SQL 활용의 주 내용

1. 표준 조인
2. 집합 연산자
3. 계층형 질의
4. 서브쿼리
5. 그룹 함수와 윈도우 함수
6. DCL
7. 절차형 SQL

제 1 절 표준 조인

2. FROM 절 JOIN 형태

- INNTER JOIN
- NATURAL JOIN
- USING 조건절
- ON 조건절
- CROSS JOIN
- OUTER JOIN

3. INNER JOIN

- JOIN 으로도 쓰고, INNTER JOIN 으로도 씀, 조인 조건 줘야 함

```
  SELECT * FROM emp, dept WHERE dept.deptno = emp.deptno; /* 옛날 스타일 */
  SELECT * from emp join dept ON dept.deptno = emp.deptno; /* ON 사용하여
JOIN 조건 구분 */
  SELECT * from emp join dept USING (deptno) /*JOIN 할 attr 이름 같은경우
USING () 사용가능 */
```

4. NATURAL JOIN

- 알아서 EQUI JOIN 수행함. USING, ON 조건절 없이 알아서 이름, 데이터타입이 같은 column 찾아서 natural join함
- 공통 col 에 대해서 projection 된 형태로 나옴.
- SQL SERVER 에서는 불가능함.
- SELECT 에 *(wildcard) 사용 시 join의 기준 컬럼부터 출력함.
- NATURAL JOIN에 사용 한 열은 SELECT 에 따로 식별자 사용 불가능 (emp.deptno 같이 불가능)

```
  SELECT * FROM emp NATURAL JOIN dept;
```

5. USING 조건절

- USING ( ) 괄호 안에 적어줘야함 유의

- 같은 이름의 key를 넣어주면 됨. 자동으로 중복 projection 도 됨
- join 에 사용된 col의 경우 결과의 제일 앞에 위치하고 alias 나 식별자 TB 앞에 못적어줌.

6. ON 조건절

- 옛날 스타일에선 WHERE 에 조인 조건과 row 조건들을 다 주니 헷갈려서 분리한 패턴
- COL 이름이 달라도 옛날 join 할 때 처럼 가능함. (using 에서는 col 이름 같아야 함.)

```
SELECT E.EMPNO, E.ENAME, E.DEPTNO, D.DNAME
FROM EMP E JOIN DEPT D
ON (E.DEPTNO = D.DEPTNO);
```

7. CROSS JOIN

- CARTESIAN PRODUCT 모든 row x row

8. OUTER JOIN

- OUTER JOIN 이라 쓴 경우도 JOIN 조건을 FROM 절에서 정의하겠다는 의미니, USING 이나 ON 사용 필수
- LEFT/RIGHT OUTER JOIN 시 기준 TB 가 join 수행시 무조건 driving table (이 경우 OUTER 생략가능)
- LEFT OUTER JOIN 인 경우 왼쪽 놈을 다 살리고 오른쪽놈 없어도 NULL 로 채워넣는다는 의미
- 옛날 스타일로 LEFT OUTER JOIN 을 표현 할 경우 WHERE 절에 join 조건에서 null 이 나와도 되는 곳에 (+) 를 써 줌
    - 이는 LEFT OUTER JOIN 으로 쓰는 형태의 드라이빙 테이블(왼쪽) 과 반대의 위치라고 생각하면 쉬움
- FULL OUTER JOIN = LEFT JOIN UNION RIGHT JOIN

제 2 절 집합 연산자

2개 이상의 테이블에서 조인을 사용하지 않고 연관된 데이터를 조회하는 방법. 2개 이상의 질의 결과를 하나로 만들어 줌

집합은 순서가 없음에 유의. 순서를 주고 싶으면 ORDER BY 사용

- UNION : 합집합. 중복된 행은 하나의 행으로 만듦
- UNION ALL : 합집합. 중복된 행 표시
    - 여기서 나오는 ORDER BY 1,2,3,4,5 는 SELECT 에 적힌 순서대로 sorting 하라는 것
    - 1이 같은경우 2로 2가 같은 경우 3으로 ...
- INTERSECT : 교집합
- EXCEPT : 차집합 (ORACLE 은 MINUS)
    - 차집합의 경우 동일한 결과를 논리 연산자로 얻을 수 있음.
    - MINUS 연산자는 NOT EXIST 와 NOT IN 으로 표현 가능함.

제 3 절 계층형 질의와 셀프 조인

[시험에 많이 나옴]

Hierarchical Query. TB에 존재하는 계층형 data 를 보기 위함.

사원과 MANAGER 생각하면 좋을 듯 아래와 같은 스타일로 생겼다.

```
  SELECT ...
  FROM TB
  WHERE condition AND condition ...
  START WITH condition
  CONNECT BY [NOCYCLE] condition AND condition ...
  [ORDER SIBLINGS BY column, column, ...]
```

- START WITH : 계층 구조 전개 시작 위치 (ROOT DATA 지정)
- CONNECT BY : 다음에 전개될 자식 데이터 지정. (CONNECT 에 있는 condition 을 만족해야 함 (조인))
- PRIOR : CONNECT BY 절에 사용되며, 현재 읽은 칼럼을 지정함.
    - 변수 넣듯 왼쪽 = 오른쪽일때 오른쪽에서 설정한 기준으로 왼쪽 애들 정해진다 보자.
    - PRIOR 자식 = 부모 이면 부모 -> 자식으로 순방향
    - PRIOR 부모 = 자식 이면 자식 -> 부모로 역방향
    - NOCYCLE : 데이터를 전개하면서 이미 나타났던 동일한 데이터가 전개 중에 다시 나타나면 CYCLE 이라 함. NOCYCLE 사용 시 사이클 발생한 이후 데이터는 전개하지 않음.
- ORDER SIBILINGS BY : Sibling 사이의 정렬수행
- WHERE : 모든 전개 후 지정 조건 데이터만 추출

ORACLE 은 계층형 질의를 위한 SELECT 에서 사용할 수 있는 가상 칼럼(Pesudo Column) 제공함

- LEVEL : ROOT data 이면 1 그 하위 data 면 2 ... 이런식으로 leaf 까지 1 증가함
- CONNECT_BY_ISLEAF : 전개 과정에서 LEAF 온 경우 1 아니면 0
- CONNECT_BY_ISCYCLE : 전개 과정에서 자식을 갖는데 그게 조상으로서 존재하면 cycle이 되니까 1 아니면 0 (cycle 옵션 사용시에만 사용 가능함.)

```
SELECT LEVEL, LPAD(' ', 4*(LEVEL-1)) || EMPNO 사원, MGR 관리자,
CONNECT_BY_ISLEAF ISLEAF
FROM EMP
START WITH MGR IS NULL
CONNECT BY PRIOR EMPNO = MGR;
/* LPAD 는 왼쪽에 만들어주는 문자열. 여기서는 계층을 보여주기 위해 4칸씩 띄었다. (ROOT 이하
부터적용)*/
```

- 위 예제에서 START WITH 를 LEAF 인 사원으로 하고 PRIOR 위치 바꿔주는 경우 EMPNO 가직 MGR 를 정함. 즉 역방향 전개 (LEAF 였던 친구가 ROOT 가 됨)

Oracle 이 제공하는 계층형 질의를 위한 FUNCTION

- SYS_CONNECT_BY_PATH : ROOT data 로 부터 현재 전개할 데이터까지의 경로 표시(유용)
- CONNECT_BY_ROOT : 현재 전개 할 데이터의 루트 데이터를 표시.

```
  SELECT CONNECT_BY_ROOT(EMPNO) 루트사원, SYS_CONNECT_BY_PATH(EMPNO, '/') 경
로, EMPNO 사원, MGR 관리자
  FROM EMP
  START WITH MGR IS NULL
  CONNECT BY PRIOR EMPNO = MGR;
```

```
/* 모든 row에 root 의 EMPNO 가 적혀있고, /를 구분자로 폴더처럼 path 가 표현됨 */
```

- 순방향(부모-> 자식)
    - START WITH MGR iS NULL CONNECT BY PRIOR EMPNO = MGR
    - 이것만 하나 잘 외워두어도 될 듯
- 괜히 깊게 정리한게 아님. 잘알아두기. SQL SERVER 는 버린다.

셀프 조인

- 반드시 alias 를 써서 구분 해 줘야함.
- 하나의 테이블에서 두 컬럼이 연관 관계를 가지고 있는 경우 가능.

제 4 절 서브쿼리

- 하나의 SQL 문 안에 포함되어 잇는 또 다른 SQL 문
- 서브쿼리는 메인쿼리의 칼럼을 사용할 수 있지만, 메인쿼리는 서브쿼리의 칼럼을 사용할 수 없음.
    - 서브쿼리가 메인 쿼리의 칼럼을 가지고 있는 경우 Correlated Sub Query 라 함.
- 서브쿼리에서는 ORDER BY 를 사용할 수 없다.
- 서브쿼리의 결과는 single row 일 수도 있고 multi row 일 수도 있다. multi row 인경우 in 이나 any 같은거 써준다.
- Single Row Subquery
    - =,<,> <> 이런것들 써줌 당연히 멀티로우면 고장남.
- Multi Row Subquery
    - IN, ANY(=SOME), ALL, EXIST 사용하여 비교함.
        - ANY
        - EXIST : 존재성만 판단함. 1건이라도 찾으면 더이상 검색하지 않음.
- 좋은 예제

```
 /* 평균보다 키가 작은 친구들을 구하시오.*/
SELECT PLAYER_NAME, POSITION, BACK_NO
FROM PLAYER
WHERE HEIGHT <= (SELECT AVG(HEIGHT) FROM PLAYER)
```

SELECT 절에 사용하는 SubQuery : 스칼라 서브쿼리

- 스칼라 서브쿼리 : 한 행 한 컬럼만을 반환하는 서브쿼리 (한 value (특히 aggregate 된) 를 얻기위함)

```
SELECT
PLAYER_NAME 선수명,
```

```
  HEIGHT 키,
  ROUND((SELECT AVG(HEIGHT) FROM PLAYER X WHERE X.TEAM_ID = P.TEAM_ID),3)
팀평균키
  FROM PLAYER P
```

FROM 절에서 사용하는 subQuery

- FROM절에서 사용하는 SubQuery 를 라인 뷰(Inline View) 라 함
- 마치 하나의 TB 인것처럼 alias 를 주고 다룰 수 있기 때문에 자유롭게 참조 가능(inline view)

```
  SELECT T.TEAM_NAME 팀명, P.PLAYER_NAME 선수명, P.BACK_NO 백넘버
  FROM
  (SELECT TEAM_ID, PLAYER_NAME, BACK_NO FROM PLAYER WHERE POSITION='MF')
P, TEAM T
  WHERE P.TEAM_ID = T.TEAM_ID
  ORDER BY 선수명
```

그외..

- HAVING, UPDATE의 SET, INSERT의 VALUES 등에 사용 가능

VIEW

- 실제 TB 처럼 데이터를 가지고 있지 않고 데이터 정의만 가짐. 쿼리 날릴 때 VIEW 사용하면 DBMS 가 내부적으로 재작성하여 수행함. Virtual TABLE
    - 독립성, 편리성, 보안성의 장점

제 5 절 그룹 함수(GROUP FUNCTION)

SQL 표준은 데이터 분석을 위해 세 함수를 정의함

- AGGREGATE FUNCTION, GROUP FUNCTION, WINDOW FUNCTION
- GROUP FUNCTION
    - 여러 GROUP BY 연산 합쳐서 주는것들
    - ROLLUP : 뒤에서 부터 말아올라감 (a,b,c) => (a,b) => (a) => ()
    - CUBE : 가능한 sub group 의 모든 경우 나옴
    - GROUPING SETS : 표시된 인수들에 대한 개별 집계를 구할 수 있음. ROLLUP 과 달리 평등관계, 인수 바뀌어도 상관없음
        - DECODE(GROUPING(DNAME), 1, 'All Documents', DNAME) AS DNAME

제 6 절 윈도우 함수

- RANK 관련 함수
    - RANK, DENSE_RANK, ROW_NUMBER
- GROUP 내 행 순서 관련 함수

    - FIRST_VALUE, LAST_VALUE, LAG, LEAD

- WINDOW 함수에는 **OVER** 문구가 키워드로 필수 포함된다.
- RANK 함수

```
SELECT JOB,ENAME,SAL,
RANK() OVER(ORDER BY SAL DESC) ALL_RANK,
RANK() OVER(PARTITION BY JOB ORDER BY SAL DEESC) JOB_RANK
FROM EMP;
```

- DENSE_RANK 함수
    - 동일한 순위를 하나의 등수로 간주한 결과도 같이 출력함.
    - 동점자에 대하여
        - 1,2,2,3 : DENSE RANK 방식
        - 1,2,2,4 : RANK 방식
- ROW_NUMBER : 동일 값에도 고유한 순위. 그냥 행갯수느낌
- 그룹 내 행 순서 함수
    - FIRST_VALUE : 파티션별 윈도우에서 가장 먼저 나온 값. MIN
    - LAST_VALUE : 파티션별 윈도우에서 제일 마지막에 나온 값. MAX
    - LAG(처지다) : 파티션별 윈도우에서 이전 몇 번째 행의 값을 가져올 수 있음
    - LEAD(이끌다) : 파티션별 윈도우에서 이후 몇 번쨔 행의 값을 가져올 수 있음.
    - LAG 나 LEAD 의 첫번째 인자 :target, 두번째 인자: 몇칸?, 세번쨔 인자 : (젤앞 젤뒤같이)없을때 뭘로대체?

```
/* 고용일 빠른 순으로 정렬했을 때 내 앞에 row 의 SAL 나옴 */
SELECT ENAME, HIREDATE, SAL, LAG(SAL) OVER (ORDER BY HIREDATE) AS
PREV_SAL
FROM EMP
WHERE JOB = 'SALESMAN';
```

```
/* 고용일 기준으로 나보다 두칸 전 row 의 SAL 나옴 */
SELECT .... LAG(SAL,2,0)
```

제 7 절 DCL

- 권한 부여 : GRANT
    - GRANT (권한명) TO (유저명)
- 권한 뺏기 : REVOKE
- 권한 뭉태기 : ROLE

제 8 절 절차형 SQL

- 절차지향적 프로그램 가능하도록 벤터별로 PL/SQL 같은것을 지원
- 프로시저와 트리거 차이점
    - 프로시저 : BEGIN ~ END 내에 COMMIT ROLLBACK 과 같은 트랜잭션 명령어 사용 가능
    - 트리거 : BEGIN ~ END 내에 사용 할 수 없음.