2과목 2장 SQL활용

● 집합 연산자

(1) UNION : 합집합(중복 행은 1개의 행으로 처리)
- 두 개의 테이블의 칼럼 수, 칼럼의 데이터 형식 모두가 일치해야 한다. 만약 다르다면 오류 발생

(2) UNION ALL : 합집합(중복 행도 모두 표시)
- Alias 는 첫 번째 SQL 기준 / 정렬은 마지막 SQL 기준

(3) INTERSECT : 교집합

(4) EXCEPT / MINUS : 차집합
- Oracle 은 MINUS / SQL Server는 EXCEPT

● EQUI JOIN (등가조인) : 교집합
- 두 개의 테이블에서 일치하는 것을 조인
- 등가조인은 "="을 사용해서 두 개의 테이블을 연결
- 조인 후에 실행계획을 확인해서 내부적으로 두 개의 테이블을 어떻게 연결했는지 확인할 수 있다.

● NON-EQUI JOIN(비등가조인)
- 두 개의 테이블을 조인하는 경우 연산자 "="가 아닌 다른 연산자를 사용한다.
- 비등가조인은 정확하게 일치하지 않는 것을 조인하는 것

● USING 절

같은 이름을 가진 칼럼들 중에서 원하는 칼럼에 대해서만 선택적으로 등가조인을 할 수 있다. JOIN 칼럼에 대해서 Alias나 테이블 이름과 같은 접두사를 붙일 수 없다.
(SQL Server 는 지원하지 않음)

● ON 절

ON 조건절과 WHERE 조건절을 분리하여 이해가 쉬우며, 칼럼명이 다르더라도 JOIN 조건을 사용할 수 있다. 두 테이블에서 사용하는 칼럼명이 같은 경우 Alias나 테이블명을 반드시 적어야 한다.

● INNER JOIN

JOIN 조건에서 동일한 값이 있는 행만 반환, USING 이나 ON 절을 필수적으로 사용

● NATURAL JOIN

두 테이블 간의 동일한 이름을 갖는 모든 칼럼에 대해 등가조인을 수행

● CROSS JOIN

JOIN 조건이 없는 경우 생길 수 있는 모든 데이터 조합
양쪽 집합의 N*M 건의 데이터 조합이 발생 (곱집합)

● OUTER JOIN

두 개의 테이블 간에 교집합을 조회하고 한쪽 테이블에만 있는 데이터도 포함시켜서 조회한다. (값이 없는 칼럼은 null)
USING이나 ON 조건절 반드시 사용

● LEFT OUTER JOIN

ex) table_A LEFT OUTER JOIN table_B ON (A.col = B.col)
테이블 A의 값은 모두 가져오고 B에서는 ON 조건절에 해당되는 행만 가져온다.
B에는 해당되지 않아 값이 없는 경우에는 null 표기

● RIGHT OUER JOIN : LEFT OUTER JOIN 의 반대

● FULL OUTER JOIN

조인 수행 시 좌측/우측 테이블의 모든 데이터를 읽어 JOIN 하여 결과를 생성한다. 중복 데이터는 삭제한다
아래 예시와 같은 결과를 가져온다.

```
> SELECT * FROM A LEFT JOIN B ...
  UNION
  SELECT * FROM A RIGHT JOIN B ...
```

● 계층형 조회(connect by)

테이블에 계층형 데이터가 존재하는 경우 데이터를 조회하기 위해 사용

(1) START WITH : 계층 구조 전개의 시작 위치 지정

(2) CONNECT BY : 다음에 전개될 자식 데이터 지정

(3) PRIOR : CONNECT BY 절에 사용되며, 현재 읽은 칼럼을 지정, SELECT와 WHERE 절에도 사용 가능
PRIOR 부모 = 자식 : 역방향 (자식부터 부모까지)
PRIOR 자식 = 부모 : 순방향 (부모부터 자식까지)

(4) NOCYCLE : 동일한 데이터가 전개되지 않음 (순환구조가 발생지점까지만 전개)

(5) ORDER SIBLINGS BY : 형제 노드간의 정렬 수행

(6) WHERE : 모든 전개를 수행한 다음 지정된 조건을 만족하는 데이터만 추출(필터링)

● CONNECT BY 키워드

(1) LEVEL : 검색 항목의 깊이, 계층구조에서 가장 상위 레벨이 1이 된다.

(2) CONNECT_BY_ROOT : 계층구조에서 가장 최상위 값 표시
단항 연산자이다

(3) CONNECT_BY_ISLEAF : 계층구조에서 가장 최하위를 표시
최하위 데이터면 1, 아니면 0

(4) CONNECT_BY_ISCYCLE : 순환구조 발생 지점을 표시
데이터가 조상이면 1, 아니면 0 (CYCLE 옵션 사용했을 시만 사용 가능)

(5) SYS_CONNECT_BY_PATH : 계층구조 전체 전개 경로 표시

(6) NOCYCLE : 순환구조가 발생지점까지만 전개

● SELF JOIN(셀프조인)

- 동일 테이블 사이의 조인, FROM 절에 동일 테이블이 2번 이상 나타난다. 반드시 테이블 별칭을 사용
- 동일 테이블에서 서로 관련 있는 칼럼이 있을 때 사용

● SubQuery(서브쿼리)

- 하나의 SQL문 안에 포함되어 있는 또 다른 SQL문
- 알려지지 않은 기준을 이용한 검색에 사용
- 서브쿼리는 메인쿼리의 칼럼을 모두 사용할 수 있지만 메인쿼리는 서브쿼리의 칼럼을 사용할 수 없다.

● 메인쿼리와 서브쿼리

- 인라인 뷰 : FROM 절에 서브쿼리 사용
- 스칼라 서브쿼리 : SELECT 절에 서브쿼리 사용
- 서브쿼리 : WHERE 절에 서브쿼리 사용
- 메인쿼리 : 위 3개 외에는 메인쿼리

● 단일 행 서브쿼리

- 서브쿼리를 실행하면 그 결과는 반드시 한 행만 조회

● 다중 행 서브쿼리

- 서브쿼리를 실행하면 그 결과는 여러 개의 행이 조회
- IN, ALL, ANY, SOMEEXISTS 등의 연산자 사용

● 다중 행 서브쿼리 연산자

(1) IN

- 메인쿼리의 비교조건이 서브쿼리의 결과 중 하나만 동일하면 참 (OR조건)

(2) ALL

- 메인쿼리와 서브쿼리의 결과가 모두 동일하면 참

< ALL : 최솟값을 반환

> ALL : 최댓값을 반환

ex) SELECT * FROM emp WHERE deptno <= ALL (20, 30)

: deptno 가 20, 30(둘 다)보다 작거나 같은 것을 조회

(3) ANY

- 메인쿼리의 비교조건이 서브쿼리의 결과 중 하나 이상 동일하면 참

ex) SELECT * FROM emp WHERE deptno <= ANY (20, 30)

: deptno 가 20,30 둘 중 하나보다라도 작은 것을 조회

(4) EXISTS

- 서브쿼리의 결과가 하나라도 존재하면 참
- 결과는 참과 거짓이 반환된다
- 주로 메인쿼리의 칼럼과 함께 쓰인다

● 스칼라 서브쿼리

- 스칼라 서브쿼리는 반드시 한 행과 한 칼럼만 반환
- 만약 여러 행이 반환되면 오류 발생

√ 그룹함수

● ROLLUP

- GROUP BY 칼럼에 대해서 Subtotal을 만들어 준다.
- ROLLUP을 할 때 GROUP BY 구에 칼럼이 두 개 이상 오면 순서에 따라서 결과가 달라진다. (인수 순서에 주의)
- Columns 수를 N이라고 했을 때 N+1 Level의 Subtotal이 생성된다.
- ROLLUP(A,B) 는 아래와 같다.

ex) GROUP BY A,B
UNION ALL
GROUP BY A
UNION ALL
모든 집합 그룹 결과

● GROUPING 함수

- ROLLUP, CUBE, GROUPING SETS 에서 생성되는 합계 값을 구분한다.
- 소계, 합계 등이 계산되면 GROUPING 함수는 1을 반환, 그렇지 않으면 0을 반환

● GROUPING SETS

- GROUP BY에 나오는 칼럼의 순서와 관계없이 개별적인 집계를 구할 수 있다. 다양한 소계 집합을 만들 수 있다.
- GROUPING SETS(A,B) 는 아래와 같다.

ex) GROUP BY A
UNION ALL
GROUP BY B

● CUBE

- 제시한 칼럼에 대해서 결합 가능한 모든 집계를 계산한다.
- 다차원 집계를 제공하여 다양하게 데이터를 분석 가능
- ROLLUP에 비해 시스템 부하가 심하다.
- CUBE(A,B) 는 아래와 같다.

ex) GROUP BY A,B
UNION ALL
GROUP BY A
UNION ALL
GROUP BY B
UNION ALL
모든 집합 그룹 결과

● 윈도우 함수

- 행과 행 간의 관계를 정의하기 위해서 제공되는 함수
- 순위, 합계, 평균, 행 위치 등을 조작할 수 있다.

ex) SELECT WINDOW_FUNCTION (ARGUMENTS)
OVER (PARTITION BY 칼럼 ORDER BY WINDOWING 절)

● **윈도우 함수 구조**

(1) ARGUMENTS(인수) : 윈도우 함수에 따라 0~N 개의 인수를 설정한다.

(2) PARTITION BY : 전체 집합을 기준에 의해 소그룹으로 나눈다.

(3) ORDER BY : 어떤 항목에 대해서 정렬한다.

(4) WINDOWING : 행 기준의 범위를 정한다. ROWS는 물리적 결과의 행 수이고 RANGE는 논리적인 값에 의한 범위

● **WINDOWING**

(1) ROWS : 부분집합인 윈도우 크기를 물리적 단위로 행의 집합을 지정

(2) RANGE : 논리적인 주소에 의해 행 집합을 지정

(3) BETWEEN ~ AND : 윈도우의 시작과 끝의 위치를 지정

(4) UNBOUNDED PRECEDING : 윈도우의 첫 번째 행

(5) UNBOUNDED FOLLOWING : 윈도우의 마지막 행

(6) CURRENT ROW : 윈도우의 현재 행

● **순의 함수(RANK Function)**

(1) **RANK** : 동일한 순위는 동일한 값이 부여된다.

ex) 2등이 2명이면 3등이 없다.

(2) **DENSE_RANK** : 동일한 순위를 하나의 건수로 계산

ex) 2등이 2명이어도 3등이 있다.

(3) **ROW_NUMBER** : 동일한 순위가 있어도 고유의 순위

ex) 2등이 2명이어도 둘의 순위를 나누어 2위, 3위가 된다.

- 동일한 순위는 주로 id 값으로 정렬하여 나눈다.

● **집계 함수**

(1) SUM : 합계

(2) AVG : 평균

(3) COUNT : 행 수

(4) MAX / MIN : 최댓값 / 최솟값

● **행 순서 관련 함수**

- 상위 행의 값을 하위에 출력하거나 하위 행의 값을 상위 행에 출력하게 할 수 있다.
- 특정 위치의 행을 가지고 와서 출력할 수 있다.

(1) **FIRST_VALUE** : 가장 처음 나오는 값, MIN 함수를 이용해서 같은 결과를 구할 수 있다.

(2) **LAST_VALUE** : 가장 나중에 나오는 값, MAX 함수를 이용해서 같은 결과를 구할 수 있다.

(3) **LAG** : 이전 행을 가지고 온다.

(4) **LEAD** : 특정 위치의 행을 가지고 온다 (기본 값은 +1행)

● **비율 관련 함수**

(1) **CUME_DIST** : 파티션 전체 건수에서 현재 행보다 작거나 같은 건수에 대한 누적 백분율을 조회

- 누적 분포상의 위치를 0~1 사이의 값을 가진다.
- 결과는 0보다 크고 1보다 같거나 작다.

(2) **PERCENT_RANK** : 파티션에서 제일 먼저 나온 것을 0, 제일 늦게 나온 것을 1로 하여 행의 순서별 백분율을 조회한다.

- 결과는 0보다 같거나 크고 1보다 같거나 작다.

(3) **NTILE** : 파티션별 전체 건수를 인수 값으로 N등분한 결과를 조회한다.

(4) **RATIO_TO_REPORT** : 파티션 내 SUM(칼럼)에 대한 행 별 칼럼 값의 백분율을 소수점까지 조회된다.

- 결과는 0보다 크고 1보다 같거나 작다.

● **테이블 파티션**

- 대용량의 테이블을 여러 개의 테이블 파일에 분리해서 저장
- 테이블의 데이터가 물리적으로 분리된 데이터 파일에 저장되면 입력, 수정, 삭제, 조회 성능이 향상된다.
- 파티션을 각각의 파티션 별로 독립적으로 관리할 수 있다.
- 파티션 별로 백업하고 복구가 가능하며 파티션 전용 인덱스 생성도 가능하다.
- 파티션은 DB의 논리적 관리 단위인 테이블 스페이스 간 이동이 가능하다.
- 데이터를 조회할 때 데이터의 범위를 줄여서 성능을 향상

● **Range Partition**

테이블의 칼럼 중에서 값의 범위를 기준으로 여러 개의 파티션으로 데이터를 나누어 저장하는 방법

● **List Partition**

특정 값을 기준으로 분할하는 방법

● **Hash Partition**

DBMS가 내부적으로 해시함수를 사용해서 데이터를 분할하는 방법, 결과적으로 DBMS가 알아서 분할하고 관리하는 것

● **Composite Partition**

여러 개의 파티션 기법을 조합해서 사용하는 방법

● **DCL :** 유저를 생성하고 권한을 제어할 수 있는 명령어

● **시스템 권한 :** 사용자가 SQL문을 실행하기 위해 필요한 적절한 권한

● **Oracle 과 SQL Server의 사용자 아키텍처 차이**

(1) Oracle : 유저를 통해 DB에 접속을 하는 형태, ID와 PW방식으로 인스턴스에 접속을 하고 그에 해당하는 스키마에 오브젝트 생성 등의 권한을 부여받는다

(2) SQL Server : 인스턴스에 접속하기 위해 로그인이라는 것을 생성하게 되며, 인스턴스 내에 존재하는 다수의 DB에 연

결하여 작업하기 위해 유저를 생성한 후 로그인과 유저를 매핑해야 한다. Windows 인증 방식과 혼합 인증 방식이 있다.

√ 모든 유저는 자신이 생성한 테이블 외에 다른 유저의 테이블에 접근하려면 해당 테이블에 대한 권한을 부여받아야 한다.

● **ROLE :** 유저에게 알맞은 권한들을 한 번에 부여하기 위해 사용하는 것

● **CASCADE :** 하위 오브젝트의 권한까지 삭제

● **절차형 SQL**

SQL문의 연속적인 실행이나 조건에 따른 분기처리를 이용하여 특정 기능을 수행하는 저장 모듈을 생성할 수 있다.
Procedure, User Defined Function, Trigger 등이 있다.

● **PL/SQL 특징**

- Block 구조로 되어 있어 각 기능별로 모듈화 가능
- 변수, 상수 등을 선언하여 SQL 문장 간 값을 교환
- IF, LOOP 등의 절차형 언어를 사용하여 절차적인 프로그램이 가능
- DBMS 정의 에러나 사용자 정의 에러를 정의하여 사용 가능
- PL/SQL은 Oracle에 내장되어 있으므로 호환성이 좋다
- 응용 프로그램의 성능을 향상시킨다.
- Block 단위로 처리하므로 통신량을 줄일 수 있다.
- PL/SQL에서는 DDL을 사용할 때 EXECUTE IMMEDIATE 를 사용한다.

√ **DECLARE :** BEGIN ~ END 절에서 사용될 변수와 인수에 대한 정의 및 데이터 타입 선언부
√ **BEGIN ~ END :** 개발자가 처리하고자 하는 SQL문과 여러 가지 비교문, 제어문의 구현부, 실제 로직 처리 부분
√ **EXCEPTION :** BEGIN ~ END 에서 실행되는 SQL문이 실행될 때 에러가 발생하면 그 에러를 처리하는 것에 대한 예외처리 부분

● **T-SQL :** SQL Server를 제어하는 언어

● **User Defined Function :** RETURN을 사용해서 하나의 값을 반드시 되돌려 줘야 함, 일반적으로 프로그래밍에서 기본 형타입을 반환하는 함수라고 생각하면 됨

● **Trigger :** 특정한 테이블에 INSERT/UPDATE/DELETE 와 같은 DML문이 수행되었을 때, DB에서 자동으로 동작하도록 작성된 프로그램, 사용자 호출이 아닌 DB가 자동으로 수행

● **프로시저와 트리거의 차이점**

(1) 프로시저는 BEGIN ~ END 절 내에서 사용 / 트리거는 BEGIN ~ END 절 내에서 사용 불가
(2) 프로시저는 COMMIT, ROLLBACK 사용 가능 / 트리거는 COMMIT, ROLLBACK 사용 불가
(3) 프로시저는 EXCUTE 명령어로 실행 / 트리거는 생성 후 자동으로 실행