**1. SUBPROGRAM**

PL/SQL을 지원하는 어떤 툴이나 언어에서도 SUBPROGRAM(프로시저와 함수)을 실행할 수 있다. PL/SQL내부에서 식의 일부로서 함수를 실행할 수 있다. EXECUTE는 명령 다음에 입력되는 Stored Procedure를 실행한다.

**1.1 SUBPROGRAM의 개요**

PL/SQL 프로시저와 함수는 3GL의 프로시저 및 함수와 매우 비슷하게 동작된다. 모듈화를 통해 관리가 용이하고 적절히 논리적 단위로 나누어진 프로그래밍을 할 수 있다. 즉, 잘 정의된 논리적인 단위로 코드를 분할할 수 있다. PL/SQL에서 이들 단위를 단위 프로그램 또는 SUBPROGRAM이라 부른다. PL/SQL에는 프로시저와 함수라는 두 가지 유형의 SUBPROGRAM이 있다. SUBPROGRAM은 컴파일된 상태로 데이터베이스에 저장되어 있어 Performance가 향상된다.

**1.2 SUBPROGRAM 작성 단계**

**1.2.1 구문 작성**

TEXT 편집기를 이용하여 SCRIPT FILE에 CREATE PROCEDURE나 CREATE FUNCTION문을 작성한다.

|  |
| --- |
| SQL> ed emp\_up  CREATE OR REPLACE PROCEDURE emp\_sal\_update(  p\_empno IN emp.empno%TYPE, p\_sal IN emp.sal%TYPE)  IS  BEGIN  UPDATE emp  SET sal = p\_sal  WHERE empno = p\_empno;  IF SQL%NOTFOUND THEN  DBMS\_OUTPUT.PUT\_LINE(TO\_CHAR(p\_empno) ||  '는 없는 사원번호입니다.');  ELSE  DBMS\_OUTPUT.PUT\_LINE(TO\_CHAR(SQL%ROWCOUNT) ||  '명의 자료를 수정하였습니다.');  END IF;  END emp\_sal\_update;  / |

**1.2.2 코드 컴파일**

SCRIPT FILE을 실행 시켜 컴파일하여 컴파일된 코드를 데이터베이스에 저장한다.

|  |
| --- |
| SQL> **@emp\_up**  Procedure created. |

**1.2.3 에러 수정**

코드 컴파일시 에러가 발생하면 에러를 확인하고 수정하여 코드를 다시 컴파일한다.

|  |
| --- |
| SQL> **@emp\_up**  Warning: Procedure created with compilation errors.  SQL> **ed emp\_up**  -- emp\_up를 수정한 후 저장하고 종료한다.  SQL> **@emp\_up**  Procedure created. |

**1.2.4 실행**

SQL\*Plus에서 EXECUTE명령으로 SUBPROGRAM을 실행한다.

|  |
| --- |
| SQL> **EXECUTE emp\_sal\_update(7788,3500)**  PL/SQL procedure successfully completed.  SQL> **SELECT empno,ename,job,sal**  2 **FROM emp**  3 **WHERE empno = 7788;**  EMPNO ENAME JOB SAL  --------- ---------- --------- ---------  7788 SCOTT ANALYST 3500 |

♣ 참고

변경된 내용을 확인하기 위해 SELECT문을 사용할 수 있다.

**1.3 PROCEDURE 생성**

나중에 실행할 일련의 동작을 저장하기 위해 PL/SQL프로시저를 작성한다. 프로시저는 실행할 때 사용하는 Parameter가 없거나 여러 개를 가질 수도 있다. 프로시저에서는 DECLARE절이 생략되고 IS와 BEGIN사이에 필요한 변수를 선언하여 사용한다

**1.3.1 Syntax**

|  |
| --- |
| **CREATE** [**OR REPLACE**] **PROCEDURE** procedure\_name  [(argument1 [mode1] datatype [{:= | **DEFAULT**} expression]  [,argument2 [mode2] datatype [{:= | **DEFAULT**} expression], . . .])]  {**IS** | **AS**}  **BEGIN**  pl/sql\_block;  **END**; |

OR REPLACE procedure\_name이 존재할 경우 PROCEDURE의 내용을 지우고 다시 생성

procedure\_name PROCEDURE 명

argument 매개변수의 이름

mode 3가지가 있다

IN : 입력 매개변수로 상용

OUT : 출력 매개변수로 사용

IN OUT : 입력, 출력 매개변수로 상용

pl/sql\_block PROCEDURE를 구성하는 코드를 구성하는 PL/SQL의 블록

☞ Guidelines

1. SQL\*Plus에서 프로시저를 작성할 때 CREATE OR REPLACE를 사용하시오.
2. 어떠한 Parameter라도 사용 가능하다.
3. IS로 PL/SQL블록을 시작하시오.
4. Local변수 선언은 IS와 BEGIN사이에 선언 하시오.

**1.3.2 PROCEDURE 실행**

PL/SQL을 지원하는 어떤 툴이나 언어에서도 프로시저를 실행할 수 있다. SQL\*Plus에서 프로시저 호출은 Stored Procedure를 참조하는 PL/SQL 문을 실행하기 위해 EXECUTE 명령을 사용할 수 있다. EXECUTE는 명령 다음에 입력되는 Stored Procedure를 실행한다.

**가) Syntax**

|  |
| --- |
| procedure\_name[(argument1[,argument2, . . . .])] |

**나) SQL\*Plus에서 프로시저 실행**

EXECUTE는 명령 다음에 입력되는 Stored Procedure를 실행한다.

|  |
| --- |
| SQL> **EXECUTE emp\_sal\_update(7902,4000)**  PL/SQL procedure successfully completed. |

문제1) EMP 테이블에 새로운 사원의 정보를 이름,업무,매니저,급여를 입력받아 등록하는 프로시저를 생성하여라. 단 부서 번호는 매니저의 부서 번호와 동일하게 하고 보너스는 SALESMAN은 0을 그 외는 NULL을 입력하여라.

|  |
| --- |
| CREATE OR REPLACE PROCEDURE emp\_input(  v\_name IN emp.ename %TYPE,  v\_job IN emp.job %TYPE,  v\_mgr IN emp.mgr %TYPE,  v\_sal IN emp.sal %TYPE)  IS  v\_comm emp.comm%TYPE;  v\_deptno emp.deptno%TYPE;  manager\_error EXCEPTION;  BEGIN  IF UPPER(v\_job) NOT IN ('PRESIDENT','MANAGER','ANALYST',  'SALESMAN','CLERK') THEN  RAISE manager\_error;  ELSIF UPPER(v\_job) = 'SALESMAN' THEN  v\_comm := 0;  ELSE  v\_comm := NULL;  END IF;  SELECT deptno  INTO v\_deptno  FROM emp  WHERE empno = v\_mgr;  INSERT INTO emp  VALUES (empno\_sequence.NEXTVAL,v\_name,UPPER(v\_job),  v\_mgr,SYSDATE,v\_sal,v\_comm,v\_deptno);  EXCEPTION  WHEN manager\_error THEN  DBMS\_OUTPUT.PUT\_LINE('담당 업무가 잘못 입력되었습니다.');  WHEN NO\_DATA\_FOUND THEN  DBMS\_OUTPUT.PUT\_LINE('입력한 MANAGER는 없습니다.');  WHEN OTHERS THEN  DBMS\_OUTPUT.PUT\_LINE('기타 에러입니다.');  END;  / |
| SQL> **SET SERVEROUTPUT ON**  SQL> **EXECUTE emp\_input('YOONJB','MANAGER',7788,2500)** |

문제2) 이름을 입력받아 그 사원의 정보 중 부서명과 급여를 검색하는 프로시저를 생성하여라.

|  |
| --- |
| CREATE OR REPLACE PROCEDURE dname\_sal\_disp(  v\_ename IN emp.ename%TYPE,  v\_dname OUT dept.dname%TYPE,  v\_sal OUT emp.sal%TYPE)  IS  v\_deptno emp.deptno%TYPE;  BEGIN  SELECT sal,deptno  INTO v\_sal,v\_deptno  FROM emp  WHERE ename = UPPER(v\_ename);  SELECT dname  INTO v\_dname  FROM dept  WHERE deptno = v\_deptno;  EXCEPTION  WHEN NO\_DATA\_FOUND THEN  DBMS\_OUTPUT.PUT\_LINE('입력한 MANAGER는 없습니다.');  WHEN TOO\_MANY\_ROWS THEN  DBMS\_OUTPUT.PUT\_LINE('자료가 2건 이상입니다.');  WHEN OTHERS THEN  DBMS\_OUTPUT.PUT\_LINE('기타 에러입니다.');  END;  / |
| SQL> **VAR g\_dname VARCHAR2(14)**  SQL> **VAR g\_sal NUMBER**  SQL> **EXECUTE dname\_sal\_disp('SCOTT',:g\_dname,:g\_sal);**  PL/SQL procedure successfully completed.  SQL> **PRINT g\_dname**  G\_DNAME  --------------------------------  ACCOUNTING  SQL> **PRINT g\_sal**  G\_SAL  ---------  3000 |

문제3) 이름을 입력받아 그 사원의 정보 중 부서명과 급여를 검색하는 프로시저를 생성하여라.

|  |
| --- |
| CREATE OR REPLACE PROCEDURE tel(  v\_tel IN OUT VARCHAR2)  IS  BEGIN  v\_tel := SUBSTR(v\_tel,1,3) || '-' || SUBSTR(v\_tel,4);  DBMS\_OUTPUT.PUT\_LINE('전화번호 : ' || v\_tel);  END tel;  / |
| SQL> **SET SERVEROUTPUT ON**  SQL> **VAR g\_tel VARCHAR2(20)**  SQL> **BEGIN**  2 **:g\_tel := 1234567;**  3 **END;**  4 **/**  PL/SQL procedure successfully completed.  SQL> **EXECUTE tel(:g\_tel)**  전화번호 : 123-4567  PL/SQL procedure successfully completed.  SQL> **PRINT g\_tel**  G\_TEL  --------------------------------  123-4567 |

**1.4 FUNCTION 생성**

실행 환경에 반드시 하나의 값을 Return하기 위해 PL/SQL 함수를 사용한다. 함수 선언에서 Datatype이 있는 RETURN 절을 추가하고 PL/SQL 블록에 적어도 한 개의 이상의 RETURN 문을 포함한다. PARAMETER에서 사용하는 IN,OUT,IN OUT는 PROCEDURE에서 사용한 것과 동일하게 사용 가능하나 대부분 IN을 사용한다.

**1.4.1 Syntax**

|  |
| --- |
| **CREATE** [**OR REPLACE**] **FUNCTION** function\_name  [(argument1 [mode1] datatype [{:= | **DEFAULT**} expression]  [,argument2 [mode2] datatype [{:= | **DEFAULT**} expression], . . .])]  **RETURN** data\_type  {**IS** | **AS**}  **BEGIN**  pl/sql\_block;  **END**; |

OR REPLACE function\_name이 존재할 경우 FUNCTION의 내용을 지우고 다시 생성

function\_name Function의 이름은 표준 Oracle 명명법에 따른 함수이름

argument 매개변수의 이름

mode 3가지가 있다

IN : 입력 매개변수로 상용

OUT : 출력 매개변수로 사용

IN OUT : 입력, 출력 매개변수로 상용

data\_type 반환되는 값의 datatype

pl/sql\_block FUNCTION를 구성하는 코드를 구성하는 PL/SQL의 블록

**1.4.2 RETURN 문**

1. PL/SQL 블록에는 RETURN문이 있어야 한다.
2. 함수는 RETURN 절에 지정된 것과 동일한 datatype으로 RETURN 값을 설정해야 한다.
3. 다중 RETURN 문은 사용할 수 있지만 한 번의 호출로는 한 개의 RETURN 문만 실행된다.
4. 일반적으로 다중 RETURN 문은 IF 문에서 사용한다.

**1.4.3 FUNCTION 실행**

PL/SQL을 지원하는 어떤 툴이나 언어에서도 함수를 실행할 수 있고 PL/SQL내부에서 식의 일부로서 함수를 실행할 수 있다. SQL\*Plus에서 FUNCTION 호출은 Stored Function를 참조하는 PL/SQL 문을 실행하기 위해 EXECUTE 명령을 사용할 수 있다. EXECUTE는 명령 다음에 입력되는 Stored Function를 실행한다.

**가) Syntax**

|  |
| --- |
| output\_variable := function\_name[(argument1[,argument2, . . . . .])] |

**나) SQL\*Plus에서 함수 실행**

|  |
| --- |
| SQL> **EXECUTE :g\_deptno := ename\_deptno('ALLEN')**  PL/SQL procedure successfully completed. |

문제4) EMP 테이블에서 이름으로 부서 번호를 검색하는 함수를 작성하여라.

|  |
| --- |
| CREATE OR REPLACE FUNCTION ename\_deptno(  v\_ename IN emp.ename%TYPE)  RETURN NUMBER  IS  v\_deptno emp.deptno%TYPE;  BEGIN  SELECT deptno  INTO v\_deptno  FROM emp  WHERE ename = UPPER(v\_ename);  DBMS\_OUTPUT.PUT\_LINE('부서번호 : ' || TO\_CHAR(v\_deptno));  RETURN v\_deptno;  EXCEPTION  WHEN NO\_DATA\_FOUND THEN  DBMS\_OUTPUT.PUT\_LINE('입력한 MANAGER는 없습니다.');  WHEN TOO\_MANY\_ROWS THEN  DBMS\_OUTPUT.PUT\_LINE('자료가 2건 이상입니다.');  WHEN OTHERS THEN  DBMS\_OUTPUT.PUT\_LINE('기타 에러입니다.');  END;  / |
| SQL> **SET SERVEROUTPUT ON**  SQL> **VAR g\_deptno NUMBER**  SQL> **EXECUTE :g\_deptno := ename\_deptno('SCOTT')**  부서번호 : 10  PL/SQL procedure successfully completed.  SQL> **PRINT g\_deptno**  G\_DEPTNO  ---------  10 |

문제5) EMP 테이블에서 이름을 입력 받아 부서번호,부서명,급여를 검색하는 FUNCTION을 작성하여라. 단 부서번호를 RETURN에 사용하여라.

|  |
| --- |
| CREATE OR REPLACE FUNCTION emp\_disp(  v\_ename IN emp.ename%TYPE,  v\_dname OUT dept.dname%TYPE,  v\_sal OUT emp.sal%TYPE)  RETURN NUMBER  IS  v\_deptno emp.deptno%TYPE;  v\_dname\_temp dept.dname%TYPE;  v\_sal\_temp emp.sal%TYPE;  BEGIN  SELECT sal,deptno  INTO v\_sal\_temp,v\_deptno  FROM emp  WHERE ename = UPPER(v\_ename);  SELECT dname  INTO v\_dname\_temp  FROM dept  WHERE deptno = v\_deptno;  v\_dname := v\_dname\_temp;  v\_sal := v\_sal\_temp;  DBMS\_OUTPUT.PUT\_LINE('성 명 : ' || v\_ename);  DBMS\_OUTPUT.PUT\_LINE('부서번호 : ' || TO\_CHAR(v\_deptno));  DBMS\_OUTPUT.PUT\_LINE('부 서 명 : ' || v\_dname\_temp);  DBMS\_OUTPUT.PUT\_LINE('급 여 : ' || TO\_CHAR(v\_sal\_temp,'$999,999'));  RETURN v\_deptno;  EXCEPTION  WHEN NO\_DATA\_FOUND THEN  DBMS\_OUTPUT.PUT\_LINE('입력한 MANAGER는 없습니다.');  WHEN TOO\_MANY\_ROWS THEN  DBMS\_OUTPUT.PUT\_LINE('자료가 2건 이상입니다.');  WHEN OTHERS THEN  DBMS\_OUTPUT.PUT\_LINE('기타 에러입니다.');  END;  / |
| SQL> **SET SERVEROUTPUT ON**  SQL> **VAR g\_deptno NUMBER**  SQL> **VAR g\_dname VARCHAR2(20)**  SQL> **VAR g\_sal NUMBER**  SQL> **EXECUTE :g\_deptno := emp\_disp('scott',:g\_dname,:g\_sal)**  성 명 : scott  부서번호 : 10  부 서 명 : ACCOUNTING  급 여 : $3,000  PL/SQL procedure successfully completed.  SQL> **PRINT g\_deptno**  G\_DEPTNO  ---------  10  SQL> **PRINT g\_dname**  G\_DNAME  --------------------------------  ACCOUNTING  SQL> **PRINT g\_sal**  G\_SAL  ---------  3000 |

**1.5 함수와 프로시저 비교**

|  |  |
| --- | --- |
| 프로시저 | 함수 |
| PL/SQL 문으로서 실행 | 식의 일부로서 사용 |
| RETURN Datatype이 없음 | RETURN Datatype이 필수 |
| 값을 Return할 수 있음 | 값을 Return하는 것이 필수 |

♣ 참고

1. 프로시저는 parameter리스트를 가질 수 있지만 값 반환이 필수적 이지는 않다.
2. 함수는 다음과 같은 두 가지 점에서 프로시저와 다르다.
3. 식(expression)의 일부로서 함수를 사용한다.
4. 함수는 값을 return하는 것이 필수적이다.

**1.6 TRIGGER**

특정 테이블에 DML(INSERT,UPDATE,DELETE)문장이 수행되었을 때 데이터베이스에서 자동적으로 PL/SQL 블록을 수행 시키기 위해서 데이터베이스 TRIGGER를 사용한다. TRIGGER는 트리거링 이벤트가 일어날 때마다 암시적으로 실행된다. 트리거링 이벤트에는 데이터베이스 테이블에서 INSERT,UPDATE,DELETE 오퍼레이션이다.

**1.6.1 TRIGGER가 사용되는 경우**

1. 테이블 생성시 CONSTRAINT로 선언 제한이 불가능하고 복잡한 무결성 제한을 유지
2. DML문장을 사용한 사람,변경한 내용,시간 등을 기록함으로써 정보를 AUDIT하기
3. 테이블을 변경할 때 일어나야 할 동작을 다른 테이블 또는 다른 프로그램들에게 자동적으로 신호하기

**1.6.2 TRIGGER에 대한 제한**

1. TRIGGER는 트랜잭션 제어 문(COMMIT,ROLLBACK,SAVEPOINT)장을 사용하지 못한다.
2. TRIGGER 주요부에 의해 호출되는 프로시저나 함수는 트랜잭션 제어 문장을 사용하지 못한다.
3. TRIGGER 주요부는 LONG또는 LONG RAW변수를 선언할 수 없다.
4. TRIGGER 주요부가 액세스하게 될 테이블에 대한 제한이 있다.

**1.6.3 TRIGGER생성**

CREATE TRIGGER문장에 의해 TRIGGER를 생성할 수 있다.

**가) Syntax**

|  |
| --- |
| **CREATE** [**OR REPLACE**]  **TRIGGER** trigger\_name  {**BEFORE** | **AFTER**} triggering\_event [OF column1, . . .] **ON** table\_name  [**FOR EACH ROW** [**WHEN** trigger\_condition]  trigger\_body; |

trigger\_name TRIGGER의 식별자

BEFORE | AFTER DML문장이 실행되기 전에 TRIGGER를 실행할 것인지 실행된

후에 TRIGGER를 실행할 것인지를 정의

triggering\_event TRIGGER를 실행하는 DML(INSERT,UPDATE,DELETE)문을 기술한다.

OF column TRIGGER가 실행되는 테이블에서 COLUMN명을 기술한다.

table\_name TRIGGER가 실행되는 테이블 이름

FOR EACH ROW 이 옵션을 사용하면 행 레벨 트리거가 되어 triggering문장

에 의해 영향받은 행에 대해 각각 한번씩 실행하고 사용하지

않으면 문장 레벨 트리거가 되어 DML문장 당 한번만 실행된다.

**나) TRIGGER에서 OLD와 NEW**

행 레벨 TRIGGER에서만 사용할 수 있는 예약어로 트리거 내에서 현재 처리되고 있는 행을 액세스할 수 있다. 즉 두개의 의사 레코드를 통하여 이 작업을 수행할 수 있다. :OLD는 INSERT문에 의해 정의되지 않고 :NEW는 DELETE에 대해 정의되지 않는다. 그러나 UPDATE는 :OLD와 :NEW를 모두 정의한다. 아래의 표는 OLD와 NEW값을 정의한 표이다.

|  |  |  |
| --- | --- | --- |
| 문장 | :OLD | :NEW |
| INSERT | 모든 필드는 NULL로 정의 | 문장이 완전할 때 삽입된 새로운 값 |
| UPDATE | 갱신하기 전의 원래 값 | 문장이 완전할 때 갱신된 새로운 값 |
| DELETE | 행이 삭제되기 전의 원래 값 | 모든 필드는 NULL이다. |

**다) TRIGGER 술어 사용하기**

트리거 내에서 오퍼레이션이 무엇인지를 결정하기 위해 사용할 수 있는 3가지 BOOLEAN함수가 있다.

|  |  |
| --- | --- |
| 술 어 | 설 명 |
| INSERTING | 트리거링 문장이 INSERT이면 TRUE를 그렇지 않으면 FALSE를 RETURN |
| UPDATING | 트리거링 문장이 UPDATE이면 TRUE를 그렇지 않으면 FALSE를 RETURN |
| DELETING | 트리거링 문장이 DELETE이면 TRUE를 그렇지 않으면 FALSE를 RETURN |

**라) TRIGGER 삭제와 억제하기**

DROP TRIGGER명령어로 트리거를 삭제할 수 있고 TRIGGER를 잠시 disable할 수 있다.

|  |
| --- |
| **DROP TRIGGER** trigger\_name; |
| **ALTER TRIGGER**  trigger\_name {**DISABLE** | **ENABLE**}; |

**마) TRIGGER와 DATA DICTIONARY**

TRIGGER가 생성될 때 소스 코드는 데이터 사전 VIEW인 user\_triggers에 저장된다. 이 VIEW는 TRIGGER\_BODY, WHERE절, 트리거링 테이블, TRIGGER 타입을 포함 한다.

|  |
| --- |
| SQL> **SELECT trigger\_type,table\_name,triggering\_event**  2 **FROM user\_triggers;**  TRIGGER\_TYPE TABLE\_NAME TRIGGERING\_EVENT  ---------------- ------------------------------ --------------------------  AFTER STATEMENT EMP INSERT OR UPDATE OR DELETE  BEFORE STATEMENT EMP INSERT OR UPDATE OR DELETE  BEFORE EACH ROW EMP UPDATE |

문제6) EMP 테이블에서 급여를 수정시 현재의 값보다 적게 수정할 수 없으며 현재의 값보다 10% 이상 높게 수정할 수 없다. 이러한 조건을 만족하는 트리거를 작성하여라.

|  |
| --- |
| CREATE OR REPLACE TRIGGER emp\_sal\_chk  BEFORE UPDATE OF sal ON emp  FOR EACH ROW WHEN (NEW.sal < OLD.sal  OR NEW.sal > OLD.sal \* 1.1)  BEGIN  raise\_application\_error(-20502,  'May not decrease salary. Increase must be < 10%');  END;  / |
| SQL> **@emp\_sal**  Trigger created.  SQL> **update emp**  2 **set sal = 6000;**  set sal = 6000  \*  ERROR at line 2:  ORA-20502: May not decrease salary. Increase must be < 10%  ORA-06512: at "SCOTT.EMP\_SAL\_CHK", line 2  ORA-04088: error during execution of trigger 'SCOTT.EMP\_SAL\_CHK' |

문제7) EMP 테이블을 사용할 수 있는 시간은 월요일부터 금요일까지 09시부터 18시까지만 사용할 수 있도록 하는 트리거를 작성하여라.

|  |
| --- |
| CREATE OR REPLACE TRIGGER emp\_resource  BEFORE insert OR update OR delete ON emp  BEGIN  IF TO\_CHAR(SYSDATE,'DY') IN ('SAT','SUN')  OR TO\_NUMBER(TO\_CHAR(SYSDATE,'HH24'))  NOT BETWEEN 9 AND 18 THEN  raise\_application\_error(-20502,  '작업할 수 없는 시간 입니다.');  END IF;  END;  / |
| SQL> **@tr1**  Trigger created.  SQL> **select to\_char(sysdate,'hh24') from dual;**  TO\_CHAR(SYSDATE,'HH24')  -----------------------  19  SQL> **update emp**  2 **set deptno = 10;**  update emp  \*  ERROR at line 1:  ORA-20502: 작업할 수 없는 시간 입니다.  ORA-06512: at "SCOTT.EMP\_RESOURCE", line 5  ORA-04088: error during execution of trigger 'SCOTT.EMP\_RESOURCE' |

문제8) EMP 테이블에 INSERT,UPDATE,DELETE문장이 하루에 몇 건 발생하는지 조사하려고 한다. 조사 내용은 EMP\_AUDIT에 사용자 이름, 작업 구분,작업 시간을 저장하는 트리거를 작성하여라.

|  |
| --- |
| CREATE SEQUENCE emp\_audit\_tr  INCREMENT BY 1  START WITH 1  MAXVALUE 999999  MINVALUE 1  NOCYCLE  NOCACHE;  CREATE TABLE emp\_audit(  e\_id NUMBER(6)  CONSTRAINT emp\_audit\_pk PRIMARY KEY,  e\_name VARCHAR2(30),  e\_gubun VARCHAR2(10),  e\_date DATE);  CREATE OR REPLACE TRIGGER emp\_audit\_tr  AFTER insert OR update OR delete ON emp  BEGIN  IF INSERTING THEN  INSERT INTO emp\_audit  VALUES(emp\_audit\_tr.NEXTVAL,USER,'inserting',SYSDATE);  ELSIF UPDATING THEN  INSERT INTO emp\_audit  VALUES(emp\_audit\_tr.NEXTVAL,USER,'updating',SYSDATE);  ELSIF DELETING THEN  INSERT INTO emp\_audit  VALUES(emp\_audit\_tr.NEXTVAL,USER,'deleting',SYSDATE);  END IF;  END;  / |
| SQL> **UPDATE emp**  2 **SET deptno = 20**  3 **WHERE deptno = 10;**  2 row updated.  SQL> **SELECT \* FROM emp\_audit;**  E\_ID E\_NAME E\_GUBUN E\_DATE  --------- ------------------------------ ---------- ------------------  1 SCOTT updating 23-FEB-99 |

문제9) EMP 테이블에 INSERT,UPDATE,DELETE문장이 하루에 몇 건의 ROW가 발생되는지 조사하려고 한다. 조사 내용은 EMP\_AUDIT\_ROW에 사용자 이름, 작업 구분,작업 시간,사원번호,이전의 급여, 갱신된 급여을 저장하는 트리거를 작성하여라.

|  |
| --- |
| DROP SEQUENCE emp\_row\_seq;  CREATE SEQUENCE emp\_row\_seq  INCREMENT BY 1  START WITH 1  MAXVALUE 999999  MINVALUE 1  NOCYCLE  NOCACHE;  DROP TABLE emp\_row\_tab;  CREATE TABLE emp\_row\_tab(  e\_id NUMBER(6)  CONSTRAINT emp\_row\_pk PRIMARY KEY,  e\_name VARCHAR2(30),  e\_gubun VARCHAR2(10),  e\_date DATE);  CREATE OR REPLACE TRIGGER emp\_row\_aud  AFTER insert OR update OR delete ON emp  FOR EACH ROW  BEGIN  IF INSERTING THEN  INSERT INTO emp\_row\_tab  VALUES(emp\_row\_seq.NEXTVAL,USER,'inserting',SYSDATE);  ELSIF UPDATING THEN  INSERT INTO emp\_row\_tab  VALUES(emp\_row\_seq.NEXTVAL,USER,'updating',SYSDATE);  ELSIF DELETING THEN  INSERT INTO emp\_row\_tab  VALUES(emp\_row\_seq.NEXTVAL,USER,'deleting',SYSDATE);  END IF;  END;  / |
| SQL> **UPDATE emp**  2 **SET deptno = 40**  3 **WHERE deptno = 10;**  3 rows updated.  SQL> **SELECT \* FROM emp\_row\_tab;**  E\_ID E\_NAME E\_GUBUN E\_DATE  --------- ------------------------------ ---------- ------------------  1 SCOTT updating 23-FEB-99  2 SCOTT updating 23-FEB-99  3 SCOTT updating 23-FEB-99 |

**◈ 연 습 문 제 ◈**

1. EMP TABLE에 이름,사번,급여,부서번호를 전달받아 등록하는 PROCEDURE를 작성하여라.

2. 사원번호를 입력받아 급여를 수정하는 PROCEDURE를 작성하여라.

3. 최고의 월급을 받는 사원의 사번을 구하여 출력하는 PROCEDURE를 작성하여라.

4. 이름을 입력받아 부서명을 구하여 출력하는 PROCEDURE를 작성하여라.

5. 부서번호를 입력받아 그 부서의 최고 급여를 구하여 출력하는 FUNCTION을 작성하여라.

6. PROCEDURE와 FUNCTION의 차이점을 설명하여라.

7. TRIGGER란 ?

8. EMP\_SAL\_TOT(부서번호,급여의 합) TABLE을 생성하여라.

|  |
| --- |
| SQL> CREATE TABLE emp\_sal\_tot as  2 SELECT deptno,SUM(sal) sal\_tot  3 FROM emp  4 GROUP BY deptno;  Table created. |

9. EMP\_SAL\_TOT TABLE의 내용은 EMP TABLE을 변경하면 EMP\_SAL\_TOT TABLE의 내용도 자동적으로 변경되어야 한다. 이를 반영한 TRIGGER를 작성하여라.