1. **그룹 쿼리(Aggregation, 데이터를 묶어보자)**

|  |
| --- |
| 그룹핑 쿼리 관련 내용들....  -- [ 그룹핑 관련 쿼리가 필요한 질문들 ]  -- 어떤 사원들에게 얼마의 급여가 지급되고 있나?  -- 직급과 부서에 따른 급여가 얼마인가?  -- 이 회사에서 매월 지급되는 급여의 전체 금액은 얼마인가?  -- 판매 부서 사원들의 총 급여는 얼마인가?  -- 모든 사원의 평균 급여는 얼마인가?  -- 가장 높은 급여를 받는 직급은 무엇인가? |

**1.1 관련 키워드 및 함수**

**- 관련 키워드**

|  |
| --- |
| -- 중복 제거와 전체(default)  -- Distinct와 all 🡺 2개의 결과 비교  **SELECT** **DISTINCT** employee\_id  **FROM** employees;    **SELECT** **DISTINCT** department\_id  **FROM** employees; |
| -- 2개의 결과 비교  **SELECT DISTINCT** department\_id, employee\_id  **FROM** employees;    **SELECT** **ALL** department\_id  **FROM** employees; |

**- 관련 함수 (Aggregate Functions, 집계 함수)**

집계란 하나이상의 데이터들을 대상으로 일종의 통계정보(전체개수, 평균, 최대값 및 최소값)를

말하며 집계함수란 이러한 기능을 수행하는 함수들을 말한다.

이는 두 가지 작업이 필요하다.

1) 어떤 단위로 묶을 것인가(Group by절)

2) 어떤 정보를 추출할 것인가? (그룹 함수의 사용 == 집계함수)

이를 통해서 EMPLOYEES 테이블 전체에 대해 월급에 대한 각종 집계성 데이터들을 추출할 수 있다.

문제점) 특정 범위에서의 집계 데이터를 얻을 수 없다.

예를 들어 부서별로 속한 총 인원수, 부서별 월급 총계,

직급별 월급 평균 등 '~별' 로 데이터를 추출할 수 없는 문제가 있다.

이는 그룹핑과 관련된다.

1) **COUNT** : 테이블에 있는 데이터(로우)의 개수를 반환 한다.

|  |
| --- |
| **SELECT** **COUNT**(\*)  **FROM** employees;  // 1row 출력 |
| ==> 위와 동일 결과  **SELECT** **COUNT**(\*) col1,  **COUNT**(employee\_id) col2  **FROM** employees;  // 1row 출력 |
| ==> 위와 동일 결과  **SELECT** **COUNT**(**ALL** employee\_id)  **FROM** employees; |
| ==> 위와 동일 결과  **SELECT** **COUNT**( **ALL** employee\_id),  **COUNT**( **ALL** first\_name)  **FROM** employees; |
| ==> 중복값을 제거하고 COUNT하는 구문  **SELECT** **COUNT**( **DISTINCT** employee\_id),  **COUNT**( **DISTINCT** first\_name)  **FROM** employees; |
| ==> 만약 department\_id에 NULL이 있다면 리턴 값은 달라지게 된다.  COUNT함수는 NULL을 인식하지 못한다. 따라서 그 연산 결과에 포함하지 않는다.  **SELECT** **COUNT**(department\_id),  **COUNT**( **DISTINCT** department\_id)  **FROM** employees;  \*) 만약 '\*' 가 파라미터로 전달될 경우 NULL이나 중복값을 모두 포함하여 결과를 반환한다. |

2) **SUM** : 파라미터로 들어오는 전체 합계 계산 결과를 반환한다.

파라미터로 반드시 숫자형 데이터 타입의 값이나 컬럼이 와야 한다.

|  |
| --- |
| // 모든 사원의 총 월급의 합계  **SELECT** **SUM**(salary)  **FROM** EMPLOYEES |
| // example  CREATE TABLE sum\_test (  Col1 varchar2(10),  Col2 number);  INSERT INTO sum\_test VALUES('100', 100);  INSERT INTO sum\_test VALUES('200', 200);  INSERT INTO sum\_test VALUES('300', 300);  INSERT INTO sum\_test VALUES('400', 400); |
| **SELECT** **SUM**(col1),  **SUM**(col2)  **FROM** sum\_test;  [ 출력결과 ]  SUM(COL1) SUM(COL2)  ------------------ -------------  1000 1000 |
| **SELECT** **SUM**(DISTINCT salary),  **SUM**(salary)  **FROM** employees;  [ 출력결과]  SUM((DISTINCT ) SUM(salary )  ------------------ -------------  397900 691400  \*)SUM 함수에서 DISTINCT 를 사용하면 SALARY 컬럼값이 중복되지 않는 값만  을 대상으로 해서 그 합계를 구한다. |

3) **MAX와 MIN** : 파라미터로 들어온 expr의 최대값 / 최소값

|  |
| --- |
| // 가장 높은 급여와 가장 낮은 급여  **SELECT** **MAX**(salary), **MIN**(salary)  **FROM** employees; |
| // 해당 급여를 받는 사람 출력  **SELECT** employee\_id, first\_name, last\_name, job\_id, salary  **FROM** employees  **WHERE** salary **in** ( 24000, 2100 ); |
| // 한 문장으로 구성이 가능할까?  // 그룹 함수 사용 불가 에러가 발생한다.  // why? WHERE 절에는 집계 함수 사용이 불가능하다. 대안은 서브쿼리......  **SELECT** employee\_id, first\_name, last\_name, job\_id, salary  **FROM** employees  **WHERE** salary = **MAX**(salary); |
| // MAX나 MIN 함수도 파라미터에 DISTINCT 나 ALL을 사용할 수 있다.  // 사용결과는 같다. => 당연하다.  **SELECT** **MAX**(salary), **MAX**(DISTINCT salary), **MIN**(salary), **MIN**(**DISTINCT** salary)  **FROM** employees; |

4) **AVG** : expr의 평균값을 구해 반환하는 함수

|  |
| --- |
| // 회사의 월 평균 급여  **SELECT** **AVG**(salary)  **FROM** employees; |
| // 중복값 제외  **SELECT** **AVG**(**DISTINCT** salary)  **FROM** employees; |
| // 평균값을 구하는 공식을 사용해 직접 구하는 것도 가능  **SELECT** **AVG**(salary) avg1,  **SUM**(salary) / **COUNT**(salary) avg2  **FROM** employees; |

5) **STDDEV와 VARIANCE** : 표준편차와 분산을 구하는 함수

|  |
| --- |
| **SELECT** **STDDEV**(salary),  **VARIANCE**(salary)  **FROM** employees; |

**1.2 GROUP BY절**

전체가 아닌 특정 범위에서 집계 데이터를 얻는 방법

- 각 부서별 속한 총 인원수

- 부서별 월급 총계

- 직급별 월급 평균

Ex) department\_id에 따라 그룹핑 한다.

|  |
| --- |
| **SELECT** department\_id  **FROM** employees  **GROUP BY** department\_id; |

\*) 위의 결과는 DISTINCT 키워드를 사용한 결과와 동일함

DISTINCT와의 차이점 : (D)중복을 제거한 결과를 반환

(G)부서별 그룹을 지은것임

Ex) 부서별 급여 총계

|  |
| --- |
| **SELECT** department\_id, **SUM**(salary)  **FROM** employees  **GROUP BY** department\_id; |

Ex) 부서별 사원수, 평균 급여

|  |
| --- |
| **SELECT** department\_id, **SUM**(salary), **COUNT**(salary), **AVG**(salary)  **FROM** employees  **GROUP BY** department\_id; |

Ex) 오류발생

|  |
| --- |
| **SELECT** department\_id, job\_id, **SUM**(salary), **AVG**(salary)  **FROM** employees  **GROUP BY** department\_id;  *ERROR report : SQL Error : ORA-0979: GROUP BY 표현식이 아닙니다.* |

\*) SELECT 문장에서 GROUP BY절을 사용할 경우에는 SELECT리스트에 있는 항목 중 집계 함수

를 제외하고는 모든 항목이 GROUP BY절에 명시되어 있어야 한다.

job\_id는 GROUP BY절이 아니다.

Ex) 위의 오류를 해결

|  |
| --- |
| **SELECT** department\_id, job\_id, **SUM**(salary), **AVG**(salary)  **FROM** employees  **GROUP BY** department\_id, job\_id; |

\*) 하지만 쿼리의 결과가 모호하다.

가독성을 높이기위한 직급별 정렬 작업 필요

Ex) 위의 쿼리에서 해당 부서별, 직급별 정렬 작업 추가

|  |
| --- |
| **SELECT** department\_id, job\_id, **SUM**(salary), **AVG**(salary)  **FROM** employees  **GROUP BY** department\_id, job\_id  **ORDER BY** department\_id, job\_id; |

Ex) 위의 쿼리에 가독성 추가 쿼리의 맨 앞에 회사명 출력

|  |
| --- |
| **SELECT** 'ORACEL' company, department\_id, job\_id,  **TO\_CHAR**( **SUM**(salary), '999,999') tot\_sal,  **TO\_CHAR**(**SUM**(salary), '999,999') avg\_sal  **FROM** employees  **GROUP BY** department\_id, job\_id  **ORDER BY** department\_id, job\_id; |

\*) 상수값은 GROUP BY절에 없어도 오류 발생이 되지 않는다.

월급총합과 평균 금액을 이전 장에서 본 TO\_CHAR함수를 사용하여 일정한 포맷을 주어 가독

성을 높일 수 있다.

Ex) GROUP BY절과 WHERE절

만약 월급여가 가장 높았던 부서인 판매부서의 총 급여액, 평균값을 구하고자 할 경우

where절에서 판매부서인 사원의 정보만 가져오도록 하는 조건을 명시한다.

|  |
| --- |
| **SELECT** job\_id, **TO\_CHAR**(**SUM**(salary), '999,999') tot\_sal,  **TO\_CHAR(SUM**(salary), '999,999') avg\_sal  **FROM** employees  **WHERE** department\_id = 80  **GROUP BY** job\_id  **ORDER BY** job\_id; |

정리)

- SQL 함수에 포함되는 집계 함수는 말 그대로 데이터를 집계하는 기능을 수행하는 함수로,

COUNT, SUM, MAX, MIN, AVG, STDDEV, VARIANCE 등이 있다.

- 특정 컬럼값이나 표현식을 단위로 집계성 데이터를 보기 위해서는 SELECT 문장에서 GROUP BY 절과 함께 집계 함수를 사용한다.

- 집계 함수 이외에 SELECT 절에 오는 컬럼이나 값은 상수를 제외하고 모두 GROUP BY

절에 명시되어야 한다.

- GROUP BY 절에는 한 개 이상의 컬럼이나 값이 올 수 있다.

- 그룹핑한 결과를 일정한 순서로 보고자 할 경우 ORDER BY 절을 사용한다.

ORDERBY 절은 GROUP BY절 다음에 위치해야 하며, ORDER BY절에 명시하는 컬럼이나

값은 GROUP BY절에서 명시한 컬럼이나 값의 순서와는 달라도 상관없으나 GROUP BY절에

명시한 컬럼이나 값이외의 것은 올 수 없다.

- GROUP BY 절을 사용한 SQL 문장에서도 WHERE조건을 사용할 수 있으며, 이 때 WHERE

절은 GROUP BY절 앞에 위치해야 한다.

또한 WHERE 조건절에 사용된 컬럼이나 값은 GROUP BY 절에 명시된 컬럼이나 값과 달라

도 상관없다. 즉 처리순서로 본다면 해당 문장에서 WERE 조건이 가장 먼저 처리되고

그 다음에 GROUP BY및 집계함수가 처리된다.

**1.3 Having 절**

GROUP BY절과 사용되어 SELECT 문장에서 집계 함수를 사용할 조건을 명시하는 절이다.

즉 일반 조건은 WHERE 절에 명시하지만 집계함수의 결과가 필요한 경우에 한해 이를 HAVING

절에 명시한다.

-- HAVING 절

-- GROUP BY절을 사용한 SELECT 문장에서 일정한 조건을 주려면 WHERE조건을 명시하면 된다.

Ex) 현재 어느 부서에 몇 명의 사원들이 속해있는지 알고 싶다면?

|  |
| --- |
| **SELECT** department\_id, **COUNT**(\*)  **FROM** EMPLOYEES  **WHERE** department\_id **IS NOT NULL**  **GROUP BY** department\_id  **ORDER BY** department\_id; |

\*) 위의 결과는 전 사원 중 부서에 소속되어 있지 않은 사원이 현재 1명이 존재하므로

WHERE조건에서 DEPARTMENT\_ID값이 NULL이 아닌 건을 조회하라는 조건을 준 것이다.

Ex) 그렇다면 부서에 소속된 사원이 5명 이하인 부서만 조회하고자 할 경우?

|  |
| --- |
| **SELECT** department\_id, **COUNT**(\*)  **FROM** EMPLOYEES  **WHERE** department\_id **IS NOT NULL**  **AND** **COUNT**(\*) <= 5  **GROUP BY** department\_id  **ORDER BY** department\_id; |

\*) 문장을 실행하면 AND COUNT(\*) <=5 에서 오류가 발생한다.

오류는 그룹함수는 허가되지 않습니다.

즉, GROUP 함수는 WHERE절에 올 수 없다.(문법)

-- HAVING절은 GROUP BY절과 같이 사용되어 SELECT 문장에서 집계 함수를 사용한 조건을

명시하는 절이다.

-- 즉 일반 조건은 WHERE절에 명시하지만 우리가 지금 구하고자 하는 쿼리와 같은 조건에

-- 집계 함수의 결과가 필요한 경우에 한해 이를 HAVING절에 명시한다.

Ex) 수정 코드

|  |
| --- |
| **SELECT** department\_id, **COUNT**(\*)  **FROM** EMPLOYEES  **WHERE** department\_id **IS NOT NULL**  **GROUP BY** department\_id  **HAVING COUNT**(\*) <= 5  **ORDER BY** department\_id; |

Ex) HAVING절은 GROUP BY절없이 단독으로 사용될 수도 있다.

|  |
| --- |
| **SELECT COUNT**(\*)  **FROM** employees  **HAVING COUNT**(\*) < 10; |

- HAVING절 정리

1) HAVING절은 WHERE 조건 이외에 집계 함수의 결과로 조건을 주고자 할 때 사용되며,

GROUP BY절과 함께 사용된다.

2) HAVING절에는 일반적으로 집계 함수나 상수가 사용된 조건을 명시하는데 GROUP BY절

에 명시한 컬럼도 다음과 같이 조건으로 명시할 수는 있다.

하지만 집계함수는 오직 HAVING절에서만 조건으로 사용될 수 있다.

**SELECT** department\_id, **COUNT**(\*)

**FROM** employees

**WHERE** department\_id **IS NOT NULL**

**GROUP BY** department\_id

**HAVING** department\_id > 30;

3) WHERE절과 HAVING절을 동시에 사용할 수 있는데, 일반적인 조건은 WHERE절에 명시

하고 집계함수가 포함된 조건은 HAVING절에 명시한다.

위의 예제에서 department\_id > 30;은 HAVING이 아닌 WHERE에 추가해야 한다.

4) HAVING절의 위치는 GROUP BY절의 이전 혹은 이후 어디에나 올 수 있다.

위치는 결과에 아무런 영향을 주지 않는다.

**1.4 ROLLUP & CUBE**

함수로서 GROUP BY절에 사용되어 추가적인 그룹핑 정보를 표현하게 된다.

ROLLUP : 그룹핑된 결과에 그룹별 합계 정보를 추가한다.

CUBE : 그룹핑 된 컬럼의 모든 가능한 조합에 대한 합계 정보를 추가한다.

Ex) 부서가 속한 도시, 부서명, 직급별로 소속된 사원 수와 그들의 월급 합계를 보여주는 쿼리문

|  |
| --- |
| **SELECT** c.city, b.department\_name, a.job\_id, **COUNT**(\*) persons, **SUM**(a.salary) tot\_sal  **FROM** employees a, departments b, locations c  **WHERE** a.department\_id = b.department\_id  **AND** b.location\_id = c.location\_id  **GROUP BY** c.city, b.department\_name, a.job\_id  **ORDER BY** c.city, b.department\_name, a.job\_id; |
| [ 결과 ]  CITY DEPARMENT\_NAME JOB\_ID PERSONS TOT\_SAL  -------------------------------------------------------------------------------------------------  런던 인사부 HR\_REP 1 6,500  시애틀 경리 AC\_ACCOUNT 1 1,000  시애틀 경리 AC\_MGR 1 2,000  시애틀 기획 FI\_ACCOUNT 3 4,000  시애틀 기획 FI\_ACCOUNT 4 3,000 |

Ex) ROLLUP 추가 , ROLLUP 대상으로 도시명, 부서명, 직급을 명시함

|  |
| --- |
| **SELECT** c.city,b.department\_name,a.job\_id, **COUNT**(\*) persons, **TO\_CHAR**(**SUM**(a.salary),'999,999') tot\_sal  **FROM** employees a, departments b, locations c  **WHERE** a.department\_id = b.department\_id  **AND** b.location\_id = c.location\_id  **GROUP BY ROLLUP**(c.city, b.department\_name, a.job\_id)  **ORDER BY** c.city, b.department\_name, a.job\_id; |
| [ 결과 ]  CITY DEPARMENT\_NAME JOB\_ID PERSONS TOT\_SAL  -------------------------------------------------------------------------------------------  런던 인사부 HR\_REP 1 6,500  런던 인사부 1 6,500  런던 1 6,500  시애틀 경리 AC\_ACCOUNT 1 1,000  시애틀 경리 AC\_MGR 1 2,000  시애틀 경리 2 3,000  시애틀 기획 FI\_ACCOUNT 3 4,000  시애틀 기획 FI\_ACCOUNT 4 3,000  시애틀 기획 7 7,000  시애틀 9 10,000 |

Ex) ROLLUP 대신 CUBE를 사용

|  |
| --- |
| **SELECT** c.city, b.department\_name, a.job\_id, **COUNT**(\*) persons,  **TO\_CHAR(SUM**(a.salary),'999,999') tot\_sal  **FROM** employees a, departments b, locations c  **WHERE** a.department\_id = b.department\_id  **AND** b.location\_id = c.location\_id  **GROUP BY CUBE**(c.city, b.department\_name, a.job\_id)  **ORDER BY** c.city, b.department\_name, a.job\_id; |
| **SELECT** c.city, b.department\_name, a.job\_id, **COUNT**(\*) persons,  **TO\_CHAR(SUM**(a.salary),'999,999') tot\_sal  **FROM** employees a, departments b, locations c  **WHERE** a.department\_id = b.department\_id  **AND** b.location\_id = c.location\_id  **AND** c.city = '런던'  **GROUP BY CUBE**(c.city, b.department\_name, a.job\_id)  **ORDER BY** c.city, b.department\_name, a.job\_id; |

\*) ROLLUP과 CUBE는 그룹핑 쿼리에서 사용되는 함수로 GROUP BY절에 위치해서

파라미터로 들어온 컬럼에 대한 소계를 산출한다.

\*) 파라미터 개수가 n개라 하면 ROLLUP은 n+1개의 결과를 추출하며, CUBE는 2의 n승 개의

결과를 추출한다.

예를 들어 위 쿼리와 같이 ROLLUP이나 CUBE의 파라미터로 도시명, 부서명, 직급 3개의

컬럼이 명시된 경우 ROLLUP은 3+1 = 4, CUBE는 2\*2\*2=8개의 결과가 나오게 된다.

\*) ROLLUP과 CUBE를 사용하면 원하는 집계 정보 이외에 세부 항목별 소계까지 산출할 수 있다.

==> ROLLUP과 CUBE는 애플리케이션에서 처리하는 경우가 많다.