Oracle DBMS

CRUD
CREATE, READ(SELECT 활용), UPDATE, DELETE

<LISTAGG(List Aggreation)함수>
GROUP BY를 활용하여 그룹화하여 출력
SELECT DEPTNO, ENAME
FROM EMP
GROUP BY DEPTNO, ENAME(ORDER BY DEPTNO ASC);
()부분써서 간결한 정렬시키면 됨.

LISTAGG 사용하여 사원 이름 나열하기
```
SELECT DEPTNO,
        LISTAGG(ENAME, ', ')
        WITHIN GROUP(ORDER BY SAL DESC) AS ENAMES
  FROM EMP
GROUP BY DEPTNO;
```

→ 이름이 A, B, C 형태로 나타난다.
,는 구별자라고 한다.

<PIVOT, UNPIVOT 함수>
엑셀에서 나오는 기능과 동일
PIVOT = 축을 뜻한다.

<GROUP BY, ORDER BY 활용하여 그룹화해서 최고 급여 데이터 나타내기>
```
SELECT DEPTNO, JOB, MAX(SAL)
  FROM EMP
GROUP BY DEPTNO, JOB
ORDER BY DEPTNO, JOB;
```

<PIVOT 이용하여서 직책, 부서별 최고급여를 2차원 표 형태로 출력하기>
```
SELECT *
  FROM(SELECT DEPTNO, JOB, SAL
          FROM EMP)
PIVOT(MAX(SAL)
      FOR DEPTNO IN (10, 20, 30)
      )
ORDER BY JOB;
```

SELECT * FROM안에 SELECT가 들어가있다.
이것을 SUBQUERY라고 한다. (뒷장에서 나온다)

```
SELECT *
  FROM(SELECT JOB, DEPTNO, SAL
         FROM EMP)
PIVOT(MAX(SAL)
      FOR JOB IN ('CLERK' AS CLERK,
                  'SALESMAN' AS SALESMAN,
                  'PRESIDENT' AS PRESIDENT,
                  'MANAGER' AS MANAGER,
                  'ANALYST' AS ANALYST)
     )
ORDER BY DEPTNO;
```

<DECODE와 PIVOT 활용>
<DECODE, GROUP BY, ORDER BY 사용 - 코드가 길다>

```
SELECT DEPTNO,
        MAX(DECODE(JOB, 'CLERK', SAL)) AS "CLERK",
        MAX(DECODE(JOB, 'SALESMAN', SAL)) AS "SALESMAN",
        MAX(DECODE(JOB, 'PRESIDENT', SAL)) AS "PRESIDENT",
        MAX(DECODE(JOB, 'MANAGER', SAL)) AS "MANAGER",
        MAX(DECODE(JOB, 'ANALYST', SAL)) AS "ANALYST"
  FROM EMP
GROUP BY DEPTNO
ORDER BY DEPTNO;
```

<UNPIVOT>

```
SELECT *
  FROM(SELECT DEPTNO,
              MAX(DECODE(JOB, 'CLERK' , SAL)) AS "CLERK",
              MAX(DECODE(JOB, 'SALESMAN' , SAL)) AS "SALESMAN",
              MAX(DECODE(JOB, 'PRESIDENT', SAL)) AS "PRESIDENT",
              MAX(DECODE(JOB, 'MANAGER' , SAL)) AS "MANAGER",
              MAX(DECODE(JOB, 'ANALYST' , SAL)) AS "ANALYST"
         FROM EMP
        GROUP BY DEPTNO
        ORDER BY DEPTNO)
UNPIVOT(
   SAL FOR JOB IN (CLERK, SALESMAN, PRESIDENT, MANAGER,ANALYST))
```

ORDER BY DEPTNO, JOB;
피벗테이블을 본래상태로 돌리는 것이 UNPIVOT이다.

<여러 테이블의 결합>
JOIN
두 개 이상의 테이블을 연결하여 하나의 테이블처럼 사용하는 것
```
SELECT *
  FROM EMP, DEPT
ORDER BY EMPNO;

SELECT *
  FROM EMP, DEPT
 WHERE EMP.DEPTNO = DEPT.DEPTNO
ORDER BY EMPNO;
```

테이블 별칭 설정
```
SELECT *
  FROM EMP E, DEPT D
 WHERE E.DEPTNO = D.DEPTNO
ORDER BY EMPNO;
```
테이블 별칭을 만들 때 AS를 사용하지 않고 한칸 띄운 후 별칭의 이름을 짧게 쓰면 된다.

JOIN의 종류
등가, 비등가, 자체, 외부 JOIN이 있다.

```
SELECT EMPNO, ENAME, DEPTNO, DNAME, LOC
  FROM EMP E, DEPT D
 WHERE E.DEPTNO = D.DEPTNO;
```
에러
테이블의 이름을 잘 명시하는 것이 중요하다.

테이블 이름 명시하기
```
SELECT E.EMPNO, E.ENAME, D.DEPTNO, D.DNAME, D.LOC
  FROM EMP E, DEPT D
 WHERE E.DEPTNO = D.DEPTNO
ORDER BY D.DEPTNO, E.EMPNO;
```

WHERE를 활용하여 조건 넣어서 출력
```
SELECT E.EMPNO, E.ENAME, E.SAL, D.DEPTNO, D.DNAME, D.LOC
  FROM EMP E, DEPT D
 WHERE E.DEPTNO = D.DEPTNO
```

```
   AND SAL >= 3000;
```

비등가 조인
서로 다른 테이블을 조인하는 것이다.

```
SELECT *
  FROM EMP E, SALGRADE S
 WHERE E.SAL BETWEEN S.LOSAL AND S.HISAL;
```

자체 조인

```
SELECT E1.EMPNO, E1.ENAME, E1.MGR,
       E2.EMPNO AS MGR_EMPNO,
       E2.ENAME AS MGR_ENAME
  FROM EMP E1, EMP E2
 WHERE E1.MGR = E2.EMPNO;
```

상사의 번호와 일치하는 매니저 이름을 찾음.
테이블 안에는 번호로 되어 있지만 상사의 이름을 찾아서 상사의 데이터를 얻는다.

외부 조인

```
SELECT E1.EMPNO, E1.ENAME, E1.MGR,
       E2.EMPNO AS MGR_EMPNO,
       E2.ENAME AS MGR_ENAME
  FROM EMP E1, EMP E2
 WHERE E1.MGR = E2.EMPNO(+)
ORDER BY E1.EMPNO;
```

```
SELECT E1.EMPNO, E1.ENAME, E1.MGR,
       E2.EMPNO AS MGR_EMPNO,
       E2.ENAME AS MGR_ENAME
  FROM EMP E1, EMP E2
 WHERE E1.MGR(+) = E2.EMPNO
ORDER BY E1.EMPNO;
```

KING은 상사가 없어서 NULL값이 나타난다.
NULL값을 출력할 테이블 쪽을 +로 표시한다.

SQL-99 표준문법
99년도 이후로 표준 지정된 문법
SQL-99는 1999년에 채택된 SQL의 표준 규격이며, ISO/IEC 9075 표준 문서로 정의되어 있습니다. 이전에는 SQL-92라는 표준이 있었으나, 이를 개정한 것이 SQL-99입니다. SQL-99에서는 전체적으로 SQL의 기능이 확장되었으며, 객체지향적인 기능이 추가된 문법이다.

<NATURAL JOIN>

```
SELECT E.EMPNO, E.ENAME, E.JOB, E.MGR, E.HIREDATE, E.SAL, E.COMM,
DEPTNO, D.DNAME, D.LOC
FROM EMP E NATURAL JOIN DEPT D
ORDER BY DEPTNO, E.EMPNO;
```

자연스럽게 JOIN시키는 역할을 한다.

<JOIN ~ USING>

```
SELECT E.EMPNO, E.ENAME, E.JOB, E.MGR, E.HIREDATE, E.SAL, E.COMM,
        DEPTNO, D.DNAME, D.LOC
  FROM EMP E JOIN DEPT D USING (DEPTNO)
 WHERE SAL >= 3000
ORDER BY DEPTNO, E.EMPNO;
```

<JOIN ~ ON>

```
SELECT E.EMPNO, E.ENAME, E.JOB, E.MGR, E.HIREDATE, E.SAL, E.COMM,
        E.DEPTNO,
        D.DNAME, D.LOC
  FROM EMP E JOIN DEPT D ON (E.DEPTNO = D.DEPTNO)
 WHERE SAL <= 3000
ORDER BY E.DEPTNO, EMPNO;
```

<OUTER JOIN>

```
SELECT E1.EMPNO, E1.ENAME, E1.MGR,
        E2.EMPNO AS MGR_EMPNO,
        E2.ENAME AS MGR_ENAME
  FROM EMP E1 LEFT OUTER JOIN EMP E2 ON (E1.MGR = E2.EMPNO)
ORDER BY E1.EMPNO;
```

<전체 외부 JOIN>

```
SELECT E1.EMPNO, E1.ENAME, E1.MGR,
        E2.EMPNO AS MGR_EMPNO,
        E2.ENAME AS MGR_ENAME
  FROM EMP E1 FULL OUTER JOIN EMP E2 ON (E1.MGR = E2.EMPNO)
ORDER BY E1.EMPNO;
```

JOIN을 하는 기준이 되는 테이블의 값을 모두 살리는 것

서브쿼리

SQL문 내에서 SQL문을 작성하는 것

```
SELECT SAL FROM EMP
WHERE ENAME = 'JONES';
```

```
SELECT * FROM EMP
WHERE SAL > 2975;
```

이 두 가지의 SQL 구문을 합하여 간결하게 나타내면 아래와 같다.

```
SELECT *
  FROM EMP
 WHERE SAL > (SELECT SAL FROM EMP
                WHERE ENAME = 'JONES');
```

서브쿼리는 이와 같이 활용을 할 수 있다.

<단일형 서브쿼리>
입사일이 적은 사람을 구하기

```
SELECT *
FROM EMP
WHERE HIREDATE < (SELECT HIREDATE
                    FROM EMP
                   WHERE ENAME = 'KING');
```

서브 쿼리 내에서 함수를 사용할 수 있다.

```
SELECT E.EMPNO, E.ENAME, E.JOB, E.SAL, D.DEPTNO, D.DNAME, D.LOC
  FROM EMP E, DEPT D
 WHERE E.DEPTNO = D.DEPTNO
   AND E.DEPTNO = 20
   AND E.SAL > (SELECT AVG(SAL)
                  FROM EMP);
```

<다중행 서브쿼리>
IN 연산자

```
SELECT *
  FROM EMP
 WHERE DEPTNO IN (20, 30);
```

IN 연산자 활용 서브쿼리

```
SELECT *
  FROM EMP
 WHERE SAL IN (SELECT MAX(SAL)
                 FROM EMP
                 GROUP BY DEPTNO);
```

서브 쿼리 내 쿼리 ; 부서 번호 별로 최대 급여

```
SELECT MAX(SAL)
  FROM EMP
GROUP BY DEPTNO;
```

ANY 연산자

```
SELECT *
  FROM EMP
 WHERE SAL = ANY (SELECT MAX(SAL)
                    FROM EMP
                   GROUP BY DEPTNO);
```

앞서 했었던 IN 연산자 써서 나타낸 것과 같다.

SOME 연산자

```
SELECT *
  FROM EMP
 WHERE SAL = SOME (SELECT MAX(SAL)
                     FROM EMP
                    GROUP BY DEPTNO);
```

ANY 종류 불문하고 어떤 것
SOME은 몇 몇이라는 뜻

= 일 때는 구분이 불가, 비교하는 연산자 > 또는 <일 때는 구분이 된다.

최대 급여 보다 적은 사원 정보 출력

```
SELECT *
  FROM EMP
 WHERE SAL < ANY (SELECT SAL
                    FROM EMP
                   WHERE DEPTNO = 30)
                  ORDER BY SAL, EMPNO;
```

서브 쿼리 내 쿼리 - 30번 부서 내의 월급 나타냄

```
SELECT SAL FROM EMP
WHERE DEPTNO = 30;
```

→ ANY를 사용해서 최대보다 적은 모두를 표시한다.(종류 불문하고 / 가리지 않고의 의미)

최소 급여보다 많은 급여를 받는 받는 사람 구하기

```
SELECT *
  FROM EMP
```

```
 WHERE SAL > ANY (SELECT SAL
                    FROM EMP
                   WHERE DEPTNO = 30)
                  ORDER BY SAL, EMPNO;
```

최소 급여보다 더 적은 급여를 받는 사람 / 최대 급여보다 더 많은 급여를 받는 사람 출력

<ALL 연산자>

```
SELECT *
  FROM EMP
 WHERE SAL < ALL (SELECT SAL
                    FROM EMP
                   WHERE DEPTNO = 30)
                  ORDER BY SAL, EMPNO;
SELECT *
  FROM EMP
 WHERE SAL > ALL (SELECT SAL
                    FROM EMP
                   WHERE DEPTNO = 30)
                  ORDER BY SAL, EMPNO;
```

ALL은 뭉텅이로 모두라는 의미

<EXIST 연산자>

EXIST : 존재하다라는 의미

```
SELECT *
  FROM EMP
 WHERE EXISTS (SELECT DNAME
                 FROM DEPT
                WHERE DEPTNO = 10);
```

~가 있으면 보여달라는 의미로 쓰이는 연산자
비교의 연산자가 아님.

서브쿼리에서 반환된 결과가 존재하는지 여부를 판단하는 논리 연산자입니다. EXISTS 연산자를 사용하면 하위 질의에서 반환된 결과를 비교하여, 결과가 있으면 참(TRUE)을 반환하고, 결과가 없으면 거짓(FALSE)을 반환합니다.

```
SELECT *
  FROM EMP
 WHERE EXISTS (SELECT DNAME
```

```
                  FROM DEPT
                 WHERE DEPTNO = 50);
```

50이라는 부서번호가 없다.
값이 존재하지 않는 경우 아무것도 보이지 않는다.

<다중열 서브쿼리 = 복수열 서브쿼리>

```
SELECT *
  FROM EMP
 WHERE (DEPTNO, SAL) IN (SELECT DEPTNO, MAX(SAL)
                           FROM EMP
                          GROUP BY DEPTNO);
```

비교할 열이 여러 개일 때 사용

<FROM 뒤에 사용하는 서브쿼리 & WITH절>
인라인 뷰란?
서브쿼리를 이용해 생성된 가상의 테이블을 의미합니다. 서브쿼리에서 쿼리 결과를 가지고 새로운 SELECT 문을 만들어 사용하는 것을 인라인 뷰라고 한다.

인라인 뷰를 사용하는 경우
중첩된 쿼리에서 부모 쿼리에서 자식 쿼리의 결과를 사용할 때
서브쿼리에서 여러 개의 테이블을 조인해야 할 때
서브쿼리 결과를 그룹화, 정렬, 조작하여 새로운 테이블을 만들 때
인라인 뷰는 FROM 절에 사용되며, 다음과 같은 형식으로 사용됩니다.

```
SELECT E10.EMPNO, E10.ENAME, E10.DEPTNO, D.DNAME, D.LOC
  FROM (SELECT * FROM EMP WHERE DEPTNO = 10) E10,
       (SELECT * FROM DEPT) D
 WHERE E10.DEPTNO = D.DEPTNO;
```

<WITH 사용>

```
WITH
E10 AS (SELECT * FROM EMP WHERE DEPTNO = 10),
D AS (SELECT * FROM DEPT)
SELECT E10.EMPNO, E10.ENAME, E10.DEPTNO, D.DNAME, D.LOC
  FROM E10, D
 WHERE E10.DEPTNO = D.DEPTNO;
```

<SELECT에서 사용하는 서브쿼리>

```
SELECT EMPNO, ENAME, JOB, SAL,
       (SELECT GRADE
```

```
          FROM SALGRADE
         WHERE E.SAL BETWEEN LOSAL AND HISAL) AS SALGRADE,
       DEPTNO,
      (SELECT DNAME
         FROM DEPT
        WHERE E.DEPTNO = DEPT.DEPTNO) AS DNAME
FROM EMP E;
```