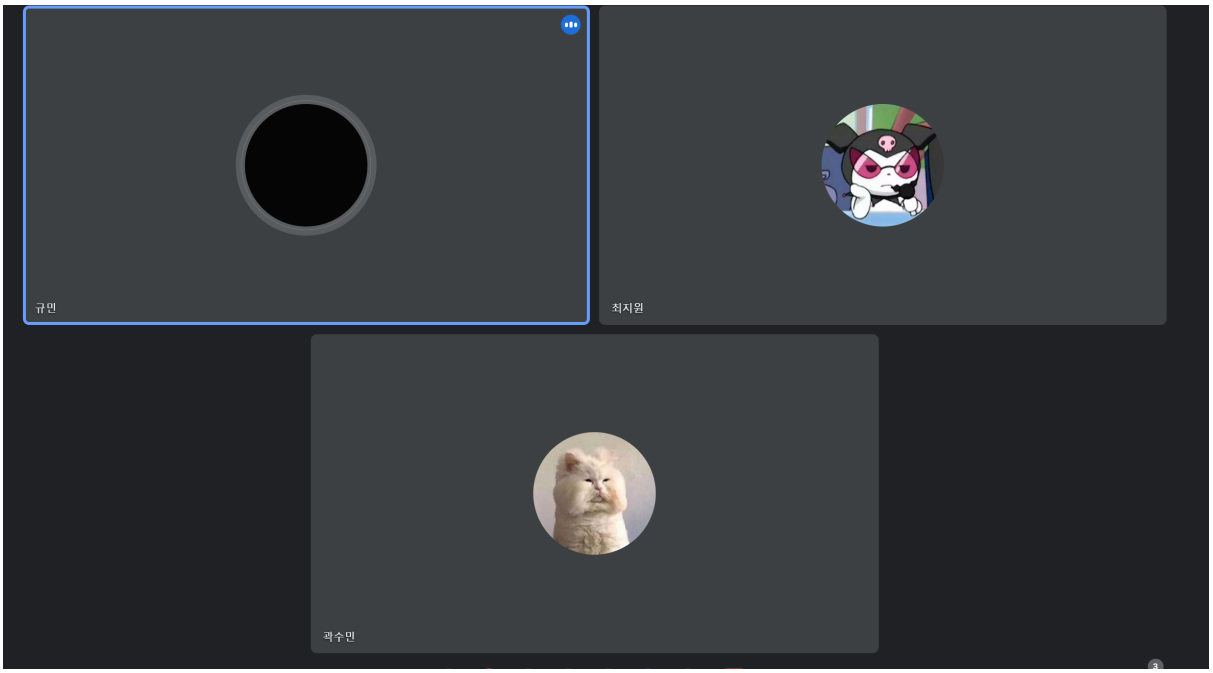
10~11강

상태 곽수민



▼ 10강. 다면적인 축을 사용해 데이터 집약하기

1. 카테고리별 매출과 소계 계산하기

- 서브 카테고리, 카테고리, 전체 집계를 따로따로해서 union all로 합쳐줌

```
with subcategory_amount as (
select category as category, sub_category as sub_category, sum(price) as amount
from purchase_detail_log
group by 1, 2
)
, category_amount as (
select category, 'all' AS sub_category, sum(price) as amount
from purchase_detail_log
group by 1, 2
)
, total_amount as (
select 'all' as category, 'all' as sub_category, sum(price) as amount
from purchase_detail_log
)

select * from subcategory_amount
union all select * from category_amount
union all select * from total_amount
```

- union all은 성능이 좋지 않기 때문에, ROLLUP을 사용하는 것이 더 좋다! (ROLLUP으로 집계되는 레코드 값은 NULL처리)

```
select coalesce(category, 'all') as category,
  coalesce(sub_category, 'all') as sub_category,
```

```
  sum(price) as mount
from purchase_detail_log
group by rollup(category, sub_category)
```

2. ABC 분석으로 잘 팔리는 상품 판별하기

- ABC 분석이란?
    - 매출 중요도에 따라 상품을 A/B/C로 나누고, 각 그룹 별로 가지는 값을 비교

```
with monthly_sale as (
select category, sum(price) as amount
from purchase_detail_log
-- where dt between '2015-12-01' and '2015-12-31'
group by category
)
, sales_composition_ratio as (
select category, amount,
  100.0*amount / sum(amount) over() as composition_ratio,
  100.0*sum(amount) over(order by amount desc rows between unbounded preceding and current row)
  / sum(amount) over() as cummulative_ratio
from monthly_sale
)

select *,
  case when cummulative_ratio between 0 and 70 then 'A'
  when cummulative_ratio between 70 and 90 then 'B'
  when cummulative_ratio between 90 and 100 then 'C' end as abc_rank
from sales_composition_ratio
order by amount desc
```

3. 팬 차트로 상품의 매출 증가율 확인하기

- 팬 차트 : 어떤 시점을 100%로 두고 이후의 숫자 변동을 확인할 수 있게 하는 그래프

```
with daily_category_amount as (
select dt, category, substring(dt, 1, 4) as year
  , substring(dt, 6, 2) as month
  , substring(dt, 9, 2) as day
  , sum(price) as amount
from purchase_detail_log
group by 1, 2
)
, monthly_category_amount as (
select year||'-'||month as ym
  , category, sum(amount) as amount
from daily_category_amount
group by 1, 2
)

select ym, category, amount,
  first_value(amount) over(partition by category order by ym, category rows unbounded preceding) as base_amount
  , 100.0*amount / first_value(amount) over(partition by category order by ym, category rows unbounded preceding)
from monthly_category_amount
order by 1, 2
```

4. 히스토그램으로 구매 가격대 집계하기

```
with stats as (
select max(price) as max_price,
      min(price) as min_price,
      max(price) - min(price) as range_price,
```

```
      10 as bucket_num
from purchase_detail_log
)
, purchase_log_with_bucket as (
select price, min_price, price - min_price as diff,
    1.0 * range_price / bucket_num as bucket_range
  , floor(1.0 * (price - min_price) / (1.0 * range_price / bucket_num))+1 as bucket
from purchase_detail_log, stats
)

select *
from purchase_log_with_bucket
order by 1
```

- 위 쿼리는 `하한 이상 ~ 상한 미만`으로 그룹을 나누었기 때문에 최댓값은 상만 미만에 포함이 안되어 그룹의 개수보다 + 1된 그룹으로 들어감

```
with stats as (
select max(price) + 1 as max_price,
      min(price) as min_price,
      max(price) + 1 - min(price) as range_price,
      10 as bucket_num
from purchase_detail_log
)
, purchase_log_with_bucket as (
select price, min_price, price - min_price as diff,
    1.0 * range_price / bucket_num as bucket_range
  , floor(1.0 * (price - min_price) / (1.0 * range_price / bucket_num))+1 as bucket
from purchase_detail_log, stats
)

select *
from purchase_log_with_bucket
order by 1
```

- 같은 쿼리에서 MAX 값만 + 1을 해주어서, 모두가 10개 그룹 이내로 들어올 수 있도록 조정
- 히스토그램을 구하기 위해 매출을 집계

```
with stats as (
select max(price) + 1 as max_price, min(price) as min_price
    , max(price) + 1 - min(price) as range_price
    , 10 as bucket_num
from purchase_detail_log
)
, purchase_log_with_bucket as (
select price, min_price, price - min_price as diff,
    1.0 * range_price / bucket_num as bucket_range
    , floor(1.0 * (price - min_price) / (1.0 * range_price / bucket_num))+1 as bucket
from purchase_detail_log, stats
)

select bucket, min_price + bucket_range * (bucket - 1) as lower_limit
  , min_price + bucket_range * bucket as upper_limit
  , count(price) as num_purchase
  , sum(price) as total_amount
from purchase_log_with_bucket
group by bucket, min_price, bucket_range
order by bucket
```

- 히스토그램의 상한과 하한을 수동으로 조정 (5000원)한 쿼리

```
with stats as (
select 50000 as max_price, 0 as min_price
  , 50000 as range_price, 10 as bucket_num
```

```
from purchase_detail_log
)
, purchase_log_with_bucket as (
select price, min_price, price - min_price as diff,
  1.0 * range_price / bucket_num as bucket_range
  , floor(1.0 * (price - min_price) / (1.0 * range_price / bucket_num))+1 as bucket
from purchase_detail_log, stats
)

select bucket, min_price + bucket_range * (bucket - 1) as lower_limit
  , min_price + bucket_range * bucket as upper_limit
  , count(price) as num_purchase
  , sum(price) as total_amount
from purchase_log_with_bucket
group by bucket, min_price, bucket_range
order by bucket
```

▼ 11강. 사용자 전체의 특징과 경향 찾기

1. 사용자의 액션 수 집계하기

- view, add_cart, purchase로 나누어 집계

```
with stats as (
select count(distinct session) as total_uu
from action_log
)

select l.action
  , count(distinct l.session) as action_uu
  , count(1) as cation_count
  , s.total_uu
  , 100.0 * count(distinct l.session) / s.total_uu as usage_rate
  , 1.0 * count(1) / count(distinct l.session) as count_per_user
from action_log l
cross join stats s
group by 1, 4
```

- 로그인 사용자와 비로그인 사용자를 구분해서 집계

```
with action_log_with_status as (
select session, user_id, action,
  case when coalesce(user_id, '') <> '' then 'login' else 'guest' end login_status
from action_log
)

select *
from action_log_with_status
```

```
select coalesce(action, 'all') as action
  , coalesce(login_status, 'all') as login_status
  , count(distinct session) as action_uu
  , count(1) as action_count
from action_log_with_status
group by rollup(action, login_status)
```

- 회원과 비회원을 구분하여 집계 (로그인을 안했더라도 이전에 한 번이라도 로그인을 했으면 회원으로 카운트)

```
with action_log_with_status as (
select session, user_id, action,
  case when coalesce(max(user_id) over(partition by session order by stamp
                    rows between unbounded preceding and current row)
                     , '') <> '' then 'member' else 'none' end member_status,
  stamp

from action_log
)

select *
from action_log_with_status
```

2. 연령별 구분 집계하기

```
with mst_users_with_int_birth_date as (
select *,
  20170101 as int_specific_date,
  replace(substring(birth_date, 1, 10), '-', '')::integer as int_birth_date
from mst_users
)
, mst_users_with_age as (
select *,
  floor((int_specific_date - int_birth_date)/10000) as age
from mst_users_with_int_birth_date
)

select user_id, sex, birth_date, age
from mst_users_with_age
```

- 앞에서 계산한 연령을 바탕으로 연령대를 그룹화

```
with mst_users_with_int_birth_date as (
select *,
  20170101 as int_specific_date,
  replace(substring(birth_date, 1, 10), '-', '')::integer as int_birth_date
from mst_users
)
, mst_users_with_age as (
select *,
  floor((int_specific_date - int_birth_date)/10000) as age
from mst_users_with_int_birth_date
)
, mst_users_with_category as (
select user_id, sex, age,
  (case when age >= 20 then sex else '' end)||
  (case when age between 4 and 12 then 'C'
    when age between 13 and 19 then 'T'
    when age between 20 and 34 then '1'
    when age between 35 and 49 then '2'
    when age >= 50 then '3' end) as category
from mst_users_with_age
)

select *
from mst_users_with_category
```

3. 연령별 구분의 특징 추출하기

- 연령별 X 카테고리별 구매량 집계

```
with mst_users_with_int_birth_date as (
select *,
  20170101 as int_specific_date,
  replace(substring(birth_date, 1, 10), '-', '')::integer as int_birth_date
from mst_users
)
, mst_users_with_age as (
select *,
  floor((int_specific_date - int_birth_date)/10000) as age
from mst_users_with_int_birth_date
)
, mst_users_with_category as (
select user_id, sex, age,
  (case when age >= 20 then sex else '' end)||
  (case when age between 4 and 12 then 'C'
    when age between 13 and 19 then 'T'
    when age between 20 and 34 then '1'
    when age between 35 and 49 then '2'
    when age >= 50 then '3' end) as category
from mst_users_with_age
)

select p.category as product_category
  , u.category as user_category
  , count(*) as purchase_count
from action_log p
join mst_users_with_category as u on p.user_id = u.user_id
where action = 'purchase'
group by 1, 2
order by 1, 2
```

4. 사용자의 방문 빈도 집계하기

- 일주일 동안 서비스를 며칠이나 이용하는지, 그 일수별로 사용자는 몇명인지 집계

```
with action_log_with_dt as (
select *, substring(stamp, 1, 10) dt
from action_log
)
, action_day_count_per_user as (
select user_id, count(distinct dt) as action_day_count
from action_log_with_dt
where dt between '2016-11-01' and '2016-11-07'
group by 1
)

select action_day_count, count(distinct user_id) as user_count
from action_day_count_per_user
group by 1
order by 1
```

5. 벤 다이어그램으로 사용자 액션 집계하기

- 사용자들이 액션을 한 번이라도 했는지 아닌지 플래그를 집계

```
with user_action_flag as (
select user_id,
  sign(sum(case when action = 'purchase' then 1 else 0 end)) has_purchase
  , sign(sum(case when action = 'preview' then 1 else 0 end)) has_review
  , sign(sum(case when action = 'favorite' then 1 else 0 end)) has_favorite
from action_log
group by 1
)
```

```
select *
from user_action_flag
```

- 모든 액션 조합 별로 이용한 이용자 수 (CUBE 함수)

```
with user_action_flag as (
select user_id,
  sign(sum(case when action = 'purchase' then 1 else 0 end)) has_purchase
  , sign(sum(case when action = 'preview' then 1 else 0 end)) has_review
  , sign(sum(case when action = 'favorite' then 1 else 0 end)) has_favorite
from action_log
group by 1
)
, action_venn_diagram as (
select has_purchase, has_review
  , has_favorite, count(1) as users
from user_action_flag
group by cube(has_purchase, has_review, has_favorite)
)

select *
from action_venn_diagram
order by 1, 2, 3
```

- 같은 쿼리를 CUBE를 쓰지 않은 버전 (비효율적)

```
with user_action_flag as (
select user_id,
  sign(sum(case when action = 'purchase' then 1 else 0 end)) has_purchase
  , sign(sum(case when action = 'preview' then 1 else 0 end)) has_review
  , sign(sum(case when action = 'favorite' then 1 else 0 end)) has_favorite
from action_log
group by 1
)
, action_venn_diagram as (
-- 3개의 액션 모두 한 경우
  select has_purchase, has_review, has_favorite, count(1) users
  from user_action_flag
  group by 1, 2, 3

-- 3개 중 2개의 액션을 한 경우
union all select null as has_purchase, has_review, has_favorite, count(1) users
  from user_action_flag
  group by 2, 3
union all select has_purchase, null as has_review, has_favorite, count(1) users
  from user_action_flag
  group by 1, 3
union all select has_purchase, has_review, null as has_favorite, count(1) users
  from user_action_flag
  group by 1, 2

-- 3개 중 1개의 액션만
union all select null as has_purchase, null as has_review, has_favorite, count(1) users
  from user_action_flag
  group by 3
union all select null as has_purchase, has_review, null as has_favorite, count(1) users
  from user_action_flag
  group by 2
union all select has_purchase, null as has_review, null as has_favorite, count(1) users
  from user_action_flag
  group by 1
)

select *
from action_venn_diagram
order by 1, 2, 3
```

- 벤 다이어그램을 만들기 위해 데이터를 가공하는 쿼리

```
with user_action_flag as (
select user_id,
  sign(sum(case when action = 'purchase' then 1 else 0 end)) has_purchase
  , sign(sum(case when action = 'preview' then 1 else 0 end)) has_review
  , sign(sum(case when action = 'favorite' then 1 else 0 end)) has_favorite
from action_log
group by 1
)
, action_venn_diagram as (
-- 3개의 액션 모두 한 경우
  select has_purchase, has_review, has_favorite, count(1) users
  from user_action_flag
  group by 1, 2, 3

-- 3개 중 2개의 액션을 한 경우
union all select null as has_purchase, has_review, has_favorite, count(1) users
  from user_action_flag
  group by 2, 3
union all select has_purchase, null as has_review, has_favorite, count(1) users
  from user_action_flag
  group by 1, 3
union all select has_purchase, has_review, null as has_favorite, count(1) users
  from user_action_flag
  group by 1, 2

-- 3개 중 1개의 액션만
union all select null as has_purchase, null as has_review, has_favorite, count(1) users
  from user_action_flag
  group by 3
union all select null as has_purchase, has_review, null as has_favorite, count(1) users
  from user_action_flag
  group by 2
union all select has_purchase, null as has_review, null as has_favorite, count(1) users
  from user_action_flag
  group by 1
)

select
-- 0, 1을 문자열로 가공
case has_purchase when 1 then 'purchase' when 0 then 'not purchase' else 'any' end has_purchase,
case has_review when 1 then 'review' when 0 then 'not review' else 'any' end has_review,
case has_favorite when 1 then 'favorite' when 0 then 'not favorite' else 'any' end has_favorite,
users, 100.0*users / nullif(sum(case when has_purchase is null
                  and has_review is null and has_favorite is null
                 then users else 0 end) over(), 0) ratio
from action_venn_diagram
order by has_purchase, has_review, has_favorite
```

- 1과 0을 텍스트 값으로 디코딩했고, 비율 계산

6. Decile 분석을 사용해 사용자를 10단계 그룹으로 나누기

Decile 분석

1. 사용자를 구매 금액이 많은 순부터 정렬
2. 정렬된 사용자 상위부터 10%씩 Decile1 ~ Decile10까지의 그룹으로 할당
3. 각 그룹의 구매 금액 합계를 집계
4. 전체 구매 금액에 대해 각 Decile의 구매 금액 비율(구성비)를 계산
5. 상위에서 누적으로 어느 정도의 비율을 차지하는지 구성비누계를 집계

- 1 ~ 2단계 (ntile 함수는 값 순서대로 N개만큼 그룹을 나누는 함수)

```
with user_purchase_amount as (
select user_id, sum(amount) as purchase_amount
from action_log
where action = 'purchase'
group by 1
)
, users_with_decile as (
select user_id, purchase_amount, ntile(10) over(order by purchase_amount desc) decile
from user_purchase_amount
)

select *
from users_with_decile
```

- 3단계

```
with user_purchase_amount as (
select user_id, sum(amount) as purchase_amount
from action_log
where action = 'purchase'
group by 1
)
, users_with_decile as (
select user_id, purchase_amount, ntile(10) over(order by purchase_amount desc) decile
from user_purchase_amount
)
, decile_with_purchase_amount as (
select decile, sum(purchase_amount) amount
  , avg(purchase_amount) avg_amount
  , sum(sum(purchase_amount)) over(order by decile) cumulative_amount
  , sum(sum(purchase_amount)) over() total_amount
from users_with_decile
group by 1
)

select *
from decile_with_purchase_amount
```

- 4 ~ 5단계

```
with user_purchase_amount as (
select user_id, sum(amount) as purchase_amount
from action_log
where action = 'purchase'
group by 1
)
, users_with_decile as (
select user_id, purchase_amount, ntile(10) over(order by purchase_amount desc) decile
from user_purchase_amount
)
, decile_with_purchase_amount as (
select decile, sum(purchase_amount) amount
  , avg(purchase_amount) avg_amount
  , sum(sum(purchase_amount)) over(order by decile) cumulative_amount
  , sum(sum(purchase_amount)) over() total_amount
from users_with_decile
group by 1
)

select decile, amount, avg_amount
  , 100.0*amount / total_amount total_ratio
  , 100.0*cumulative_amount / total_amount cumulative_ratio
from decile_with_purchase_amount
```

7. RFM 분석으로 사용자를 3가지 관점의 그룹으로 나누기

RFM

- Recency : 최근 구매일
- Frequency : 구매 횟수
- Monetary : 구매 금액 합계

- 사용자별로 RFM 집계

```
with purchase_log as (
select user_id, amount, substring(stamp, 1, 10) dt
from action_log
where action = 'purchase'
)
, user_rfm as (
select user_id, max(dt) recent_date,
  current_date - max(dt::date) recency
  , count(dt) frequency
  , sum(amount) monetary
from purchase_log
group by 1
)

select *
from user_rfm
```

- 사용자들의 RFM 값 별로 RFM 랭크를 정의

랭크	R : 최근 구매일	F : 누계 구매 횟수	M : 누계 구매 금액
5	14일 이내	20회 이상	300만원 이상
4	28일 이내	10회 이상	100만원 이상
3	60일 이내	5회 이상	30만원 이상
2	90일 이내	2회 이상	5만원 이상
1	91일 이내	1회	5만원 미만

```
with purchase_log as (
select user_id, amount, substring(stamp, 1, 10) dt
from action_log
where action = 'purchase'
)
, user_rfm as (
select user_id, max(dt) recent_date,
  current_date - max(dt::date) recency
  , count(dt) frequency
  , sum(amount) monetary
from purchase_log
group by 1
)
, user_rfm_rank as (
select user_id, recent_date, recency, frequency, monetary
  , case when recency < 14 then 5
    when recency < 28 then 4
    when recency < 60 then 3
    when recency < 90 then 2
    else 1 end r
  , case when frequency >= 20 then 5
    when frequency >= 10 then 4
    when frequency >= 5 then 3
```

```
    when frequency >= 2 then 2
    else 1 end f
  , case when monetary >= 300000 then 5
    when monetary >= 100000 then 4
    when monetary >= 30000 then 3
    when monetary >= 5000 then 2
    else 1 end m
from user_rfm
)

select *
from user_rfm_rank
```

- 각 그룹의 사람 수를 계산

```
with purchase_log as (
select user_id, amount, substring(stamp, 1, 10) dt
from action_log
where action = 'purchase'
)
, user_rfm as (
select user_id, max(dt) recent_date,
  current_date - max(dt::date) recency
  , count(dt) frequency
  , sum(amount) monetary
from purchase_log
group by 1
)
, user_rfm_rank as (
select user_id, recent_date, recency, frequency, monetary
  , case when recency < 14 then 5
    when recency < 28 then 4
    when recency < 60 then 3
    when recency < 90 then 2
    else 1 end r
  , case when frequency >= 20 then 5
    when frequency >= 10 then 4
    when frequency >= 5 then 3
    when frequency >= 2 then 2
    else 1 end f
  , case when monetary >= 300000 then 5
    when monetary >= 100000 then 4
    when monetary >= 30000 then 3
    when monetary >= 5000 then 2
    else 1 end m
from user_rfm
)
, mst_rfm_index as (
select 1 as rfm_index
union all select 2 as rfm_index
union all select 3 as rfm_index
union all select 4 as rfm_index
union all select 5 as rfm_index
)
, rfm_flag as (
select m.rfm_index,
  case when m.rfm_index = r.r then 1 else 0 end r_flag
  , case when m.rfm_index = r.f then 1 else 0 end f_flag
  , case when m.rfm_index = r.m then 1 else 0 end m_flag
from mst_rfm_index m
cross join user_rfm_rank r
)

select rfm_index, sum(r_flag) r, sum(f_flag) f, sum(m_flag) m
from rfm_flag
group by 1
order by 1 desc
```

- 사용자를 1차원으로 구분하기

- R + F + M 값을 통합 랭크로 계산

```
with purchase_log as (
select user_id, amount, substring(stamp, 1, 10) dt
from action_log
where action = 'purchase'
)
, user_rfm as (
select user_id, max(dt) recent_date,
  current_date - max(dt::date) recency
  , count(dt) frequency
  , sum(amount) monetary
from purchase_log
group by 1
)
, user_rfm_rank as (
select user_id, recent_date, recency, frequency, monetary
  , case when recency < 14 then 5
    when recency < 28 then 4
    when recency < 60 then 3
    when recency < 90 then 2
    else 1 end r
  , case when frequency >= 20 then 5
    when frequency >= 10 then 4
    when frequency >= 5 then 3
    when frequency >= 2 then 2
    else 1 end f
  , case when monetary >= 300000 then 5
    when monetary >= 100000 then 4
    when monetary >= 30000 then 3
    when monetary >= 5000 then 2
    else 1 end m
from user_rfm
)

select r + f + m total_rank, r, f, m, count(user_id)
from user_rfm_rank
group by r, f, m
order by 1 desc, r desc, f desc, m desc
```

- 2차원으로 사용자 인식하기 (R과 F만 사용해서 집계)

```
with purchase_log as (
select user_id, amount, substring(stamp, 1, 10) dt
from action_log
where action = 'purchase'
)
, user_rfm as (
select user_id, max(dt) recent_date,
  current_date - max(dt::date) recency
  , count(dt) frequency
  , sum(amount) monetary
from purchase_log
group by 1
)
, user_rfm_rank as (
select user_id, recent_date, recency, frequency, monetary
  , case when recency < 14 then 5
    when recency < 28 then 4
    when recency < 60 then 3
    when recency < 90 then 2
    else 1 end r
  , case when frequency >= 20 then 5
    when frequency >= 10 then 4
    when frequency >= 5 then 3
    when frequency >= 2 then 2
    else 1 end f
  , case when monetary >= 300000 then 5
    when monetary >= 100000 then 4
```

```
    when monetary >= 30000 then 3
    when monetary >= 5000 then 2
    else 1 end m
from user_rfm
)

select 'r_'||r r_rank
  , count(case when f = 5 then 1 end) f_5
  , count(case when f = 4 then 1 end) f_4
  , count(case when f = 3 then 1 end) f_3
  , count(case when f = 2 then 1 end) f_2
  , count(case when f = 1 then 1 end) f_1
from user_rfm_rank
group by r
order by 1 desc
```

N X N 행렬로 나타남