SQL 3장

데이터 가공을 위한 SQL

5강
하나의 값 조작하기

데이터 5-1 사용자 마스터(mst_users) 테이블

user_id	register_date	register_device
U001	2016-08-26	1
U002	2016-08-26	2
U003	2016-08-27	3

코드값을 레이블로 변경하기, CASE

-> 조건에 따라서 값을 지정

CASE 뒤에 2HEN〈조건식〉THEN〈조건을 만족할 때의 값〉' 을 나열

만약 조건식에 해당하는 경우가 없다면 NULL이 되지만,
마지막에 ,ELSE〈값〉' 형태를 사용해서 디폴트 값을 별도로 지정

PostgreSQL Hive Redshift BigQuery SparkSQL

```
SELECT
   user_id
 , CASE
     WHEN register_device = 1 THEN '데스크톱'
     WHEN register_device = 2 THEN '스마트폰'
     WHEN register_device = 3 THEN '애플리케이션'
     -- 디폴트 값을 지정할 경우 ELSE 구문을 사용합니다.
     -- ELSE ''
   END AS device_name
FROM mst_users
;
```

▼

실행결과

user_id	device_name
U001	데스크톱
U002	스마트폰
U003	애플리케이션

5강
하나의 값 조작하기

URL에서 요소 추출

페이지 단위로 집계하면 밀도가 너무 작아 복잡해지므로,
호스트 단위로 집계

데이터 5-2 접근 로그(access_log) 테이블

```
-[ RECORD 1 ]-------------------------------------------------
stamp        | 2016-08-26 12:02:00
referrer     | http://www.other.com/path1/index.php?k1=v1&k2=v2#Ref1
url          | http://www.example.com/video/detail?id=001
-[ RECORD 2 ]-------------------------------------------------
stamp        | 2016-08-26 12:02:01
referrer     | http://www.other.net/path1/index.php?k1=v1&k2=v2#Ref1
url          | http://www.example.com/video#ref
-[ RECORD 3 ]-------------------------------------------------
stamp        | 2016-08-26 12:02:01
referrer     | https://www.other.com/
url          | http://www.example.com/book/detail?id=002
```

PostgreSQL Hive Redshift BigQuery SparkSQL

```
SELECT
   stamp
  -- referrer의 호스트 이름 부분 추출하기
  -- ■ PostgreSQL의 경우 substring 함수와 정규 표현식 사용하기
, substring(referrer from 'https?://([^/]*)') AS referrer_host
  -- ■ Redshift의 경우 정규 표현식에 그룹을 사용할 수 없으므로
  -- , regexp_substr 함수과 regexp_replace 함수를 조합해서 사용하기
  -- , regexp_replace(regexp_substr(referrer, 'https?://[^/]*'), 'https?://', '')
  -- AS referrer_host
  -- ■ Hive, SparkSQL의 경우 parse_url 함수로 호스트 이름 추출하기
  -- , parse_url(referrer, 'HOST') AS referrer_host
  -- ■ BigQuery의 경우는 host 함수 사용하기
  -- , host(referrer) AS referrer_host
FROM access_log
;
```

▼

실행결과

```
       stamp          | referrer_host
----------------------+---------------
 2016-08-26 12:02:00  | www.other.com
 2016-08-26 12:02:01  | www.other.net
 2016-08-26 12:02:01  | www.other.com
```

5강
하나의 값 조작하기

URL에서 요소 추출

URL 경로와 GET 매개변수에 있는 특정 키 값을 추출하는 쿼리

데이터 5-2 접근 로그(access_log) 테이블

```
-[ RECORD 1 ]-------------------------------------------------
stamp        | 2016-08-26 12:02:00
referrer     | http://www.other.com/path1/index.php?k1=v1&k2=v2#Ref1
url          | http://www.example.com/video/detail?id=001
-[ RECORD 2 ]-------------------------------------------------
stamp        | 2016-08-26 12:02:01
referrer     | http://www.other.net/path1/index.php?k1=v1&k2=v2#Ref1
url          | http://www.example.com/video#ref
-[ RECORD 3 ]-------------------------------------------------
stamp        | 2016-08-26 12:02:01
referrer     | https://www.other.com/
url          | http://www.example.com/book/detail?id=002
```

```
SELECT
    stamp
  , url
  -- URL 경로 또는 GET 매개변수의 id 추출하기
  -- ■ PostgreSQL의 경우 substring 함수와 정규 표현식 사용하기
  , substring(url from '//[^/]+([^?#]+)') AS path
  , substring(url from 'id=([^&]*)') AS id
  -- ■ Redshift의 경우 regexp_substr 함수와 regexp_replace 함수를 조합해서 사용하기
  -- , regexp_replace(regexp_substr(url, '//[^/]+[^?#]+'), '//[^/]+', '') AS path
  -- , regexp_replace(regexp_substr(url, 'id=[^&]*'), 'id=', '') AS id
  -- ■ BigQuery의 경우 정규 표현식과 regexp_extract 함수 사용하기
  -- , regexp_extract(url, '//[^/]+([^?#]+)') AS path
  -- , regexp_extract(url, 'id=([^&]*)') AS id
  -- ■ Hive 、SparkSQL의 경우 parse_url 함수로 URL 경로 부분 또는 쿼리 매개변수 부분의 값 추출하기
  -- , parse_url(url, 'PATH') AS path
  -- , parse_url(url, 'QUERY', 'id') AS id
FROM access_log
;
```

▼

실행결과

```
      stamp         |                 url                  |    path      | id
--------------------+--------------------------------------+--------------+----
 2016-08-26 12:02:00 | http://www.example.com/video/detail?id=001 | /video/detail | 001
 2016-08-26 12:02:01 | http://www.example.com/video#ref           | /video        |
 2016-08-26 12:02:01 | http://www.example.com/book/detail?id=002  | /book/detail  | 002
```

5강
하나의 값 조작하기

URL에서 요소 추출 문자열을 배열로 분해, split

영어 문장을 공백으로 분할해서 하나하나의 단어로 구분하는 경우,
쉼표로 연결된 데이터를 잘라 하나하나 의 값을 추출하는 경우

URL 경로를 /로 분할

데이터 5-2 접근 로그(access_log) 테이블

```
-[ RECORD 1 ]-------------------------------------------------
stamp     | 2016-08-26 12:02:00
referrer  | http://www.other.com/path1/index.php?k1=v1&k2=v2#Ref1
url       | http://www.example.com/video/detail?id=001
-[ RECORD 2 ]-------------------------------------------------
stamp     | 2016-08-26 12:02:01
referrer  | http://www.other.net/path1/index.php?k1=v1&k2=v2#Ref1
url       | http://www.example.com/video#ref
-[ RECORD 3 ]-------------------------------------------------
stamp     | 2016-08-26 12:02:01
referrer  | https://www.other.com/
url       | http://www.example.com/book/detail?id=002
```

PostgreSQL | Hive | Redshift | BigQuery | SparkSQL

```
SELECT
    stamp
  , url
  -- 경로를 슬래시로 잘라 배열로 분할하기
  -- 경로가 반드시 슬래시로 시작하므로 2번째 요소가 마지막 계층
  -- ■ PostgreSQL의 경우 split_part로 n번째 요소 추출하기
  , split_part(substring(url from '//[^/]+([^?#]+)'), '/', 2) AS path1
  , split_part(substring(url from '//[^/]+([^?#]+)'), '/', 3) AS path2
  -- ■ Redshift도 split_part로 n번째 요소 추출하기
  -- , split_part(regexp_replace(
  --     regexp_substr(url, '//[^/]+[^?#]+'), '//[^/]+', ''), '/', 2) AS path1
  -- , split_part(regexp_replace(
  --     regexp_substr(url, '//[^/]+[^?#]+'), '//[^/]+', ''), '/', 3) AS path2
  -- ■ BigQuery의 경우 split 함수를 사용해 배열로 자름(별도 인덱스 지정 필요)
  -- , split(regexp_extract(url, '//[^/]+([^?#]+)'), '/')[SAFE_ORDINAL(2)] AS path1
  -- , split(regexp_extract(url, '//[^/]+([^?#]+)'), '/')[SAFE_ORDINAL(3)] AS path2
  -- ■ Hive, SparkSQL도 split 함수를 사용해 배열로 자름
  --  다만 배열의 인덱스가 0부터 시작하므로 주의하기
  -- , split(parse_url(url, 'PATH'), '/')[1] AS path1
  -- , split(parse_url(url, 'PATH'), '/')[2] AS path2
FROM access_log
;
```

실행결과

```
       stamp        |                      url                       | path1  | path2
--------------------+------------------------------------------------+--------+--------
 2016-08-26 12:02:00 | http://www.example.com/video/detail?id=001     | video  | detail
 2016-08-26 12:02:01 | http://www.example.com/video#ref               | video  |
 2016-08-26 12:02:01 | http://www.example.com/book/detail?id=002      | book   | detail
```

5강
하나의 값 조작하기

날짜와 타임스탬프 다루기

현재 날짜와 타임스탬프 추출하기, CURRENT_TIMESTAMP

PostgreSQL | Hive | Redshift | BigQuery | SparkSQL

```
SELECT
 -- ■ PostgreSQL, Hive, BigQuery의 경우
 -- CURRENT_DATE 상수와 CURRENT_TIMESTAMP 상수 사용하기
   CURRENT_DATE AS dt
 , CURRENT_TIMESTAMP AS stamp
 -- ■ Hive, BigQuery, SparkSQL의 경우
 -- CURRENT_DATE() 함수와 CURRENT_TIMESTAMP() 함수 사용하기
 --   CURRENT_DATE() AS dt
 -- , CURRENT_TIMESTAMP() AS stamp
 -- ■ Redshift의 경우 현재 날짜는 CURRENT_DATE, 현재 타임 스탬프는 GETDATE() 사용하기
    CURRENT_DATE AS dt
  , GETDATE() AS stamp
 -- ■ PostgreSQL의 경우 CURRENT_TIMESTAMP는 타임존이 적용된 타임스탬프
 -- 타임존을 적용하고 싶지 않으면 LOCALTIMESTAMP 사용하기
 -- , LOCALTIMESTAMP AS stamp
;
```

▼

실행결과

```
    dt      |          stamp
------------+----------------------------
 2017-01-30 | 2017-01-30 18:42:57.584993
```

5강
하나의 값 조작하기

날짜와 타임스탬프 다루기

지정한 값의 날짜/시각 데이터 추출하기

CAST(데이터형식을 다른 데이터형식으로 변환하는 함수)

현재 시각이 아니라 문자열로 지정한 날짜와 시각을 기반으로
날짜 자료형과 타임스탬프 자료형의 데이터를 만드는 경우

PostgreSQL | Hive | Redshift | BigQuery | SparkSQL

```
SELECT
 -- ■ PostgreSQL, Hive, BigQuery의 경우
 -- CURRENT_DATE 상수와 CURRENT_TIMESTAMP 상수 사용하기
   CURRENT_DATE AS dt
 , CURRENT_TIMESTAMP AS stamp
 -- ■ Hive, BigQuery, SparkSQL의 경우
 -- CURRENT_DATE() 함수와 CURRENT_TIMESTAMP() 함수 사용하기
 --   CURRENT_DATE() AS dt
 -- , CURRENT_TIMESTAMP() AS stamp
 -- ■ Redshift의 경우 현재 날짜는 CURRENT_DATE, 현재 타임 스탬프는 GETDATE() 사용하기
    CURRENT_DATE AS dt
  , GETDATE() AS stamp
 -- ■ PostgreSQL의 경우 CURRENT_TIMESTAMP는 타임존이 적용된 타임스탬프
 -- 타임존을 적용하고 싶지 않으면 LOCALTIMESTAMP 사용하기
 -- , LOCALTIMESTAMP AS stamp
;
```

▼

실행결과

```
     dt      |           stamp
-------------+----------------------------
 2017-01-30  | 2017-01-30 18:42:57.584993
```

5강
하나의 값 조작하기

날짜/시각에서 특정 필드 추출하기, EXTRACT

타임스탬프 자료형의 데이터에서 연 월 일 등을 추출하는 쿼리

PostgreSQL Hive Redshift BigQuery SparkSQL

```
SELECT
   stamp
  -- ■ PostgreSQL, Redshift, BigQuery의 경우 EXTRACT 함수 사용하기
  , EXTRACT(YEAR  FROM stamp) AS year
  , EXTRACT(MONTH FROM stamp) AS month
  , EXTRACT(DAY   FROM stamp) AS day
  , EXTRACT(HOUR  FROM stamp) AS hour

  -- ■ Hive, SparkSQL의 경우 EXTRACT 함수 대신 다음과 같은 함수 사용하기
  -- , YEAR(stamp)  AS year
  -- , MONTH(stamp) AS month
  -- , DAY(stamp)   AS day
  -- , HOUR(stamp)  AS hour
FROM
  (SELECT CAST('2016-01-30 12:00:00' AS timestamp) AS stamp) AS t
;
```

▼

실행결과

```
       stamp          | year  | month  | day  | hour
----------------------+-------+--------+------+------
 2016-01-30 12:00:00  | 2016  |    1   | 30   |  12
```

5강
하나의 값 조작하기

날짜/시각에서 특정 필드 추출하기, substring

substring 함수를 이용해서 문자열 추출

PostgreSQL Hive Redshift BigQuery SparkSQL

```
SELECT
   stamp
  -- ■ PostgreSQL, Hive, Redshift, SparkSQL에서는 substring 함수 사용하기
  , substring(stamp,  1, 4) AS year
  , substring(stamp,  6, 2) AS month
  , substring(stamp,  9, 2) AS day
  , substring(stamp, 12, 2) AS hour
  -- 연과 월을 함께 추출하기
  , substring(stamp,  1, 7) AS year_month

  -- ■ PostgreSQL, Hive, BigQuery, SparkSQL에서는 substr 함수 사용하기
  -- , substr(stamp,  1, 4) AS year
  -- , substr(stamp,  6, 2) AS month
  -- , substr(stamp,  9, 2) AS day
  -- , substr(stamp, 12, 2) AS hour
  -- , substr(stamp,  1, 7) AS year_month
FROM
  -- ■ PostgreSQL, Redshift의 경우 문자열 자료형으로 text 사용하기
  (SELECT CAST('2016-01-30 12:00:00' AS text) AS stamp) AS t
  -- ■ Hive, BigQuery, SparkSQL의 경우 문자열 자료형으로 string 사용하기
  -- (SELECT CAST('2016-01-30 12:00:00' AS string) AS stamp) AS t
;
```

▼

실행결과

```
      stamp          | year | month | day | hour | year_month
---------------------+------+-------+-----+------+------------
 2016-01-30 12:00:00 | 2016 | 01    | 30  | 12   | 2016-01
```

5강
하나의 값 조작하기

결손 값을 디폴트 값으로 대치하기, COALESCE

NULL과 문 자열을 결합하면 NULL이 되며,
NULL과 숫자를 사칙 연산해도 NULL

데이터 5-3 쿠폰 사용 여부가 함께 있는 구매 로그(purchase_log_with_coupon) 테이블

Purchase_id	amount	coupon
10001	3280	null
10002	4650	500
10003	3870	null

코드 5-9 구매액에서 할인 쿠폰 값을 제외한 매출 금액을 구하는 쿼리

PostgreSQL Hive Redshift BigQuery SparkSQL

```
SELECT
    purchase_id
  , amount
  , coupon
  , amount - coupon AS discount_amount1
  , amount - COALESCE(coupon, 0) AS discount_amount2
FROM
    purchase_log_with_coupon
;
```

▼

실행결과

purchase_id	amount	coupon	discount_amount1	discount_amount2
10001	3280	null	null	3280
10002	4650	500	4150	4150
10003	3870	null	null	3870

6강

여러 개의 값에 대한 조작

새로운 지표 정의하기

〈페이지 뷰〉+〈방문자 수〉를 구하면, '

사용자 한 명이 페이지를 몇 번이나 방문 했는가?'라는 새로운 지표를 계산

문자열 연결하기, CONCAT

tJOOI. 서울시, 강서구이라는 데이터가 있으면

'서울시강서구'라는 형태로 문자열 연결

데이터 6-1 사용자의 주소 정보(mst_user_location) 테이블

```
 user_id |   pref_name      | city_name
---------+------------------+-----------
 U001    | 서울특별시       | 강서구
 U002    | 경기도수원시     | 장안구
 U003    | 제주특별자치도   | 서귀포시
```

코드 6-1 문자열을 연결하는 쿼리[3]

PostgreSQL Hive Redshift BigQuery SparkSQL

```
SELECT
   user_id
   -- ■ PostgreSQL, Hive, Redshift, BigQuery, SparkSQL 모두 CONCAT 함수 사용 가능
   --   다만 Redshift의 경우는 매개변수를 2개 밖에 못 받음
  , CONCAT(pref_name, city_name) AS pref_city
   -- ■ PostgreSQL, Redshift의 경우는 || 연산자도 사용 가능
   -- , pref_name || city_name AS pref_city
FROM
  mst_user_location
;
```

실행결과

```
 user_id |       pref_city
---------+------------------------
 U001    | 서울특별시강서구
 U002    | 경기도수원시장안구
 U003    | 제주특별자치도서귀포시
```

6강
여러 개의 값에 대한 조작

여러 개의 값비교하기 - 분기별 매출 증감 판정

CASE -> q1 < q2 : + / q1 < q2 : -

diff_q2_ql -> 값의 차이

SIGN -> 양수: 1, 0: 0, 음수: -1

데이터 6-2 4분기 매출(quarterly_sales) 테이블

year	q1	q2	q3	q4
2015	82000	83000	78000	83000
2016	85000	85000	80000	81000
2017	92000	81000		

코드 6-2 q1, q2 컬럼을 비교하는 쿼리

PostgreSQL | Hive | Redshift | BigQuery | SparkSQL

```
SELECT
    year
  , q1
  , q2
    -- Q1과 Q2의 매출 변화 평가하기
  , CASE
      WHEN q1 < q2 THEN '+'
      WHEN q1 = q2 THEN ' '
      ELSE '-'
    END AS judge_q1_q2
    -- Q1과 Q2의 매출액의 차이 계산하기
  , q2 - q1 AS diff_q2_q1
    -- Q1과 Q2의 매출 변화를 1, 0, -1로 표현하기
  , SIGN(q2 - q1) AS sign_q2_q1
FROM
  quarterly_sales
ORDER BY
  year
;
```

실행결과

year	q1	q2	judge_q1_q2	diff_q2_q1	sign_q2_q1
2015	82000	83000	+	1000	1
2016	85000	85000		0	0
2017	92000	81000	-	-11000	-1

6강
여러 개의 값에 대한 조작

여러 개의 값비교하기
- 연간 최대/최소 4분기 매출 찾기

greatest: 최댓값 / least: 최솟값

데이터 6-2 4분기 매출(quarterly_sales) 테이블

year	q1	q2	q3	q4
2015	82000	83000	78000	83000
2016	85000	85000	80000	81000
2017	92000	81000		

코드 6-3 연간 최대/최소 4분기 매출을 찾는 쿼리

PostgreSQL | Hive | Redshift | BigQuery | SparkSQL

```
SELECT
    year
    -- Q1~Q4의 최대 매출 구하기
  , greatest(q1, q2, q3, q4) AS greatest_sales
    -- Q1~Q4 최소 매출 구하기
  , least(q1, q2, q3, q4)    AS least_sales
FROM
  quarterly_sales
ORDER BY
  year
;
```

▼

실행결과

year	greatest_sales	least_sales
2015	83000	78000
2016	85000	80000
2017	92000	81000

6강
여러 개의 값에 대한 조작

여러 개의 값비교하기
- 연간 평균 4분기 매출

평균: q1~q4의 합/4

데이터 6-2 4분기 매출(quarterly_sales) 테이블

year	q1	q2	q3	q4
2015	82000	83000	78000	83000
2016	85000	85000	80000	81000
2017	92000	81000		

코드 6-4 단순한 연산으로 평균 4분기 매출을 구하는 쿼리

PostgreSQL | Hive | Redshift | BigQuery | SparkSQL

```
SELECT
    year
  , (q1 + q2 + q3 + q4) / 4 AS average
FROM
  quarterly_sales
ORDER BY
  year
;
```

▼

실행결과

```
 year | average
------+---------
 2015 |   81500
 2016 |   82750
 2017 |
```

6강
여러 개의 값에 대한 조작

여러 개의 값비교하기
- 연간 평균 4분기 매출

평균: q1~q4의 합/4

데이터 6-2 4분기 매출(quarterly_sales) 테이블

year	q1	q2	q3	q4
2015	82000	83000	78000	83000
2016	85000	85000	80000	81000
2017	92000	81000		

코드 6-4 단순한 연산으로 평균 4분기 매출을 구하는 쿼리

PostgreSQL | Hive | Redshift | BigQuery | SparkSQL

```
SELECT
    year
  , (q1 + q2 + q3 + q4) / 4 AS average
FROM
  quarterly_sales
ORDER BY
  year
;
```

▼

실행결과

```
 year | average
------+---------
 2015 |   81500
 2016 |   82750
 2017 |
```

6강
여러 개의 값에 대한 조작

여러 개의 값비교하기
- 연간 평균 4분기 매출

NULL 값을 계산하려면 COALESCE를 통해 변환

데이터 6-2 4분기 매출(quarterly_sales) 테이블

year	q1	q2	q3	q4
2015	82000	83000	78000	83000
2016	85000	85000	80000	81000
2017	92000	81000		

코드 6-5 COALESCE를 사용해 NULL을 0으로 변환하고 평균값을 구하는 쿼리

PostgreSQL Hive Redshift BigQuery SparkSQL

```
SELECT
    year
  , (COALESCE(q1, 0) + COALESCE(q2, 0) + COALESCE(q3, 0) + COALESCE(q4, 0)) / 4
    AS average
FROM
  quarterly_sales
ORDER BY
  year
;
```

▼

실행결과

year	average
2015	81500
2016	82750
2017	43250

6강
여러 개의 값에 대한 조작

2개의 값 비율 계산하기
- 정수 자료형의 데이터 나누기

CTR: 클릭 / 노출 수

퍼센트: ctr * 100

데이터 6-3 광고 통계 정보(advertising_stats) 테이블

dt	ad_id	impressions	clicks
2017-04-01	001	100000	3000
2017-04-01	002	120000	1200
2017-04-01	003	500000	10000
2017-04-02	001	0	0
2017-04-02	002	130000	1400
2017-04-02	003	620000	15000

코드 6-7 정수 자료형의 데이터를 나누는 쿼리

PostgreSQL | Hive | Redshift | BigQuery | SparkSQL

```
SELECT
   dt
, ad_id

   -- ■ Hive, Redshift, BigQuery, SparkSQL의 경우
   --  정수를 나눌 때는 자동적으로 실수로 변환
  , clicks / impressions AS ctr
   -- ■ PostgreSQL의 경우 정수를 나누면 소수점이 잘리므로 명시적으로 자료형 변환

   -- , CAST(clicks AS double precision) / impressions AS ctr
   -- 실수를 상수로 앞에 두고 계산하면 암묵적으로 자료형 변환이 일어남
  , 100.0 * clicks / impressions AS ctr_as_percent
FROM
  advertising_stats
WHERE
  dt = '2017-04-01'
ORDER BY
  dt, ad_id
;
```

▼

실행결과

dt	ad_id	ctr	ctr_as_percent
2017-04-01	001	0.03	3.00
2017-04-01	002	0.01	1.00
2017-04-01	003	0.02	2.00

6강
여러 개의 값에 대한 조작

2개의 값 비율 계산하기
- 0으로 나누는 것 피하기

4/2의 impression은 0 따라서 case를 통해서 처리

데이터 6-3 광고 통계 정보(advertising_stats) 테이블

dt	ad_id	impressions	clicks
2017-04-01	001	100000	3000
2017-04-01	002	120000	1200
2017-04-01	003	500000	10000
2017-04-02	001	0	0
2017-04-02	002	130000	1400
2017-04-02	003	620000	15000

PostgreSQL Hive Redshift BigQuery SparkSQL

```
SELECT
    dt
  , ad_id
    -- CASE 식으로 분모가 0일 경우를 분기해서, 0으로 나누지 않게 만드는 방법
  , CASE
      WHEN impressions > 0 THEN 100.0 * clicks / impressions
    END AS ctr_as_percent_by_case

    -- 분모가 0이라면 NULL로 변환해서, 0으로 나누지 않게 만드는 방법
    -- ■ PostgreSQL, Redshift, BigQuery, SparkSQL의 경우 NULLIF 함수 사용하기
  , 100.0 * clicks / NULLIF(impressions, 0) AS ctr_as_percent_by_null
    -- ■ Hive의 경우 NULLIF 대신 CASE 식 사용하기
    -- , 100.0 * clicks
    --   / CASE WHEN impressions = 0 THEN NULL ELSE impressions END
    --   AS ctr_as_percent_by_null
FROM
  advertising_stats
ORDER BY
  dt, ad_id
;
```

▼

실행결과

dt	ad_id	ctr_as_percent_by_case	ctr_as_percent_by_null
2017-04-01	001	3.00	3.00
2017-04-01	002	1.00	1.00
2017-04-01	003	2.00	2.00
2017-04-02	001		
2017-04-02	002	1.07	1.07
2017-04-02	003	2.41	2.41

6강
여러 개의 값에 대한 조작

두 값의 거리 계산하기

abs: 절댓값 계산

power: 제곱

sort: 제곱근

숫자 데이터의 절댓값, 제곱 평균 제곱근(RMS) 계산하기

데이터 6-4 일차원 위치 정보(location_1d) 테이블

x1	x2
5	10
10	5
-2	4
3	3
0	1

코드 6-9 일차원 데이터의 절댓값과 제곱 평균 제곱근을 계산하는 쿼리

PostgreSQL | Hive | Redshift | BigQuery | SparkSQL

```
SELECT
  abs(x1 - x2) AS abs
  , sqrt(power(x1 - x2, 2)) AS rms
FROM location_1d
;
```

▼

실행결과

abs	rms
5	5
5	5
6	6
0	0
1	1

6강
여러 개의 값에 대한 조작

날짜/시간 계산하기

interval

데이터 6-6 등록 시간과 생일을 포함하는 사용자 마스터(mst_users_with_dates) 테이블

user_id	register_stamp	birth_date
U001	2016-02-28 10:00:00	2000-02-29
U002	2016-02-29 10:00:00	2000-02-29
U003	2016-03-01 10:00:00	2000-02-29

코드 6-11 미래 또는 과거의 날짜/시간을 계산하는 쿼리

PostgreSQL | Hive | Redshift | BigQuery | SparkSQL

```
SELECT
    user_id
  -- ■ PostgreSQL의 경우 interval 자료형의 데이터에 사칙 연산 적용하기
  , register_stamp::timestamp AS register_stamp
  , register_stamp::timestamp + '1 hour'::interval    AS after_1_hour
  , register_stamp::timestamp - '30 minutes'::interval AS before_30_minutes

  , register_stamp::date AS register_date
  , (register_stamp::date + '1 day'::interval  )::date AS after_1_day
  , (register_stamp::date - '1 month'::interval)::date AS before_1_month

  -- ■ Redshift의 경우 dateadd 함수 사용하기
  -- , register_stamp::timestamp AS register_stamp
  -- , dateadd(hour, 1, register_stamp::timestamp) AS after_1_hour
  -- , dateadd(minute, -30, register_stamp::timestamp) AS before_30_minutes
  --
  -- , register_stamp::date AS register_date
  -- , dateadd(day, 1, register_stamp::date) AS after_1_day
  -- , dateadd(month, -1, register_stamp::date) AS before_1_month

  FROM mst_users_with_dates
  ;
```

▼

실행결과

```
-[ RECORD 1 ]-------+--------------------
user_id             | U001
register_stamp      | 2016-02-28 10:00:00
after_1_hour        | 2016-02-28 11:00:00
before_30_minutes   | 2016-02-28 09:30:00
register_date       | 2016-02-28
after_1_day         | 2016-02-29
before_1_month      | 2016-01-28
-[ RECORD 2 ]-------+--------------------
...
```

6강
여러 개의 값에 대한 조작

날짜 데이터들의 차이 계산

date_diff

데이터 6-6 등록 시간과 생일을 포함하는 사용자 마스터(mst_users_with_dates) 테이블

user_id	register_stamp	birth_date
U001	2016-02-28 10:00:00	2000-02-29
U002	2016-02-29 10:00:00	2000-02-29
U003	2016-03-01 10:00:00	2000-02-29

코드 6-12 두 날짜의 차이를 계산하는 쿼리

PostgreSQL Hive Redshift BigQuery SparkSQL

```
SELECT
    user_id

  -- ■ PostgreSQL, Redshift의 경우 날짜 자료형끼리 뺄 수 있음
  , CURRENT_DATE AS today
  , register_stamp::date AS register_date
  , CURRENT_DATE - register_stamp::date AS diff_days

  -- ■ BigQuery의 경우 date_diff 함수 사용하기
  -- , CURRENT_DATE AS today
  -- , date(timestamp(register_stamp)) AS register_date
  -- , date_diff(CURRENT_DATE, date(timestamp(register_stamp)), day) AS diff_days

  -- ■ Hive, SparkSQL의 경우 datediff 함수 사용하기
  -- , CURRENT_DATE() AS today
  -- , to_date(register_stamp) AS register_date
  -- , datediff(CURRENT_DATE(), to_date(register_stamp)) AS diff_days
FROM mst_users_with_dates
;
```

▼

실행결과

user_id	today	register_date	diff_days
U001	2017-02-05	2016-02-28	343
U002	2017-02-05	2016-02-29	342
U003	2017-02-05	2016-03-01	341

6강
여러 개의 값에 대한 조작

사용자의 생년월일로 나이 계산하기

PostgreSQL - age 함수

extract로 년도(year)만 추출

age 함수로 현재 나이리턴 / 특정 날짜 지정 가능

데이터 6-6 등록 시간과 생일을 포함하는 사용자 마스터(mst_users_with_dates) 테이블

user_id	register_stamp	birth_date
U001	2016-02-28 10:00:00	2000-02-29
U002	2016-02-29 10:00:00	2000-02-29
U003	2016-03-01 10:00:00	2000-02-29

PostgreSQL

```
SELECT
   user_id

  -- ■ PostgreSQL의 경우 age 함수와 EXTRACT 함수를 사용해 나이 집계하기
  , 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
;
```

▼

실행결과

user_id	today	register_date	birth_date	current_age	register_age
U001	2017-02-05	2016-02-28	2000-02-29	16	15
U002	2017-02-05	2016-02-29	2000-02-29	16	16
U003	2017-02-05	2016-03-01	2000-02-29	16	16

6강
여러 개의 값에 대한 조작

사용자의 생년월일로 나이 계산하기

PostgreSQL 외

2016년 2월 28일은 '20160228, 로 표현한 뒤 / 10,000

코드 6-15 날짜를 정수로 표현해서 나이를 계산하는 함수

PostgreSQL | Hive | Redshift | BigQuery | SparkSQL

```
-- 생일이 2000년 2월 29일인 사람의 2016년 2월 28일 시점의 나이 계산하기
SELECT floor((20160228 - 20000229) / 10000) AS age;
```

데이터 6-6 등록 시간과 생일을 포함하는 사용자 마스터(mst_users_with_dates) 테이블

user_id	register_stamp	birth_date
U001	2016-02-28 10:00:00	2000-02-29
U002	2016-02-29 10:00:00	2000-02-29
U003	2016-03-01 10:00:00	2000-02-29

코드 6-16 등록 시점과 현재 시점의 나이를 문자열로 계산하는 쿼리

PostgreSQL | Hive | Redshift | BigQuery | SparkSQL

```
SELECT
    user_id
  , substring(register_stamp, 1, 10) AS register_date
  , birth_date
    -- 등록 시점의 나이 계산하기
  , floor(
      ( CAST(replace(substring(register_stamp, 1, 10), '-', '') AS integer)
        - CAST(replace(birth_date, '-', '') AS integer)
      ) / 10000
    ) AS register_age
    -- 현재 시점의 나이 계산하기
  , floor(
      ( CAST(replace(CAST(CURRENT_DATE AS text), '-', '') AS integer)
        - CAST(replace(birth_date, '-', '') AS integer)
      ) / 10000
    ) AS current_age

    -- ■ BigQuery의 경우 text를 string, integer를 int64로 바꾸기
    -- ( CAST(replace(CAST(CURRENT_DATE AS string), '-', '') AS int64)
    --   - CAST(replace(birth_date, '-', '') AS int64)
    -- ) / 10000

    -- ■ Hive, SparkSQL의 경우 replace를 regexp_replace, text를 string,
    --  integer를 int로 바꾸기
    --  SparkSQL의 경우는 추가로 CURRENT_DATE를 CURRENT_DATE()로 바꾸기
    -- ( CAST(regexp_replace(CAST(CURRENT_DATE() AS string), '-', '') AS int)
    --   - CAST(regexp_replace(birth_date, '-', '') AS int)
    -- ) / 10000
FROM mst_users_with_dates
;
```

▼

실행결과

user_id	register_date	birth_date	register_age	current_age
U001	2016-02-28	2000-02-29	15	16
U002	2016-02-29	2000-02-29	16	16
U003	2016-03-01	2000-02-29	16	16

7강
하나의 테이블에 대한조작

그룹의 특징잡기

집약 함수: 여러 레코드를 기반으로 하나의 값을 리턴하는 함수

ex. count sum

데이터 7-1 상품 평가(review) 테이블

user_id	product_id	score
U001	A001	4.0
U001	A002	5.0
U001	A003	5.0
U002	A001	3.0
U002	A002	3.0
U002	A003	4.0
U003	A001	5.0
U003	A002	4.0
U003	A003	4.0

테이블 전체의 특징량 계산하기

COUNT: 지정한 컬럼의 레코드 수를 리턴

DISTINCT: 중복을 제외

SUM: 합게 / 실수 정수형

AVG: 평균 / 실수 정수형

MAX MIN: 최대 최소 /대소 비교가 가 능한 자료형 (숫자, 문자열, 타임스탬프 등)

코드 7-1 집약 함수를 사용해서 테이블 전체의 특징량을 계산하는 쿼리

PostgreSQL Hive Redshift BigQuery SparkSQL

```
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
;
```

실행결과

total_count	user_count	product_count	sum	avg	max	min
9	3	3	37.0	4.11	5.0	3.0

7강
하나의 테이블에 대한 조작

그루핑한 데이터의 특징량 계산하기

GROUP BY: 데이터를 분류할 키를 지 정하고.

그러한 키를 기반으로 데이터를 집약

데이터 7-1 상품 평가(review) 테이블

user_id	product_id	score
U001	A001	4.0
U001	A002	5.0
U001	A003	5.0
U002	A001	3.0
U002	A002	3.0
U002	A003	4.0
U003	A001	5.0
U003	A002	4.0
U003	A003	4.0

코드 7-2 사용자 기반으로 데이터를 분할하고 집약 함수를 적용하는 쿼리

PostgreSQL | Hive | Redshift | BigQuery | SparkSQL

```
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
GROUP BY
  user_id
;
```

▼

실행결과

user_id	total_count	product_count	sum	avg	max	min
U001	3	3	14.0	4.67	5.0	4.0
U002	3	3	10.0	3.33	4.0	3.0
U003	3	3	13.0	4.33	5.0	4.0

7강
하나의 테이블에 대한 조작

집약 함수를 적용한 값과 집약 전의 값을 동시에 다루기

OVER: 구문에 매개 변수를 지정하지 않으면 -> 테이블 전체에 집약 함수를 적용

매개 변수에 PARTITION BY〈컬럼 이름〉을 지정하면
-> 해당 컬럼 값을 기반으로 그룹화하고 집약 함수를 적용

데이터 7-1 상품 평가(review) 테이블

user_id	product_id	score
U001	A001	4.0
U001	A002	5.0
U001	A003	5.0
U002	A001	3.0
U002	A002	3.0
U002	A003	4.0
U003	A001	5.0
U003	A002	4.0
U003	A003	4.0

코드 7-3 윈도 함수를 사용해 집약 함수의 결과와 원래 값을 동시에 다루는 쿼리

PostgreSQL | Hive | Redshift | BigQuery | SparkSQL

```
SELECT
    user_id
  , product_id
    -- 개별 리뷰 점수
  , score
    -- 전체 평균 리뷰 점수
  , AVG(score) OVER() AS avg_score
    -- 사용자의 평균 리뷰 점수
  , AVG(score) OVER(PARTITION BY user_id) AS user_avg_score
    -- 개별 리뷰 점수와 사용자 평균 리뷰 점수의 차이
  , score - AVG(score) OVER(PARTITION BY user_id) AS user_avg_score_diff
FROM
  review
;
```

▼

실행결과

user_id	product_id	score	avg_score	user_avg_score	user_avg_score_diff
U001	A001	4.0	4.11	4.67	-0.67
U001	A002	5.0	4.11	4.67	0.33
U001	A003	5.0	4.11	4.67	0.33
U002	A001	3.0	4.11	3.33	-0.33
U002	A002	3.0	4.11	3.33	-0.33
U002	A003	4.0	4.11	3.33	0.67
U003	A001	5.0	4.11	4.33	0.67
U003	A002	4.0	4.11	4.33	-0.33
U003	A003	4.0	4.11	4.33	-0.33

7강
하나의 테이블에 대한 조작

ORDER BY 구문으로 순서 정의하기

윈도 함수: 테이블 내부에 '윈도'라고 부르는 범위를 정의하고, 해당 범위 내부에 포함된 값을 특정 레코드에서 자 유롭게 사용

ORDER BY: 테이블 내부의 순서

RANK,DENSE_RANK: 순위 번호

RANK 함수는 같은 순위의 레코드 뒤의 순위 번호를 건너뛰고,

DENSE_RANK 함수의 경우 순위 번호를 건너뛰지 않음

LAG,LEAD: 현재 행을 기준으로 앞의 행 또는 뒤의 행의 값을 추출

데이터 7-2 인기 상품(popular_products) 테이블

product_id	category	score
A001	action	94
A002	action	81
A003	action	78
A004	action	64
D001	drama	90
D002	drama	82
D003	drama	78
D004	drama	58

PostgreSQL Hive Redshift BigQuery SparkSQL

```
SELECT
    product_id
  , score

    -- 점수 순서로 유일한 순위를 붙임
  , ROW_NUMBER()          OVER(ORDER BY score DESC) AS row
    -- 같은 순위를 허용해서 순위를 붙임
  , RANK()                OVER(ORDER BY score DESC) AS rank
    -- 같은 순위가 있을 때 같은 순위 다음에 있는 순위를 건너 뛰고 순위를 붙임
  , DENSE_RANK()          OVER(ORDER BY score DESC) AS dense_rank

    -- 현재 행보다 앞에 있는 행의 값 추출하기
  , LAG(product_id)       OVER(ORDER BY score DESC) AS lag1
  , LAG(product_id, 2)    OVER(ORDER BY score DESC) AS lag2

    -- 현재 행보다 뒤에 있는 행의 값 추출하기
  , LEAD(product_id)      OVER(ORDER BY score DESC) AS lead1
  , LEAD(product_id, 2)   OVER(ORDER BY score DESC) AS lead2
FROM popular_products
ORDER BY row
;
```

▼

실행결과

product_id	score	row	rank	dense_rank	lag1	lag2	lead1	lead2
A001	94	1	1	1			D001	D002
D001	90	2	2	2	A001		D002	A002
D002	82	3	3	3	D001	A001	A002	A003
A002	81	4	4	4	D002	D001	A003	D003
A003	78	5	5	5	A002	D002	D003	A004
D003	78	6	5	5	A003	A002	A004	D004
A004	64	7	7	6	D003	A003	D004	
D004	58	8	8	7	A004	D003		

7강
하나의 테이블에 대한 조작

ORDER BY 구문과 집약 함수 조합

ORDER BY 구문과 SUM/AVG 등의 집약 함수를 조합하면,
집약 함수의 적용 범위를 유연 하게 지정할 수 있습

PARTEON BY와도 조합 가능

데이터 7-2 인기 상품(popular_products) 테이블

product_id	category	score
A001	action	94
A002	action	81
A003	action	78
A004	action	64
D001	drama	90
D002	drama	82
D003	drama	78
D004	drama	58

코드 7-5 ORDER BY 구문과 집약 함수를 조합해서 계산하는 쿼리

PostgreSQL | Hive | Redshift | BigQuery | SparkSQL

```
SELECT
    product_id
  , score

    -- 점수 순서로 유일한 순위를 붙임
  , ROW_NUMBER() OVER(ORDER BY score DESC) AS row

  -- 순위 상위부터의 누계 점수 계산하기
  , SUM(score)
      OVER(ORDER BY score DESC
        ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW)
    AS cum_score

    -- 현재 행과 앞 뒤의 행이 가진 값을 기반으로 평균 점수 계산하기
  , AVG(score)
      OVER(ORDER BY score DESC
        ROWS BETWEEN 1 PRECEDING AND 1 FOLLOWING)
    AS local_avg

    -- 순위가 높은 상품 ID 추출하기
  , FIRST_VALUE(product_id)
      OVER(ORDER BY score DESC
        ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING)
    AS first_value

    -- 순위가 낮은 상품 ID 추출하기
  , 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
;
```

▼

실행결과

product_id	score	row	cum_score	local_avg	first_value	last_value
A001	94	1	94	92.00	A001	D004
D001	90	2	184	88.67	A001	D004
D002	82	3	266	84.33	A001	D004
A002	81	4	347	80.33	A001	D004
A003	78	5	425	79.00	A001	D004
D003	78	6	503	73.33	A001	D004
A004	64	7	567	66.67	A001	D004
D004	58	8	625	61.00	A001	D004

7강
하나의 테이블에 대한 조작

세로기반 데이터를 가로 기반으로 변환하기

SQL은 행(레코드) 기반으로 처리하는 것이 기본

최종 출력에서는 데이터를 열로 전개해야 가독성이 높은 경우가 많음

햄을 열로 변환하기

SQL의 열은 고정적

1. GROUP BY: 날짜를 1 개의 레코드로 집약
2. MAX (CASE-): 노출 수', 세션 수', 사용자 수에 해당하는 레코드만을 CASE 식으로 추출하고 MAX 함수로 해당값을 추출

데이터 7-3 날짜별 KPI 데이터(daily_kpi) 테이블

dt	indicator	val
2017-01-01	impressions	1800
2017-01-01	sessions	500
2017-01-01	users	200
2017-01-02	impressions	2000
2017-01-02	sessions	700
2017-01-02	users	250

코드 7-10 행으로 저장된 지표 값을 열로 변환하는 쿼리

PostgreSQL | Hive | Redshift | BigQuery | SparkSQL

```
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
;
```

▼

실행결과

dt	impressions	sessions	users
2017-01-01	1800	500	200
2017-01-02	2000	700	250

7강
하나의 테이블에 대한 조작

행을 쉼표로 구분한 문자열로 집약하기

행을 집약해서 쉼표로 구분된 문자열로 변환

데이터 7-4 구매 상세 로그(purchase_detail_log) 테이블

purchase_id	product_id	price
100001	A001	300
100001	A002	400
100001	A003	200
100002	D001	500
100002	D002	300
100003	A001	300

코드 7-11 행을 집약해서 쉼표로 구분된 문자열로 변환하기

PostgreSQL | Hive | Redshift | BigQuery | SparkSQL

```
SELECT
    purchase_id

    -- 상품 ID를 배열에 집약하고 쉼표로 구분된 문자열로 변환하기
    -- ■ PostgreSQL, BigQuery의 경우는 string_agg 사용하기
    , string_agg(product_id, ',') AS product_ids

    -- ■ Redshift의 경우는 listagg 사용하기
    -- , listagg(product_id, ',') AS product_ids

    -- ■ Hive, SparkSQL의 경우는 collect_list와 concat_ws 사용하기
    -- , concat_ws(',', collect_list(product_id)) AS product_ids
    , SUM(price) AS amount
FROM purchase_detail_log
GROUP BY purchase_id
ORDER BY purchase_id
;
```

실행결과

purchase_id	product_ids	amount
100001	A001,A002,A003	900
100002	D001,D002	800
100003	A001	300

7강
하나의 테이블에 대한 조작

행을 쉼표로 구분한 문자열로 집약하기

행을 집약해서 쉼표로 구분된 문자열로 변환

데이터 7-4 구매 상세 로그(purchase_detail_log) 테이블

purchase_id	product_id	price
100001	A001	300
100001	A002	400
100001	A003	200
100002	D001	500
100002	D002	300
100003	A001	300

코드 7-11 행을 집약해서 쉼표로 구분된 문자열로 변환하기

PostgreSQL | Hive | Redshift | BigQuery | SparkSQL

```
SELECT
    purchase_id

    -- 상품 ID를 배열에 집약하고 쉼표로 구분된 문자열로 변환하기
    -- ■ PostgreSQL, BigQuery의 경우는 string_agg 사용하기
    , string_agg(product_id, ',') AS product_ids

    -- ■ Redshift의 경우는 listagg 사용하기
    -- , listagg(product_id, ',') AS product_ids

    -- ■ Hive, SparkSQL의 경우는 collect_list와 concat_ws 사용하기
    -- , concat_ws(',', collect_list(product_id)) AS product_ids
    , SUM(price) AS amount
FROM purchase_detail_log
GROUP BY purchase_id
ORDER BY purchase_id
;
```

실행결과

purchase_id	product_ids	amount
100001	A001,A002,A003	900
100002	D001,D002	800
100003	A001	300

7강
하나의 테이블에 대한 조작

가로 기반 데이터를 세로기반으로 변환하기

세로 기반 데이터를 가로 기반으로 변환해서 가공이 쉬운 데이터 형식으로 만드는 방법

임의의 길이를 가진 배열을 행으로 전개

데이터의 길이가 확정되지 않은 경우

unnest or explode 함수: 매개 변수로 배열을 받고, 배열을 레코드 분할해서 리턴

데이터 7-6 구매 로그(purchase_log) 테이블

```
 purchase_id  | product_ids
--------------+----------------
      100001  | A001,A002,A003
      100002  | D001,D002
      100003  | A001
```

코드 7-13 테이블 함수를 사용해 배열을 행으로 전개하는 쿼리

PostgreSQL | Hive | BigQuery | SparkSQL

```
-- ■ PostgreSQL의 경우 unnest 함수 사용하기
SELECT unnest(ARRAY['A001','A002','A003']) AS product_id;

-- ■ BigQuery의 경우도 unnest 함수를 사용하지만
--    테이블 함수는 FROM 구문 내부에서만 사용할 수 있음
-- SELECT * FROM unnest(ARRAY['A001','A002','A003']) AS product_id;

-- ■ Hive, SparkSQL의 경우 explode 함수 사용하기
-- SELECT explode(ARRAY('A001','A002','A003')) AS product_id;
```

▼

실행결과

```
 product_id
------------
 A001
 A002
 A003
```

7강
하나의 테이블에 대한 조작

[데이터 7-6]의 구매 로그를 레코드로 전개

FROM 구문 내부에 작성하고.

JOIN 구문을 사용해 원래 테이블과 테이블 함수의 리턴 값을 결합

데이터 7-6 구매 로그(purchase_log) 테이블

```
 purchase_id  | product_ids
--------------+----------------
      100001  | A001,A002,A003
      100002  | D001,D002
      100003  | A001
```

코드 7-14 테이블 함수를 사용해 쉼표로 구분된 문자열 데이터를 행으로 전개하는 쿼리

PostgreSQL Hive BigQuery SparkSQL

```
SELECT
    purchase_id
  , product_id
FROM
    purchase_log AS p
  -- string_to_array 함수로 문자열을 배열로 변환하고, unnest 함수로 테이블로 변환하기
  CROSS JOIN unnest(string_to_array(product_ids, ',')) AS product_id

  -- ■ BigQuery의 경우 문자열 분해에 split 함수 사용하기
  -- CROSS JOIN unnest(split(product_ids, ',')) AS product_id

  -- ■ Hive와 SparkSQK의 경우 LATERAL VIEW explode 사용하기
  -- LATERAL VIEW explode(split(product_ids, ',')) e AS product_id;
```

▼

실행결과

```
 purchase_id  | product_id
--------------+------------
      100001  | A001
      100001  | A002
      100001  | A003
      100002  | D001
      100002  | D002
      100003  | A001
```

8강

여러 개의 테이블 조작하기

여러 개의 테이블을 세로로 결합하기

FUNION ALL: 여러 개의 테이블을 세로로 결합

UNION DISTINCT(또는 UNION): 중복을 제외

데이터 8-1 애플리케이션1의 사용자 마스터 (app1_mst_users) 테이블

user_id	name	email
U001	Sato	sato@example.com
U002	Suzuki	suzuki@example.com

데이터 8-2 애플리케이션2의 사용자 마스터 (app2_mst_users) 테이블

user_id	name	phone
U001	Ito	080-xxxx-xxxx
U002	Tanaka	070-xxxx-xxxx

코드 8-1 UNION ALL 구문을 사용해 테이블을 세로로 결합하는 쿼리

PostgreSQL | Hive | Redshift | BigQuery | SparkSQL

```
  SELECT 'app1' AS app_name, user_id, name, email FROM app1_mst_users
UNION ALL
  SELECT 'app2' AS app_name, user_id, name, NULL AS email FROM app2_mst_users;
```

▼

실행결과

app_name	user_id	name	email
app1	U001	Sato	sato@example.com
app1	U002	Suzuki	suzuki@example.com
app2	U001	Ito	
app2	U002	Tanaka	

8강

여러 개의 테이블 조작하기

여러 개의 테이블을 가로로 정렬하기

JOIN : 여러 개의 테이블을 가로 정렬할 때

마스터 테이블에 JOIN을 사용하면 결합하지 못한 데이터가 사라지거나,

반대로 중복된 데이터가 발생

데이터 8-3 카테고리 마스터(mst_categories) 테이블

category_id	name
1	dvd
2	cd
3	book

데이터 8-4 카테고리별 매출(category_sales) 테이블

category_id	sales
1	850000
2	500000

데이터 8-5 카테고리별 상품 매출 순위(product_sale_ranking) 테이블

category_id	rank	product_id	sales
1	1	D001	50000
1	2	D002	20000
1	3	D003	10000
2	1	C001	30000
2	2	C002	20000
2	3	C003	10000

코드 8-2 여러 개의 테이블을 결합해서 가로로 정렬하는 쿼리

PostgreSQL | Hive | Redshift | BigQuery | SparkSQL

```
SELECT
  m.category_id
  , m.name
  , s.sales
  , r.product_id AS sale_product
FROM
   mst_categories AS m
  JOIN
    -- 카테고리별로 매출액 결합하기
    category_sales AS s
    ON m.category_id = s.category_id
  JOIN
    -- 카테고리별로 상품 결합하기
    product_sale_ranking AS r
    ON  m.category_id = r.category_id
;
```

▼

실행결과

category_id	name	sales	sale_product
1	dvd	850000	D001
1	dvd	850000	D002
1	dvd	850000	D003
2	cd	500000	C001
2	cd	500000	C002
2	cd	500000	C003

8강

여러 개의 테이블 조작하기

여러 개의 테이블을 가로로 정렬하기

LEFT JOIN: 마스터 테이블의 행 수를 변경하지 않고 데이터를 가로 정렬

코드 8-3 마스터 테이블의 행 수를 변경하지 않고 여러 개의 테이블을 가로로 정렬하는 쿼리

PostgreSQL | Hive | Redshift | BigQuery | SparkSQL

```
SELECT
   m.category_id
  , m.name
  , s.sales
  , r.product_id AS top_sale_product
FROM
    mst_categories AS m
  -- LEFT JOIN을 사용해서 결합한 레코드를 남김
  LEFT JOIN
    -- 카테고리별 매출액 결합하기
    category_sales AS s
    ON m.category_id = s.category_id
  -- LEFT JOIN을 사용해서 결합하지 못한 레코드를 남김
  LEFT JOIN
    -- 카테고리별 최고 매출 상품 하나만 추출해서 결합하기
    product_sale_ranking AS r
    ON  m.category_id = r.category_id
    AND r.rank = 1
;
```

▼

실행결과

```
 category_id | name  | sales  | top_sale_product
-------------+-------+--------+-----------------
           1 | dvd   | 850000 | D001
           2 | cd    | 500000 | C001
           3 | book  |        |
```

8강

여러 개의 테이블 조작하기

조건플래그를 0과 1로표현하기

LEFT JOIN:, 사용자ID로 GROUP BY:

사용자 마스터의 레코드 수를 그대로 유지한 상태로 구매 로그 정보를 결합

CASE or SIGN 함수를 이용하여 0 or 1로 표현

데이터 8-6 신용카드 번호를 포함한 사용자 마스터(mst_users_with_card_number) 테이블

user_id	card_number
U001	1234-xxxx-xxxx-xxxx
U002	
U003	5678-xxxx-xxxx-xxxx

데이터 8-7 구매 로그(purchase_log) 테이블

purchase_id	user_id	amount	stamp
10001	U001	200	2017-01-30 10:00:00
10002	U001	500	2017-02-10 10:00:00
10003	U001	200	2017-02-12 10:00:00
10004	U002	800	2017-03-01 10:00:00
10005	U002	400	2017-03-02 10:00:00

코드 8-5 신용 카드 등록과 구매 이력 유무를 0과 1이라는 플래그로 나타내는 쿼리

PostgreSQL Hive Redshift BigQuery SparkSQL

```
SELECT
  m.user_id
  , m.card_number
  , COUNT(p.user_id) AS purchase_count
    -- 신용 카드 번호를 등록한 경우 1, 등록하지 않은 경우 0으로 표현하기
  , CASE WHEN m.card_number IS NOT NULL THEN 1 ELSE 0 END AS has_card
    -- 구매 이력이 있는 경우 1, 없는 경우 0으로 표현하기
  , SIGN(COUNT(p.user_id)) AS has_purchased
FROM
  mst_users_with_card_number AS m
 LEFT JOIN
  purchase_log AS p
  ON m.user_id = p.user_id
GROUP BY m.user_id, m.card_number
;
```

▼

실행결과

user_id	card_number	purchase_count	has_card	has_purchased
U001	1234-xxxx-xxxx-xxxx	3	1	1
U002		2	0	1
U003	5678-xxxx-xxxx-xxxx	0	1	0

8강

여러 개의 테이블 조작하기

계산한 테이블에 이름 붙여 재사용하기

공통 테이블 식 (CTE: Common Table Expression)

일시적인 테이블에 이름을 붙여 재사용

코드의 가독성이 크게 높아짐

"WITH<테이블 이름>AS (SELECT ~)"

데이터 8-8 카테고리별 상품 매출(product_sales) 테이블

category_name	product_id	sales
dvd	D001	50000
dvd	D002	20000
dvd	D003	10000
cd	C001	30000
cd	C002	20000
cd	C003	10000
book	B001	20000
book	B002	15000
book	B003	10000
book	B004	5000

코드 8-6 카테고리별 순위를 추가한 테이블에 이름 붙이기

PostgreSQL | Hive | Redshift | BigQuery | SparkSQL

```
WITH
product_sale_ranking AS (
  SELECT
      category_name
    , product_id
    , sales
    , ROW_NUMBER() OVER(PARTITION BY category_name ORDER BY sales DESC) AS rank
  FROM
    product_sales
)
SELECT *
FROM product_sale_ranking
;
```

▼

실행결과

category_name	product_id	sales	rank
book	B001	20000	1
book	B002	15000	2
book	B003	10000	3
book	B004	5000	4
cd	C001	30000	1
cd	C002	20000	2
cd	C003	10000	3
dvd	D001	50000	1
dvd	D002	20000	2
dvd	D003	10000	3

8강
여러 개의 테이블 조작하기

유사테이블 만들기

SELECT 구문

PostgreSQL Hive Redshift BigQuery SparkSQL

```
WITH
mst_devices AS (
          SELECT 1 AS device_id, 'PC' AS device_name
  UNION ALL SELECT 2 AS device_id, 'SP' AS device_name
  UNION ALL SELECT 3 AS device_id, '애플리케이션' AS device_name
)
SELECT *
FROM mst_devices
;
```

▼

실행결과

```
 device_id | device_name
-----------+-------------
         1 | PC
         2 | SP
         3 | 애플리케이션
```

순번을 자동 생성하는 테이블 함수

임의의 레코드 수를 가진 유사 테이블

PostgreSQL BigQuery

```
WITH
series AS (
  -- 1부터 5까지의 순번 생성하기
  -- ■ PostgreSQL의 경우 generate_series 사용하기
  SELECT generate_series(1, 5) AS idx
-- ■ BigQuery의 경우 generate_array 사용하기
  -- SELECT idx FROM unnest(generate_array(1, 5)) AS idx
)
SELECT *
FROM series
;
```

▼

실행결과

```
 idx
-----
   1
   2
   3
   4
   5
```

감사합니다!