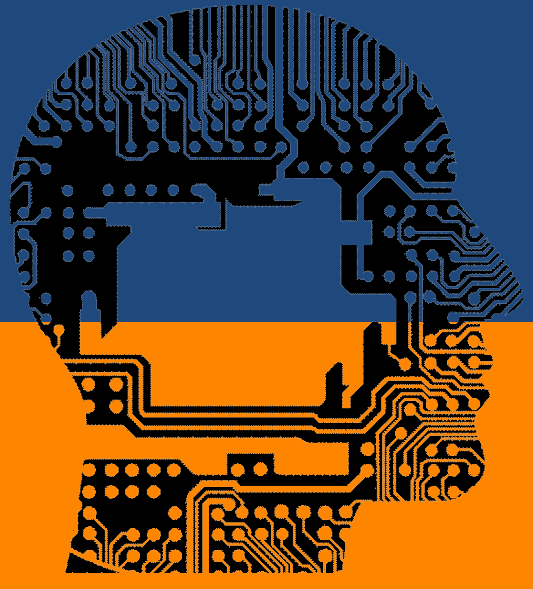
한국IT진흥부설 정보보호교육학원 아이섹

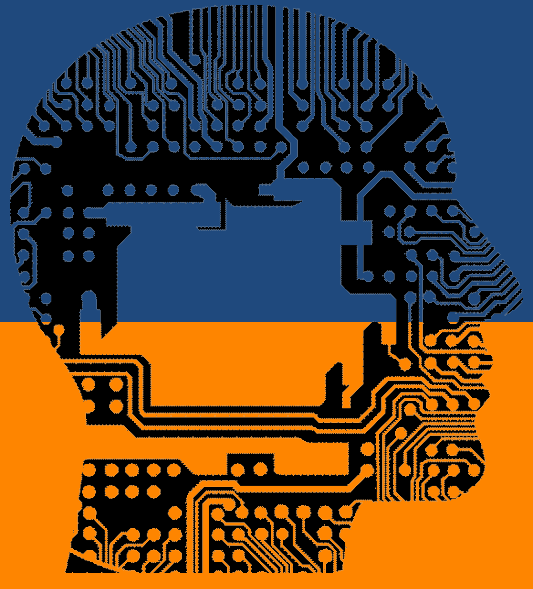


JAVA 웹 개발자 양성과정
DataBase - SQLD

4강 - DCL, 절차형 SQL

By SoonGu Hong

한국IT진흥부설
정보보호교육학원 아이섹



JAVA 웹 개발자 양성과정

DataBase

1. DCL

➢ DCL이란?

① 유저를 생성하고 권한을 제어할 수 있는 명령어
② 데이터의 보호와 보안을 위해서 유저와 권한을 관리 해야함

➢ 오라클에서 제공하는 유저들

유저	설명
SCOTT	• 테스트용 샘플 유저
SYS	• DBA 권한을 부여 받은 유저 (최상위 유저)
SYSTEM	• SYSTEM 데이터베이스의 모든 시스템 권한을 부여 받은 유저(SYS바로 밑)

➢ 유저 생성(SYSTEM계정으로 접속)

```
DROP USER SQLD_TEST CASCADE;
CREATE USER SQLD_TEST IDENTIFIED BY 1234;
```

➢ SQLD_TEST로 접속 시도

```
ORA-01045: 사용자 SQLD_TEST는 CREATE SESSION 권한을 가지고 있지 않음, 로그온이 거절되었습니다.
```

❖ 에러 발생

➢ 접속 권한 주기(SYSTEM계정으로 접속)

```
GRANT CREATE SESSION TO SQLD_TEST;
```

➢ SQLD_TEST로 접속 시도(SQLD_TEST 계정으로 접속)

❖ SQLD_TEST로 접속 성공

```
CREATE TABLE TB_SQLD_TEST
(
  SQLD_TEST_NO CHAR(10)
, SQLD_TEST_NM VARCHAR2(50)
, CONSTRAINT PK_TB_SQLD_TEST PRIMARY KEY(SQLD_TEST_NO)
)
;
```

❖ 테이블 생성 실패

```
ORA-01031: 권한이 불충분합니다
```

➢ 테이블 생성 권한 주기(SYSTEM계정으로 접속)

```
GRANT CREATE TABLE TO SQLD_TEST;
```

➢ 테이블 생성 재시도(SQLD_TEST 계정으로 접속)

```
CREATE TABLE TB_SQLD_TEST
(
  SQLD_TEST_NO CHAR(10)
, SQLD_TEST_NM VARCHAR2(50)
, CONSTRAINT PK_TB_SQLD_TEST PRIMARY KEY(SQLD_TEST_NO)
)
;
```

❖ 테이블 생성 성공

```
SELECT * FROM TB_SQLD_TEST;
```

❖ 테이블 조회 성공

➢ SQLD_TEST로 SQLD 유저의 테이블 조회 시도(SQLD_TEST 계정으로 접속)

```
SELECT * FROM SQLD.TB_EMP;
```

❖ 테이블 조회 실패

```
ORA-00942: 테이블 또는 뷰가 존재하지 않습니다
```

➢ 테이블 조회 권한 주기(SYSTEM계정으로 접속)

```
GRANT SELECT ON SQLD.TB_EMP TO SQLD_TEST;
```

➢ SQLD_TEST로 SQLD 유저의 테이블 조회 재시도(SQLD_TEST 계정으로 접속)

```
SELECT * FROM SQLD.TB_EMP;
```

❖ 테이블 조회 성공

➢ 테이블 DML 권한 주기(SYSTEM계정으로 접속)

```
GRANT INSERT, DELETE, UPDATE ON SQLD.TB_EMP TO SQLD_TEST;
```

❖ INSERT, DELETE, UPDATE 권한도 줌
❖ SQLD_TEST 계정은 해당 테이블에 대한 DML도 수행할 수 있게 됨

➢ ROLE을 이용한 권한 부여

① 유저를 생성하면 다양한 많은 권한들을 부여 해야함
② DBA는 ROLE을 생성하고 ROLE에 각종 권한을 부여한 후 해당 ROLE을 다른 유저에게 부여
③ ROLE에 포함된 권한들이 필요한 유저에게 빠르게 권한을 부여할 수 있음

➢ 테이블 생성 권한 회수(SYSTEM계정으로 접속)

```
REVOKE CREATE TABLE FROM SQLD_TEST;
```

❖ SQLD_TEST가 가진 CREATE TABLE 권한 회수

➢ SQLD_TEST로 테이블 생성 시도 (SQLD_TEST 계정으로 접속)

```
CREATE TABLE TB_SQLD_TEST_2
(
  SQLD_TEST_NO CHAR(10)
, SQLD_TEST_NM VARCHAR2(50)
, CONSTRAINT PK_TB_SQLD_TEST PRIMARY KEY(SQLD_TEST_NO)
)
;
```

❖ 테이블 생성 실패

```
ORA-01031: 권한이 불충분합니다
```

➢ 롤을 생성한 후 SQL_TEST에게 롤 권한 부여(SYSTEM계정으로 접속)

```
DROP ROLE CREATE_SESSION_TABLE;
CREATE ROLE CREATE_SESSION_TABLE;
GRANT CREATE SESSION, CREATE TABLE TO CREATE_SESSION_TABLE;
GRANT CREATE_SESSION_TABLE TO SQLD_TEST;
```

➢ SQLD_TEST로 테이블 생성 재시도 (SQLD_TEST 계정으로 접속)

```
CREATE TABLE TB_SQLD_TEST_2
(
  SQLD_TEST_NO CHAR(10)
, SQLD_TEST_NM VARCHAR2(50)
, CONSTRAINT PK_TB_SQLD_TEST PRIMARY KEY(SQLD_TEST_NO)
)
;
```

❖ 테이블 생성 성공

➢ 오라클 DBMS에서 일반적으로 부여하는 ROLE

ROLE명	부여 권한
CONNECT	• CREATE SESSION
RESOURCE	• CREATE CLUSTER • CREATE PROCEDURE • CREATE TYPE • CREATE SEQUENCE • CREATE TRIGGER • CREATE OPERATOR • CREATE TABLE • CREATE INDEXTYPE

➢ SQLD_TEST유저 제거 및 생성, 기본 롤 부여(SYSTEM계정으로 접속)

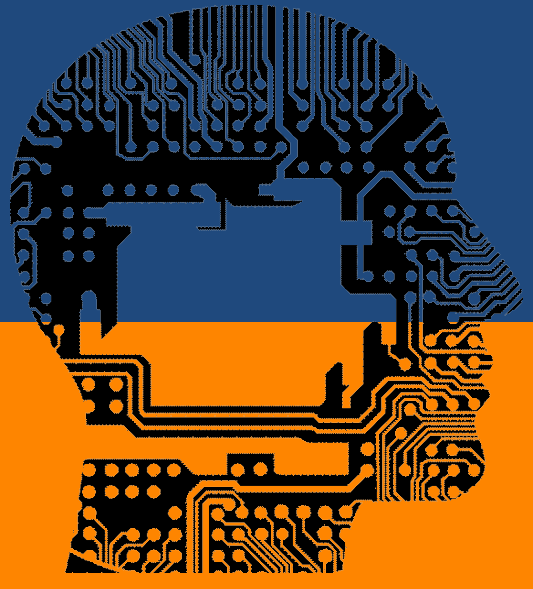
```
DROP USER SQLD_TEST CASCADE; --생성한 테이블도 같이 제거됨
CREATE USER SQLD_TEST IDENTIFIED BY 1234;

GRANT CONNECT, RESOURCE TO SQLD_TEST;
```

➢ 테이블 생성 시도(SQLD_TEST 계정으로 접속)

```
DROP TABLE TB_SQLD_TEST;
CREATE TABLE TB_SQLD_TEST
(
  SQLD_TEST_NO CHAR(10)
, SQLD_TEST_NM VARCHAR2(50)
, CONSTRAINT PK_TB_SQLD_TEST PRIMARY KEY(SQLD_TEST_NO)
)
;
```

❖ 테이블 생성 성공

한국IT진흥부설
정보보호교육학원 아이섹



JAVA 웹 개발자 양성과정

DataBase

2. 절차형 SQL

➢ 절차 형SQL이란?

① 일반적인 개발언어처럼 SQL문도 절차지향적인 프로그램 작성이 가능하도록 절차 형 SQL을 제공한다.
② 절차 형 SQL을 사용하면 SQL문의 연속적인 실행이나 조건에 따른 분기 처리를 수행하는 모듈을 생성할 수 있다.
③ 오라클 기준 이러한 절차 형 모듈의 종류는 사용자정의함수, 프로시저, 트리거가 있다.
④ 오라클 기준 이러한 절차 형 모듈을 PL/SQL이라고 부른다.

➢ PL/SQL개요

① PL/SQL은 Block 구조로 되어 있고 Block 내에는 SQL문, IF, LOOP 등이 존재함
② PL/SQL을 이용해서 다양한 모듈을 개발 가능

➢ PL/SQL개요

① Block구조로 되어있으며 각 기능별로 모듈화가 가능
② 변수/상수 선언 및 IF/LOOP문 등의 사용이 가능
③ DBMS에러나 사용자 에러 정의를 할 수 있음
④ PL/SQL은 오라클에 내장 시킬수 있으므로 어떠한 오라클 서버로도 이식이 가능
⑤ PL/SQL은 여러 SQL문장을 Block으로 묶고 한번에 Block전부를 서버로 보내기때문에 네트워크 패킷 수를 감소시킴

➢ PL/SQL Block구조

구조 명	필수/선택	설명
DECLARE(선언 부)	필수	• BEGIN~END에서 사용할 변수나 인수에 대한 정의 및 데이터 타입 선언
BEGIN(실행 부)	필수	• 개발자가 처리하고자 하는 SQL문과 필요한 LOGIC (비교 문, 제어 문 등)이 정의되는 실행 부
EXCEPTION(예외 처리 부)	선택	• BEGIN~END에서 실행되는 SQL문에 발생된 에러를 처리하는 에러 처리 부
END	필수	

❖ PL/SQL은 DECLARE, BEGIN, EXCEPTION, END로 이루어져 있으며 그중 EXCEPTION은 선택항목이고 나머지는 필수 항목이다.

➢ 오라클 프로시저 실습 준비

```
DROP TABLE TB_EMP_PAY_BY_YEAR;
CREATE TABLE TB_EMP_PAY_BY_YEAR
(
  EMP_NO CHAR(10) NOT NULL
, STD_YEAR CHAR(4) NOT NULL
, PAY_AMT NUMBER(15) NOT NULL
, CONSTRAINT PK_TB_EMP_PAY_BY_YEAR PRIMARY KEY(EMP_NO, STD_YEAR)
)
;
```

❖ "연별직원급여지급내역" 테이블을 생성한다.

➢ 오라클 프로시저 생성

```
CREATE OR REPLACE PROCEDURE SP_INSERT_TB_EMP_PAY_BY_YEAR
(IN_STD_YEAR IN TB_EMP_PAY_BY_YEAR.STD_YEAR%TYPE)
IS
V_EMP_NO TB_SAL_HIS.EMP_NO%TYPE;
V_SEX_CD TB_EMP.SEX_CD%TYPE;
V_STD_YEAR TB_EMP_PAY_BY_YEAR.STD_YEAR%TYPE;
V_PAY_AMT TB_SAL_HIS.PAY_AMT%TYPE;
CURSOR SELECT_TB_EMP IS
SELECT B.EMP_NO
     , (SELECT L.SEX_CD FROM TB_EMP L WHERE L.EMP_NO = B.EMP_NO) AS SEX_CD
     , SUBSTR(B.PAY_DE, 1, 4) AS STD_YEAR
     , SUM(B.PAY_AMT) AS PAY_AMT
  FROM TB_SAL_HIS B
 WHERE B.PAY_DE BETWEEN IN_STD_YEAR||'0101' AND IN_STD_YEAR||'1231'
 GROUP BY B.EMP_NO, SUBSTR(B.PAY_DE, 1, 4)
 ORDER BY B.EMP_NO;

BEGIN
    OPEN SELECT_TB_EMP;
    DBMS_OUTPUT.PUT_LINE('-----------------------------------------------------');
    LOOP
        FETCH SELECT_TB_EMP INTO V_EMP_NO, V_SEX_CD, V_STD_YEAR, V_PAY_AMT;
        EXIT WHEN SELECT_TB_EMP%NOTFOUND;

        DBMS_OUTPUT.PUT_LINE('V_EMP_NO   : '  || '[' || V_EMP_NO ||']');
        DBMS_OUTPUT.PUT_LINE('V_SEX_CD   : '  || '[' || V_SEX_CD ||']');
        DBMS_OUTPUT.PUT_LINE('V_STD_YEAR : '  || '[' || V_STD_YEAR ||']');
        DBMS_OUTPUT.PUT_LINE('V_PAY_AMT  : '  || '[' || V_PAY_AMT ||']');

        IF V_SEX_CD = '1' THEN
            INSERT INTO TB_EMP_PAY_BY_YEAR VALUES (V_EMP_NO, V_STD_YEAR, V_PAY_AMT);
        END IF;
    END LOOP;
    CLOSE SELECT_TB_EMP;

    COMMIT;
    DBMS_OUTPUT.PUT_LINE('-----------------------------------------------------');
END SP_INSERT_TB_EMP_PAY_BY_YEAR;
/
```

❖ 입력한 년(YYYY) 기준 직원 별 연봉 액수를 TB_EMP_PAY_BY_YEAR 테이블에 저장함

❖ 성별이 남자인 직원에 한해서만 저장함

➢ 오라클 프로시저 실행

```
TRUNCATE TABLE TB_EMP_PAY_BY_YEAR;
EXEC SP_INSERT_TB_EMP_PAY_BY_YEAR('2019');
```

❖ 입력 값을 2019로 함
❖ 2019년도 직원 별 급여 지급 합계를 구하게 됨

❖ SQLPLUS에서 실행함

```
Microsoft Windows [Version 10.0.18363.900]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\dbmsexpert>SQLPLUS SQLD/1234

SQL*Plus: Release 12.2.0.1.0 Production on Sat Jul 4 11:21:29 2020

Copyright (c) 1982, 2016, Oracle.  All rights reserved.

Last Successful login time: Fri Jul 03 2020 22:23:22 +09:00

Connected to:
Oracle Database 12c Enterprise Edition Release 12.2.0.1.0 - 64bit Production

SQL> EXEC SP_INSERT_TB_EMP_PAY_BY_YEAR('2019');

PL/SQL procedure successfully completed.

SQL>
```

```
SELECT * FROM TB_EMP_PAY_BY_YEAR;
```

EMP_NO	STD_YEAR	PAY_AMT
1000000001	2019	46270000
1000000002	2019	53530000
1000000003	2019	45740000
1000000004	2019	49220000
1000000009	2019	48800000
1000000012	2019	48990000
1000000014	2019	58690000
1000000015	2019	49180000
1000000017	2019	49240000
1000000021	2019	46780000
1000000023	2019	46430000
1000000024	2019	47140000
1000000026	2019	50780000
1000000030	2019	50210000
1000000032	2019	53490000
1000000033	2019	50670000
1000000035	2019	49090000
1000000039	2019	51000000
9999999999	2019	44330000

❖ 데이터가 저장되어 있는 것을 확인

➢ 사용자 정의 함수란?

① 사용자 정의 함수는 프로시저처럼 SQL문을 IF/LOOP등의 LOGIC와 함께 데이터베이스에 저장해 놓은 명령문의 집합이다.
② SUM, AVG, NVL의 함수처럼 호출해서 사용할 수 있다.
③ 프로시저와 차이점은 반드시 한 건을 되돌려 줘야 한다는 것이다.

➢ 사용자 정의 함수 생성

```
CREATE OR REPLACE FUNCTION
FUNC_EMP_CNT_BY_DEPT_CD(IN_DEPT_CD IN TB_DEPT.DEPT_CD%TYPE)
RETURN NUMBER IS V_EMP_CNT NUMBER;

BEGIN
SELECT COUNT(*) CNT
  INTO V_EMP_CNT
  FROM TB_EMP
 WHERE DEPT_CD = IN_DEPT_CD
 ;

RETURN V_EMP_CNT;
END;
/
```

❖ SQLPLUS에서 실행함

➢ 사용자 정의 함수 생성 - SQLPLUS

```
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\dbmsexpert>SQLPLUS SQLD/1234

SQL*Plus: Release 12.2.0.1.0 Production on Sat Jul 4 11:29:13 2020

Copyright (c) 1982, 2016, Oracle.  All rights reserved.

Last Successful login time: Sat Jul 04 2020 11:21:46 +09:00

Connected to:
Oracle Database 12c Enterprise Edition Release 12.2.0.1.0 - 64bit Production

SQL> CREATE OR REPLACE FUNCTION FUNC_EMP_CNT_BY_DEPT_CD(IN_DEPT_CD IN TB_DEPT.DEPT_CD%TYPE)
  2  RETURN NUMBER IS V_EMP_CNT NUMBER;
  3
  4  BEGIN
  5  SELECT COUNT(*) CNT
  6    INTO V_EMP_CNT
  7    FROM TB_EMP
  8   WHERE DEPT_CD = IN_DEPT_CD
  9   ;
 10
 11  RETURN V_EMP_CNT;
 12  END;
 13  /

Function created.

SQL>
```

➢ 사용자 정의 함수 호출

```
SELECT A.DEPT_CD
     , A.DEPT_NM
     , FUNC_EMP_CNT_BY_DEPT_CD(A.DEPT_CD) AS EMP_CNT
FROM TB_DEPT A
;
```

DEPT_CD	DEPT_NM	EMP_CNT
100001	운영본부	1
100002	지원팀	4
100003	기획팀	4
100004	디자인팀	4
100005	플랫폼사업본부	1
100006	데이터팀	4
100007	개발팀	4
100008	솔루션사업본부	1
100009	운영팀	4
100010	개발팀	4
100011	신사업본부	1
100012	인공지능팀	4
100013	빅데이터팀	4
999999	회장실	1

➢ 트리거 란?

① 특정한 테이블에 INSERT, UPDATE, DELETE를 수행할 때 DBMS내에서 자동으로 동작하도록 작성된 프로그램이다.
② 즉 사용자가 직접 호출하는 것이 아니고 DBMS가 자동적으로 수행한다.
③ 트리거는 테이블과 뷰, DB작업을 대상으로 정의 할수 있으며 전체 트랜잭션 작업에 대해 발생되는 트리거와 각행에 대해 발생되는 트리거가 있다.

➢ 트리거 실습 준비

❖ 직원 별 급여지급 합계를 저장하는 테이블을 생성

```
DROP TABLE TB_EMP_PAY_SUM;

CREATE TABLE TB_EMP_PAY_SUM
(
   EMP_NO    CHAR(10)Not Null
,  PAY_AMT_SUM NUMBER(15) Not Null
, CONSTRAINT PK_TB_SAL_HIS_SUM PRIMARY KEY(EMP_NO)
)
;

INSERT INTO TB_EMP_PAY_SUM
SELECT EMP_NO, SUM(PAY_AMT) FROM TB_SAL_HIS
GROUP BY EMP_NO
ORDER BY EMP_NO
;

COMMIT;
```

➢ 트리거 생성

```
CREATE OR REPLACE TRIGGER TRIG_TB_SAL_HIS_INSERT
AFTER INSERT
ON TB_SAL_HIS
FOR EACH ROW
DECLARE
V_EMP_NO TB_SAL_HIS.EMP_NO%TYPE;

BEGIN
V_EMP_NO := :NEW.EMP_NO;

UPDATE TB_EMP_PAY_SUM A
SET A.PAY_AMT_SUM = A.PAY_AMT_SUM + :NEW.PAY_AMT
WHERE A.EMP_NO = V_EMP_NO;

IF SQL%NOTFOUND THEN
    INSERT INTO TB_EMP_PAY_SUM VALUES (V_EMP_NO, :NEW.PAY_AMT);
END IF;
END;
/
```

- ❖ SQLPLUS에서 실행함
- ❖ 해당 트리거는 TB_EMP 테이블에 INSERT시 TB_DEPT 테이블의 EMP_CNT를 UPDATE하는 트리거임

➢ 트리거 생성 - SQLPLUS

```
SQL*Plus: Release 12.2.0.1.0 Production on Sat Jul 4 11:35:37 2020

Copyright (c) 1982, 2016, Oracle.  All rights reserved.

Last Successful login time: Sat Jul 04 2020 11:35:02 +09:00

Connected to:
Oracle Database 12c Enterprise Edition Release 12.2.0.1.0 - 64bit Production

SQL> CREATE OR REPLACE TRIGGER TRIG_TB_SAL_HIS_INSERT
  2  AFTER INSERT
  3  ON TB_SAL_HIS
  4  FOR EACH ROW
  5  DECLARE
  6  V_EMP_NO TB_SAL_HIS.EMP_NO%TYPE;
  7
  8  BEGIN
  9  V_EMP_NO := :NEW.EMP_NO;
 10
 11  UPDATE TB_EMP_PAY_SUM A
 12  SET A.PAY_AMT_SUM = A.PAY_AMT_SUM + :NEW.PAY_AMT
 13  WHERE A.EMP_NO = V_EMP_NO;
 14
 15  IF SQL%NOTFOUND THEN
 16      INSERT INTO TB_EMP_PAY_SUM VALUES (V_EMP_NO, :NEW.PAY_AMT);
 17  END IF;
 18  END;
 19  /

Trigger created.

SQL>
```

➢ 트리거 실습

```
INSERT INTO TB_SAL_HIS VALUES (
(  SELECT TO_CHAR(NVL(MAX(SAL_HIS_NO), 0) + 1) AS SAL_HIS_NO FROM TB_SAL_HIS)
 , TO_CHAR(SYSDATE, 'YYYYMMDD'), 1000000 , '9999999999'
);
 COMMIT;
```

```
SELECT A.EMP_NO
     , A.PAY_AMT_SUM
  FROM TB_EMP_PAY_SUM a
 WHERE EMP_NO = '9999999999'
;
```

EMP_NO	PAY_AMT_SUM
9999999999	103110000

❖ TB_SAL_HIS 테이블에 데이터를 입력하면 TB_EMP_PAY_SUM 테이블에 데이터가 입력됨

➢ 트리거 실습 - 새로운 직원을 입력

```
ALTER TABLE TB_SAL_HIS DROP CONSTRAINT FK_TB_SAL_HIS01;
```

❖ 실습을 위해 존재하지 않는 직원번호를 입력하므로 참조 무결성 제약 조건 잠시 제거

```
INSERT INTO TB_SAL_HIS VALUES (
( SELECT TO_CHAR(NVL(MAX(SAL_HIS_NO), 0) + 1) AS SAL_HIS_NO FROM TB_SAL_HIS)
, TO_CHAR(SYSDATE, 'YYYYMMDD'), 1000000, '1234567890'
);
COMMIT;
```

```
SELECT A.EMP_NO
     , A.PAY_AMT_SUM
  FROM TB_EMP_PAY_SUM A
 WHERE EMP_NO = '1234567890'
;
```

EMP_NO	PAY_AMT_SUM
1234567890	1000000

```
DELETE FROM TB_SAL_HIS WHERE EMP_NO = '1234567890';
COMMIT;
```

❖ 테스트 데이터를 삭제 후 참조 무결성 제약조건을 원상 복귀 시킴

```
ALTER TABLE SQLD.TB_SAL_HIS ADD CONSTRAINT FK_TB_SAL_HIS01 FOREIGN KEY (EMP_NO) REFERENCES SQLD.TB_EMP (EMP_NO) NOVALIDATE;
```

➢ 프로시저와 트리거의 차이점

프로시저	트리거
CREATE PROCEDURE 문법 사용	CREATE TRIGGER 문법 사용
EXECUTE/EXEC 명령어로 실행	생성 후 자동으로 실행
내부에서 COMMIT, ROLLBACK 실행가능	내부에서 COMMIT, ROLLBACK 실행 안됨

감사합니다
THANK YOU