717
718
719
719
720
721
722
723
724
725
726
727
728
729
729
730
731
732
733
734
735
736
737
738
739
739
740
741
742
743
744
745
746
747
748
749
749
750
751
752
753
754
755
756
757
758
759
759
760
761
762
763
764
765
766
767
768
769
769
770
771
772
773
774
775
776
777
778
779
779
780
781
782
783
784
785
786
787
788
788
789
789
790
791
792
793
794
795
796
797
798
799
799
800
801
802
803
804
805
806
807
808
809
809
810
811
812
813
814
815
816
817
818
819
819
820
821
822
823
824
825
826
827
828
829
829
830
831
832
833
834
835
836
837
838
839
839
840
841
842
843
844
845
846
847
848
849
849
850
851
852
853
854
855
856
857
858
859
859
860
861
862
863
864
865
866
867
868
869
869
870
871
872
873
874
875
876
877
878
879
879
880
881
882
883
884
885
886
887
888
889
889
890
891
892
893
894
895
896
897
898
899
899
900
901
902
903
904
905
906
907
908
909
909
910
911
912
913
914
915
916
917
918
919
919
920
921
922
923
924
925
926
927
928
929
929
930
931
932
933
934
935
936
937
938
939
939
940
941
942
943
944
945
946
947
948
949
949
950
951
952
953
954
955
956
957
958
959
959
960
961
962
963
964
965
966
967
968
969
969
970
971
972
973
974
975
976
977
978
979
979
980
981
982
983
984
985
986
987
988
989
989
990
991
992
993
994
995
996
997
998
999
999
1000
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1088
1089
1090
1091
1092
1093
1094
1095
1096
1097
1097
1098
1099
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1149
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1179
1179
1180
1181
1182
1183
1184
1185
1186
1187
1188
1188
1189
1190
1191
1192
1193
1194
1195
1196
1196
1197
1198
1199
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1249
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1279
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1288
1289
1290
1291
1292
1293
1294
1295
1296
1297
1297
1298
1299
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1379
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1388
1389
1390
1391
1392
1393
1394
1395
1396
1397
1397
1398
1399
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1439
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1479
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1488
1489
1490
1491
1492
1493
1494
1495
1496
1497
1497
1498
1499
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1597
1598
1599
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1679
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1688
1689
1690
1691
1692
1693
1694
1695
1696
1697
1697
1698
1699
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1788
1789
1790
1791
1792
1793
1794
1795
1796
1797
1797
1798
1799
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1888
1889
1889
1890
1891
1892
1893
1894
1895
1896
1897
1897
1898
1899
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2088
2089
2089
2090
2091
2092
2093
2094
2095
2096
2097
2097
2098
2099
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2139
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2148
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2179
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2188
2189
2189
2190
2191
2192
2193
2194
2195
2196
2197
2197
2198
2199
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2209
2210
2211
2212
2213
2214
2215
2216
2217
2218
2219
2219
2220
2221
2222
2223
2224
2225
2226
2227
2228
2229
2229
2230
2231
2232
2233
2234
2235
2236
2237
2238
2239
2239
2240
2241
2242
2243
2244
2245
2246
2247
2248
2248
2249
2250
2251
2252
2253
2254
2255
2256
2257
2258
2259
2259
2260
2261
2262
2
```

행	CustomerID	purchase_cnt
1	14911	242
2	12748	217
3	17841	169
4	14606	125
5	15311	118
6	13089	118
7	12971	88
8	13408	75
9	14646	73
10	16029	66

- 각 고객 별로 구매한 아이템의 총 수량 더하기

```
SELECT CustomerID,
       SUM(Quantity) AS item_cnt
  FROM continual-nomad-479202-p8.test.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
9	12355	240
10	12356	1573

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

```
CREATE OR REPLACE TABLE `continual-nomad-479202-p8.modulabs.user_rf` AS

-- [WITH 절 시작]
WITH purchase_cnt AS (
    -- (1) 전체 거래 건수 (F: Frequency)
    SELECT
        CustomerID,
        COUNT(DISTINCT InvoiceNo) AS purchase_cnt
    FROM `continual-nomad-479202-p8.test.data`
    GROUP BY CustomerID
    -- 여기 ORDER BY 제거! (불필요)
),
item_cnt AS (
    -- (2) 총 수량 (M: Monetary 관련)
    SELECT
        CustomerID,
        SUM(Quantity) AS item_cnt
    FROM `continual-nomad-479202-p8.test.data`
    GROUP BY CustomerID
)
-- [최종 SELECT] 3개 테이블(pc, ic, ur) 조인
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 `continual-nomad-479202-p8.test.user_rf` AS ur
ON pc.CustomerID = ur.CustomerID;
```

쿼리 결과			
작업 정보	결과	실행 세부정보	실행 그래프
❶ 이 문으로 이름이 user_rf인 테이블이 교체되었습니다.			

Monetary

- 고객별 총 지출액 계산 (소수점 첫째 자리에서 반올림)

```
SELECT CustomerID, ROUND(SUM(UnitPrice * Quantity),1) AS user_total
FROM continual-nomad-479202-p8.test.data
GROUP BY CustomerID
```

행	CustomerID	user_total
1	12346	0.0
2	12347	4310.0
3	12348	1437.2
4	12349	1457.6
5	12350	294.4
6	12352	1265.4
7	12353	89.0
8	12354	1079.4
9	12355	459.4
10	12356	2487.4
11	12357	6207.7

- 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 continual-nomad-479202-p8.test.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, 1) AS user_average
FROM continual-nomad-479202-p8.modulabs.user_rf AS rf
LEFT JOIN (
--고객 별 총 지출액
SELECT CustomerID, ROUND(SUM(UnitPrice * Quantity),1) AS user_total
FROM continual-nomad-479202-p8.test.data
GROUP BY CustomerID
) AS ut
ON rf.CustomerID = ut.CustomerID;
```

쿼리 결과

작업 정보 결과 실행 세부정보 실행 그래프

ⓘ 이 문으로 이름이 user_rfm인 새 테이블이 생성되었습니다.

RFM 통합 테이블 출력하기

- 최종 user_rfm 테이블을 출력하기

```
SELECT *
FROM continual-nomad-479202-p8.test.user_rfm
```

행	CustomerID	purchase_cnt	item_cnt	recency	user_total	user_average
1	12713	1	505	0	794.5	794.5
2	14569	1	79	1	227.4	227.4
3	13298	1	96	1	360.0	360.0
4	15520	1	314	1	343.5	343.5
5	13436	1	76	1	196.9	196.9
6	14204	1	72	2	150.6	150.6
7	15471	1	256	2	454.5	454.5
8	15195	1	1404	2	3861.0	3861.0
9	12650	1	250	3	242.4	242.4
10	15992	1	17	3	42.0	42.0
11	14578	1	946	2	162.6	162.6

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 continual-nomad-479202-p8.test.user_data AS
WITH unique_products AS (
  SELECT
    CustomerID,
    COUNT(DISTINCT StockCode) AS unique_products
  FROM continual-nomad-479202-p8.test.data
  GROUP BY CustomerID
)
SELECT ur.* , up.* EXCEPT (CustomerID)
FROM continual-nomad-479202-p8.test.user_rfm AS ur
JOIN unique_products AS up
ON ur.CustomerID = up.CustomerID;
```

```
341 -- 이 문은 데이터베이스에서 실행되는 SQL 쿼리를 보여줍니다. 주제는 데이터베이스 관리입니다.
342 CREATE OR REPLACE TABLE continual-nomad-479202-p8.test.user_data AS
343 WITH unique_products AS (
344     SELECT
345         CustomerID,
346         COUNT(DISTINCT StockCode) AS unique_products
347     FROM continual-nomad-479202-p8.test.data
348     GROUP BY CustomerID
349 )
350 SELECT ur.* , up.* EXCEPT (CustomerID)
351 FROM continual-nomad-479202-p8.test.user_rfm AS ur
352 INNER JOIN unique_products AS up
353 WHERE ur.CustomerID = up.CustomerID
```

2. 평균 구매 주기

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

```
CREATE OR REPLACE TABLE continual-nomad-479202-p8.test.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
            continual-nomad-479202-p8.test.data
        WHERE CustomerID IS NOT NULL
    )
    GROUP BY CustomerID
)
SELECT u.*, pi.* EXCEPT (CustomerID)
FROM continual-nomad-479202-p8.test.user_data AS u
LEFT JOIN purchase_intervals AS pi
ON u.CustomerID = pi.CustomerID;
```

쿼리 완료됨
주문형 처리 할당량 사용 중

쿼리 결과

작업 정보 결과 실행 세부정보 실행 그래프

이 문으로 이름이 user_data인 테이블이 교체되었습니다.

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 continual-nomad-479202-p8.test.user_data AS

WITH TransactionInfo AS (
  SELECT
    CustomerID,
    COUNT(*) AS total_transactions,
    SUM(CASE WHEN InvoiceNO LIKE 'C%' THEN 1 ELSE 0 END) AS cancel_frequency
  FROM continual-nomad-479202-p8.test.data
  GROUP BY CustomerID
)

SELECT u.* , t.* EXCEPT(CustomerID), ROUND(t.cancel_frequency / t.total_transactions,2) AS cancel_rate
FROM `continual-nomad-479202-p8.test.user_data` AS u
LEFT JOIN TransactionInfo AS t
ON t.CustomerID = u.CustomerID;
```

쿼리 결과

작업 정보 결과 실행 세부정보 실행 그래프

이 문으로 이름이 user_data인 테이블이 교체되었습니다.

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

```
SELECT *
FROM continual-nomad-479202-p8.test.user_data
```

행	CustomerID	purchase_cnt	item_cnt	recency	user_total	user_average	unique_products	average_interval	total_transactions	cancel_frequency	cancel_rate
1	13667	1	149	149	299.0	299.0	82	0.0	83	0	0.0
2	15870	1	333	32	351.4	351.4	68	0.0	74	0	0.0
3	16054	1	380	145	783.9	783.9	70	0.0	70	0	0.0
4	16270	1	688	353	1141.2	1141.2	54	0.0	54	0	0.0
5	12349	1	630	18	1457.6	1457.6	72	0.0	72	0	0.0
6	15000	1	218	47	490.5	490.5	76	0.0	77	0	0.0
7	16504	1	510	25	465.9	465.9	76	0.0	83	0	0.0
8	12391	1	293	21	439.7	439.7	92	0.0	93	0	0.0
9	17914	1	457	3	329.4	329.4	72	0.0	75	0	0.0
10	15832	1	465	254	836.8	836.8	54	0.0	54	0	0.0
11	13339	1	326	200	860.1	860.1	54	0.0	54	0	0.0
...

회고

[회고 내용을 작성해주세요]

Keep : SQL에서도 정규표현식을 이용해 문자열 필터링이 가능하다는 것을 알게 되었다.

Problem : 특정 코드 값을 가진 데이터 수의 비율을 구하는 쿼리문에서 SELECT 절을 컬럼의 수만큼 적는게 너무 번거로웠다. 그래서 전체 컬럼(*)에 대해 조건을 걸고, 그룹화하여 HAVING으로 특정 코드를 조건으로 적어주니 훨씬 간편했다.

Try : UNION ALL을 하는 과정에서 최종 결과가 가로로 쭉 늘어져있는 형태여서 시각적으로 좋지 못했다. 그래서 서브쿼리를 통해 2개의 열로만 깔끔하게 나오는 결과를 얻을 수 있었다.

행	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country
1	536365	85123A	WHITE HANGING HEART T-LIGH...	6	2010-12-01 08:26:00 UTC	2.55	17850	United Kingdom
2	536365	71053	WHITE METAL LANTERN	6	2010-12-01 08:26:00 UTC	3.39	17850	United Kingdom
3	536365	84406B	CREAM CUPID HEARTS COAT H...	8	2010-12-01 08:26:00 UTC	2.75	17850	United Kingdom
4	536365	84029G	KNITTED UNION FLAG HOT WA...	6	2010-12-01 08:26:00 UTC	3.39	17850	United Kingdom
5	536365	84029E	RED WOOLLY HOTTIE WHITE H...	6	2010-12-01 08:26:00 UTC	3.39	17850	United Kingdom
6	536365	22752	SET 7 BABUSHKA NESTING BO...	2	2010-12-01 08:26:00 UTC	7.65	17850	United Kingdom
7	536365	21730	GLASS STAR FROSTED T-LIGHT ...	6	2010-12-01 08:26:00 UTC	4.25	17850	United Kingdom
8	536366	22633	HAND WARMER UNION JACK	6	2010-12-01 08:28:00 UTC	1.85	17850	United Kingdom