6~7강

상태 곽수민

▼ 6강. 여러 개의 값에 대한 조작

새로운 지표 정의하기

- 페이지 뷰 : 페이지가 출력된 횟수
- 방문자 수 : 페이지를 출력한 사용자 수
- $\frac{\text{페이지뷰}}{\text{방문자수}}$: 사용자 한 명 당 방문하는 페이지 수
- 이 외에도 CTR(Click Through Rate), CVR(Conversion Rate) 등등 새로이 정의 가능

1. 문자열 연결하기

- 서울시, 강서구라는 컬럼이 있으면 이 두 컬럼을 연결

```
select user_id,
  concat(pref_name, city_name) as pref_city
from mst_user_location
```

- CONCAT 함수 대신에 || 연산자로 붙일 수도 있음(파이썬의 문자열 +와 같은 기능)

2. 여러 개의 값 비교하기

- 분기별 매출 증감 판정

```
select year, q1, q2,
  case when q1 < q2 then '+'
  when q1 = q2 then ''
  else '-' end as judge_q1_q2,
  q2 - q1 as diff_q2_q1,
  sign(q2-q1) as sign_q2_q1
from quarterly_sales

-- case when으로 판정해도 되고, SIGN 함수로 판정해도 됨
```

- 연간 최대/최소 4분기 매출 찾기

```
select year, greatest(q1, q2, q3, q4) as greatest_sales
  , least(q1, q2, q3, q4) as least_sales
from quarterly_sales
order by year

-- 이렇게 해도 되고 group by year 해서 min max 함수 조져도 됨
```

- 연간 평균 4분기 매출 계산하기

```
select year, (coalesce(q1, 0) + coalesce(q2, 0) + coalesce(q3, 0) + coalesce(q4,0) / 4 as average
from quarterly_sales
order by year

-- NULL이 아닌 컬럼만의 평균
select year,
  (coalesce(q1,0) + coalesce(q2,0) + coalesce(q3,0) + coalesce(q4,0)
  / (sign(coalesce(q1,0)) + sign(coalesce(q2,0)) + sign(coalesce(q3,0)) + sign(coalesce(q4,0))) as average
from quarterly_sales
order by year
```

3. 2개의 값 비율 계산하기

- 정수 자료형의 데이터 나누기

```
select dt, ad_id,
  cast(clicks as double precision) / impressions as ctr,
  100.0*clicks/impressions as ctr_as_percent
from advertising_stats
where dt = '2017-04-01'
order by dt, ad_id

-- postgreSQL에선 정수를 나누면 소수점이 잘려서 앞에 수를 실수로 전환
```

- 0으로 나누는 것 피하기 (조건문)

```
select dt, ad_id,
  case when impressions > 0 then 100.0*clicks/impressions end as ctr_as_percent
from advertising_stats
order by dt, ad_id
```

4. 두 값의 거리 계산하기

- 숫자 데이터의 절댓값, 제곱 평균 제곱근(RMS) (말만 어렵지 그냥 절대거리)

```
select abs(x1 - x2)
from location_1d
```

- 유클리드 거리

```
select sqrt((x1-x2)^2 + (y1-y2)^2)
from location_2d

-- 제곱 연산자가 없으면 power(x1-x2, 2)로 제곱을 표현
```

5. 날짜/시간 계산하기

- 회원 등록 시간으로부터 1시간 뒤, 30분 전의 시간과 등록일의 다음날과 한달 전의 날짜를 계산

```
select user_id,
  register_stamp::timestamp,
  register_stamp::timestamp + interval '1 hour' as after_1hour,
  register_stamp::timestamp - interval '30 minutes' as before_30minutes,
```

```
  register_stamp::date as register_date,
  (register_stamp::date + interval '1 day')::date as after_1day,
  (register_stamp::date - interval '1 month')::date as before_1month
from mst_users_with_dates

-- interval 'N hour/minutes/second/day/month/year' 로 계산이 가능하다
-- dateadd('month', -3, time)
```

- 날짜 데이터의 차이 계산

```
select user_id,
  current_date as today,
  register_stamp::date as register_date,
  current_date - register_stamp::date as diff_days
from mst_users_with_dates

-- datediff 함수를 쓸 수도 있다!
```

- 사용자의 생년월일로 나이를 계산

```
select user_id,
  current_date as today,
  register_stamp::date as register_date,
  birth_date::date as birth_date,
  extract(year from age(birth_date::date)) as current_age,
  extract(year from age(register_stamp::date, birth_date::date)) as register_age
from mst_users_with_dates

-- age 함수로 기준시점에서부터의 나이를 계산. 만 나이를 계산함.
-- datediff(year, birth_date::date, current_date)처럼 연 차이를 계산할 수도 있긴한데, 이 경우 한국식 나이로 계산됨

-- 같은 방식으로는 floor((20160228 - 20000229) / 10000) 으로 숫자로 바꿔서 할 수도 있음
select user_id,
  substring(register_stamp, 1, 10) as register_date,
  birth_date,
  floor((cast(replace(substring(register_stamp, 1, 10), '-', '') as integer)
     - cast(replace(birth_date, '-', '') as integer)) / 10000 ) as register_age
from mst_users_with_dates
```

6. IP 주소 다루기

- IP 주소 자료형 활용하기 (inet 자료형)

```
select
  cast('127.0.0.1' as inet) < cast('127.0.0.2' as inet) as lt,
  cast('127.0.0.1' as inet) > cast('127.0.0.2' as inet) as gt
```

- IP 주소를 정수 자료형으로 변환하기

```
select ip,
  cast(split_part(ip, '.', 1) as integer) as ip_part1,
  cast(split_part(ip, '.', 2) as integer) as ip_part2,
  cast(split_part(ip, '.', 3) as integer) as ip_part3,
  cast(split_part(ip, '.', 4) as integer) as ip_part4,

  cast(split_part(ip, '.', 1) as integer) * 2^24 +
  cast(split_part(ip, '.', 2) as integer) * 2^16 +
  cast(split_part(ip, '.', 3) as integer) * 2^8 +
```

```
  cast(split_part(ip, '.', 4) as integer) * 2^0 as ip_integer
from (select '192.168.0.1' as ip) as t

-- ip는 8의 배수만큼 곱해서 더해준 뒤 대소비교를 함
```

- IP 주소를 0으로 메우기

```
select ip,
  lpad(split_part(ip, '.', 1), 3, '0')||
  lpad(split_part(ip, '.', 2), 3, '0')||
  lpad(split_part(ip, '.', 3), 3, '0')||
  lpad(split_part(ip, '.', 4), 3, '0')
from (select '192.168.0.1' as ip) as t

-- 3자리가 채워지지 않은 경우 00으로 빈칸을 채우기. lpad 함수를 통해 패딩 추가
```

▼ 7강. 하나의 테이블에 대한 조작

데이터 집약

- 레코드의 수(COUNT), 저장된 값의 합계(SUM), 평균(AVG), 최대(MAX), 최소(MIN)를 계산해주는 함수 존재

1. 그룹의 특징 잡기

- 테이블 전체의 특징량(통계량) 계산하기

```
select count(*) as total_count,
  count(distinct user_id) as user_count,
  count(distinct product_id) as product_count,
  sum(score) as sum,
  avg(score) as avg,
  max(score) as max,
  min(score) as min
from review
```

- 그루핑한 데이터의 특징량 계산하기

```
select user_id,
  count(*) as total_count,
  count(distinct product_id) as product_count,
  sum(score) as sum,
  avg(score) as avg,
  max(score) as max,
  min(score) as min
from review
```

- 집약 함수를 적용한 값과 집약 전의 값을 동시에 다루기

```
select user_id, product_id, score,
  avg(score) over() as avg_score,  -- 전체평점
  avg(score) over(partition by user_id) as user_avg_score,  -- 유저 평점
```

```
  score - avg(score) over(partition by user_id) as user_avg_score_diff
from review
```

2. 그룹 내부의 순서

- ORDER BY 구문으로 순서 정의하기

```
select product_id, score,
  row_number() over(order by score desc) as row,
  rank() over(order by score desc) as rank,
  dense_rank() over(order by score desc) as dense_rank, -- 중복순위 있으면 건너뜀
  lag(product_id) over(order by score desc) as lag1,  -- 현재 행보다 한 칸 앞의 값
  lag(product_id, 2) over(order by score desc) as lag2,  -- 두 칸 앞의 값
  lead(product_id) over(order by score desc) as lead1,   -- 한 칸 뒤의 값
  lead(product_id, 2) over(order by score desc) as lead2   -- 두 칸 뒤의 값
from popular_products
order by row
```

- ORDER BY 구문과 집약 함수 조합

```
select product_id, score,
  row_number() over(order by score desc) as row,

  -- unbounded preceding(처음부터) 현재 행까지 누적합계
  sum(score) over(order by score desc
                rows between unbounded preceding and current row) as cum_score,
  -- 1 preceding : 1개 이전의 값, 1 following : 1개 이후의 값
  avg(score) over(order by score desc
                  rows between 1 preceding and 1 following) as local_avg,

  first_value(product_id) over(order by score desc
            rows between unbounded preceding and unbounded following) as first_value,
  last_value(product_id) over(order by score desc
            rows between unbounded preceding and unbounded following) as last_value
from popular_products
order by row
```

- 윈도 프레임 지정별 상품 ID 집약

```
select product_id,
  row_number() over(order by score desc) as row,
  -- 범위에 들어가는 모든 값을 array로 저장
  array_agg(product_id) over(order by score desc
                 rows between unbounded preceding and unbounded following)
                 as whole_agg,
  array_agg(product_id) over(order by score desc
                rows between unbounded preceding and current row) as cum_agg,
  array_agg(product_id) over(order by score desc
                rows between 1 preceding and 1 following) as local_agg
from popular_products
where category = 'action'
order by row
```

- PARTITION BY와 ORDER BY 조합하기

```
select category, product_id, score,
  row_number() over(partition by category order by score desc) as row,
  rank() over(partition by category order by score desc) as rank,
```

```
  dense_rank() over(partition by category order by score desc) as dense_rank
from popular_products
order by category, row
-- 그룹 내 순위
```

- 각 카테고리의 상위 n개 추출하기

```
with pp as (
select category, product_id, score,
  row_number() over(partition by category order by score desc) as rank
from popular_products)

select *
from pp
where rank <= 2
order by category, rank

-- with OO as (서브쿼리) 로 가독성 상승
```

- 카테고리별 순위 최상위 상품 추출

```
select distinct category,
  first_value(product_id) over(partition by category order by score desc
            rows between unbounded preceding and unbounded following) as product_id
from popular_products
```

3. 세로 기반 데이터를 가로 기반으로 변환하기

- 행을 열로 변환하기

```
select dt,
  max(case when indicator = 'impressions' then val end) as impressions,
  max(case when indicator = 'sessions' then val end) as sessions,
  max(case when indicator = 'users' then val end) as users
from daily_kpi
group by dt
order by dt

-- indicator를 열로 변환
```

- 행을 쉼표로 구분한 문자열로 집약하기

```
select purchase_id,
  -- 문자열로 더하는데 각 pid별로 쉼표로 구분
  string_agg(product_id, ',') as product_ids,
  sum(price) as amount
from purchase_detail_log
group by 1
order by 1
```

4. 가로 기반 데이터를 세로 기반으로 변경하기

- 열로 표현된 값을 행으로 변경하기

```
select q.year,
  case when p.idx = 1 then 'q1'
  when p.idx = 2 then 'q2'
  when p.idx = 3 then 'q3'
  when p.idx = 4 then 'q4'
  end as quarter,
  case when p.idx = 1 then q.q1
  when p.idx = 2 then q.q2
  when p.idx = 3 then q.q3
  when p.idx = 4 then q.q4
  end as sales
from quarterly_sales as q
cross join -- 행으로 전개하고 싶은 열의 수만큼 순번 테이블 만들기
(select 1 as idx
union all select 2 as idx
union all select 3 as idx
union all select 4 as idx) as p
```

- 임의의 길이를 가진 배열을 행으로 전개하기

```
select unnest(array['A001', 'A002', 'A003']) as product_id

-- unnest 함수로 array를 행으로 바꿀 수 있음
```

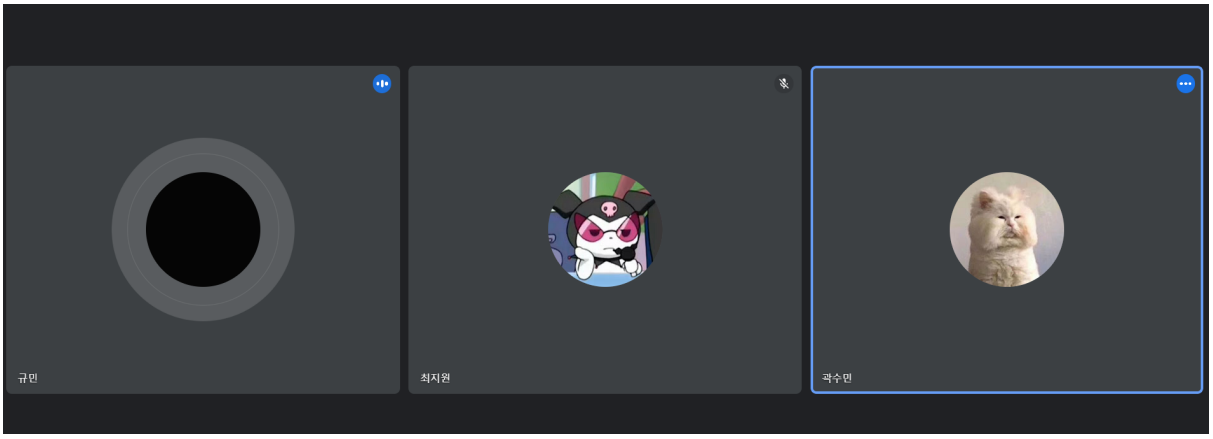
- 쉼표로 구분된 문자열 데이터를 행으로 전개하기

```
select purchase_id, product_id
from purchase_log as p
cross join unnest(string_to_array(product_ids, ',')) as product_id

-- postgreSQL에선 함수 하나로 간단하게 가능
select purchase_id,
    regexp_split_to_table(product_ids, ',') as product_id
from purchase_log
```

Redshift 버전 (배열 자료형이 지원되지 않을 때)

- 피벗 테이블을 사용해 문자열을 행으로 전개

```
select l.purchase_id, l.product_ids, p.idx,
  split_part(l.product_ids, ',', p.idx) as product_id
from purchase_log as l
join (select 1 as idx union all select 2 as idx union all select 3 as idx) as p
on p.idx <= (1 + char_length(l.product_ids) - char_length(replace(product_ids, ',', '')))

-- join on __ 문에서 쓴 수식으로 문자열 안에 들어있는 데이터 개수 출력(쉼표 개수 카운트 +1)
```


규민
최지원
곽수민