SQL 응용

1. 절차형 SQL 작성

(1) 프로시저

▼ 개념

- 절차형 SQL 활용, 특정 기능을 수행할 수 있는 트랜잭션 언어
- 프로시저 호출을 통해 실행, 일련의 SQL 작업을 포함하는 데이터 조작어(DML)를 수행

▼ 구성 (디비컨SET)

- 선언부(DECLARE)

프로시저 명칭, 변수와 인수, 데이터 타입 정의

- 시작/종료부(BEGIN/END)

프로시저 시작 종료 표현

다수 실행을 제어하는 기본적 단위, 논리적 프로세스 구성

- 제어부(CONTROL)

기본적으로 순차적 처리

조건문, 반복문

- SQL

DML을 주로 사용

DDL 중 TRUNCATE 사용 (테이블 구조만 남은 최초 테이블이 만들어진 상태로 돌아감)

- 예외부 (EXCEPTION)

SQL 문 실행 시 예외 발생하는 경우 예외 처리 방법 정의

- 실행부 (TRANSACTION)

트리거에서 수행된 DML 수행 내역의 DBMS의 적용 또는 취소 여부를 결정

▼ 구성 상세

- 선언부

```
-- CREATE : DBMS 내에 객체(트리거, 함수, 프로시저) 생성
-- [OR REPLACE] : 기존 프로시저 존재 시에 현재 컴파일하는 내용으로 덮어씀. (만약에 해당 명령이 없고 같은 이름의 프로시저가 존재하면 에러 발생
CREATE [OR REPLACE] PROCEDURE 프로시저명

-- 파라미터: 프로시저와 운영체제 간 필요한 값을 전송하기 위한 인자
-- [MODE] : IN (운영체제->프로시저 값 전달) / OUT (프로시저->운영체제로 처리된 결과 전달 / INOUT (두 가지 기능 동시 수행)
-- 데이터 타입 : CHAR / VARCHAR / NUMBER
파라미터명 [MODE] 데이터 타입

-- PL/SQL 블록 시작
IS[AS]

-- 프로시저 내에서 사용할 변수와 변수에 대한 초기값 설정
변수 선언
```

- 시작/종료부
- 제어부

```
-- 조건문
-- IF문
IF 조건 THEN
  문장;
ELSIF 조건 THEN
  문장;
...
ELSE
  문장;
END IF;

-- 간단한 Case 문
-- 명확한 값 가지는 집합에 대한 표현식의 값 매칭 (범위->검색된 케이스문 사용)
CASE 변수
  WHEN 값1 THEN
    SET 명령어;
  WHEN 값2 THEN
    SET 명령어;
  ...
  ELSE
    SET 명령어;
END CASE;

-- 검색된 케이스문
CASE
  WHEN 조건1 THEN
    SET 명령어;
  WHEN 조건2 THEN
    SET 명령어;
  ...
  ELSE
    SET 명령어;
END CASE;
```

```
-- 반복문
-- LOOP문
LOOP
  문장;
  EXIT WHEN 탈출조건;
END LOOP;

-- WHILE문
WHILE 반복 조건 LOOP
  문장;
  EXIT WHEN 탈출조건;
END LOOP;

-- FOR LOOP문
FOR 인덱스 IN 시작값 ... 종료값
  LOOP
  문장;
END LOOP;
```

- 프로시저 SQL

SELECT, INSERT, UPDATE, DELETE

- 예외부

```
EXCEPTION
  WHEN 조건 THEN
    SET 명령어;
```

- 실행부

프로시저에서 수행한 DML을 DBMS에 반영할지 복구할지 결정

COMMIT - 성공적으로 끝나고, 데이터베이스가 일관성 있는 상태에 있을 때 하나의 트랜잭션이 끝났을 때 사용

ROLLBACK - 비정상 종료되어 트랜잭션 원자성이 깨질 경우 처음부터 다시 시작하거나, 부분적으로 연산을 취소

▼ 프로시저 호출문 작성

- 응용 프로그램에서 호출하거나 내부 스케줄러에 의해 배치 작업을 수행하는 경우 사용
- EXECUTE / EXEC 명령어 이용
- 프로시저에 입출력 변수가 존재하는 경우 변수를 입력하여 실행해야 한다
- 가급적 프로시저에서 선언한 데이터 타입과 동일하게 입출력 변수를 넣어서 실행

```
EXCUTE 프로시저명 (파라미터1, 파라미터2, ...);
```

▼ 예시

```
-- 선언부
CREATE PROCEDURE SALES_CLOSING
-- 외부에서 입력 받을 CHAR형 변수 (마감 일자)
(V_CLOSING_DATE IN CHAR(8))
IS
-- 내부에서 사용할 변수 선언 (총매출액)
V_SALES_TOT_AMT NUMBER := 0;

-- 시작/종료부
BEGIN

-- 제어부
IF V_CLOSING_DATE < "20000101" THEN
SET
  V_CLOSING_DATE = "20200101";
END IF;

-- SQL
SELECT SUM(SALES_AMT)
  INTO V_SALES_TOT_AMT
  FROM SALES_LIST_T
WHERE SALES_DATE = V_CLOSING_DATE;

-- 예외부
EXCEPTION
  WHEN NO_DATA_FOUND THEN
    SET V_SALES_TOT_AMT = 0;

-- 변수값 테이블 삽입
INSERT INTO SALES_CLOSED_T
  (SALES_DATE, SALES_TOT_AMT)
  VALUES (V_CLOSING_DATE, V_SALES_TOT_AMT);

-- 실행부
COMMIT;
END;
```

```
-- 프로시저 호출
EXECUTE SALES_CLOSING("20180425");
```

(2) 사용자 정의 함수

▼ 개념

- 절차형 SQL을 활용하여 일련의 SQL 처리를 수행하고, **수행 결과를 단일 값으로 반환**할 수 있는 절차형 SQL
- 반환에서의 부분만 프로시저와 다름
- 사용자 정의함수의 호출을 통해 실행, 반환되는 단일 값을 조회 또는 삽입, 주성 작업에 이용하는 것이 일반적

▼ 구성 (디비컨SER)

- 선언부
- 시작/종료부
- 제어부
- SQL
- 예외부
- 반환부 - 호출문에 대한 함수 값을 반환

▼ 구문

```
CREATE [OR REPLACE] FUNCTION 함수명
-- 함수가 반환 할 데이터 타입 지정
RETURN 데이터 타입
파라미터명 [MODE] 데이터 타입
...
IS
변수 선언
```

▼ 사용자 정의 함수 호출문 작성

```
함수명 (파라미터1, 파라미터2, ...)
```

▼ 예시

```
-- 선언부
CREATE FUNCTION GET_AGE
(V_BIRTH_DATE IN CHAR(8))
IS

-- 시작/종료부
BEGIN
  V_CURRENT_YEAR CHAR(4);
  V_BIRTH_YEAR CHAR(4);
  V_AGE NUMBER;

-- 제어부
IF V_BIRTH_DATE > "30000000" THEN
SET
  V_CLOSING_DATE = "20200101"
END IF;

-- SQL
-- 현재 일자(SYSDATE)를 조회하고 연도 4자리만 파싱, 생년월일 8자리에서 4자리만 파싱하여 각각 변수에 입력
SELECT TO_CHAR(SYSDATE, 'YYYY'),
       SUBSTR(V_BIRTH_DATE, 1, 4)
INTO V_CURRENT_YEAR, V_BIRTH_YEAR
FROM DUAL;

-- 현재 연도와 생년을 숫자형식으로 변환, 두 수의 차에 1을 더하고 나이 변수에 입력
SET AGE = TO_NUMBER(V_CURRENT_YEAR) - TO_NUMBER(V_BIRTH_YEAR) + 1;

-- 반환부
RETURN AGE;
END;
```

```
-- 생년월일 값을 가지고 나이를 조회
SELECT GET_AGE("199900101")
  FROM DUAL

-- 직원 아이디를 활용하여 생일 컬럼 내의 값을 직접 활용하여 나이를 수정
UPDATE EMPLOYEE_INFO_T
  SET AGE = GET_AGE(BIRTH_DATE)
WHERE EMPLOYEE_ID = '2018001';
```

(3) 트리거

▼ 개념

- 특정 테이블에 삽입, 수정, 삭제 등의 데이터 변경 이벤트 발생하면 DBMS에서 자동적으로 실행되도록 구현된 프로그램
- 이벤트는 전체 트랜잭션 대상과 각 행에 의해 발생되는 경우 모두를 포함할 수 있으며 테이블과 뷰, DB작업을 대상으로 정의 가능

▼ 목적

- 특정 테이블에 대한 데이터 변경을 시작점으로 설정, 그와 관련된 작업을 자동적으로 수행하기 위해 트리거 사용
- 이벤트와 관련된 테이블의 데이터 삽입, 추가, 삭제 작업을 DBMS가 자동적으로 실행시키는 데 활용
- 데이터 무결성 유지 및 로그 메시지 출력 등의 별도 처리를 위해 트리거 사용

▼ 종류

- 행 트리거 - 데이터 변화가 생길 때마다 실행
- 문장 트리거 - 트리거에 의해 단 한 번 실행

▼ 구성 (디이비컨SE)

- 반환 값X, 프로시저와 유사
- EVENT 명령어를 통해 트리거 실행을 위한 이벤트를 인지, 외부 변수 IN/OUT 이 없다는 점이 차이점
- 선언부
- 이벤트부 - 트리거가 실행되는 타이밍, 이벤트를 명시
- 시작/종료부
- 제어부
- SQL
- 예외부

▼ 구문

```
CREATE [OR REPLACE] TRIGGER 트리거명
-- 이벤트부
순서 유형 ON 테이블명
[FOR EACH NOW]

-- 시작/종료부

-- 제어부

-- SQL
-- 행 트리거 안에서 OLD 및 NEW 수식자 접두어를 붙여 데이터 변경 전후 열의 값을 참조
-- 데이터 작업 / OLD / NEW
-- INSERT / NULL / 삽입된 값
-- UPDATE / 갱신 전 값 / 갱신 후 값
-- DELETE / 삭제 전 값 / NULL

-- EXCEPTION
```

- 이벤트 순서

BEFORE : 이벤트부의 테이블에 대한 INSERT, UPDATE, DELETE, SELECT 를 수행하기 전에 트리거가 실행하도록 지정

AFTER : 이벤트부의 테이블에 대한 INSERT, UPDATE, DELETE, SELECT 가 성공적으로 실행 되었을 때만 트리거가 실행하도록 지정

▼ 예시

```
-- 선언부
CREATE TRIGGER PUT_EMP_HIST

-- 이벤트부
AFTER UPDATE OR DELETE
ON EMP
FOR EACH NOW

- 시작/종료부
BEGIN

-- 제어부
IF UPDATING
THEN

-- SQL
  INSERT INTO EMP_HIST
         (EMP_ID, EMP_NAME, EMP_STATUS)
      VALUES ( :OLD.EMP_ID, :NEW.EMP_NAME, "부서이동");
ELSIF DELETING
THEN
  INSERT INTO EMP_HIST
         (EMP_ID, EMP_NAME, EMP_DEPT)
      VALUES (:OLD.EMP_ID, :OLD.EMP_NAME, "퇴사");
END IF;
END;
```

▼ 주의사항

- TCL 사용 불가

트리거 내에는 COMMIT, ROLLBACK 등의 트랜잭션 제어어(TCL) 사용 시 컴파일 에러 발생

- 오류 주의

트리거 실행 중 오류가 발생하면 트리거 실행의 원인을 제공한 데이터 작업에도 영향

특정 테이블에 데이터를 추가한 후 발생하는 트리거에서 오류가 발생할 경우 트리거 이후의 작업이 진행되지 않거나 데이터가 추가되지 않음

2. 응용 SQL 작성

(1) 집계성 SQL 작성

1) 데이터 분석 함수

▼ 개념

- 복수 행 기준의 데이터를 모아서 처리하는 것을 목적으로 하는 다중 행 함수
- 단일 행을 기반으로 산출하지 않고 **복수 행을 그룹별로 모아 놓고 그룹당 단일 계산 결과를 반환**
- GROUP BY 구문 → 복수 행 그룹핑
- SELECT, HAVING, ORDER BY 등의 구문에 활용

▼ 종류

- 집계 함수

여러 행 또는 테이블 전체 행으로 부터 하나의 결과 값을 반환

- 그룹 함수

소그룹 간의 소계 및 중계 등의 중간 합계 분석 데이터를 산출

- 윈도 함수

데이터베이스를 사용한 **온라인 분석 처리 용도**로 사용하기 위해 표준 SQL에 추가된 기능

2) 집계 함수

▼ 개념

- 여러 행 또는 테이블 전체 행으로 부터 하나의 결과 값을 반환

▼ 구문

```
SELECT 컬럼1,컬럼2, ..., 집계함수
  FROM 테이블명
[WHERE 조건]
GROUP BY 컬럼1, 컬럼2, ...
[HAVING 조건식(집계함수 포함)]
```

- WHERE 조건으로 지정된 데이터 집합 → 그룹화 된 집합에 대한 조건 선택 시에 HAVING 사용
- GROUP BY 구문 뒤 테이블을 구분하는 컬럼을 기재하여 그룹화
- HAVING 구문은 그룹화 된 집합에 대한 조건 지정 시 사용, 상수나 집약 함수, 집약 키 사용 가능

i) GROUP BY문

- 복수 ROW 대상의 데이터 분석 시 그룹핑 대상이 되는 부분 선별 필요
- 실제 구체적 데이터 분석 값을 보고자 하는 컬럼 단위를 선정할 때 사용 되는 기준, 이 부분의 조정을 통해 사용자가 원하는 분석 데이터를 볼 수 있게 해줌
- NULL 값 가지는 ROW는 제외 후 산출
- SELECT에서 사용하는 것과 같은 ALIAS 사용 불가 → GROUP BY가 먼저 실행되기 때문
- WHERE 구문 안에 포함X

ii) HAVING 구문

- WHERE 구문 내에는 사용할 수 없는 집계 함수의 구문을 적용해 복수 행의 계산 결과를 조건 별로 적용하는 데 사용

▼ 종류

- COUNT
- SUM
- AVG
- MAX
- MIN
- STDDEV - 복수 행의 해당 컬럼 간의 표준편차
- VARIANCE - 복수 행의 해당 컬럼 간의 분산

▼ 예시

```
SELECT COUNT(*)
  FROM STD
WHERE 국어 >= 80;

SELECT SUM(국어), AVG(영어)
  FROM STD

SELECT MAX(국어), MIN(국어)
  FROM STD

SELECT STDDEV(국어), VARIANCE(국어)
  FROM STD
```

3) 그룹 함수

▼ 개념

- 테이블의 전체 행을 하나 이상의 컬럼을 기준으로 컬럼 값에 따라 그룹화하여 그룹별로 결과를 출력하는 함수

▼ 유형

i) ROLLUP 함수

▼ 개념

- ROLLUP에 의해 지정된 컬럼은 소계(소그룹의 합계) 등 중간 집계 값을 산출하기 위한 그룹 함수
- 지정 컬럼의 수보다 하나 더 큰 레벨 만큼의 중간 집계 값이 생성
- 지정 컬럼은 계층별로 구성되기 때문에 순서가 바뀌면 수행 결과가 바뀜에 유의

```
-- SELECT 뒤에 포함되는 컬럼이 GROUP BY 또는 ROLLUP 뒤에 기재되어야 한다
SELECT 컬럼1, 컬럼2, ..., 그룹함수
  FROM 테이블명
[WHERE ... ]
-- 소계 집계 대상이 되는 컬럼을 ROLLUP 뒤에 기재, 소계 집계 대상이 아닌 경우 GROUP BY 뒤에 기재
GROUP BY [컬럼 ...] ROLLUP 컬럼
[HAVING ...]
-- 계층 내 정렬
[ORDER BY ...]
```

▼ 사례

```
-- 부서별 직위에 해당하는 연봉 정보, 부서별 연봉 합계, 전체 연봉 합계 출력, 소계, 중간 집계도 같이 출력
SELECT DEPT, JOB, SUM(SALARY)
  FROM DEPT_SALARY
GROUP BY ROLLUP (DEPT, JOB)
```

ii) CUBE 함수

▼ 개념

- 결합 가능한 모든 값에 대해 다차원 집계를 생성하는 그룹 함수
- 연산이 많아 시스템에 부담을 줌

```
SELECT 컬럼1, ..., 그룹 함수
  FROM 테이블명
[WHERE ...]
-- 결합 가능한 모든 값에 대해 다차원 집계 생성
-- 세분화된 소계가 구해짐
GROUP BY [컬럼1,...] CUBE(컬럼a, ...)
[HAVING ...]
[ORDER BY ...]
```

▼ 사례

```
-- 부서명, 직위에 해당되는 직위별 연봉 합계, 부서별 연봉 합계, 전체 연봉 합계 출력
SELECT DEPT, JOB, SUM(SALARY)
  FROM EMP
GROUP BY CUBE(DEPT, NAME)
```

iii) GROUPING SETS 함수

▼ 개념

- 집계 대상 컬럼들에 대한 개별 집계를 구할 수 있으며, ROLLUP이나 CUBE 와는 달리 컬럼 간 순서와 무관한 결과를 얻을 수 있는 그룹함수
- 다양한 소계 집합을 만들 수 있음

```
-- 개별 집계를 구할 수 있으며, ROLLUP 계층 구조와 달리 평등한 관계라 순서에 상관 없이 동일한 결과
SELECT 컬럼1, ..., 그룹 함수
  FROM 테이블명
[WHERE ...]
GROUP BY [컬럼1, ...] GROUPING SETS(컬럼1, ...)
[HAVING ...]
[ORDER BY ...]
```

▼ 사례

```
-- 전체 연봉 합계, 부서별 연봉 합계, 직위별 연봉 합계 출력
SELECT DEPT, NAME, SUM(SALARY)
  FROM DEPT_SALARY
GROUP BY GROUPING SETS(DEPT, NAME)
```

4) 윈도 함수

▼ 개념

- 데이터베이스를 사용한 온라인 분석 처리 용도로 사용하기 위해 표준 SQL에 추가된 함수
- OLAP 함수 - 의사결정 지원 시스템으로, 사용자가 동일한 데이터를 여러 기준을 이용하는 다양한 방식으로 바라보며 다차원 데이터 분석을 할 수 있도록 도와주는 기술

▼ 구문

```
-- PARTITION BY는 선택 항목이며, 순위를 정할 대상 범위의 컬럼을 설정. PARTITION BY 구에는 GROUP BY구가 가진 집약 기능이 없으며, 이로 인해 레코드가 줄
-- PARTITION BY 통해 구분된 레코드 집합이 윈도

SELECT 함수명(파라미터)
  OVER
  ([PARTITION BY 컬럼1, ...])
  [ORDER BY 컬럼A, ...]
   FROM 테이블명
```

▼ 분류

- 순위 함수

레코드의 순위를 계산하는 함수

▼ 해당 함수

- RANK - 동일 순위의 레코드 존재 시 후 순위는 넘어감(2위가 3개인 레코드인 경우: 2위, 2위, 2위, 5위, 6위 ...)
- DENSE_RANK - 후 순위 넘어가지 않음 (2위, 2위, 2위, 3위, 4위, ...)
- ROW_NUMBER - 동일 순위의 값이 존재해도 이와 무관하게 연속 번호를 부여(2위, 3위, 4위, 5위 ...)

```
SELECT NAME, SALARY,
      RANK() OVER (ORDER BY SALARY DESC) A,
      DENSE_RANK() OVER (ORDER BY SALARY DESC) B,
      ROW_NUMBER() OVER (ORDER BY SALARY DESC) C
  FROM EMP;
```

- 행순서 함수

레코드에서 가장 먼저 나오거나 가장 뒤에 나오는 값, 이전/이후의 값들을 출력하는 함수

▼ 해당 함수

- FIRST_VALUE - 파티션 별 윈도에서 가장 먼저 나오는 값을 찾음. 집계 함수의 MIN과 동일 결과 출력
- LAST_VALUE - 파티션 별 윈도에서 가장 늦게 나오는 값을 찾음. 집계 함수의 MAX과 동일 결과 출력
- LAG - 파티션 별 윈도에서 이전 로우 값 반환
- LEAD - 파티션 별 윈도에서 이후 로우 값 반환

```
SELECT NAME, SALARY
      FIRST_VALUE(NAME) OVER (ORDER BY SALARY DESC) A,
      LAST_VALUE(NAME) OVER (ORDER BY SALARY DESC) B,
      LAG(NAME) OVER (ORDER BY SALARY DESC) C,
      LEAD(NAME) OVER (ORDER BY SALARY DESC) D
  FROM EMP;
```

- 그룹 내 비율 함수

백분율을 보여 주거나 행의 순서 별 백분율 등 비율과 관련된 통계를 보여주는 함수

▼ 해당 함수

- RATIO_TO_REPORT - 주어진 그룹에 대해 합을 기준으로 각 로우의 상대적 비율을 반환. 결과 값은 0~1의 범위 값 가짐
- PERCENT_RANK - 주어진 그룹에 대해 제일 먼저 나오는 것을 0으로, 제일 늦게 나오는 것을 1로하여, 값이 아닌 행의 순서 별 백분율을 구함. 결과 값은 0~1의 범위 값 가짐

```
SELECT NAME, SALARY
      RATIO_TO_PERCENT(NAME) OVER (ORDER BY SALARY DESC) A,
      PERCENT_RANK(NAME) OVER (ORDER BY SALARY DESC) B
  FROM EMP;
```

(2) 특정 기능 수행 SQL문 작성

1) 응용 시스템 DBMS 접속 기술

▼ JDBC

- SQL을 사용하여 DBMS에 질의하고 데이터를 조작하는 API 제공

```
private static final String SQL_QUERY
  = "SELECT * FROM USER_INFO WHERE USER_NAME = ? AND AGE = ?";

conn = dataSource.getConnection();
sql_exec = conn.preparestatement(SQL_QUERY);
sql_exec.setString(1, user_name);
sql_exec.setString(2, age);
sql_res = sql_exec.executeQuery();
```

▼ MyBatis

- SQL Mapping 기반 오픈 소스 Access Framework로, DBMS에 질의하기 위한 SQL 쿼리를 별도의 XML 파일로 분리하고 Mapping을 통해서 SQL 실행
- DBMS 의존도가 높고 SQL 질의 언어를 사용하기 때문에 SQL 친화적인 국내 실무 개발 환경에 많이 사용

▼ 장점

- 복잡한 JDBC 코드 단순화
- SQL 거의 그대로 사용 가능
- Spring 기반 프레임워크와 통합 기능 제공
- 우수한 성능

2) MyBatis 작성 문법

i) SQL 문장의 입력 파라미터 사용 방법

- 응용 시스템을 통해 SQL을 실행할 때 입력 변수에 대한 처리 방법이 중요. #{파라미터 명}으로 처리

```
<mapper>
  <select id = "findId" resultType = "map">
    SELECT USER_ID
      FROM USER_INFO
    WHERE USER_NAME = #{user_name}
      AND BIRTH_DAY = #{user_birth_day}
  </select>
</mapper>
```

- 응용 시스템의 UI를 통해 입력 파라미터를 받고 이를 SQL문에 적용한 후 실행하는 것이 일반적
- SQL문은 미리 작성해 놓고, 사용자가 입력하는 항목에 따라 다른 조건의 SQL을 실행하는 것이 목적

ii) 동적 SQL

- 조건에 따라 SQL 구문 자체를 변경 가능
- 응용 프로그램 실행 시 수행할 SQL문이 결정. SQL에 포함된 다양한 부분을 조건에 따라 변경 가능

```
<mapper>
  <select id="findId" resultType = "map">
    SELECT USER_ID
      FROM USER_INFO
    WHERE USER_NAME = #{user_name}

    <if input_para="user_birth_day != null">
      AND BIRTH_DAY = #{user_birth_day}
    </if>

    <if input_para="user_email != null">
      AND EMAIL = #{user_email}
    </if>
  </select>
</mapper>
```

- if 외에도 다중 문자열 등의 반복적 입력 시 사용하는 <foreach>, if 보다 좀 더 경우 별 SQL문 적용을 위한 <choose when otherwise> 등 사용

iii) **절차형 SQL 호출**

- 사용자 정의 함수, 트리거, 프로시저 실행 가능
- 사용자 정의함수는 DQL이나 DML에 포함하여 사용하기 때문에 별도 호출이 의미 없음, 트리거 역시 DBMS에서 바로 실행 되므로 별도 호출 불필요
- 주로 프로시저 호출
- statementType 반드시 'CALLABLE'로 설정, 프로시저 호출 전에도 'CALL' 문장 사용

```
<select id = "execSalesDailyBatch"
        resultType = "map"
        statementType = "CALLABLE">
  {CALL RUN_SALES_AMT_BATCH(#{TARGET_DATE})}
</select>
```

(3) 데이터 제어어 명령문 작성

▼ 개념

- 데이터베이스 관리자가 데이터 보안, 무결성 유지, 병행 제어, 회복을 위해 **관리자(DBA)가 사용하는 제어용 언어**

▼ 유형

- GRANT

사용 권한 부여

관리자가 사용자에게 데이터베이스에 대한 권한을 부여

- REVOKE

사용 권한 취소

관리자가 사용자에게 부여 했던 권한을 회수

▼ 데이터 제어어(DCL) 명령문

- GRANT

```
GRANT 권한 ON 테이블 TO 사용자
-- 사용자가 권한을 받고난 후 다른 사람들과 권한을 나눠가질 수 있는 옵션
[WITH 권한 옵션];

GRANT UPDATE ON 학생 TO 장길산 WITH GRANT OPTION;
```

- REVOKE

```
REVOKE 권한 ON 테이블 FROM 사용자
-- 연쇄적인 권한을 해제할 때 입력(WITH GRANT OPTION으로 부여된 사용자들의 권한까지 취소)
[CASCADE CONSTRAINT];

REVOKE UPDATE ON 학생 FROM 장길산 CASCADE CONSTRAINTS;
```