데이터베이스와 SQL

8장

그룹화와 집계

빅데이터 분석가 과정

목차

8.1 그룹화의 개념

- group by
  - 많은 데이터를 일일이 조회하기 어려운 경우가 많음
  - group by 절을 사용하여 특정 컬럼의 데이터를 그룹화
  - 집계 함수(aggregate function)를 사용하여 각 그룹의 행의 수를 계산

```
use sakila;

SELECT customer_id, count(*)
FROM rental
GROUP BY customer_id ;
```

count(*)
- 각 그룹의 모든 행의 수를 계산

```
customer_id|count(*)|
-----------+--------+
          1|      32|
          2|      27|
          3|      26|
          4|      22|
          5|      38|
          6|      28|
          7|      33|
          8|      24|
          9|      23|
     . . .
        597|      25|
        598|      22|
        599|      19|
```

각 사용자 ID별 대여 회수를 계산

8.1 그룹화의 개념

- 가장 많이 대여한 회원 찾기
  - group by 연산 및 order by 연산 사용 (내림 차순 정렬)

```
SELECT customer_id, count(*)
FROM rental
GROUP BY customer_id
ORDER BY 2 desc;
```

order by에 인덱스 사용

동일

```
SELECT customer_id, count(*)
FROM rental
GROUP BY customer_id
ORDER BY count(*) desc;
```

order by에 컬럼 이름 사용

```
customer_id¦count(*)¦
-----------+--------+
        148¦      46¦
        526¦      45¦
        236¦      42¦
        144¦      42¦
         75¦      41¦
        469¦      40¦
          . . .
         61¦      14¦
        110¦      14¦
        281¦      14¦
        318¦      12¦
```

가장 많이 대여한 횟수: 46회

  - 그룹 연산 필터링
    - where절에 필터링을 적용할 수 없음
      - 그룹화한 결과에는 having 절을 사용함

8.1 그룹화의 개념

- 잘못된 필터링 사용
  - where절 다음에 group by 연산이 수행: 집계함수 count(*)를 사용하지 못함

```
SELECT customer_id, count(*)
FROM rental
WHERE count(*) > 40
GROUP BY customer_id;
```

SQL Error [1111] [HY000]: Invalid use of group function

- having절 사용
  - group by 다음에 having 절 사용

```
SELECT customer_id, count(*)
FROM rental
GROUP BY customer_id
HAVING count(*) >= 40;
```

```
customer_id¦count(*)¦
-----------+--------+
         75¦       41¦
        144¦       42¦
        148¦       46¦
        197¦       40¦
        236¦       42¦
        469¦       40¦
        526¦       45¦
```

8.2 집계 함수

- 집계 함수
  - 그룹의 모든 행에 대해 특정 연산을 수행
  - max(): 집합 내의 최댓값을 반환
  - min(): 집합 내의 최솟값 반환
  - avg(): 집합의 평균값 반환
  - sum(): 집합의 총합을 반환
  - count(): 집합의 전체 레코드 수를 반환

- payment 테이블 구성 확인

```
desc payment;
```

```
Field       ¦Type              ¦Null¦Key¦Default          ¦Extra                                       ¦
------------+------------------+----+---+-----------------+--------------------------------------------+
payment_id  ¦smallint unsigned¦NO  ¦PRI¦                 ¦auto_increment                              ¦
customer_id ¦smallint unsigned¦NO  ¦MUL¦                 ¦                                            ¦
staff_id    ¦tinyint unsigned ¦NO  ¦MUL¦                 ¦                                            ¦
rental_id   ¦int              ¦YES ¦MUL¦                 ¦                                            ¦
amount      ¦decimal(5,2)     ¦NO  ¦   ¦                 ¦                                            ¦
payment_date¦datetime         ¦NO  ¦   ¦                 ¦                                            ¦
last_update ¦timestamp        ¦YES ¦   ¦CURRENT_TIMESTAMP¦DEFAULT_GENERATED on update CURRENT_TIMESTAMP¦
```

- decimal(M, D):
  - M은 소수 부분을 포함한 전체 자리수, D는 소수 부분의 자리수

8.2 집계 함수

- payment 테이블의 amount 열에 집계 함수 계산
  - 암시적 그룹 결과
    - group by절을 사용하지 않음: 집계 함수에 의해 생성된 값

```
SELECT max(amount) as max_amt,
    min(amount) as min_amt,
    avg(amount) as avg_amt,
    sum(amount) as tot_amt,
    count(*) as num_payments
FROM payment;
```

쿼리가 반환한 값은 모두 집계함수를 사용

```
max_amt¦min_amt¦avg_amt ¦tot_amt ¦num_payments¦
-------+-------+--------+--------+------------+
  11.99¦   0.00¦4.201356¦67406.56¦       16044¦
```

- 총 16,044개 행에서 영화 대여료로 지불한 최대 금액: 11.99 달러
- 최소 금액: 0달러
- 평균 지불 금액: 4.20 달러
- 총 대여료: 67,406.51 달러

8.2.1 명시적 그룹과 암시적 그룹

■ 명시적 그룹

▪ 집계 함수를 적용하기 위해 group by 절에 그룹화할 열의 이름 지정

- customer_id값이 동일한 행들을 그룹화한 다음, 5개의 집계 함수 적용

```
SELECT customer_id,
    max(amount) as max_amt,
    min(amount) as min_amt,
    avg(amount) as avg_amt,
    sum(amount) as tot_amt,
    count(*) as num_payments
FROM payment;
```

그룹화하지 않은 일반 컬럼

집계함수

```
SELECT customer_id,
    max(amount) as max_amt,
    min(amount) as min_amt,
    avg(amount) as avg_amt,
    sum(amount) as tot_amt,
    count(*) as num_payments
FROM payment
GROUP BY customer_id;
```

customer_id를 그룹화

```
SQL Error [1140] [42000]: In aggregated query without GROUP BY, expression #1 of SELECT list contains nonaggregated column
```

- 집계 함수 및 추가 열을 검색 (에러 발생)

```
customer_id|max_amt|min_amt|avg_amt |tot_amt|num_payments|
-----------+-------+-------+--------+-------+------------+
          1|   9.99|   0.99|3.708750| 118.68|          32|
          2|  10.99|   0.99|4.767778| 128.73|          27|
          3|  10.99|   0.99|5.220769| 135.74|          26|
          4|   8.99|   0.99|3.717273|  81.78|          22|
          5|   9.99|   0.99|3.805789| 144.62|          38|
          6|   7.99|   0.99|3.347143|  93.72|          28|
          7|   8.99|   0.99|4.596061| 151.67|          33|
           .
           .
           .
        597|   8.99|   0.99|3.990000|  99.75|          25|
        598|   7.99|   0.99|3.808182|  83.78|          22|
        599|   9.99|   0.99|4.411053|  83.81|          19|
```

599 그룹에 5개의 집계함수를 적용

8.2.2 고유한 값 계산

- 고유한 값 계산
  - count() 함수 사용
    - 그룹의 모든 customer_id 수를 계산: 중복 포함
    - 모든 customer_id 중에 고유한 값에 대해서만 계산: 중복 제거(distinct 키워드)

```
SELECT count(customer_id) as num_rows,
       count(distinct customer_id) as num_customers
FROM payment;
```

```
num_rows|num_customers|
--------+-------------+
   16044|          599|
```

- 첫 번째 count(customer_id): payment 테이블의 행의 수를 계산
- 두 번째 count(distinct customer_id): 중복을 제거한 customer_id 수만 계산

8.2.3 표현식 사용

- 집계 함수를 사용할 때 표현식 사용 가능
  - 영화를 대여한 후 반환하기까지 걸린 최대 일 수 계산

```
SELECT max(datediff(return_date, rental_date))
FROM rental;
```

```
max(datediff(return_date, rental_date))¦
---------------------------------------+
                                      10¦
```

  - 영화 대여 후 반납까지 평균 기간 계산

```
SELECT avg(datediff(return_date, rental_date)) FROM rental;
```

```
avg(datediff(return_date, rental_date))¦
---------------------------------------+
                                5.0252¦
```

8.2.4 Null 처리 방법

- Null 값 처리
  - 함수들이 null 값을 만나면 무시
  - 간단한 테이블 생성

```
use sqlclass_sb;
CREATE TABLE number_tbl (val smallint);
desc number_tbl;
```

```
Field¦Type    ¦Null¦Key¦Default¦Extra¦
-----+--------+----+---+-------+-----+
val  ¦smallint¦YES ¦   ¦       ¦     ¦
```

```
INSERT INTO number_tbl VALUES(1);
INSERT INTO number_tbl VALUES(3);
INSERT INTO number_tbl VALUES(5);
```

```
val¦
---+
  1¦
  3¦
  5¦
```

  - 숫자에 대해 5개의 집계 함수 실행

```
SELECT count(*) as num_rows,
       count(val) as num_vals,
       sum(val) as total,
       max(val) as max_val,
       avg(val) as avg_val
FROM number_tbl;
```

```
num_rows¦num_vals¦total¦max_val¦avg_val¦
--------+--------+-----+-------+-------+
       3¦       3¦    9¦      5¦ 3.0000¦
```

8.2.4 Null 처리 방법

- number_tbl에 NULL 값 추가

```
INSERT INTO number_tbl VALUES (NULL);
SELECT * FROM number_tbl;
```

```
val|
---+
  1|
  3|
  5|
   |
```

NULL 값

- 집계 함수 사용

```
SELECT count(*) as num_rows,
       count(val) as num_vals,
       sum(val) as total,
       max(val) as max_val,
       avg(val) as avg_val
FROM number_tbl;
```

NULL값 추가 후 결과

```
num_rows|num_vals|total|max_val|avg_val|
--------+--------+-----+-------+-------+
       4|       3|    9|      5| 3.0000|
```

NULL값 추가 전 결과

```
num_rows|num_vals|total|max_val|avg_val|
--------+--------+-----+-------+-------+
       3|       3|    9|      5| 3.0000|
```

- 함수들이 null 값을 만나면 무시
- sum(), max() 및 avg() 함수의 결과는 이전과 동일
- count(val): 이전과 동일한 3을 반환 (null값 무시)
- count(*): 전체 행의 수를 계산 (null이 있는 행도 계산)

8.3 그룹 생성

- 단일 열 그룹화
  - 각 배우가 출연한 영화 수 계산

```
use sakila;

SELECT actor_id, count(*)
FROM film_actor
GROUP BY actor_id;
```

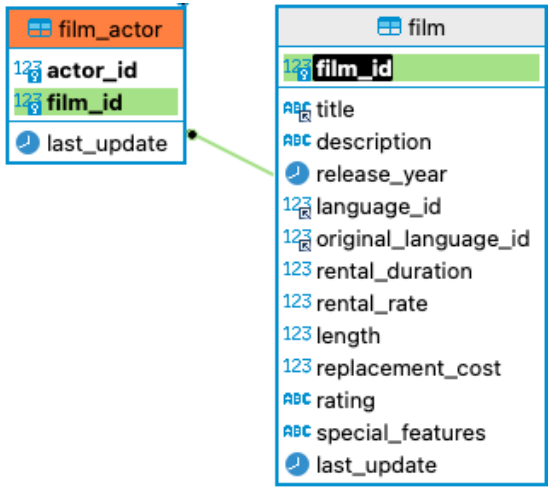
```
actor_id¦count(*)¦
--------+--------+
       1¦      19¦
       2¦      25¦
       3¦      22¦
       4¦      22¦
       5¦      29¦
       6¦      20¦
       7¦      30¦
. . .
     198¦      40¦
     199¦      15¦
     200¦      20¦
```

8.3 그룹 생성

- 다중 열 그룹화
  - 하나 이상의 열을 이용해서 그룹 생성
    - 각 배우들의 영화 등급별 출연 영화 수 계산

```
SELECT fa.actor_id, f.rating, count(*)
FROM film_actor as fa
    INNER JOIN film as f
    on fa.film_id = f.film_id
GROUP BY fa.actor_id, f.rating
ORDER BY 1, 2;
```

film_actor와 film 테이블 내부 조인



- Group by 절에 여러 개의 컬럼을 사용하는 경우
  - select 절에도 해당 컬럼을 같이 사용
  - 첫 번째 컬럼(fa.actor_id)으로 그룹화
  - 두 번째 컬럼(f.rating)으로 집계(aggregate)

```
actor_id|rating|count(*)|
--------+------+--------+
       1|G     |       4|
       1|PG    |       6|
       1|PG-13 |       1|
       1|R     |       3|
       1|NC-17 |       5|
       2|G     |       7|
       2|PG    |       6|
          . . .
     199|NC-17 |       2|
     200|G     |       5|
     200|PG    |       3|
     200|PG-13 |       2|
     200|R     |       6|
     200|NC-17 |       4|
```

각 배우의 영화 등급별 영화 출연 수 계산

8.3 그룹 생성

- 그룹화와 표현식
  - 표현식으로 생성한 값을 기반으로 그룹 생성 가능
    - extract() 함수를 사용하여 rental 테이블의 행을 그룹화

```
EXTRACT (unit, FROM date)
```

- 날짜 데이터에서 원하는 연, 월, 일, 시, 분, 초 등 특정 값을 추출

```
SELECT extract(year from rental_date) as year,
       count(*) as how_many
FROM rental
GROUP BY extract(year from rental_date);
```

```
year¦how_many¦
----+--------+
2005¦   15862¦
2006¦     182¦
```

- 연도별 대여수를 그룹화

unit(단위명)	설명
MICROSECOND	마이크로 초
SECOND	초
MINUTE	분
HOUR	시간
DAY	일
WEEK	주
MONTH	개월
QUARTER	분기
YEAR	년
SECOND_MICROSECOND	
MINUTE_MICROSECOND	
MINUTE_SECOND	
HOUR_MICROSECOND	
HOUR_SECOND	
HOUR_MINUTE	
DAY_MICROSECOND	
DAY_SECOND	
DAY_MINUTE	
DAY_HOUR	
YEAR_MONTH	

8.3.4 롤업 생성

- 각 배우/등급의 총합과 각 개별 배우의 총합 계산
  - with rollup 옵션
    - group by 결과로 출력된 항목들의 합계를 나타내는 방법

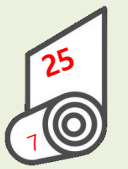
```
SELECT fa.actor_id, f.rating, count(*)
FROM film_actor as fa
    inner join film as f
    on fa.film_id = f.film_id
GROUP BY fa.actor_id, f.rating with rollup
ORDER BY 1, 2;
```

```
actor_id|rating|count(*)|
--------+------+--------+
        |      |    5462|
      1 |      |      19|
      1 |G     |       4|
      1 |NC-17 |       5|
      1 |PG    |       6|
      1 |PG-13 |       1|
      1 |R     |       3|
      2 |      |      25|
      2 |G     |       7|
      2 |NC-17 |       8|
      2 |PG    |       6|
      2 |PG-13 |       2|
      2 |R     |       2|
. . .
```

5462: film_actor 테이블의 행의 수

19: actor_id 1의 count 합

25: actor_id 2의 count 합



7 + 8 + 6 + 2 + 2

8.4 그룹 필터조건

- 두 가지 필터 조건 사용
  - WHERE 절
    - G 또는 PG 등급의 영화 선택
  - HAVING 절
    - 10개 이상의 영화에 출연한 배우만 선택

```
SELECT fa.actor_id, f.rating, count(*)
FROM film_actor fa
    INNER JOIN film f
    ON fa.film_id = f.film_id
WHERE f.rating IN ('G','PG')
GROUP BY fa.actor_id, f.rating
HAVING count(*) > 9;
```

```
actor_id¦rating¦count(*)¦
--------+------+--------+
     137¦PG    ¦       10¦
      37¦PG    ¦       12¦
     180¦PG    ¦       12¦
       7¦G     ¦       10¦
      83¦G     ¦       14¦
     129¦G     ¦       12¦
     111¦PG    ¦       15¦
      44¦PG    ¦       12¦
      26¦PG    ¦       11¦
      92¦PG    ¦       12¦
      17¦G     ¦       12¦
     158¦PG    ¦       10¦
     147¦PG    ¦       10¦
      14¦G     ¦       10¦
     102¦PG    ¦       11¦
     133¦PG    ¦       10¦
```

  - where절에는 집계함수를 포함할 수 없음

8.4 그룹 필터조건

- 두 가지 필터 조건
  - 앞의 예제에서 HAVING을 제외한 쿼리 및 결과

```
SELECT fa.actor_id, f.rating, count(*)
FROM film_actor fa
    INNER JOIN film f
    ON fa.film_id = f.film_id
WHERE f.rating IN ('G','PG')
GROUP BY fa.actor_id, f.rating
ORDER BY 1 asc;
```

```
actor_id¦rating¦count(*)¦
--------+------+--------+
       1¦G     ¦       4¦
       1¦PG    ¦       6¦
       2¦G     ¦       7¦
       2¦PG    ¦       6¦
       3¦G     ¦       2¦
       3¦PG    ¦       5¦
       4¦G     ¦       3¦
       4¦PG    ¦       5¦
       5¦G     ¦       7¦
       5¦PG    ¦       4¦
       6¦G     ¦       4¦
       6¦PG    ¦       3¦
       7¦G     ¦      10¦
       7¦PG    ¦       7¦
       8¦G     ¦       3¦
       8¦PG    ¦       2¦
       9¦G     ¦       7¦
       9¦PG    ¦       3¦
```

8.5 학습 점검

- 실습 8-2: 각 고객의 지불 횟수와 각 고객이 지불한 총 금액을 계산

```
SELECT customer_id, count(*), sum(amount)
FROM payment
GROUP BY customer_id;
```

```
customer_id¦count(*)¦sum(amount)¦
-----------+--------+-----------+
          1¦      32¦     118.68¦
          2¦      27¦     128.73¦
          3¦      26¦     135.74¦
          4¦      22¦      81.78¦
          5¦      38¦     144.62¦
          6¦      28¦      93.72¦
          7¦      33¦     151.67¦
          8¦      24¦      92.76¦
          9¦      23¦      89.77¦
         10¦      25¦      99.75¦
         11¦      24¦     106.76¦
         12¦      28¦     103.72¦
         13¦      27¦     131.73¦
         14¦      28¦     117.72¦
         15¦      32¦     134.68¦
. . .
```

Questions?