

4-1. 집계쿼리 - 집계함수와 GROUP BY 절

홍형경

chariehong@gmail.com

2020.11

1. 집계 쿼리

- **GROUP BY 절과 집계 함수**를 사용한 쿼리
- 특정 항목(컬럼)별 최소, 최대, 평균 값 등을 산출
- 과목별 평균 점수, 월별 전체 매출액 등 기본적인 데이터 분석에 사용됨
- GROUP BY 절과 집계 함수 단독 사용 가능하나, 일반적으로 둘 모두를 함께 사용

2. GROUP BY 절

- 구문

SELECT expr1, expr2, ...

FROM ...

WHERE ...

GROUP BY expr1, expr2 ...

ORDER BY ... ;

- WHERE 절과 ORDER BY 절 사이에 위치

- GROUP BY 절에 기술한 컬럼이나 표현식 별로 데이터가 집계

2. GROUP BY 절

- GROUP BY 절에 기술한 컬럼, 표현식 이외의 항목은 SELECT 절에 **명시 불가**
단, 집계 함수는 가능
- GROUP BY 절과 집계 함수를 함께 사용해야 의미 있는 결과를 도출

3. 집계함수

- 여러 건의 데이터를 집계 연산한 결과를 반환하는 함수
- **COUNT** (expr)
 - expr의 전체 개수 반환
 - expr은 컬럼을 포함한 표현식, 보통 * 사용
- **MAX** (expr)
 - expr의 최댓값 반환
- **MIN** (expr)
 - expr의 최솟값 반환

3. 집계함수

- **SUM** (expr)
 - expr의 합계 반환
- **AVG** (expr)
 - expr의 평균값 반환
- **VARIANCE** (expr)
 - expr의 분산 반환
- **STDDEV** (expr)
 - expr의 표준편차 반환

3. 집계함수

- GROUP BY 절 없이 집계 함수만 사용 시 **조회되는 데이터 전체에 대한 집계 값 계산**
- GROUP BY 절과 함께 사용 시, **GROUP BY 절에 명시한 항목별 집계 값 계산**
- 매개변수 '*'는 COUNT 함수에서만 사용

4. Group by절, 집계 함수 실습

(1) Group by 절

```
SELECT employee_id  
FROM employees  
GROUP BY employee_id;
```

→ employee_id 컬럼은 기본 키이므로
유일한 값만 들어 있어
GROUP BY 절을 사용해 집계하는 의미가 없음

	EMPLOYEE_ID
1	100
2	101
3	102
4	103
5	104
6	105
7	106
8	107
9	108
10	109

99	198
100	199
101	200
102	201
103	202
104	203
105	204
106	205
107	206

4. Group by절, 집계 함수 실습

(1) Group by 절

```
SELECT employee_id, job_id  
FROM employees  
ORDER by 2;
```

```
SELECT job_id  
FROM employees  
GROUP BY job_id;
```

→ job_id 컬럼을 기준으로 집계

즉, job_id 컬럼의 **유일한 값들을** 모아 집계됨

→ **유일한 job_id 컬럼 값 수로** 로우 수가 줄어 듬

EMPLOYEE_ID	JOB_ID
206	AC_ACCOUNT
205	AC_MGR
200	AD_ASST
100	AD_PRES
101	AD_VP
102	AD_VP
109	FI_ACCOUNT
113	FI_ACCOUNT
110	FI_ACCOUNT
111	FI_ACCOUNT
112	FI_ACCOUNT
108	FI_MGR
203	HR_REP
104	IT_PROG
103	IT_PROG
105	IT_PROG
107	IT_PROG
106	IT_PROG

JOB_ID
AC_ACCOUNT
AC_MGR
AD_ASST
AD_PRES
AD_VP
FI_ACCOUNT
FI_MGR
HR_REP
IT_PROG
MK_MAN
MK_REP
PR_REP
PU_CLERK
PU_MAN
SA_MAN
SA_REP
SH_CLERK
ST_CLERK

4. Group by절, 집계 함수 실습

(1) Group by 절

```
SELECT TO_CHAR(hire_date, 'YYYY') HIRE_YEAR  
FROM employees  
GROUP BY TO_CHAR(hire_date, 'YYYY');
```

- **입사년도** 별 집계를 하므로 총 조회되는 로우 수는 8개
- GROUP BY 절에는 SELECT 절에 기술한 형태 그대로 사용해야 함
별칭은 기술하면 안됨

	HIRE_YEAR
1	2002
2	2004
3	2008
4	2005
5	2001
6	2007
7	2003
8	2006

4. Group by절, 집계 함수 실습

(1) Group by 절

```
SELECT TO_CHAR(hire_date, 'YYYY') HIRE_YEAR  
FROM employees  
GROUP BY hire_date;
```

→ **입사년도** 별 집계를 하고자 했으나, GROUP BY 절에 입사일자를 명시해
결과적으로 입사년도가 아닌 입사일자별로 집계 되었음

→ **잘못된 집계 쿼리**

	HIRE_YEAR
1	2005
2	2007
3	2007
4	2005
5	2007
6	2005
7	2003
8	2005
9	2005
10	2007
11	2005
12	2006
13	2007
14	2008
15	2007
16	2008
17	2006
18	2006
19	2005
20	2006
21	2006
22	2003
23	2006

	HIRE_DATE
	2005-09-30 00:00:00
	2007-12-07 00:00:00
	2007-08-10 00:00:00
	2005-10-10 00:00:00
	2007-01-14 00:00:00
	2005-10-30 00:00:00
	2003-10-17 00:00:00
	2005-01-29 00:00:00
	2005-01-05 00:00:00
	2007-10-15 00:00:00
	2005-01-30 00:00:00
	2006-11-03 00:00:00
	2007-03-19 00:00:00
	2008-02-23 00:00:00
	2007-05-24 00:00:00
	2008-02-03 00:00:00
	2006-04-24 00:00:00
	2006-05-23 00:00:00
	2005-09-21 00:00:00
	2006-02-05 00:00:00
	2006-03-07 00:00:00
	2003-05-18 00:00:00
	2006-11-15 00:00:00

4. Group by절, 집계 함수 실습

(2) 집계함수

```
SELECT COUNT(*)  
FROM employees;
```

COUNT(*)
107

→ EMPLOYEES 테이블의 전체 로우 건 수

4. Group by절, 집계 함수 실습

(2) 집계함수

```
SELECT COUNT(*) total_cnt, MIN(salary) min_salary, MAX(salary) max_salary  
FROM employees;
```

TOTAL_CNT	MIN_SALARY	MAX_SALARY
107	2100	24000

- ➔ EMPLOYEES 테이블의 전체 로우 건 수
- ➔ Salary 컬럼의 최소와 최대값

4. Group by절, 집계 함수 실습

(3) Group By 와 집계함수

```
SELECT job_id,  
       COUNT(*) total_cnt,  
       MIN(salary) min_salary,  
       MAX(salary) max_salary  
FROM employees  
GROUP BY job_id  
ORDER BY job_id;
```

→ EMPLOYEES 테이블의 job_id 별
건수, salary 컬럼의 최소와 최대값

JOB_ID	TOTAL_CNT	MIN_SALARY	MAX_SALARY
1 AC_ACCOUNT	1	8300	8300
2 AC_MGR	1	12008	12008
3 AD_ASST	1	4400	4400
4 AD_PRES	1	24000	24000
5 AD_VP	2	17000	17000
6 FI_ACCOUNT	5	6900	9000
7 FI_MGR	1	12008	12008
8 HR_REP	1	6500	6500
9 IT_PROG	5	4200	9000
10 MK_MAN	1	13000	13000
11 MK_REP	1	6000	6000
12 PR_REP	1	10000	10000
13 PU_CLERK	5	2500	3100
14 PU_MAN	1	11000	11000
15 SA_MAN	5	10500	14000
16 SA_REP	30	6100	11500
17 SH_CLERK	20	2500	4200
18 ST_CLERK	20	2100	3600
19 ST_MAN	5	5800	8200

EMPLOYEE_ID	JOB_ID	SALARY
109	FI_ACCOUNT	9000
110	FI_ACCOUNT	8200
111	FI_ACCOUNT	7700
112	FI_ACCOUNT	7800
113	FI_ACCOUNT	6900

4. Group by절, 집계 함수 실습

(3) Group By 와 집계함수

```
SELECT TO_CHAR(hire_date, 'YYYY') HIRE_YEAR,
       department_id,
       COUNT(*), SUM(salary), AVG(salary)
FROM employees
WHERE TO_CHAR(hire_date, 'YYYY') >= '2004'
GROUP BY TO_CHAR(hire_date, 'YYYY'), department_id
ORDER BY 1, 2;
```

➔ 2004년 이후 입사 년도와 부서별

총 인원수와 급여 총액, 급여 평균

[illegible]

4. Group by절, 집계 함수 실습

(3) Group By 와 집계함수

```
SELECT TO_CHAR(hire_date, 'YYYY') HIRE_YEAR,  
       department_id,  
       COUNT(*), SUM(salary), ROUND(AVG(salary),0)  
FROM employees  
WHERE TO_CHAR(hire_date, 'YYYY') >= '2004'  
GROUP BY TO_CHAR(hire_date, 'YYYY'), department_id  
ORDER BY 1, 2;
```

→ ROUND 함수를 사용해 급여 평균 값의 소수점 제거

	HIRE_YEAR	DEPARTMENT_ID	COUNT(*)	SUM(SALARY)	ROUND(AVG(SALARY),0)
1	2004	20	1	13000	13000
2	2004	50	4	19500	4875
3	2004	80	5	53500	10700
4	2005	20	1	6000	6000
5	2005	30	2	5700	2850
6	2005	50	12	48200	4017
7	2005	60	1	4800	4800
8	2005	80	10	100300	10030
9	2005	90	1	17000	17000
10	2005	100	2	15900	7950
11	2006	30	1	2600	2600
12	2006	50	13	37800	2908
13	2006	60	2	13800	6900
14	2006	80	7	59100	8443
15	2006	100	1	7800	7800
16	2007	30	1	2500	2500
17	2007	50	9	26100	2900
18	2007	60	2	10200	5100
19	2007	80	5	42200	8440
20	2007	100	1	6900	6900
21	2007	(null)	1	7000	7000
22	2008	50	4	9800	2450
23	2008	80	7	49400	7057

4. Group by절, 집계 함수 실습

(3) Group By 와 집계함수

```
SELECT TO_CHAR(hire_date, 'YYYY') HIRE_YEAR,  
       department_id,  
       COUNT(*), SUM(salary), ROUND(AVG(salary),0)  
FROM employees  
WHERE ROUND(AVG(salary),0) >= 5000  
GROUP BY TO_CHAR(hire_date, 'YYYY'), department_id  
ORDER BY 1, 2;
```

➔ 그룹 함수는 **WHERE 절에서 사용 불가**

```
ORA-00934: 그룹 함수는 허가되지 않습니다  
00934, 00000 - "group function is not allowed here"  
*Cause:  
*Action:  
3행, 13열에서 오류 발생
```

5. HAVING 절

- 집계 쿼리에서 집계 함수 반환 값에 대한 조건을 걸 때 사용
- 일반적인 조건 → WHERE 절, HAVING 절 → 집계 쿼리에 대한 추가 조건 절
- 예) 한 반에서 과목별 평균 점수가 60점 이상인 과목을 조회
 - 집계 쿼리로 평균 값 산출 : AVG(점수)
 - WHERE AVG(점수) >= 60 → X
 - HAVING AVG(점수) >= 60 → O

6. DISTINCT

- **SELECT DISTINCT expr1, expr2 ...
FROM ...**
- **DISTINCT 뒤에 명시한 표현식(컬럼)의 고유한 값을 조회**
- **집계 함수 없이 GROUP BY 절을 사용한 것과 동일한 효과**

7. Having절과 Distinct 실습

(1) Having 절

```
SELECT TO_CHAR(hire_date, 'YYYY') HIRE_YEAR,  
       department_id,  
       COUNT(*), SUM(salary), ROUND(AVG(salary),0)  
FROM employees  
--WHERE ROUND(AVG(salary),0) >= 5000  
GROUP BY TO_CHAR(hire_date, 'YYYY'), department_id  
HAVING ROUND(AVG(salary),0) >= 5000  
ORDER BY 1, 2;
```

	HIRE_YEAR	DEPARTMENT_ID	COUNT(*)	SUM(SALARY)	ROUND(AVG(SALARY),0)
1	2001	90	1	17000	17000
2	2002	30	1	11000	11000
3	2002	40	1	6500	6500
4	2002	70	1	10000	10000
5	2002	100	2	21008	10504
6	2002	110	2	20308	10154
7	2003	50	3	15000	5000
8	2003	90	1	24000	24000
9	2004	20	1	13000	13000
10	2004	80	5	53500	10700
11	2005	20	1	6000	6000
12	2005	80	10	100300	10030
13	2005	90	1	17000	17000
14	2005	100	2	15900	7950
15	2006	60	2	13800	6900
16	2006	80	7	59100	8443
17	2006	100	1	7800	7800
18	2007	60	2	10200	5100
19	2007	80	5	42200	8440
20	2007	100	1	6900	6900
21	2007	(null)	1	7000	7000
22	2008	80	7	49400	7057

7. Having절과 Distinct 실습

(1) Having 절

```
SELECT TO_CHAR(hire_date, 'YYYY') HIRE_YEAR,  
       department_id,  
       COUNT(*), SUM(salary), ROUND(AVG(salary),0)  
FROM employees  
GROUP BY TO_CHAR(hire_date, 'YYYY'), department_id  
HAVING COUNT(*) > 1  
ORDER BY 1, 2;
```

	HIRE_YEAR	DEPARTMENT_ID	COUNT(*)	SUM(SALARY)	ROUND(AVG(SALARY),0)
1	2002	100	2	21008	10504
2	2002	110	2	20308	10154
3	2003	50	3	15000	5000
4	2004	50	4	19500	4875
5	2004	80	5	53500	10700
6	2005	30	2	5700	2850
7	2005	50	12	48200	4017
8	2005	80	10	100300	10030
9	2005	100	2	15900	7950
10	2006	50	13	37800	2908
11	2006	60	2	13800	6900
12	2006	80	7	59100	8443
13	2007	50	9	26100	2900
14	2007	60	2	10200	5100
15	2007	80	5	42200	8440
16	2008	50	4	9800	2450
17	2008	80	7	49400	7057

7. Having절과 Distinct 실습

(2) DISTINCT

```
SELECT job_id  
FROM employees  
GROUP BY job_id;
```

```
SELECT DISTINCT job_id  
FROM employees;
```

JOB_ID
1 AC_ACCOUNT
2 AC_MGR
3 AD_ASST
4 AD_PRES
5 AD_VP
6 FI_ACCOUNT
7 FI_MGR
8 HR_REP
9 IT_PROG
10 MK_MAN
11 MK_REP
12 PR_REP
13 PU_CLERK
14 PU_MAN
15 SA_MAN
16 SA_REP
17 SH_CLERK
18 ST_CLERK
19 ST_MAN

JOB_ID
1 AC_ACCOUNT
2 AC_MGR
3 AD_ASST
4 AD_PRES
5 AD_VP
6 FI_ACCOUNT
7 FI_MGR
8 HR_REP
9 IT_PROG
10 MK_MAN
11 MK_REP
12 PR_REP
13 PU_CLERK
14 PU_MAN
15 SA_MAN
16 SA_REP
17 SH_CLERK
18 ST_CLERK
19 ST_MAN

7. Having절과 Distinct 실습

(2) DISTINCT

```
SELECT DISTINCT TO_CHAR(hire_date, 'YYYY') HIRE_YEAR, department_id
FROM employees
ORDER BY 1, 2;
```

	HIRE_YEAR	DEPARTMENT_ID
1	2001	90
2	2002	30
3	2002	40
4	2002	70
5	2002	100
6	2002	110
7	2003	10
8	2003	30
9	2003	50
10	2003	90
11	2004	20
12	2004	50
13	2004	80
14	2005	20
15	2005	30
16	2005	50
17	2005	60
18	2005	80
19	2005	90
20	2005	100
21	2006	30
22	2006	50
23	2006	60
24	2006	80
25	2006	100
26	2007	20

8. Rollup과 Cube

- Rollup : 소계(Sub Total)
- `SELECT COL1, COL2, SUM(COL3)`
`FROM TABLE1`
`GROUP BY ROLLUP(COL1, COL2) ...`
→ COL1에 대한 소계, COL1과 COL2의 계, 그리고 전체 합계 계산
- ROLLUP에 명시한 표현식 수(콤마로 구분) + 1개를 Grouping
예) `ROLLUP(col1, col2)`
→ col1과 col2에 대한 합계, col1에 대한 합계, 전체 합계

8. Rollup과 Cube

- Cube : 모든 가능한 조합에 대한 소계
- ```
SELECT COL1, COL2, SUM(COL3)
FROM TABLE1
GROUP BY CUBE(COL1, COL2) ...
```
- CUBE 절에 명시한 표현식 수(콤마로 구분)가 n개 → 2의 n승개의 조합  
예) .... ROLLUP(col1, col2) → 2의 2승인 4개의 조합  
→ col1, col2, col1과 col2 에 대한 합계, 전체 합계

## 9. Rollup과 Cube 실습

```
SELECT substr(phone_number,1,3),
 JOB_ID,
 SUM(salary)
FROM EMPLOYEES
GROUP BY JOB_ID,
 substr(phone_number,1,3)
ORDER BY 1, 2;
```

|    | ⚡ SUBSTR(PHONE_NUMBE... | ⚡ JOB_ID   | ⚡ SUM(SALARY) |
|----|-------------------------|------------|---------------|
| 1  | 011                     | SA_MAN     | 61000         |
| 2  | 011                     | SA_REP     | 250500        |
| 3  | 515                     | AC_ACCOUNT | 8300          |
| 4  | 515                     | AC_MGR     | 12008         |
| 5  | 515                     | AD_ASST    | 4400          |
| 6  | 515                     | AD PRES    | 24000         |
| 7  | 515                     | AD_VP      | 34000         |
| 8  | 515                     | FI_ACCOUNT | 39600         |
| 9  | 515                     | FI_MGR     | 12008         |
| 10 | 515                     | HR_REP     | 6500          |
| 11 | 515                     | MK_MAN     | 13000         |
| 12 | 515                     | PR_REP     | 10000         |
| 13 | 515                     | PU_CLERK   | 13900         |
| 14 | 515                     | PU_MAN     | 11000         |
| 15 | 590                     | IT_PROG    | 28800         |
| 16 | 603                     | MK_REP     | 6000          |
| 17 | 650                     | SH_CLERK   | 64300         |
| 18 | 650                     | ST_CLERK   | 55700         |
| 19 | 650                     | ST_MAN     | 36400         |

## 9. Rollup과 Cube 실습

```
SELECT substr(phone_number,1,3),
 JOB_ID,
 SUM(salary)
FROM EMPLOYEES
GROUP BY
 CUBE(substr(phone_number,1,3), JOB_ID)
ORDER BY 1, 2;
```

## 9. Rollup과 Cube 실습

|    | ⚡ SUBSTR(PHONE_NUMBER,1,3) | ⚡ JOB_ID   | ⚡ SUM(SALARY) |
|----|----------------------------|------------|---------------|
| 1  | 011                        | SA_MAN     | 61000         |
| 2  | 011                        | SA_REP     | 250500        |
| 3  | 011                        | (null)     | 311500        |
| 4  | 515                        | AC_ACCOUNT | 8300          |
| 5  | 515                        | AC_MGR     | 12008         |
| 6  | 515                        | AD_ASST    | 4400          |
| 7  | 515                        | AD_PRES    | 24000         |
| 8  | 515                        | AD_VP      | 34000         |
| 9  | 515                        | FI_ACCOUNT | 39600         |
| 10 | 515                        | FI_MGR     | 12008         |
| 11 | 515                        | HR_REP     | 6500          |
| 12 | 515                        | MK_MAN     | 13000         |
| 13 | 515                        | PR_REP     | 10000         |
| 14 | 515                        | PU_CLERK   | 13900         |
| 15 | 515                        | PU_MAN     | 11000         |
| 16 | 515                        | (null)     | 188716        |
| 17 | 590                        | IT_PROG    | 28800         |
| 18 | 590                        | (null)     | 28800         |
| 19 | 603                        | MK_REP     | 6000          |
| 20 | 603                        | (null)     | 6000          |
| 21 | 650                        | SH_CLERK   | 64300         |
| 22 | 650                        | ST_CLERK   | 55700         |
| 23 | 650                        | ST_MAN     | 36400         |
| 24 | 650                        | (null)     | 156400        |

|    |        |            |        |
|----|--------|------------|--------|
| 25 | (null) | AC_ACCOUNT | 8300   |
| 26 | (null) | AC_MGR     | 12008  |
| 27 | (null) | AD_ASST    | 4400   |
| 28 | (null) | AD_PRES    | 24000  |
| 29 | (null) | AD_VP      | 34000  |
| 30 | (null) | FI_ACCOUNT | 39600  |
| 31 | (null) | FI_MGR     | 12008  |
| 32 | (null) | HR_REP     | 6500   |
| 33 | (null) | IT_PROG    | 28800  |
| 34 | (null) | MK_MAN     | 13000  |
| 35 | (null) | MK_REP     | 6000   |
| 36 | (null) | PR_REP     | 10000  |
| 37 | (null) | PU_CLERK   | 13900  |
| 38 | (null) | PU_MAN     | 11000  |
| 39 | (null) | SA_MAN     | 61000  |
| 40 | (null) | SA_REP     | 250500 |
| 41 | (null) | SH_CLERK   | 64300  |
| 42 | (null) | ST_CLERK   | 55700  |
| 43 | (null) | ST_MAN     | 36400  |
| 44 | (null) | (null)     | 691416 |

## 9. Rollup과 Cube 실습

```
SELECT substr(phone_number,1,3),
 JOB_ID,
 SUM(salary)
FROM EMPLOYEES
GROUP BY
 ROLLUP(substr(phone_number,1,3), JOB_ID)
ORDER BY 1, 2;
```

|    | ⚡ SUBSTR(PHONE_NUMBER,1,3) | ⚡ JOB_ID   | ⚡ SUM(SALARY) |
|----|----------------------------|------------|---------------|
| 1  | 011                        | SA_MAN     | 61000         |
| 2  | 011                        | SA_REP     | 250500        |
| 3  | 011                        | (null)     | 311500        |
| 4  | 515                        | AC_ACCOUNT | 8300          |
| 5  | 515                        | AC_MGR     | 12008         |
| 6  | 515                        | AD_ASST    | 4400          |
| 7  | 515                        | AD_PRES    | 24000         |
| 8  | 515                        | AD_VP      | 34000         |
| 9  | 515                        | FI_ACCOUNT | 39600         |
| 10 | 515                        | FI_MGR     | 12008         |
| 11 | 515                        | HR_REP     | 6500          |
| 12 | 515                        | MK_MAN     | 13000         |
| 13 | 515                        | PR_REP     | 10000         |
| 14 | 515                        | PU_CLERK   | 13900         |
| 15 | 515                        | PU_MAN     | 11000         |
| 16 | 515                        | (null)     | 188716        |
| 17 | 590                        | IT_PROG    | 28800         |
| 18 | 590                        | (null)     | 28800         |
| 19 | 603                        | MK_REP     | 6000          |
| 20 | 603                        | (null)     | 6000          |
| 21 | 650                        | SH_CLERK   | 64300         |
| 22 | 650                        | ST_CLERK   | 55700         |
| 23 | 650                        | ST_MAN     | 36400         |
| 24 | 650                        | (null)     | 156400        |
| 25 | (null)                     | (null)     | 691416        |

## 10. Quiz 데이터 생성

- covid19\_insert.sql 파일을 열어 SQL DEVELOPER에서 실행
- covid19 테이블

| 컬럼명                        | 설명              | 데이터형          |
|----------------------------|-----------------|---------------|
| ISO_CODE                   | 국가코드            | VARCHAR2(26)  |
| CONTINENT                  | 대륙명             | VARCHAR2(100) |
| COUNTRY                    | 국가명             | VARCHAR2(500) |
| DATES                      | 일자              | DATE          |
| TOTAL_CASES                | 전체 감염 수         | NUMBER        |
| NEW_CASES                  | 신규 감염 수         | NUMBER        |
| TOTAL_DEATHS               | 전체 사망자수         | NUMBER        |
| NEW_DEATHS                 | 신규 사망자수         | NUMBER        |
| TOTAL_TESTS                | 전체 검사 수         | NUMBER        |
| NEW_TESTS                  | 신규 검사 수         | NUMBER        |
| TOTAL_TESTS_PER_THOUSAND   | 1000명당 전체 검사 비율 | NUMBER        |
| POSITIVE_RATE              | 양성 비율           | NUMBER        |
| POPULATION                 | 인구 수            | NUMBER        |
| HOSPITAL_BEDS_PER_THOUSAND | 1000명당 병상 비율    | NUMBER        |

# 학습정리

- 집계 쿼리는 GROUP BY 절과 집계 함수로 구성된다.
- GROUP BY절과 집계 함수는 단독으로 사용가능하나 의미 있는 결과를 얻으려면 같이 사용하는 것이 좋다.
- 집계 함수의 결과 값으로 조건을 주려면 HAVING 절을 사용한다.
- 집계함수 없이 GROUP BY 절만 사용하면 원하는 컬럼이나 표현식의 고유한 값을 얻을 수 있는데, DISTINCT 키워드를 사용하면 GROUP BY 절 없이 동일한 결과를 얻을 수 있다.



## Quiz

1. `locations` 테이블에는 전 세계에 있는 지역 사무소 주소 정보가 나와 있습니다. 각 국가별로 지역사무소가 몇 개나 되는지 찾는 쿼리를 작성해 보세요.

# Quiz

2. `employees` 테이블에서 년도에 상관 없이 분기별로 몇 명의 사원이 입사했는지 구하는 쿼리를 작성해 보세요.

## Quiz

3. 다음 쿼리는 employees 테이블에서 job\_id별로 평균 급여를 구한 것인데, 여기서 평균을 직접 계산하는 avg\_salary1 이란 가상컬럼을 추가해 보세요. ( 평균 = 총 금액 / 인원수)

```
SELECT job_id, ROUND(AVG(salary),0) avg_salary
FROM employees
GROUP BY job_id
ORDER BY 1;
```

# Quiz

4. COVID19 테이블에서 한국(ISO\_CODE 값이 KOR)의 월별 코로나 확진자 수를 조회하는 문장을 작성하시오.

# Quiz

5. COVID19 테이블에서 한국 데이터에 대해 다음 결과가 나오도록 문장을 작성하시오.

| MONTHS  | 확진자수 | 사망자수 | 사망률  |
|---------|------|------|------|
| 2019-12 | 0    | 0    | 0    |
| 2020-01 | 7    | 0    | 0    |
| 2020-02 | 2924 | 16   | 0.55 |
| 2020-03 | 6855 | 147  | 2.14 |
| 2020-04 | 979  | 84   | 8.58 |
| 2020-05 | 703  | 23   | 3.27 |
| 2020-06 | 1332 | 12   | 0.9  |
| 2020-07 | 1505 | 19   | 1.26 |
| 2020-08 | 5642 | 23   | 0.41 |
| 2020-09 | 3865 | 89   | 2.3  |
| 2020-10 | 2459 | 49   | 1.99 |

월별 사망률 = 사망자 수 / 확진자 수  
(주의: 분모가 0일 경우는 사망률은 0)

## Quiz

6. COVID19 테이블에서 2020년 10월에 가장 많은 확진자가 나온 상위 5개 국가는 어떤 나라일까요?