# 9. 분석함수

# (1) 윈도우 함수

# WINDOW FUNCTION 개요

- 관계형 데이터베이스의 행과 행간의 관계를 쉽게 정의하기 위해 만든 함수입니다.
- 복잡한 프로그램을 하나의 SQL 문장으로 쉽게 해결할 수 있습니다.
- 데이터웨어하우스에서 발전한 기능입니다.
  - 분석 함수(ANALYTIC FUNCTION) 또는 순위 함수(RANK FUNCTION)라고도 알려져 있습니다.
  - ANSI/ISO SQL 표준용어는  WINDOW FUNCTION 입니다.
- 기존에 사용하던 그룹함수에  WINDOW 함수 전용으로 만들어진 기능이 포함되어 있습니다.
- 중첩(NEST)해서 사용하지는 못하지만, 서브쿼리에서 사용할 수 있습니다.

# WINDOW FUNCTION 종류

| 종류 | 함수이름 | 지원 DBMS |
|---|---|---|
| 그룹 내 순위(RANK) 관련 함수 | RANK, DENSE_RANK, ROW_NUMBER | ANSI/ISO SQL 표준과 Oracle, SQL Server 등 대부분의 DBMS에서 지원 |
| 그룹 내 집계(AGGREGATE) 관련 함수 | SUM, MAX, MIN, AVG, COUNT | ANSI/ISO SQL 표준과 Oracle, SQL Server 등 대부분의 DBMS에서 지원 |
| 그룹 내 행 순서 관련 함수 | FIRST_VALUE, LAST_VALUE, LAG, LEAD | Oracle에서만 지원 |
| 그룹 내 비율 관련 함수 | CUME_DIST, PERCENT_RANK, NTILE, RATIO_TO_REPORT | CUME_DIST, PERCENT_RANK 함수는 ANSI/ISO SQL 표준<br>NTILE, RATIO_TO_REPORT 함수는 Oracle에서만 지원 |

# WINDOW FUNCTION 구문

- WINDOW 함수에는 OVER 문구가 키워드로 필수 포함됩니다.

```
SELECT WINDOW_FUNCTION (ARGUMENTS) OVER ( [PARTITION BY column] [ORDER BY  ]
[WINDOWING  ] ) FROM table;
```

  - ARGUMENTS (인수) : 함수에 따라 0 ~ N개의 인수 지정
  - PARTITION BY 절 : 전체 집합을 기준에 의해 소그룹으로 나눔
  - ORDER BY 절 : 어떤 항목에 대해 순위를 지정할 지 ORDER BY 절을 기술
  - WINDOWING 절 : 함수의 대상이 되는 행 기준의 범위를 ROWS 또는 RANGE로 지정
    - ROWS는 물리적인 결과 행의 수
    - RANGE는 논리적인 값에 의한 범위

# 그룹 내 순위 함수

- RANK
  - ORDER BY를 포함한 QUERY 문에서 특정 항목(열)에 대한 순위를 구하는 함수입니다.
  - 특정 범위(PARTITION) 내에서  또는 전체 데이터에 대한 순위를 구할 수도 있습니다.
  - 동일한 값에 대해서는 동일한 순위를 부여하게 됩니다.
  - 다음은 사원 데이터에서 급여가 높은 순서로 순위를 같이 출력합니다.
    - 결과가 파티션의 기준이 된 job_id와  salary 별로 정렬이 되어 있는 것을 알 수 있습니다.

```
SELECT job_id, last_name, salary,
            RANK( ) OVER (ORDER BY salary DESC) ALL_RANK
FROM employees;
```

# 그룹 내 순위 함수

- DENSE_RANK
  - RANK 함수와 유사하나, 동일한 순위를 하나의 건수로 취급하는 것이 다른 점입니다.
  - 다음 예제는 사원데이터에서 RANK 함수를 사용하여 급여가 높은 순서와, DENSE_RANK 함수를 사용하여 동일한 순위를 하나의 등수로 간주한 결과를  출력합니다.

```
SELECT job_id, last_name, salary,
           RANK( ) OVER (ORDER BY salary DESC) RANK,
           DENSE_RANK( ) OVER (ORDER BY salary DESC) DENSE_RANK
FROM employees;
```

# 그룹 내 순위 함수

- ROW_NUMBER
  - 이 함수는 동일한 값이라도 고유한 순위를 부여합니다.
- 다음 예제는 사원데이터에서 RANK 함수를 사용하여 급여가 높은 순서와, ROW_NUMBER 함수를 사용하여 동일한 순위를 인정하지 않는 등수를 함께 출력합니다.

```
SELECT job_id, last_name, salary,
            RANK( ) OVER (ORDER BY salary DESC) RANK,
            ROW_NUMBER() OVER (ORDER BY salary DESC) ROW_NUMBER
FROM employees;
```

# 일반 그룹 함수

- SUM
  - SUM 함수를 이용해 파티션별 윈도우의 합을 구할 수 있습니다.
- 다음 예제는 사원들의 급여와 같은 매니저를 두고 있는 사원들의 SALARY 합을 함께표시합니다.
  - PARTITION BY manager_id 구문을 통해 매니저별로 데이터를 파티션화 하고 있습니다.

```
SELECT manager_id, last_name, salary,
          SUM(salary) OVER (PARTITION BY manager_id) manager_id_SUM
FROM employees;
```

# 일반 그룹 함수

- SUM의 누적값 출력
- OVER 절 내에 ORDER BY 절을 추가해 파티션 내 데이터를 정렬하고 이전 SALARY 데이터까지의 누적값을 출력할 수 있습니다.
- RANGE UNBOUNDED PRECEDING을 사용하여 현재 행을 기준으로 파티션 내의 첫 번째 행까지의 범위를 지정합니다.

```
SELECT manager_id, last_name, salary,
            SUM(salary) OVER (PARTITION BY manager_id ORDER BY
                salary RANGE UNBOUNDED PRECEDING) as manager_id_SUM
FROM employees;
```

# 일반 그룹 함수

- MAX & MIN
  - MAX와 MIN 함수를 이용해 파티션별 윈도우의 최대값과 최소값을 구할 수 있습니다.
- 다음 예제는 사원들의 급여와 같은 매니저를 두고 있는 사원들의 SALARY 중 최대값과 최소값을 함께 표시합니다.

```
SELECT manager_id, last_name, salary,
       MAX(salary) OVER (PARTITION BY manager_id) as manager_MAX,
       MIN(salary) OVER(PARTITION BY manager_id ) as manager_MIN
FROM employees;
```

# 일반 그룹 함수

- AVG
  - 파티션별 윈도우의 평균값을 구합니다.
- 다음은 사원 테이블에서 같은 매니저를 두고 있는 사원들에 대하여 현재행까지의 평균급여를 함께 표시합니다.

```
SELECT manager_id, last_name, hire_date, salary,
       ROUND(AVG(salary) OVER (PARTITION BY manager_id
       ORDER BY hire_date)) as manager_id_AVG
FROM employees;
```

# 일반 그룹 함수

- AVG 결과에 범위 지정
- AVG 함수와 파티션별 ROWS 윈도우를 이용해 원하는 조건에 맞는 데이터에 대한 통계값을 구할 수 있습니다.
- ROWS BETWEEN 절을 사용하여 현재 행을 기준으로 파티션 내에서 범위로 지정할 수 있습니다.

```
SELECT manager_id, last_name, hire_date, salary,
       ROUND (AVG(salary) OVER (PARTITION BY manager_id ORDER BY
       hire_date ROWS BETWEEN 1 PRECEDING AND 1 FOLLOWING))
       AS manager_id_avg
FROM employees;
```

ROWS는 현재 행을 기준으로 파티션 내에서 앞뒤 건수를 범위로 지정

# 일반 그룹 함수

- COUNT
  - COUNT 함수와 파티션별 ROWS 윈도우를 이용해 원하는 조건에 맞는 데이터에 대한 통계값을 구할 수 있습니다.
- 다음 예제는 사원들을 급여 기준으로 정렬하고, 본인의 급여보다 50 이하로 적거나 150 이하로 많은 급여를 받는 인원수를 함께 출력합니다.

```
SELECT last_name, salary,
          COUNT(*) OVER (ORDER BY salary
          RANGE BETWEEN 50 PRECEDING AND 150 FOLLOWING) as SIM_CNT
          FROM employees;
```

RANGE는 현재 행의 데이터 값을 기준으로 앞뒤 데이터 값의 범위를 표시

# 그룹 내 행순서 함수

- FIRST_VALUE
  - 파티션별 윈도우에서 가장 먼저 나온 값을 구할 수 있습니다.
- 다음 예제는 부서별 직원들을 연봉이 높은 순서부터 정렬하고, 파티션 내에서 가장 먼저 나온 값을 함께 출력합니다.

```
SELECT department_id, last_name, salary,
       FIRST_VALUE(last_name) OVER (PARTITION BY department_id
       ORDER BY salary DESC ROWS UNBOUNDED PRECEDING) as DEPT_RICH
FROM employees;
```

현재 행을 기준으로 파티션 내의 첫 번째 행까지의 범위를 지정

# 그룹 내 행 순서 함수

- LAST_VALUE
  - 파티션별 윈도우에서 가장 나중에 나온 값을 구할 수 있습니다.
- 다음 예제는 부서별 직원들을 연봉이 높은 순서부터 정렬하고, 파티션 내에서 가장 마지막에 나온 값을 함께 출력합니다.

```
SELECT department_id, last_name, salary,
       LAST_VALUE(last_name) OVER (PARTITION BY department_id
       ORDER BY salary DESC
       ROWS BETWEEN CURRENT ROW AND UNBOUNDED FOLLOWING)
       as DEPT_POOR
FROM employees;
```

현재 행을 포함해서 파티션 내의 마지막 행까지의 범위를 지정

# 그룹 내 행 순서 함수

- LAG
  - 파티션별 윈도우에서 이전 몇 번째 행의 값을 가져올 수 있습니다.
- 다음 예제는 직원들을 입사일자가 빠른 기준으로 정렬을 하고, 본인보다 입사일자가 한 명 앞선 사원의 입사일자를 함께 출력합니다.

```
SELECT last_name, salary, hire_date,
           LAG(hire_date) OVER (ORDER BY hire_date) as PREV_salary
FROM employees ;
```

# 그룹 내 행 순서 함수

- LEAD
  - 파티션별 윈도우에서 이후 몇 번째 행의 값을 가져올 수 있습니다.
- 다음 예제에서 직원들을 입사일자가 빠른 기준으로 정렬을 하고, 바로 다음에 입사한 인력의 입사일자를 함께 출력합니다.

```
SELECT last_name, hire_date,
          LEAD(hire_date) OVER (ORDER BY hire_date) as "NEXTHIRED"
FROM employees;
```

# 그룹 내 비율 함수

- RATIO_TO_REPORT
  - 파티션 내 전체 SUM값에 대한 행별 열 값의 백분율을 소수점으로 구할 수 있습니다.
  - 결과 값은 > 0 & <= 1 의 범위를 가지며 개별 RATIO의 합을 구하면 1이 됩니다.
- 예제는 job_id가 IT_PROG인 사원들을 대상으로 전체 급여에서 본인이 차지하는 비율을 출력합니다.

```
SELECT last_name, job_id, salary,
       ROUND(RATIO_TO_REPORT(salary) OVER (), 2) as R_R
FROM employees
WHERE job_id = 'IT_PROG';
```

# 그룹 내 비율 함수

- PERCENT_RANK
  - 파티션별 윈도우에서 제일 먼저 나오는 것을 0으로, 제일 늦게 나오는 것을 1로 하여, 값이 아닌 행의 순서별 백분율을 구할 수 있습니다.
  - 결과 값은 >= 0 & <= 1 의 범위를 가집니다.
- 다음 예제는 같은 부서 소속 사원들의 집합에서 본인의 급여가 순서상 몇 번째 위치쯤에 있는지 0과 1 사이의 값으로 출력합니다.

```
SELECT department_id, last_name, salary,
       PERCENT_RANK() OVER (PARTITION BY department_id
       ORDER BY salary DESC) as P_R
FROM employees;
```

# 그룹 내 비율 함수

- CUME_DIST
  - 파티션별 윈도우의 전체건수에서 현재 행보다 작거나 같은 건수에 대한 누적백분율을 구할 수 있습니다.
  - 결과 값은 > 0 & <= 1 의 범위를 가집니다.
- 다음 예제는 같은 부서 소속 사원들의 집합에서 본인의 급여가 누적 순서상 몇 번째 위치쯤에 있는지 0과 1 사이의 값으로 출력합다.

```
SELECT department_id, last_name, salary,
          ROUND(CUME_DIST() OVER (PARTITION BY department_id
          ORDER BY salary DESC),2) as CUME_DIST
FROM employees;
```

# 그룹 내 비율 함수

- NTILE
  - 파티션별 전체 건수를 ARGUMENT 값으로 N 등분한 결과를 구할 수 있습니다.
- 다음 예제는 전체 사원을 급여가 높은 순서로 정렬하고, 급여를 기준으로 4개의 그룹으로 분류합니다. 나머지는 앞의 조부터 할당됩니다.

```
SELECT last_name, salary,
       NTILE(4) OVER (ORDER BY salary DESC) as QUAR_TILE
FROM employees;
```

# (3) LISTAGG와 Pivot 함수

# LISTAGG 함수

- 데이터를 그룹화 한 상태에서 각 그룹안에 특정 필드값을 한 행으로 출력합니다.
  - 여러 행의 열 값을 한 행의 값으로 가져와야할 때 사용하는 그룹함수 입니다.
  - 그룹화된 개별 데이터를 하나의 열에 가로로 출력합니다.
- 구문

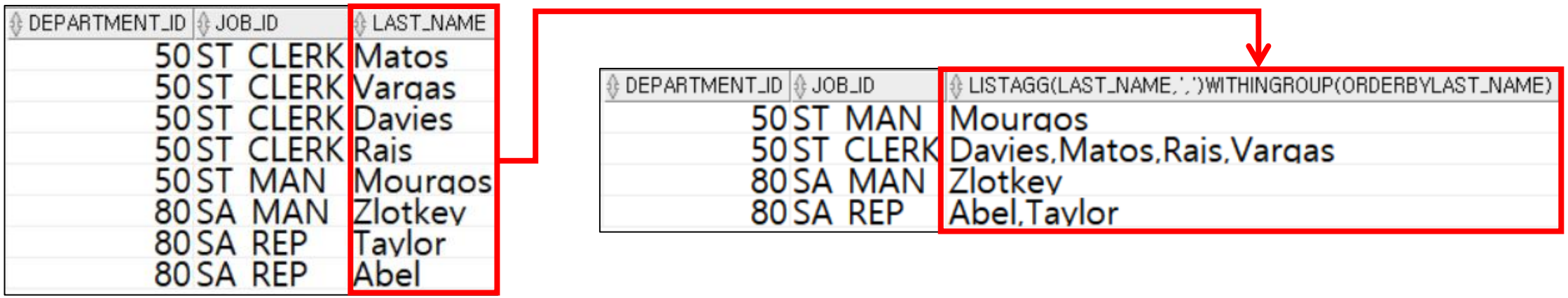
```
SELECT LISTAGG(column1, 구분자) WITHIN GROUP (ORDER BY column2)
FROM table
GROUP BY column_name;
```

  - LISTAGG 함수의 인수인 column1은 가로로 나열할 열 이름입니다.
  - ORDER BY 절을 사용하여 가로로 출력될 데이터를 정렬합니다.

# LISTAGG 함수 : 예제

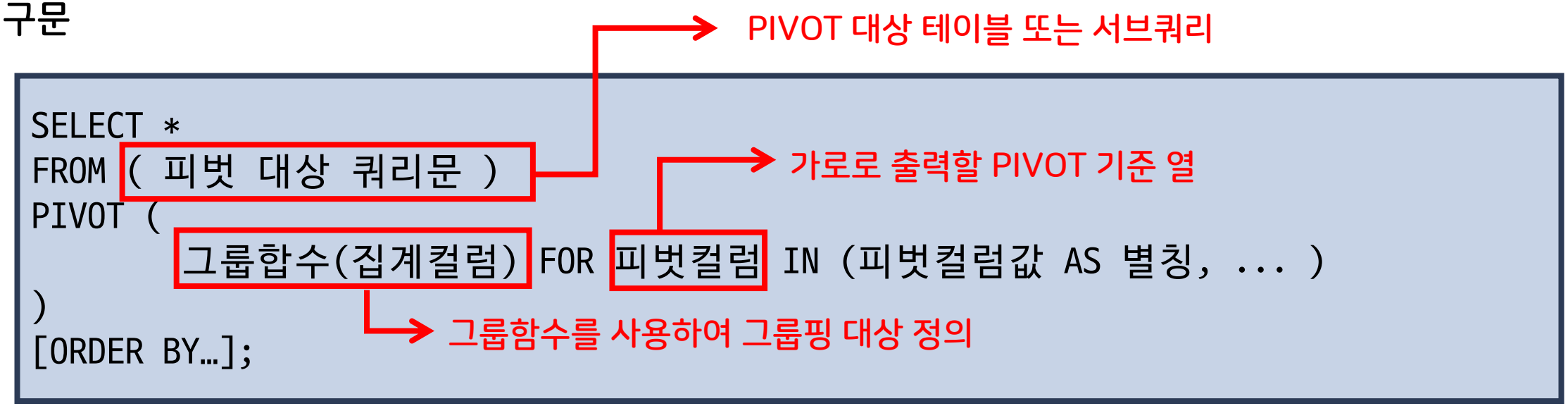
- 다음 예제는 부서별 인원 수와 함께 부서의 사원이름을 함께 출력합니다.
- 출력된 사원 이름은 부서 내에서 급여가 높은 순서 입니다.

```
SELECT department_id, COUNT(*),
       LISTAGG(last_name, ',') WITHIN GROUP(ORDER BY salary DESC)
       AS ename
FROM employees
GROUP BY department_id;
```



# PIVOT 함수

- 통계에서 많이 사용하며, 행을 열로 변환해주는 함수입니다.
- 구문



# PIVOT 함수의 이해

DEPARTMENT_ID PIVOT

| DEPARTMENT_ID | JOB_ID | MAX(SALARY) |
|---|---|---|
| 50 | ST_CLERK | 3500 |
| 50 | ST_MAN | 5800 |
| 60 | IT_PROG | 9000 |
| 80 | SA_MAN | 10500 |
| 80 | SA_REP | 11000 |

| JOB_ID | 50 | 60 | 80 |
|---|---|---|---|
| IT_PROG | (null) | 9000 | (null) |
| ST_MAN | 5800 | (null) | (null) |
| SA_MAN | (null) | (null) | 10500 |
| SA_REP | (null) | (null) | 11000 |
| ST_CLERK | 3500 | (null) | (null) |

JOB_ID PIVOT

```
SELECT department_id, job_id, MAX(salary)
FROM employees
WHERE department_id IN (50,60,80)
GROUP BY department_id, job_id
ORDER BY 1,2;
```

| DEPARTMENT_ID | 'IT_PROG' | 'SA_MAN' | 'SA_REP' | 'ST_CLERK' | 'ST_MAN' |
|---|---|---|---|---|---|
| 50 | (null) | (null) | (null) | 3500 | 5800 |
| 60 | 9000 | (null) | (null) | (null) | (null) |
| 80 | (null) | 10500 | 11000 | (null) | (null) |

# PIVOT 함수 : 예제

- 다음 예제는 직책별, 부서별 최고 급여를 출력합니다.

```
SELECT *
FROM (SELECT department_id, job_id, salary FROM employees
      WHERE department_id IN (50,60, 80))
PIVOT(MAX(salary) FOR department_id IN (50 AS d50, 60 AS d60, 80 AS d80));
```

| JOB_ID | D50 | D60 | D80 |
|---|---|---|---|
| IT_PROG | (null) | 13236.87 | (null) |
| ST_MAN | 7368.9 | (null) | (null) |
| SA_MAN | (null) | (null) | 13340.25 |
| SA_REP | (null) | (null) | 13975.5 |
| ST_CLERK | 4446.75 | (null) | (null) |

```
SELECT *
FROM (SELECT job_id, department_id, salary FROM employees
          WHERE department_id IN (50,60, 80))
PIVOT (MAX(salary) FOR job_id IN ('IT_PROG','SA_MAN','SA_REP',
       'ST_CLERK', 'ST_MAN'))
ORDER BY department_id;
```

| DEPARTMENT_ID | 'IT_PROG' | 'SA_MAN' | 'SA_REP' | 'ST_CLERK' | 'ST_MAN' |
|---|---|---|---|---|---|
| 50 | (null) | (null) | (null) | 3500 | 5800 |
| 60 | 9000 | (null) | (null) | (null) | (null) |
| 80 | (null) | 10500 | 11000 | (null) | (null) |

# UNPIVOT 함수

- 열을 행으로 변환해 주는 함수 입니다.
- 구문

```
SELECT *
FROM ( 대상테이블 or 서브쿼리 ) AS tab
UNPIVOT (
          집계값칼럼명 FOR UNPIVOT대상칼럼명 IN ([UNPIVOT할칼럼명], ...)
        ) AS unpvt
[ORDER BY…];
```

# UNPIVOT 예제

- 다음 예제는 앞서 PIVOT 예제의 결과를 다시 UNPIVOT 합니다.

```
SELECT *
FROM (SELECT *
      FROM (SELECT department_id, job_id, salary FROM employees
            WHERE department_id IN (50, 60, 80))
      PIVOT (MAX(salary) FOR department_id IN (50 AS d50, 60 AS d60, 80 AS d80))
      ) unpivoted_data
UNPIVOT (salary FOR department_id IN (d50, d60, d80))
ORDER BY job_id, department_id;
```

| JOB_ID | D50 | D60 | D80 |
|---|---|---|---|
| IT_PROG | (null) | 13236.87 | (null) |
| ST_MAN | 7368.9 | (null) | (null) |
| SA_MAN | (null) | (null) | 13340.25 |
| SA_REP | (null) | (null) | 13975.5 |
| ST_CLERK | 4446.75 | (null) | (null) |

| JOB_ID | DEPARTMENT_ID | SALARY |
|---|---|---|
| IT_PROG | D60 | 13236.87 |
| SA_MAN | D80 | 13340.25 |
| SA_REP | D80 | 13975.5 |
| ST_CLERK | D50 | 4446.75 |
| ST_MAN | D50 | 7368.9 |

# PIVOT 함수의 응용

- 다음 예제는 직군별, 월별 입사건수를 반환합니다.

```
SELECT *
  FROM (
         SELECT job_id ,
                TO_CHAR(hire_date, 'fmMM') || '월' AS hire_month
           FROM employees
       )
 PIVOT (
         COUNT(*)
         FOR hire_month IN ('1월','2월','3월','4월','5월','6월',
                            '7월','8월','9월','10월','11월','12월')
       );
```

# DECODE 함수와 GROUP 함수로 PIVOT 대체

- 이전 예제는 DECODE와 GROUP 함수를 사용하여 작성될 수 있습니다.

```
SELECT job_id
     , SUM(DECODE(TO_CHAR(hire_date, 'MM'), '1', 1, 0)) "1월"
     , SUM(DECODE(TO_CHAR(hire_date, 'MM'), '2', 1, 0)) "2월"
     , SUM(DECODE(TO_CHAR(hire_date, 'MM'), '3', 1, 0)) "3월"
     , SUM(DECODE(TO_CHAR(hire_date, 'MM'), '4', 1, 0)) "4월"
     , SUM(DECODE(TO_CHAR(hire_date, 'MM'), '5', 1, 0)) "5월"
     , SUM(DECODE(TO_CHAR(hire_date, 'MM'), '6', 1, 0)) "6월"
     , SUM(DECODE(TO_CHAR(hire_date, 'MM'), '7', 1, 0)) "7월"
     , SUM(DECODE(TO_CHAR(hire_date, 'MM'), '8', 1, 0)) "8월"
     , SUM(DECODE(TO_CHAR(hire_date, 'MM'), '9', 1, 0)) "9월"
     , SUM(DECODE(TO_CHAR(hire_date, 'MM'), '10', 1, 0)) "10월"
     , SUM(DECODE(TO_CHAR(hire_date, 'MM'), '11', 1, 0)) "11월"
     , SUM(DECODE(TO_CHAR(hire_date, 'MM'), '12', 1, 0)) "12월"
 FROM employees
 GROUP BY job_id;
```


Thank you