4. 그룹함수

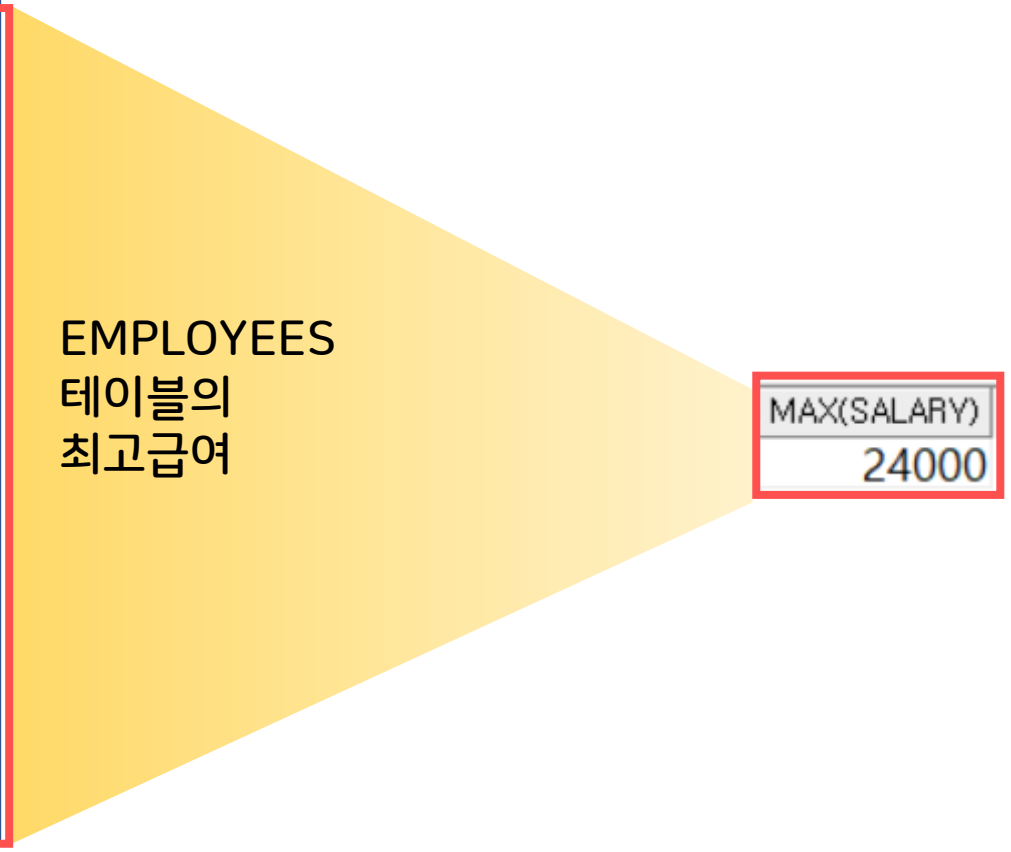
(1) 그룹함수 기초

그룹 함수란?

- 그룹 함수는 행 집합에 작용하여 그룹 당 하나의 결과를 생성합니다.

EMPLOYEES

EMPLOYEE_ID	SALARY
100	24000
101	17000
102	17000
103	9000
104	6000
107	4200
124	5800
141	3500
142	3100
143	2600
144	2500
149	10500
174	11000
176	8600
178	7000
200	4400
201	13000
202	6000
205	12000
206	8300



MAX(SALARY)
24000

그룹 함수 구문 및 종류

- 구문

```
SELECT group_function(column), ...
FROM           table
[WHERE condition]
```

- 그룹 함수의 종류
  - AVG
  - COUNT
  - MAX
  - MIN
  - SUM

AVG 및 SUM 함수 사용

- 숫자 데이터에 AVG 및 SUM을 사용할 수 있습니다.
- 예제는 모든 영업 사원에 대한 월급 평균액, 최고액, 최저액 및 총액을 표시합니다.

```
SELECT AVG(salary), MAX(salary),
       MIN(salary), SUM(salary)
FROM   employees
WHERE  job_id LIKE '%REP%';
```

MIN 및 MAX 함수 사용

- 모든 데이터 유형에 대해 MIN 및 MAX를 사용할 수 있습니다.
- 예제는 최근에 입사한 사원과 가장 오래전에 입사한 사원을 표시합니다.
  - 모든 사원 이름을 영문자순으로 나열했을 때 맨 처음 및 맨 마지막에 해당하는 사원 이름을 표시합니다.

```
SELECT MIN(hire_date), MAX(hire_date)
FROM     employees;
```

COUNT 함수 사용

- COUNT(*)는 테이블의 행 수를 반환합니다.
- COUNT 함수에는 다음 세 가지 형식이 있습니다.
  - COUNT(*)
  - COUNT(expr)
  - COUNT(DISTINCT expr)
- 예제는 전체 사원 수를 표시합니다.

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

COUNT 함수 사용

- COUNT(expr)은 expr에 대해 널이 아닌 값을 가진 행 수를 반환합니다.
- 다음 예제는 EMPLOYEES 테이블에서 80번과 90번 부서의 COMMISSION_PCT 열의 값이 널인 사원을 제외한 사원수를 표시합니다.

```
SELECT COUNT(commission_pct)
FROM   employees
WHERE  department_id = 80;
```

COUNT 함수에 DISTINCT 키워드 사용

- COUNT(DISTINCT expr)은 expr에 대해 중복되지 않는 널이 아닌 값의 수를 반환합니다.
- 다음 예제는 EMPLOYEES 테이블에서 중복되지 않는 부서 값의 수를 표시합니다.

```
SELECT COUNT(DISTINCT department_id)
FROM   employees;
```

그룹 함수와 널 값

- 그룹 함수는 해당 열의 널 값을 무시합니다.
- 다음 예제에서 결과값인 커미션의 평균은 테이블의 COMMISSION_PCT 열에 유효한 값이 저장된 행만으로 계산되어 모든 사원이 받는 커미션 총액을 커미션을 받는 사원 수(4)로 나눈 값입니다.

```
SELECT AVG(commission_pct)
FROM   employees;
```

그룹 함수에 NVL 함수 사용

- NVL 함수는 그룹 함수가 널 값을 포함하도록 강제로 지정합니다.
- 다음 예제에서 평균은 COMMISSION_PCT 열의 널 값 저장 여부에 관계 없이 테이블의 모든 행을 기반으로 계산되며, 모든 사원이 받는 커미션 총액을 회사의 전체 사원 수(20)로 나눈 값이 결과로 출력됩니다.

```
SELECT AVG(NVL(commission_pct, 0))
FROM   employees;
```

데이터 그룹 생성

EMPLOYEES

DEPARTMENT_ID	SALARY
10	4400
20	6000
20	13000
50	2500
50	2600
50	3100
50	3500
50	5800
60	4200
60	6000
60	9000
80	8600
80	10500
80	11000
90	17000
90	17000
90	24000
110	8300
110	12000
(null)	7000

EMPLOYEES
테이블의
부서별
평균급여

DEPARTMENT_ID	TRUNC(AVG(SALARY))
10	4400
20	9500
50	3500
60	6400
80	10033
90	19333
110	10150
(null)	7000

데이터 그룹 생성: GROUP BY 절 구문

- GROUP BY 절을 사용하여 테이블 행을 더 작은 그룹으로 나눕니다.
- 테이블 행을 그룹으로 나눈 후 그룹 함수를 사용하여 각 그룹에 대한 요약 정보를 반환할 수 있습니다.
- 구문

```
SELECT column, group_function(column)
FROM           table
[WHERE condition]
[GROUP BY      group_by_expression]
[ORDER BY      column];
```

GROUP BY 절 사용

- GROUP BY 절을 사용하여 서브그룹을 생성합니다.
- GROUP BY 절에 지정된 열을 SELECT 절에 포함시키지 않아도 결과는 생성됩니다.
- 그러나 다음 예제의 결과처럼 각 부서의 번호는 표시하지 않고 평균급여만 표시하므로 그 결과는 의미가 없습니다.

```
SELECT   AVG(salary)
FROM     employees
GROUP BY department_id ;
```

GROUP BY 절 사용

- 의미있는 결과를 표시하기 위해 GROUP BY 절에 사용한 열은 SELECT 목록에도 포함되어야 합니다.
- 다음 예제는 각 부서의 부서 번호 및 평균 급여를 표시합니다.

```
SELECT   department_id, AVG(salary)
FROM     employees
GROUP BY department_id
ORDER BY AVG(salary);
```

  - ORDER BY 절에 그룹 함수를 사용할 수 있습니다.

여러 열을 기준으로 그룹화

EMPLOYEES

DEPARTMENT_ID	JOB_ID	SALARY
10	AD_ASST	4400
20	MK_MAN	13000
20	MK_REP	6000
50	ST_CLERK	3500
50	ST_CLERK	2500
50	ST_CLERK	2600
50	ST_CLERK	3100
50	ST_MAN	5800
60	IT_PROG	4200
60	IT_PROG	6000
60	IT_PROG	9000
80	SA_MAN	10500
80	SA_REP	11000
80	SA_REP	8600
90	AD_PRES	24000
90	AD_VP	17000
90	AD_VP	17000
110	AC_ACCOUNT	8300
110	AC_MGR	12000
(null)	SA_REP	7000

"EMPLOYEES 테이블을 부서별로 그룹화한 후 각 업무에 대한 급여를 더합니다."

DEPARTMENT_ID	JOB_ID	SUM_SALARY
10	AD_ASST	4400
20	MK_MAN	13000
20	MK_REP	6000
50	ST_CLERK	11700
50	ST_MAN	5800
60	IT_PROG	19200
80	SA_MAN	10500
80	SA_REP	19600
90	AD_PRES	24000
90	AD_VP	34000
110	AC_ACCOUNT	8300
110	AC_MGR	12000
(null)	SA_REP	7000

GROUP BY 절에 여러 열의 사용

- 하나 이상의 GROUP BY 열을 나열하여 그룹 및 하위 그룹에 대한 요약 결과를 반환할 수 있습니다.
- GROUP BY 절에서의 열 순서에 따라 결과는 달라집니다.
- 다음 예제에서 SUM 함수는 각 부서 번호 그룹 내에 있는 업무 ID의 급여 열에 적용됩니다.

```
SELECT   department_id dept_id, job_id, SUM(salary)
FROM     employees
GROUP BY department_id, job_id
ORDER BY 1, 2;
```

그룹 함수를 사용한잘못된 질의

- SELECT 목록의 열 또는 표현식 중 그룹 함수가 아닌 것은 GROUP BY 절에 포함시켜야 합니다.

```
SELECT department_id, COUNT(last_name)
FROM   employees;
```

```
SELECT department_id, COUNT(last_name)
       *
ERROR at line 1:
ORA-00937: not a single-group group function
```

그룹 함수를 사용한 잘못된 질의

- WHERE 절을 사용하여 그룹을 제한할 수 없습니다.
  - WHERE 절에서 그룹 함수를 사용할 수 없습니다.
- HAVING 절을 사용하여 그룹을 제한할 수 있습니다.

```
SELECT   department_id, AVG(salary)
FROM     employees
WHERE    AVG(salary) > 8000
GROUP BY department_id;
```

```
WHERE  AVG(salary) > 8000
       *
ERROR at line 3:
ORA-00934: group function is not allowed here
```

그룹 결과 제외

EMPLOYEES

DEPARTMENT_ID	SALARY
10	4400
20	13000
20	6000
50	5800
50	3500
50	3100
50	2600
50	2500
...	
110	12000
110	8300

각 부서의
최고
급여 중
$10,000를
넘는 급여

DEPARTMENT_ID	MAX(SALARY)
20	13000
80	11000
90	24000
110	12000

그룹 결과 제외: HAVING 절

- HAVING 절을 사용하여 그룹을 제한합니다.
  - 행이 그룹화됩니다.
  - 그룹 함수가 적용됩니다.
  - HAVING 절과 일치하는 그룹이 표시됩니다.

```
SELECT column, group_function
FROM           table
[WHERE condition]
[GROUP BY      group_by_expression]
[HAVINGgroup_condition]
[ORDER BY      column];
```

HAVING 절 사용

- 다음은 최고 급여가 10000이 넘는 부서의 부서 번호 및 최고 급여를 표시합니다.

```
SELECT   department_id, MAX(salary)
FROM     employees
GROUP BY department_id
HAVING   MAX(salary)>10000
ORDER BY 1,2 ;
```

- 다음은 월급 총액 13,000이 넘는 각 업무에 대해 업무 ID와 월급 총액을 표시하되, 영업 사원을 제외시킨 후 월급 총액에 따라 목록을 정렬합니다.

```
SELECT   job_id, SUM(salary) PAYROLL
FROM     employees
WHERE    job_id NOT LIKE 'SA%'
GROUP BY job_id
HAVING   SUM(salary) > 13000
ORDER BY SUM(salary);
```

그룹 함수 중첩

- 그룹 함수는 두 번까지 중첩될 수 있습니다.
- 다음은 최고 평균 급여를 표시합니다.

```
SELECT   MAX(AVG(salary))
FROM     employees
GROUP BY department_id;
```

(2) 그룹함수 확장

GROUP BY에 ROLLUP 및 CUBE 연산자 사용

- ROLLUP 또는 CUBE를 GROUP BY에 사용하여 상호 참조 열에 따라 상위 집계 행을 산출합니다.
- ROLLUP 그룹화는 정규 그룹화 행과 하위 총계 값을 포함하는 결과 집합을 산출합니다.
- CUBE 그룹화는 ROLLUP의 결과 행 및 교차 도표화 행을 포함하는 결과 집합을 산출합니다.

ROLLUP 연산자

- ROLLUP은 GROUP BY 절의 확장 기능입니다.
- ROLLUP 연산을 사용하면 하위 총계와 같은 누적 집계를 산출할 수 있습니다.

```
SELECT [column,] group_function(column). . .
FROM           table
[WHERE condition]
[GROUP BY      [ROLLUP] group_by_expression]
[HAVING        having_expression];
[ORDER BY      column];
```

ROLLUP 연산자 예제

- 아래 예제는 다음을 표시합니다.
  - 각 부서 내 업무별 총급여
  - 부서 별 총급여
  - 전체 사원의 총급여

```
SELECT   department_id, job_id, SUM(salary)
FROM     employees
WHERE    department_id < 60
GROUP BY ROLLUP(department_id, job_id)
ORDER BY 1,2;
```

CUBE 연산자

- CUBE는 GROUP BY 절의 확장 기능입니다.
- CUBE 연산자를 사용하면 하나의 SELECT 문으로 교차 도표화(cross tabulation) 값을 산출할 수 있습니다.

```
SELECT [column,] group_function(column)...
FROM           table
[WHERE condition]
[GROUP BY      [CUBE] group_by_expression]
[HAVING        having_expression]
[ORDER BY      column];
```

CUBE 연산자: 예제

- 예제에 있는 SELECT 문의 출력은 부서 ID가 60보다 작은 부서에 대해 다음과 같은 4가지 결과를 포함합니다.
  - 해당 부서에 속한 업무별 총 급여를 GROUP BY 절을 사용하여 표시
  - 부서별 총 급여
  - 부서와 상관 없는 업무별 총 급여
  - 업무와 상관 없이 부서 ID가 60보다 작은 부서에 대한 전체 총 급여

```
SELECT   department_id, job_id, SUM(salary)
FROM     employees
WHERE    department_id < 60
GROUP BY CUBE (department_id, job_id)
ORDER BY 1,2 ;
```

GROUPING 함수

- GROUPING 함수를 CUBE 또는 ROLLUP 연산자와 함께 사용하면 ROLLUP 또는 CUBE를 통해 만들어진 NULL 값과 저장된 NULL 값을 구별할 수 있습니다.
- GROUPING 함수는 0(표현식 사용)또는 1을 반환합니다.

```
SELECT    [column,] group_function(column), GROUPING(expr)
FROM        table
[WHERE    condition]
[GROUP BY [ROLLUP][CUBE] group_by_expression]
[HAVING   having_expression]
[ORDER BY column];
```

GROUPING 함수: 예제

- 다음 예제는 각 행의 SUM(salary)의 결과에 department_id 열과 job_id열이 고려되었는지 여부를 GROUPING 함수를 사용한 열을 통해 알 수 있습니다.

```
SELECT   department_id DEPTID, job_id JOB, SUM(salary),
         GROUPING(department_id) GRP_DEPT,
         GROUPING(job_id) GRP_JOB
FROM     employees
WHERE    department_id < 50
GROUP BY ROLLUP(department_id, job_id)
ORDER BY 1, 2;
```

  - 0 : 해당 열을 사용하여 그룹함수를 계산했습니다.
  - 1 : 해당 열은 그룹함수 결과에 반영되지 않았습니다.

GROUPING_ID 함수

- GROUPING_ID 함수도 CUBE 또는 ROLLUP 연산자와 함께 사용합니다.
- GROUPING_ID 함수를 사용하여 행에서 하위 총계를 형성한 그룹을 찾을 수 있습니다.
- GROUPING_ID 함수는 여러개의 매개변수를 입력할 수 있습니다.
- GROUPING 함수는 그룹화 비트 백터 값을 십진수로 변환하여 반환합니다.

```
SELECT    [column,] group_function(column), GROUPING_ID(expr,…)
FROM        table
[WHERE    condition]
[GROUP BY [ROLLUP][CUBE] group_by_expression]
[HAVING   having_expression]
[ORDER BY column];
```

GROUPING_ID 함수: 예제

- 다음 예제는 각 행의 SUM(salary)의 결과에 department_id 열과 job_id 열이 고려되었는지 여부를 GROUPING_ID 함수를 사용한 열을 통해 알 수 있습니다.

```
SELECT   department_id DEPTID, job_id JOB, SUM(salary),
         GROUPING(department_id) GRP_DEPT,
         GROUPING(job_id) GRP_JOB,
         GROUPING_ID(department_id, job_id) GRP
FROM     employees
GROUP BY ROLLUP(department_id, job_id)
ORDER BY 1, 2;
```

GROUPING SETS

- GROUPING SETS는 GROUP BY 절의 확장 기능입니다.
- GROUPING SETS를 사용하면 같은 질의에서 여러 그룹화를 정의할 수 있습니다.
- 그룹화 집합을 사용하면 다음과 같은 면에서 효율적입니다.
  - 기본 테이블을 한 번만 검색합니다.
  - 복잡한 UNION 문을 작성할 필요가 없습니다.
  - GROUPING SETS에 요소가 많을수록 성능이 좋아집니다.

GROUPING SETS: 예제

- 질의는 두 그룹화에 대한 집계를 계산합니다. 테이블은 다음 그룹으로 나눠집니다.
  - 부서 ID, 업무 ID
  - 업무 ID, 관리자 ID
- 해당 그룹에 대해 각각 평균 급여가 계산되어 결과 집합에 두 그룹에 대한 평균 급여가 각각 표시됩니다.

```
SELECT   department_id, job_id,
         manager_id,avg(salary)
FROM     employees
GROUP BY GROUPING SETS((department_id,job_id), (job_id,manager_id));
```


Thank you