GROUPING

데이터 집계

❖ 집계 함수

- ✓ 데이터를 그룹화 해서 통계를 계산해주는 함수로 숫자 나 날짜 데이터에 사용
- ✓ GROUP BY 이후에 그룹화가 이루어지므로 HAVING, SELECT 등에서만 사용 가능
- ✓ GROUP BY 절의 그룹화 항목 하고만 같이 조회 가능
- ✓ 문자열의 합계 나 평균은 구할 수 없지만 최소값 과 최대값을 조회할 수 있음

함수	설명
AVG()	평균
MIN()	최소값
MAX()	최대값
COUNT()	행의 개수
COUNT(DISTINCT)	중복은 1개만 인정한 후 행의 개수
STDEV()	표준 편차
VAR_SAMP()	분산

데이터 집계

- 집계 함수
  - ✓ COUNT
    - 개수를 세는 함수
    - 필드 이름이나 *를 지정
    - tStaff 테이블의 데이터 개수 조회

```
SELECT COUNT(*)
FROM tStaff;

SELECT COUNT(*) AS "총 직원수"
FROM tStaff;
```

데이터 집계

- ❖ 집계 함수
  - ✓ COUNT
    - ● 조건과 함께 사용

```
SELECT COUNT(*)
FROM tStaff
WHERE salary >= 400;

SELECT COUNT(*)
FROM tStaff
WHERE salary >= 10000;
```

데이터 집계

- ❖ 집계 함수
  - ✓ COUNT
    - ● 필드 이름 사용

```
SELECT COUNT(name)
FROM tStaff;

SELECT COUNT(depart)
FROM tStaff;

SELECT COUNT(DISTINCT depart)
FROM tStaff;
```

데이터 집계

- 집계 함수
  - ✓ COUNT
    - 필드 이름 사용

```
SELECT COUNT(score)
FROM tStaff;

SELECT COUNT(*) - COUNT(score)
FROM tStaff;

SELECT COUNT(*)
FROM tStaff
WHERE score IS NULL;
```

데이터 집계

- ❖ 집계 함수
  - ✓ 연습문제
    - tStaff 테이블에서 score가 없는 두 직원은 몇 명인지 조회
    - tStaff 테이블에서 score가 80점 이상인 직원이 몇 명이나 되는지 조회

데이터 집계

- 일반 집계 함수
  - ✓ 도시 목록에서 인구의 총합 과 평균을 조회

```
SELECT SUM(popu), AVG(popu)
FROM tCity;
```

데이터 집계

- 일반 집계 함수
  - ✓ 도시 목록에서 면적의 최소값 과 최대값 조회

```
SELECT MIN(area), MAX(area)
FROM tCity;
```

데이터 집계

- 일반 집계 함수
  - ✓ WHERE 절 사용

```
SELECT SUM(score), AVG(score)
FROM tStaff
WHERE depart = '인사과';

SELECT MIN(salary), MAX(salary)
FROM tStaff
WHERE depart = '영업부';
```

데이터 집계

- 일반 집계 함수
  - ✓ 문자열 사용

```
SELECT MIN(name)
FROM tStaff;

SELECT MAX(popu), name
FROM tCity;
```

데이터 집계

- 집계 함수 와 NULL
  - ✓ NULL은 값을 알 수 없는 특수한 상태이므로 모든 집계 함수는 NULL을 무시하고 통계를 계산
  - ✓ COUNT는 일치하는 데이터가 없으면 0을 반환하지만 다른 집계 함수는 일치하는 데이터가 없으면 NULL을 리턴
  - ✓ 실습

```
SELECT AVG(salary)
FROM tStaff;

SELECT SUM(salary)/COUNT(*)
FROM tStaff;

SELECT AVG(score)
FROM tStaff;

SELECT SUM(score)/COUNT(*)
FROM tStaff;
```

데이터 집계

- 집계 함수 와 NULL
  - ✓ 실습

```
SELECT COUNT(*)
FROM tStaff
WHERE depart = '비서실';

SELECT MAX(salary)
FROM tStaff
WHERE depart = '비서실';
```

데이터 집계

- GROUPING
  - ✓ GROUP BY
    - ● 형식

```
SELECT select_expr
[FROM table_references]
[WHERE where_condition]
[GROUP BY {col name ! expr ! position}]
[ORDER BY {col_name I expr ! position}]
```

데이터 집계

- GROUPING
  - ✓ GROUP BY
    - 3개의 부서에 대한 월급 평균 조회

```
SELECT '영업부', AVG(salary)
FROM tStaff
WHERE depart='영업부';

SELECT '총무부', AVG(salary)
FROM tStaff
WHERE depart='총무부';

SELECT '인사과', AVG(salary)
FROM tStaff
WHERE depart='인사과';
```

데이터 집계

- GROUPING
  - ✓ GROUP BY
    - 기준이 되는 필드를 뒤에 적어 주면 기준 필드가 같은 레코드를 모아 통계 값을 구함
    - 기준 필드는 집계 함수와 같이 쓸 수 있어 목록도 보기 좋게 출력할 수 있음
    - GROUP BY의 기준 필드는 중복 값이 있을 때만 의미가 있는데 레코드 별로 고유한 값을 가지는 필드는 그룹핑 기준으로 부적합하며 구분이나 분류 필드가 적합
    - 2개 이상의 필드를 기준으로 그룹핑 가능
    - 부서별 평균 월급을 구하려면 depart 필드 기준으로 그룹핑

```
SELECT depart, AVG(salary)
FROM tStaff
GROUP BY depart;
```

데이터 집계

- GROUPING
  - ✓ GROUP BY
    - ● 여러 개의 집계 함수 사용

```
SELECT depart, COUNT(*), MAX(joindate), AVG(score)
FROM tStaff
GROUP BY depart;
```

데이터 집계

- GROUPING
  - ✓ 연습문제
    - 도시 목록에서 지역별 인구수를 조회

데이터 집계

- GROUPING
  - ✓ GROUP BY
    - 2개 이상의 필드로 그룹화

```
SELECT depart, gender, COUNT(*)
FROM tStaff
GROUP BY depart, gender;
```

데이터 집계

- GROUPING
  - ✓ GROUP BY
    - 2개 이상의 필드로 그룹화 한 후 정렬해서 조회

```
SELECT depart, gender, COUNT(*)
FROM tStaff
GROUP BY depart, gender
ORDER BY depart, gender;
```

데이터 집계

- GROUPING
  - ✓ GROUP BY 필드 목록
    - GROUP BY 절이 있으면 조회할 수 있는 목록(SELECT)에는 기준 필드나 집계 함수만 올 수 있음
    - 그룹과 상관없는 필드는 집계 함수없이 단독으로 출력할 수 없음
    - 아래 명령은 오라클에서는 에러지만 Maria DB에서는 첫번째 데이터를 조회

```
SELECT depart, salary
FROM tStaff
GROUP BY depart;
```

데이터 집계

- GROUPING
  - ✓ GROUP BY 필드 목록
    - 아래 명령으로 수정해서 실행

```
SELECT SUM(salary)
FROM tStaff
GROUP BY depart;

SELECT depart, SUM(salary)
FROM tStaff
GROUP BY depart;
```

데이터 집계

- GROUPING
  - ✓ HAVING
    - GROUP BY 다음에 오며 통계 결과 중 출력할 그룹의 조건을 지정
    - 형식

```
SELECT select_expr
[FROM table_references]
[WHERE where_condition]
[GROUP BY {col name ! expr ! position}]
[HAVING where_condition]
[ORDER BY {col_name I expr ! position}]
```

데이터 집계

- GROUPING
  - ✓ HAVING
    - buytbl 테이블에서 총 구매액(price * amount)이 1000 보다 큰 userID 와 총 구매액을 조회

```
SELECT userID AS '사용자', SUM(price * amount) AS '총구매액'
FROM buyTbl
WHERE SUM(price * amount) > 1000
GROUP BY userID;

SELECT userID AS '사용자', SUM(price * amount) AS '총구매액'
FROM buyTbl
GROUP BY userID
HAVING SUM(price * amount) > 1000;
```

데이터 집계

- GROUPING
  - ✓ HAVING
    - 평균 월급이 340을 넘는 부서만 조회

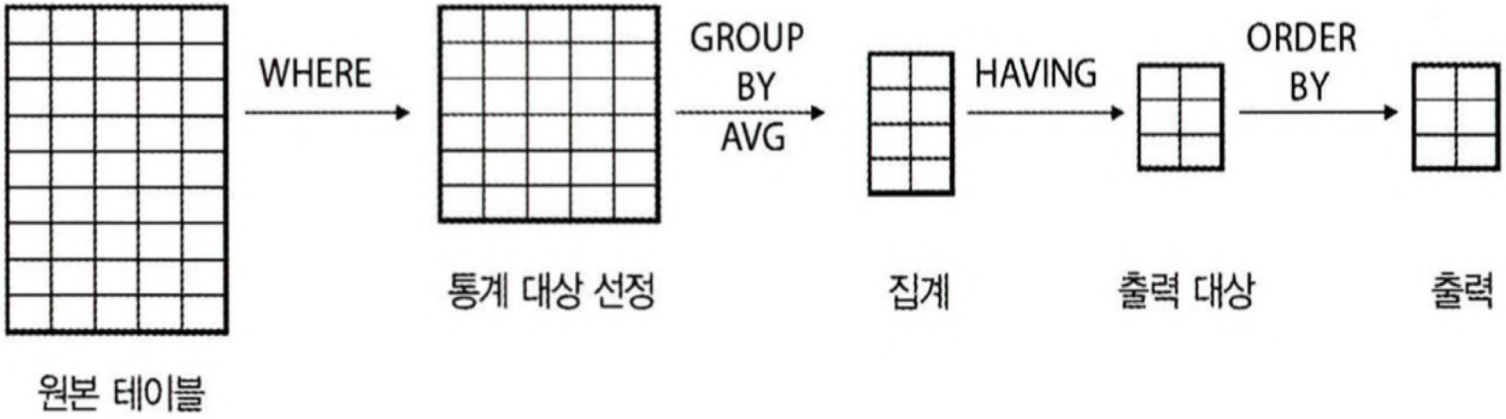
```
SELECT depart, AVG(salary)
FROM tStaff
GROUP BY depart
HAVING AVG(salary) >= 340;
```

    - 평균 월급이 340을 넘는 부서를 정렬해서 조회

```
SELECT depart, AVG(salary)
FROM tStaff
GROUP BY depart
HAVING AVG(salary) >= 340
ORDER BY AVG(salary);
```

데이터 집계

- GROUPING
  - ✓ HAVING
    - 쿼리가 실행되는 과정



데이터 집계

- GROUPING
  - ✓ HAVING
    - 동일한 쿼리

```
SELECT depart, MAX(salary)
FROM tStaff
WHERE depart IN ('인사과', '영업부')
GROUP BY depart;

SELECT depart, MAX(salary)
FROM tStaff
GROUP BY depart
HAVING depart IN ('인사과', '영업부');
```

데이터 집계

- GROUPING
  - ✓ 연습문제
    - EMP 테이블에서 인원수, 최대 급여(sal), 최소 급여, 급여의 합을 계산하여 출력하는 SELECT 문장을 작성
    - EMP 테이블에서 각 업무별(job)로 최대 급여(sal), 최소 급여, 급여의 합을 출력하는 SELECT 문장을 작성
    - EMP 테이블에서 업무별(job) 인원수를 구하여 출력하는 SELECT 문장을 작성
    - EMP 테이블에서 최고 급여(sal)와 최소 급여의 차이는 얼마인가 출력하는 SELECT 문장을 작성

함수

- 윈도우 함수
  - ✓ Window Function는 행 과 행 사이의 관계를 쉽게 정의하기 위해서 제공되는 함수
  - ✓ 윈도우 함수를 활용한다면 복잡한 SQL을 쉽게 활용할 수 있음
  - ✓ 윈도우 함수는 OVER 절이 들어간 함수라고 보면 되는데 윈도우 함수와 함께 사용되는 집계 함수로는 AVG(), COUNT0, MAX(), MIN(), STDDEV(), SUM(), VARIANCE() 등이 있음
  - ✓ 윈도우 함수와 함께 사용되는 비집계 함수에는 CUME_DIST(), DENSE_RANK(), FIRST_VALUE(), LAG(), LAST_VALUE(), LEAD(), NTH_VALUE(), NTILE(), PERCENT_RANK(), RANK(), ROW_NUMBER() 등이 있음

윈도우 함수

- 순위 함수
  - ✓ RANK(), NTILE(), DENSE_RANK(), ROW_NUMBER(), PERCENT_RANK()
  - ✓ 기본 형식

    <순위함수이름>( ) OVER(

    [PARTITION BY <partition_by_list>]

    ORDER BY <order_by_list>

윈도우 함수

- 순위 함수
  - ✓ 일련 번호 형태로 순위 설정

```
SELECT ROW_NUMBER( ) OVER(ORDER BY birthyear ASC) '나이가 많은 순서', name,
    birthyear
FROM usertbl;

SELECT ROW_NUMBER( ) OVER(ORDER BY birthyear ASC, name ASC) '나이가 많은
    순서', name, birthyear
FROM usertbl;

SELECT ROW_NUMBER( ) OVER(PARTITION BY addr ORDER BY birthyear ASC) '나이
    가 많은 순서', addr, name, birthyear
FROM usertbl;
```

윈도우 함수

❖ 순위 함수

✓ 동일한 값은 동일한 순위 설정

```
SELECT DENSE_RANK( ) OVER(ORDER BY birthyear ASC) '나이가 많은 순서', name, birthyear
FROM usertbl;
```

✓ 동일한 값은 동일한 순위의 경우 다음 순위 건너뛰기

```
SELECT RANK( ) OVER(ORDER BY birthyear ASC) '나이가 많은 순서', name, birthyear
FROM usertbl;
```

✓ N등분 하기

```
SELECT NTILE(5) OVER(ORDER BY birthyear ASC) '나이가 많은 순서', name, birthyear
FROM usertbl;
```

윈도우 함수

❖ 분석 함수

✓ CUME_DIST(), LEAD(), FIRST_VALUE(), LAG(), LAST_VALUE(), PERCENT_RANK() 등

● 다음 행과의 차이

```
SELECT name, addr, birthyear AS "태어난 해", birthyear - (LEAD(birthyear, 1)
    OVER(ORDER BY birthyear desc)) as "나이 차이"
FROM usertbl;
```

● 이전 행 과의 차이

```
SELECT name, addr, birthyear AS "태어난 해", birthyear - (LAG(birthyear, 1)
    OVER(ORDER BY birthyear desc)) as "나이 차이"
FROM usertbl;
```

윈도우 함수

❖ 분석 함수

✓ CUME_DIST(), LEAD(), FIRST_VALUE(), LAG(), LAST_VALUE() 등

● 첫번째 행 과의 차이

```
SELECT name, addr, birthyear AS "태어난 해", birthyear -
    (FIRST_VALUE(birthyear) OVER(PARTITION BY addr ORDER BY birthyear
    desc)) as "나이 차이"
FROM usertbl;
```

● 누적 백분율

```
SELECT name, addr, birthyear AS "태어난 해", CUME_DIST() OVER(PARTITION
    BY addr ORDER BY birthyear desc) * 100 AS "누적인원 백분율"
FROM usertbl;
```

윈도우 함수

❖ WITH ROLLUP

✓ GROUP BY 절과 함께 사용하여 그룹 합계와 총합계를 만들어 주는 기능

- order_d 테이블로부터 goodscd 별로 그룹화 한 후 goodscd 와 qty 의 합계를 조회하는데 총 합계를 같이 조회

```
SELECT goodscd, SUM(qty)
FROM order_d
GROUP BY goodscd WITH ROLLUP;
```

- order_d 테이블로부터 goodscd 별로 그룹화 한 후 goodscd 와 orderno별로 qty 의 합계를 조회

```
SELECT orderno, goodscd, SUM(qty)
FROM order_d
GROUP BY goodscd, orderno WITH ROLLUP;
```

피벗 보고서

- 피벗
  - ✓ 피벗은 한 열에 포함된 여러 값을 출력하고 이를 여러 열로 변환하여 테이블 반환 식을 회전하고 필요하면 집계까지 수행하는 것

```
CREATE TABLE pivotTest
  (  uName CHAR(20),
     season CHAR(20),
     amount INT );

INSERT  INTO  pivotTest VALUES
            ('aespa' , '겨울',  10) , ('블랙핑크' , '여름',  15) , ('aespa' , '가을',  25) ,
('aespa' , '봄',    3) ,
            ('aespa' , '봄',    37) , ('블랙핑크' , '겨울',  40) , ('aespa' , '여름',
14) ,('aespa' , '겨울',  22) ,
            ('블랙핑크' , '여름',  64) ;

SELECT * FROM pivotTest;
```

피벗 보고서

- 피벗
  - ✓ 피벗은 한 열에 포함된 여러 값을 출력하고 이를 여러 열로 변환하여 테이블 반환 식을 회전하고 필요하면 집계까지 수행하는 것

```
SELECT uName,
  SUM(IF(season='봄', amount, 0)) AS '봄',
  SUM(IF(season='여름', amount, 0)) AS '여름',
  SUM(IF(season='가을', amount, 0)) AS '가을',
  SUM(IF(season='겨울', amount, 0)) AS '겨울',
  SUM(amount) AS '합계'
 FROM pivotTest
GROUP BY uName ;
```

JSON

❖ JSON 출력

```
SELECT JSON_OBJECT('name', name, 'birthyear', birthyear) AS 'JSON 값'
FROM usertbl
WHERE birthyear >= 1980;
```

JSON

- JSON 조작

```
-- JSON 함수 사용
SET @json='{ "usertbl" :
  [
    {"name":"카리나","birthyear":2000},
    {"name":"지젤","birthyear":2000},
    {"name":"윈터","birthyear":2001},
   {"name":"닝닝","birthyear":2002}
  ]
}' ;
```

JSON

❖ JSON 조작

```
SELECT JSON_VALID(@json);

SELECT JSON_SEARCH(@json, 'one', '카리나');

SELECT JSON_EXTRACT(@json, '$.usertbl[2].name');

SELECT JSON_INSERT(@json, '$.usertbl[0].mDate', '2000-04-01');

SELECT JSON_REPLACE(@json, '$.usertbl[0].name', '유지민');

SELECT JSON_REMOVE(@json, '$.usertbl[0]');
```