PL/SQL(Procedual Language/SQL)

==> 자바의 프로그램처럼
오라클에서 사용하는 질의 명령으로 여러개 모아서
순차적으로 실행시키는 일종의 프로그램을 말한다.

장점]

1 프로그램화 해서 다수의 SQL을 수행하게 함으로
수행 속도를 향상 시킬 수 있다.

2 모듈화(부품화)가 가능하여 필요한 기능만 골라서
여러개의 프로그램을 연결해서 사용할 수 있다.
==> 질의 명령을 제작함에 있어서 편리하다.

3 동적 변수를 사용할 수 있기 때문에
변경된 데이터를 이용해서 질의 명령을 수행하도록 할 수 있다.
==> 동일한 질의 명령은 한번만 만든 후
데이터만 바꾸어서 실행할 수 있다.

4. 예외처리가 가능하여 문제가 발생할 경우 이것을
수정 처리할 수 있다.

종류]

1. 익명 프로시저
==> 질의 명령을 모아놓은 후 프로그램이 완료되면
즉각 실행할 수 있는 PL/SQL을 말한다.

2. 일반 프로시저
==> 프로시저에 이름을 부여한 것을 말하며
필요할 때 선택에 의해서 실행 할 수 있는 PL/SQL을 말한다.

3. Function
==> 우리가 알고있는 함수의 개념으로
사용자가 만든 함수라고 생각하면되며
나중에 질의 명령을 처리할 때 그 질의 명령에
포함시켜서 사용하는 PL/SQL 을 말한다.

4. 기타 프로시저
트리거, 스케줄러, ... 의 특별한 기능을 가진 PL/SQL을 말한다.

전체적인 PL/SQL의 구조

```
DECLARE
        선언부(프로램에서 필요한 변수나,... 을 선언하는 부분)
BEGIN
        실행부(실제 필요한 질의명령을 만들고 이것을 제어하는 부분)
[EXCEPTION
        예외처리 부분]
END;
/
```

1. 무명 프로시저

```
DECLARE
        x          NUMBER;          --          변수 선언
BEGIN
        x := 1000;                  --          대입명령
        DBMS_OUTPUT.PUT_LINE('결과 = ');
        DBMS_OUTPUT.PUT_LINE(x);    --          System.out.println()에 해당
END;
```

여기까지하면 무명 프로시저가 만들어지지만 실행되지는 않는다.
실행을 하기 위해서는 **반드시 / 를 기록하고 엔터키**를 쳐야
실행이 된다.

참고 ==> DBMS_OUTPUT.PUT_LINE은 출력명령이다.
하지만 항상 출력 가능한것이 아니고
SQLPLUS가 출력 가능한 상태가 되어야 결과가 나오게 된다.

```
sql>set serveroutput on;
```

무명 프로시저 사용방법

1. 프로시저를 만들어서 저장해 놓는다.
   이때 확장자는 중요하지 않는다.
   하지만 확장자를 .sql 이라고 부여한다.

2. SQLPLUS 에서 실행을 한다.
   1) @D:/SAMPLE.SQL ==> 실행만 시킨다.
   2) RUN D:/SAMPLE.SQL ==> 소스를 보여주고 실행시킨다.
   3) GET D:/SAMPLE.SQL ==> 소스만 보여준다.

2. 함수 만들기

==> 함수는 오라클 내부에 함수를 저장한 후
다른 질의 명령을 사용할 때 부가적으로 사용하는 것

```
CREATE OR REPLACE FUNCTION test_func
RETURN NUMBER AS
        x NUMBER;
BEGIN
        x := 999;
        RETURN x;
END;
/
```

실행]

```
SELECT ename, test_func() FROM emp;
```

3. 저장(일반) 프로시저 만들기

테이블 준비]

```
CREATE TABLE Memb
(
        name    varchar2(100),
        age     number(5)
);
```

데이터 추가]

```
INSERT INTO Memb VALUES('홍길동', 20);
INSERT INTO Memb VALUES('박아지', 30);
```

문제]

나이가 20인 사람의 나이를 100살로 수정하는 프로시저를 만드세요.

```
CREATE OR REPLACE PROCEDURE change_age
        (i_age IN INTEGER)
AS
BEGIN
        UPDATE
                MEMBER
        SET
                age = 100
        WHERE
                age = i_age;
END;
/
```

프로시저를 실행하는 방법

sql>execute 프로시저 이름(파라메터);

```
sql>execute change_age(20);
```

확인]

```
sql>select * from member;
```

참고]

위의 3가지는 혼합해서 사용할 수 있다.

1. 무명 안에 함수를 포함해서 만드는 방법

test01.sql

```
DECLARE
        temp NUMBER;                    -- 함수 프로시저

        FUNCTION test                   -- 함수 프로시저 시작
                (b_ex IN boolean,
                t_num in number,
                f_num in number)
        return number is
        begin
                if b_ex then
                        return t_num;
                elsif not b_ex then
                        return f_num;
                else
                        return null;
                end if;
        end;            -- 함수 프로시저 종료
begin           -- 무명 프로시저 코딩 부분
        DBMS_OUTPUT.PUT_LINE(test(2>1, 1, 0));          -- 무명쪽에서 함수를 사용
        DBMS_OUTPUT.PUT_LINE(test(2>3, 1, 0));          -- 무명쪽에서 함수를 사용
        temp := test(null, 1, 0);  -- 무명쪽에서 함수를 사용
        if temp is null then
                DBMS_OUTPUT.PUT_LINE('NULL');
        ELSE
                DBMS_OUTPUT.PUT_LINE(temp);
        END IF;
END;
/
```

2. 무명 안에 일반 프로시저를 포함해서 만드는 방법

test02.sql

```
DECLARE                          -- 무명시작
        f_num number;
        s_num number;

        PROCEDURE swapping                    -- 일반프로시저
                (num_1 IN OUT NUMBER,
                num_2 IN OUT NUMBER)
        IS
                temp_num NUMBER;
        BEGIN
                temp_num := num_1;
                num_1 := num_2;
                num_2 := temp_num;                    -- 일반프로시저 종료
        END;
BEGIN
        f_num := 10;
        s_num := 20;
        DBMS_OUTPUT.PUT_LINE('1번숫자 ' || TO_CHAR(f_num));
        DBMS_OUTPUT.PUT_LINE('2번숫자 ' || TO_CHAR(s_num));
        swapping(f_num, s_num);
                                                        -- 프로시저 호출
        DBMS_OUTPUT.PUT_LINE('1번숫자 ' || TO_CHAR(f_num));
        DBMS_OUTPUT.PUT_LINE('2번숫자 ' || TO_CHAR(s_num));
END;
/
```

3. 무명 안에 무명 프로시저를 포함해서 만드는 방법

test03.sql

```
DECLARE                                 -- 무명 시작
        error_flag BOOLEAN := false;
BEGIN
        DBMS_OUTPUT.PUT_LINE('100~1000 숫자 처리');
        DECLARE                                                 --  다시 무명 시작
                h_num number(1, -2);
        BEGIN
                h_num := 100;
                LOOP
                        DBMS_OUTPUT.PUT_LINE(h_num);
                        h_num := h_num + 100;
                        if h_num > 1000 then
                                exit;
                        end if;
                end loop;
        exception
                WHEN OTHERS THEN
                        error_flag := true;
        END;                                                    -- 안쪽 무명 종료
        IF error_flag THEN
                DBMS_OUTPUT.PUT_LINE('숫자를 계산할 수 없어요');
        ELSE
                DBMS_OUTPUT.PUT_LINE('계산이 끝났어요');
        END IF;
END;                                                            -- 바깥 무명 종료
/
```

저장(일반) 프로시저 만드는 방법

장점]

한번만 만들면 필요할 때 언제든지 그 이름을 이용해서 사용할 수 있다.

형식]

```
CREATE OR REPLACE PROCEDURE 적당한이름
        (파라메터 변수 선언)
IS
        내부 변수 선언
BEGIN
        처리할 내용

[EXCEPTION
                예외처리 방식]
END;
/
```

파라메터 변수선언

형식]

```
변수이름 IN[OUT] 변수형태;
```

의미]

이 프로시저를 실행할 때 전달되는 값이나(IN)
이 프로시저의 결과를 알려줄때 사용할 변수를(OUT) 선언한다.

예]

```
execute test_proc(100);
```

==> 이처럼 프로시저를 실행한 순간
데이터를 전달 할 수 있으며 이 데이터를
기억할 변수를 파라메터 변수라고 한다.

내부 변수 선언

==> 프로시저를 실행할 때 필요한 변수를 선언하는 부분이다.

형식]

```
변수이름 변수타입;
예 ] age        NUMBER;

변수이름 변수타입 := 값;
예 ] name       VARCHAR2(100) := '홍길동';

변수이름 CONSTNT 변수타입 := 값;
==>  자바로 말하면 Final 변수에 해당하는 것으로
     데이터를 바꿀 수 없는 변수

변수이름 변수타입 NOT NULL := 값;
==>  일반변수는 값을 기억하지 않으면 NULL값을 가지게 되는데
     이 변수는 NULL 을 허용하지 않는 변수로
     반드시 초기화를 해야하는 변수임을 밝히는 것이다.
     (초기화를 강제로 하면 NOT NULL을 쓰지 않아도 되지만
     가독성을 위한 것이다.)
```

참고]

변수타입은 오라클의 변수 타입도 사용할 수 있고 ANSI 타입도 사용할 수 있다.

예]

BOOLEAN, INTEGER, ...

예제 1]

emp 테이블에 있는 데이터 중에서 급여가 원하는 급여
미만 사원은 급여를 10% 인상처리 하세요.

테이블 준비]

```
CREATE TABLE Test01
AS
        SELECT * FROM emp;
```

프로시저 작성

```
CREATE OR REPLACE PROCEDURE update_sal
        (v_sal IN NUMBER)
        --          이 프로시저를 실행할 때 원하는 급여을 입력할 예정이고
        --          그 급여를 받아서 기억할 파라메터 변수를 하나 준비한다.
IS
        --          일반 변수가 들어갈 장소인데 없으므로 생략한다.
        --          하지만 IS 예약어는 반드시 써야한다.
BEGIN
        UPDATE
                Test01
        SET
                sal = sal + sal * 0.1
        WHERE
                sal <= v_sal;
        --          트랜젝션이 일어나지 않으므로
        --          필요하다면 트랜젝션을 강제로 일으켜야 한다.
        COMMIT;
END;
/
```

실행

```
execute update_sal(1000);
execute update_sal(2000);
```

예제 2]

사원번호를 알려주면 그 사원의 이름과 직책, 급여를 출력하는 프로시저를 작성하세요.

프로시저 작성

```
CREATE OR REPLACE PROCEDURE select_empno
        (v_empno IN NUMBER)
IS
        --      출력할 내용을 기억할 변수를 준비한다.
        d_ename         VARCHAR2(50);
        d_job           VARCHAR2(50);
        d_sal           NUMBER;
BEGIN
        --      결과를 출력하기 위해서는 SQLPLUS가 출력 가능 상태가 되어야 한
다.
        --      출력가능상태로 만드는 명령이 필요하다.
        --      즉, 이 프로시저에서 출력 가능하도록 하기 위한 조치
        DBMS_OUTPUT.ENABLE;

        SELECT ename, job, sal
        INTO            d_ename, d_job, d_sal
        --      SELECT된 결과를 출력하기 위해서 변수에 옮겨야 한다.
        --      SELECT된 결과를 변수에 옮기기 위한 예약어가
        --      INTO이다.
        --      규칙
        --              SELECT 된 결과의 개수와 변수의 개수는 동일해야하며
        --              변수의 형태와 실제 데이터의 형태가 반드시 일치해야 한다.
        FROM            Test01
        WHERE
                        empno = v_empno;

        --      이제 변수에 담긴 결과를 출력하자.
        DBMS_OUTPUT.PUT_LINE('사원이름 = ' || d_ename);
        DBMS_OUTPUT.PUT_LINE('사원직책 = ' || d_job);
        DBMS_OUTPUT.PUT_LINE('사원급여 = ' || d_sal);
END;
/
```

실행

```
execute select_empno(7369);
execute select_empno(7839);
```

%TYPE에 의한 변수 선언

앞에서 변수를 만들때 변수의 형태를 지정했었다.
문제는 이 변수 중에는 질의의 결과와 연관된 변수가 존재한다.
이때 크기가 다르면 문제가 생길 수 있다.
또한 이 문제는 데이터베이스가 변경되어도 문제가 생길 수 있다.
==> 그러면 프로시저를 수정해야 하는 문제가 생길 수 있다.

이런 경우를 대비해서 만드는 것이 **%TYPE 에 의한 변수 선언**이다.
즉, 이것은 이미 만들어진 타입을 그대로 복사해서 같은
타입의 변수로 만드는 행위를 말한다.

1. 이미 만들어진 변수와 동일한 타입으로 만드는 방법
   예]

```
d_name      VARCHAR2(20);
d_job       d_name%TYPE;
```

   * d_name의 변수 형태와 동일한 형태의 변수가 만들어진다.

2. 데이터베이스의 타입과 동일한 타입으로 만드는 방법
   예]

```
d_name      emp.ename%TYPE;
```

   * emp 테이블에 ename 필드와 동일한 형태로
     변수를 만들어서 사용하게 된다.

예제 3]

사원번호를 알려주면 그 사원의 이름, 직책, 상사이름을
출력하는 프로시저를 만드세요.

프로시저 작성

```
CREATE OR REPLACE PROCEDURE proc01
        (v_empno IN emp.empno%TYPE)
IS
        d_name emp.ename%TYPE;
        d_job               emp.job%TYPE;
        d_mgr               d_name%TYPE;
BEGIN
        DBMS_OUTPUT.ENABLE;

        SELECT
                e1.ename as ename, e1.job as job, e2.ename as mgr
        INTO
                d_name, d_job, d_mgr
        FROM
                emp e1, emp e2
        WHERE
                                e1.mgr = e2.empno
                AND     e1.empno = v_empno;

        DBMS_OUTPUT.PUT_LINE('사원이름 = ' || d_name);
        DBMS_OUTPUT.PUT_LINE('사원직책 = ' || d_job);
        DBMS_OUTPUT.PUT_LINE('상사이름 = ' || d_mgr);
END;
/
```

실행

```
execute proc02(7369);
```

%ROWTYPE을 이용한 변수 선언

==> 앞에서 설명한 %TYPE은 필드 단위로 데이터형태를
복사해서 사용하는 방법이다.

%ROWTYPE은 테이블 전체의 데이터 형태를 복사해서 사용하는 방법이다.

형식]

```
변수이름                    테이블이름%ROWTYPE;
```

* 이 변수는 마치 클래스처럼 내부에 멤버 변수를 가지게 된다.
멤버 변수의 이름은

변수이름.필드이름

이 된다.

예제 4]

사원 이름을 입력하면
그 사원의 이름, 직책, 급여, 입사일자를 출력하는
프로시저를 작성하세요.

프로시저 작성

```
CREATE OR REPLACE PROCEDURE proc02
        (v_ename IN emp.ename%TYPE)
IS
        d_emp           emp%ROWTYPE;
        --      이 안에는 묵시적으로 멤버변수를 가지게 되며
        --              d_emp.ename
        --              d_emp.job
        --              d_emp.sal
        --              d_emp.hiredate 를 가지게 된다.
BEGIN
        DBMS_OUTPUT.ENABLE;

        SELECT
                ename, job, sal, hiredate
        INTO
                d_emp.ename, d_emp.job, d_emp.sal, d_emp.hiredate
        FROM
                emp
        WHERE
                        ename = v_ename;

        DBMS_OUTPUT.PUT_LINE('사원이름 = ' || d_emp.ename);
        DBMS_OUTPUT.PUT_LINE('사원직책 = ' || d_emp.job);
        DBMS_OUTPUT.PUT_LINE('사원급여 = ' || d_emp.sal);
        DBMS_OUTPUT.PUT_LINE('입사일자 = ' || d_emp.hiredate);
END;
/
```

실행

```
execute proc02('SMITH');
```

문제 1

Test01테이블에서 이름이 특정 글자수인 사람의 급여를 10%
인상하는 프로시저를 만든 후 실행해 보세요.

문제 2

사원번호를 입력하면
해당사원의 이름, 직책, 부서이름, 부서위치가 출력되는
프로시저를 만들어 보세요.
==> 프로시저 안에 조인을 마음대로 써도 상관이 없다.

문제 3

사원이름을 입력하면
그 사원의 이름, 직책, 급여 및 급여등급을 출력하는
프로시저를 작성하세요.

execute proc02('SMITH');

문제 4

문제2] 를 %ROWTYPE으로 처리하세요

힌트]

사용할 테이블이 2개 이므로 변수도 2개 나와야 한다.

테이블 타입의 변수

==> PL/SQL에서 배열을 표현하는 방법

테이블타입을 정의하는 방법

```
TYPE 테이블이름 IS TABLE OF  데이터타입1
INDEX BY 데이터타입2;
```

데이터타입1 : 변수에 기억될 데이터의 형태
데이터타입2 : 배열의 인덱스로 사용할 데이터의 형태
(99.9%가 BINARY_INTEGER : 0, 1, 2, ...)

형식]

```
변수이름        테이블이름;
```

* 정의된 테이블이름의 형태로 변수를 만드세요.

참고]

테이블 타입이란 변수는 존재하지 않는다.
그래서 먼저 테이블 타입을 정의하고
정의된 테이블 타입을 이용해서 변수를 선언해야 한다.

for 명령

형식 1]

```
FOR 변수이름 IN (질의명령) LOOP
        처리내용;
        ....
END LOOP;
```

* 질의 명령의 결과를 변수에 한줄씩 기억한 후 원하는 내용을 처리하도록 한다.

형식 2]

```
FOR 변수이름 IN 시작값 .. 종료값 LOOP
        처리내용;
         ...
END LOOP;
```

* 시작값에서 종료값까지 1씩 증가하면서 처리내용을 반복시킨다.

예제]

부서번호를 알려주면 그 부서에 소속된 사원의 이름, 직책, 급여를 출력하도록 프로시저를 만드세요.

프로시저 작성

```
CREATE OR REPLACE PROCEDURE Proc01
        (v_deptno IN emp.deptno%TYPE)
IS
        /*      결과가 여러줄 나올 예정이므로 이결과를 한꺼번에
                기억하기 위해서는 배열로 선언해야 한다
                배열은 3개가 있어야 하고
                그러므로 테이블도 3개가 존재해야 한다.

                테이블 3개를 정의한다.
        */
        TYPE ename_table IS TABLE OF emp.ename%TYPE
        INDEX BY BINARY_INTEGER;

        TYPE job_table IS TABLE OF emp.job%TYPE
        INDEX BY BINARY_INTEGER;

        TYPE sal_table IS TABLE OF emp.sal%TYPE
        INDEX BY BINARY_INTEGER;
        /*
                아직 변수가 만들어진것은 아니고
                이런이런 배열을 사용할 예정임을 정의한것에 불과한 것이다.
                이 정의된 테이블을 이용해서 실제 변수를 만들어야 한다.
        */
        d_ename         ename_table;    --      자동 배열이 된다.
        d_job           job_table;
        d_sal           sal_table;

        --      배열에 사용할 인덱스 변수를 하나 만들자
        i       BINARY_INTEGER := 0;

BEGIN
        DBMS_OUTPUT.ENABLE;
```

```
        FOR     temp IN (
                SELECT
                        ename, job, sal
                FROM
                        emp
                WHERE
                        deptno = v_deptno
        ) LOOP
                --      FOR  명령에서 사용할 변수는 미리 만들지 않아도 된다.
                --      이 변수는 자동 %ROWTYPE  변수가 된다.
                --      ROWTYPE 변수는 묵시적으로 멤버를 가지는 변수이다.
                --              temp.ename
                --              temp.job
                --              temp.sal
                --      를 가지는데
                --      이때 멤버는 IN의 질의 결과에 따라 나온 결과가
                --      멤버가 결정된다.
                --      첫번째 반복할때는
                --              temp.ename = 'CLACK'
                --              temp.job = 'MANAGER'
                --              temp.sal = 2450
                --       이 기억되어서 처리된다.

                i := i + 1;
                --      오라클은 배열이 1부터 시작한다.

                --      이제 이 결과를 준비된 배열에 기억해 보자
                d_ename(i) := temp.ename;
                d_job(i) := temp.job;
                d_sal(i) := temp.sal;
        END LOOP;

        --      이제 배열변수에 있는 내용을 출력해보자
        FOR cnt IN 1 .. i LOOP
                DBMS_OUTPUT.PUT_LINE('이름 = ' ||  d_ename(cnt));
                DBMS_OUTPUT.PUT_LINE('직책 = ' ||  d_job(cnt));
                DBMS_OUTPUT.PUT_LINE('급여 = ' ||  d_sal(cnt));
        END LOOP;
END;
/
```

실행

```
set serveroutput on;
execute Proc01(20);
```

문제 1]

직책을 입력하면
해당 직책을 가진 사원의 이름, 급여, 부서이름을 출력하는
프로시저를 작성하세요

레코드 타입

==> 강제로 멤버를 가지는 변수를 만드는 방법

%ROWTYPE은 하나의 테이블을 이용해서 멤버 변수를 자동으로 만든 방법이지만
레코드 타입은 사용자가 지정한 멤버 변수를 만들 수 있다는 차이가 있다.

만드는 방법

1. 레코드 타입을 정의

```
TYPE 레코드이름 IS RECODE
(
        멤버변수이름1              데이터타입,
        멤버변수이름2              데이터타입,
        .....
);
```

2. 정의된 레코드 타입을 이용해서 변수를 선언

```
변수이름                레코드 이름;
```

예제]

이름을 알려주면 그 사원의 이름, 직책, 급여, 급여 등급을 출력하는 프로시저를 만들어보자.

프로시저 작성

```
CREATE OR REPLACE PROCEDURE Proc02
        (v_ename                IN      emp.ename%TYPE)
IS
        --      레코드 타입을 정의한다.
        --      이 안에는 4가지 정보를 기억할 수 있는 변수를 멤버로
        --      준비하도록 한다.
        TYPE    emp_record IS RECORD
        (
                m_ename                 emp.ename%TYPE,
                m_job                   emp.job%TYPE,
                m_sal                   emp.sal%TYPE,
                m_grade                 salgrade.grade%TYPE
        );
        --      변수 선언을 한다.
        d_emp           emp_record;
        --      이 안에는       d_emp.m_ename
        --                      d_emp.m_job
        --                      d_emp.m_sal
        --                      d_emp.m_grade           가 존재하게 된다.
BEGIN
        DBMS_OUTPUT.ENABLE;

        SELECT ename, job, sal, grade
        INTO            d_emp.m_ename, d_emp.m_job, d_emp.m_sal,
                        d_emp.m_grade
        FROM    emp, salgrade
        WHERE
                emp.sal BETWEEN salgrade.losal AND salgrade.hisal
                AND     emp.ename = v_ename;

        DBMS_OUTPUT.PUT_LINE('이름 = ' || d_emp.m_ename);
        DBMS_OUTPUT.PUT_LINE('직책 = ' || d_emp.m_job);
        DBMS_OUTPUT.PUT_LINE('급여 = ' || d_emp.m_sal);
        DBMS_OUTPUT.PUT_LINE('등급 = ' || d_emp.m_grade);
END;
/
```

실행

```
execute Proc02('SMITH');
```

문제]

사원번호를 입력하면
그 사원의 이름, 직책, 상사이름을 알려주는 프로시저를 만드세요
단, 정보를 레코드 타입으로 처리하세요.

레코드 타입의 배열

==> 앞에서 배운 두가지 오라클 PL/SQL 타입을 혼용해서 사용하는 것을 말하며

1. 레코드 타입을 선언한 후 (멤버 변수를 가진 변수)
2. 이것을 이용해서 다시 테이블 타입을 만들어서 사용하는것을 말한다.
   (배열 변수로 만드는 것)

예제]

직책을 입력하면
그 직책을 가진 사원의 이름, 급여, 입사일을 출력하는 프로시저를 만드세요
(다행히도 사용되는 변수가 모두 한 테이블에 있는 내용이다.)

프로시저 작성

```
CREATE OR REPLACE PROCEDURE Proc03
        (v_job IN emp.job%TYPE)
IS
        --      이미 emp 테이블의 묵시적인 레코드 타입 변수가
        --      %ROWTYPE으로 만들어진 상태이므로
        --      테이블 타입만 선언하자.

        TYPE emp_table IS TABLE OF emp%ROWTYPE
        INDEX BY BINARY_INTEGER;

        d_emp           emp_table;
        /*
                d_emp(1).ename
                d_emp(2).job            의 모양으로
                배열과 멤버를 동시에 사용할 수 있다.
        */
        i       BINARY_INTEGER := 0;
BEGIN
        DBMS_OUTPUT.ENABLE;
```

```
        FOR temp IN (SELECT ename, sal, hiredate
                                        FROM   emp
                                        WHERE          job = v_job)
LOOP
                i := i + 1;
                d_emp(i).ename := temp.ename;
                d_emp(i).sal := temp.sal;
                d_emp(i).hiredate := temp.hiredate;
        END LOOP;

        FOR     c IN 1 .. i LOOP
                DBMS_OUTPUT.PUT_LINE('이름 = ' || d_emp(c).ename);
                DBMS_OUTPUT.PUT_LINE('급여 = ' || d_emp(c).sal);
                DBMS_OUTPUT.PUT_LINE('입사일 = ' ||
d_emp(c).hiredate);
        END LOOP;
END;
/
```

실행

```
execute Proc03('MANAGER');
```

예제]

부서번호를 입력하면 해당부서 직원의 이름, 직책, 부서이름, 위치를 출력하는 프로시저를 만드세요.

프로시저 작성

```
CREATE OR REPLACE PROCEDURE Proc04
	(v_deptno IN emp.deptno%TYPE)
IS
	--	먼저 레코드 타입을 정의한다.
	TYPE emp_record IS RECORD
	(
		m_name			emp.ename%TYPE,
		m_job			emp.job%TYPE,
		m_dname			dept.dname%TYPE,
		m_loc			dept.loc%TYPE
	);
	--	위에서 만드는 레코드 타입을 이용해서 테이블 타입을 정의한다.
	TYPE emp_table IS TABLE OF emp_record
	INDEX BY BINARY_INTEGER;

	--	사용할 변수를 만든다.
	d_emp		emp_table;
	/*
		d_emp(1).m_name
		d_emp(2).m_dname	...
	*/
	i BINARY_INTEGER := 0;
BEGIN
	DBMS_OUTPUT.ENABLE;

	FOR temp IN (
		SELECT
			e.ename, e.job, d.dname, d.loc
		FROM
			emp e, dept d
		WHERE
			e.deptno = d.deptno
			AND	e.deptno = v_deptno
	) LOOP
```

```
                --          한줄씩 결과를 배열에 옮긴다.
                i := i + 1;
                d_emp(i).m_name := temp.ename;
                d_emp(i).m_job := temp.job;
                d_emp(i).m_dname := temp.dname;
                d_emp(i).m_loc := temp.loc;
        END LOOP;
END;
/
```

실행

```
execute Proc04(10);
```

문제]

입사년을 알려주면
해당 입사년도에 입사한 사원의 이름, 직책, 부서, 급여, 급여등급
을 출력하는 프로시저를 작성하세요.

execute Proc???('81');

while 명령

형식 1]

```
WHILE 조건식 LOOP
        처리내용
END LOOP;
```

형식 2]

```
WHILE 조건식1 LOOP
        처리내용
        EXIT WHEN 조건식2;
END LOOP;
```

* 조건식1이 참이면 반복하지만 조건식2가 참이면 반복을 종료한다.
  즉, break 명령의 역활을
  EXIT WHEN
  이 할 수 있다.

do~while 명령

형식]

```
LOOP
        처리내용
        EXIT WHEN 조건식;
END LOOP;
```

if 명령

형식 1]

```
IF      조건식    THEN
        처리내용
END IF;
```

형식 2]

```
IF      조건식              THEN
        처리내용1
ELSE
         처리내용2
END IF;
```

형식 3]

```
IF      조건식     THEN
        처리내용1
ELSIF 조건식 THEN
        처리내용2
        ....
ELSE
        처리내용n
END IF;
```

커서(Cursor)

==> 실제로 실행되는 질의 명령을 기억하는 변수

자주 사용하는 질의 명령을 한번만 만든 후
이것(질의명령 자체)를 변수에 기억해서 마치 변수를 사용 하듯이 질의를 실행하는 것

암시적 커서

==> 이미 오라클에서 제공하는 커서를 말하며
우리가 지금까지 사용했듯이 직접 만들어서
실행된 질의 명령 자체를 의미한다.

참고]

커서에는 내부 변수(멤버 변수)가 존재한다.

SQL%ROWCOUNT

==> 실행 결과 나타난 레코드의 개수를 기억한 멤버 변수

SQL%FOUND

==> 실행 결과가 존재함을 나타내는 멤버 변수

SQL%NOTFOUND

==> 실행 결과가 존재하지 않음을 나타내는 멤버 변수

SQL%ISOPEN

==> 커서가 만들어졌는지를 알아내는 멤버 변수

예제]

사원번호를 입력하면
그 사원의 이름을 알려주는 프로시저를 만드세요
단, 사원번호가 잘못 입력되면 사원이 없음을 알리는 메시지를 출력하라.

프로시저 작성

```
CREATE OR REPLACE PROCEDURE Proc01
        (v_empno IN emp.empno%TYPE)
IS
        d_ename         emp.ename%TYPE;
BEGIN
        DBMS_OUTPUT.ENABLE;

        SELECT ename
        INTO            d_ename
        FROM            emp
        WHERE           empno = v_empno;
        --      이 문장이 실해되는 순간 암시적 커서가 생겨난다.
        IF      SQL%FOUND THEN
                DBMS_OUTPUT.PUT_LINE('사원이 존재합니다.');
                DBMS_OUTPUT.PUT_LINE('사원이름 : ' || d_ename);
        END IF;
        EXCEPTION
                WHEN NO_DATA_FOUND THEN
                DBMS_OUTPUT.PUT_LINE('사원이 존재하지 않습니다.');
 END;
 /
```

실행

```
execute Proc01(7369);
execute Proc01(7368);
```

예제]

부서번호를 알려주면 해당 부서 직원의 급여를 10% 인상하도록 하세요.
그리고 인상된 사람이 모두 몇사람인지를 출력하세요.

테이블 준비

```
DROP TABLE Test01;
CREATE TABLE Test01
AS
        SELECT * FROM emp;
```

프로시저 작성

```
CREATE OR REPLACE PROCEDURE Proc02
        (v_deptno IN emp.deptno%TYPE)
IS
BEGIN
        DBMS_OUTPUT.ENABLE;

        UPDATE
                Test01
        SET
                sal = sal + sal * 0.1
        WHERE
                deptno = v_deptno;
        --      이 명령이 실행되는 순간 암시적 커서가 생겨났다.
        DBMS_OUTPUT.PUT_LINE('인상된 사람 = ' || SQL%ROWCOUNT);
END;
 /
```

실행

```
execute Proc02(10);
```

명시적 커서

==> 원래 커서의 의미처럼 자주사용하는 질의 명령을 미리 만든 후
필요하면 질의 명령을 변수를 이용해서 실행하도록 하는 것

명시적 커서의 처리 순서

1. 명시적 커서를 만든다.
(명시적 커서를 정의한다.)

형식]

```
CURSOR 커서이름 IS
        필요한 질의 명령
```

2. 반드시 커서를 프로시저에서 실행 가능하도록 조치한다.
(커서를 오픈시킨다.)

형식]

```
OPEN  커서이름;
```

3. 질의 명령을 실행한다. (커서를 Fetch 시킨다.)

형식]

```
FETCH                 커서이름;
```

4. 사용이 끝난 커서는 회수한다. (커서를 Close 시킨다)

형식]

```
CLOSE         커서이름;
```

예제]

부서 변수를 알려주면 부서별로 부서이름, 평균급여, 사원수를 출력하라.
단, 질의명령을 커서를 이용해서 처리하세요.

프로시저 작성

```
CREATE OR REPLACE PROCEDURE Proc03
        (v_deptno IN emp.deptno%TYPE)
IS
        --          1)          커서를 정의한다.(커서 변수를 만들자)
        CURSOR          dept_test IS
                SELECT d.dname, avg(e.sal), count(e.ename)
                FROM    emp e, dept d
                WHERE                   e.deptno = d.deptno
                                AND     e.deptno = v_deptno
                GROUP BY        d.dname;

        --          일반변수를 만들자.
        d_dname         dept.dname%TYPE;
        d_avg           NUMBER;
        d_countNUMBER;
BEGIN
        --          커서를 오픈시킨다.
        OPEN dept_test;
        --          필요한 시점에서 커서를 실행한다.
        FETCH  dept_test
                        INTO d_dname, d_avg, d_count;
        DBMS_OUTPUT.PUT_LINE('부서이름' ||  d_dname);
        DBMS_OUTPUT.PUT_LINE('평균급여' || d_avg);
        DBMS_OUTPUT.PUT_LINE('사원수' || d_count);
        --          사용이 종료되면 닫아준다.
        CLOSE dept_test;
END;
/
```

실행

```
execute Proc03(20);
```

참고]

만약 커서가 FOR LOOP안에서 사용되면
자동 OPEN, FETCH, CLOSE가 된다.

예제]

부서별로 부서이름, 평균급여, 사원수를 출력하라.

프로시저 작성

```
CREATE OR REPLACE PROCEDURE Proc04
IS
	CURSOR		dept_test IS
			SELECT
				d.dname, ROUND(AVG(e.sal)) AS avg,
				COUNT(e.ename) AS cnt
			FROM
				dept d, emp e
			WHERE
				d.deptno = e.deptno
			GROUP BY
				d.dname;
BEGIN
	FOR	d_emp		IN dept_test LOOP
	--	지금처럼 커서가 FOR LOOP 안에서 실행되면
	--	OPEN, FETCH, CLOSE 시킬 필요가 없다.
			DBMS_OUTPUT.PUT_LINE('부서이름' ||  d_emp.dname);
			DBMS_OUTPUT.PUT_LINE('평균급여' || d_emp.avg);
			DBMS_OUTPUT.PUT_LINE('사원수' || d_emp.cnt);
	END LOOP;
END;
/
```

실행

```
execute Proc04;
```

참고]

명시적 커서에도 멤버 변수를 사용할 수 있다.
멤버 변수의 종류는 앞의 암시적 커서와 동일하다.

예제]

사원의 이름, 직책, 급여를 출력하라.
단, 최종적으로 출력된 사원의 수를 같이 출력하라.

프로시저 작성

```
CREATE OR REPLACE PROCEDURE Proc05
IS
	CURSOR		emp_test IS
			SELECT ename, job, sal
			FROM   emp;

	d_ename			emp.ename%TYPE;
	d_job			emp.job%TYPE;
	d_sal			emp.sal%TYPE;
BEGIN
	OPEN emp_test;

	LOOP
			FETCH emp_test INTO d_ename, d_job, d_sal;
			DBMS_OUTPUT.PUT_LINE('총 사원이름 ' || d_ename);

			EXIT WHEN emp_test%NOTFOUND;
	END LOOP;
	DBMS_OUTPUT.PUT_LINE('총 사원수 ' || emp_test%ROWCOUNT);

	CLOSE  emp_test;
END;
/
```

실행

```
execute Proc05;
```

참고]

커서에도 필요하면 파라메터를 받아서 사용할 수 있다.

형식]

```
CURSOR      커서이름(파라메터변수 선언, ...) IS
        질의명령
```

예제]

부서번호를 알려주면 해당 부서의 직원 이름을 출력하는 프로시저를 만드세요.

프로시저 작성

```
CREATE OR REPLACE PROCEDURE Proc06
IS
        CURSOR          emp_test(p_deptno emp.deptno%TYPE) IS
        --              이처럼 파라메터를 받으면
        --              질의 명령을 만들때 이 파라메터를 사용해서
        --              질의 명령을 만들 수 있다.
                        SELECT ename
                        FROM            emp
                        WHERE           deptno = p_deptno;
        d_ename         emp.ename%TYPE;
BEGIN
        FOR     d_emp IN emp_test(10) LOOP
                        DBMS_OUTPUT.PUT_LINE('10번 부서 사원 ' ||
d_emp.ename);
        END LOOP;
        FOR     d_emp IN emp_test(20) LOOP
                        DBMS_OUTPUT.PUT_LINE('20번 부서 사원 ' ||
d_emp.ename);
        END LOOP;
        FOR     d_emp IN emp_test(30) LOOP
                        DBMS_OUTPUT.PUT_LINE('30번 부서 사원 ' ||
d_emp.ename);
        END LOOP;
END;
/
```

실행

```
execute Proc06;
```

WHERE CURRENT OF절

==> 커서를 이용해서 다른 질의명령을 실행하기 위한 방법
마치 우리가 배웠던 서브 질의처럼
하나의 질의 명령을 실행할 때 필요한 서브 질의를
커서로 만들어서 사용하는 방법

예제]

사원번호를 알려주고 직책을 알려주면
해당 사원의 직책을 지정한 직책으로 변경하는 프로시저로 만드세요.

프로시저 작성

```
CREATE OR REPLACE PROCEDURE Proc07
        (v_empno IN emp.empno%TYPE,
        v_job IN emp.job%TYPE)
IS
        CURSOR emp_info IS
                SELECT empno
                FROM            Test01
                WHERE           empno = v_empno
                FOR     UPDATE;
                --      WHERE CURRENT OF절이라 부르며
                --      이 커서는 UPDATE  질의 명령에서 부가적으로
                --      사용되는 커서임을 밝히는 것이다.
BEGIN
        FOR     d_emp IN emp_info LOOP
                UPDATE
                        Test01
                SET
                        job = v_job
                WHERE CURRENT OF emp_info;
                --      이 update 명령의 조건은 앞의 커서를 이용해서
                --      커서의 결과를 이용해서 조건을 만든다.
        END LOOP;
END;
/
```

실행

```
select empno, ename, job FROM test01;

execute Proc07(7369, '영업직');

select empno, ename, job FROM test01;
```

예외처리

==> PL/SQL를 실행하는 도중 발생하는 런타임 오류를 여기서도 예외라고 말하며
이것은 필요하면 자바와 동일하게 예외의 이유를 알아볼 수 있다.

다만 자바와 차이점인 예외가 발생하면 그 이후의 모든 PL/SQL 명령은 실행되지 않는다.
오직 예외의 정보를 파악하여 실행이 왜 되지 않았는지 정도만 파악할 수 있을 뿐이다.

PL/SQL에서의 예외의 종류

1. 미리 정의된 예외
   ==> PL/SQL에서 자주 발생하는 20여가지 예외에 대해서
   예외 코드값과 예외의 이름을 연결시켜 놓은 예외를 말한다.

   자동적으로 발생하기 때문에 굳이 특별한 조치를 하지 않아도
   예외 처리를 할 수 있다.

2. 미리 정의되지 않은 예외
   ==> 자주 발생하지 않기 때문에 이름을 연결시켜 놓지는 않았지만
   PL/SQL 컴파일러가 알고 있는 예외를 말한다.

   미리 이름하고 코드값과 개발자가 연결한 후(예외를 선언한다)
   사용해야 하는 예외이다.

3. 사용자 예외
   ==> PL/SQL 컴파일러가 알지 못하는 예외를 말한다.
   자바로 말하면 사용자가 강제로 발생하는 예외와 동일한 것이다.

   미리 이름을 강제로 부여하여 하나의 예외를 만들어 놓은 후
   +
   필요한 시점에서 강제로 예외라고 인정해 주어야 한다.

예외처리의 문법

```
EXCEPTION
        WHEN  예외이름 THEN
                처리할 내용;
        WHEN  예외이름 THEN
                처리할 내용;
        ....
        WHEN OTHERS THEN
                처리할 내용;
```

* EXCEPTION 절은 Procedue의 가장 마지막에 와야 한다.
  즉 EXCEPTION 처리 후에는 다른 내용이 나와서는 안된다.

http://rosebud90.tistory.com/entry/18-Oracle-Exception-%EC%98%88%EC%99%B8%EC%B2%98%EB%A6%AC

예제]

부서 번호를 알려주면 해당 부서의 직원정보를 출력하도록 한다.
단, 문제가 발생하면 예외처리를 이용해서 문제의 원인을 출력해보자.

프로시저 작성

```
CREATE OR REPLACE PROCEDURE Proc01
        (v_deptno IN emp.deptno%TYPE)
IS
        d_ename         emp.ename%TYPE;
BEGIN
        --      원래는 부서번호를 가지고 검색을 하면
        --      두줄 이상의 결과가 나올 수 있으므로
        --      for 처리를 해서 사용해야 한다.
        --      일부러 에러를 내기 위해서 한줄만 나오는 방식으로 처리할 예정
이다.
        SELECT ename
        INTO            d_ename
        FROM            emp
        WHERE           deptno = v_deptno;

        DBMS_OUTPUT.PUT_LINE('이름 = ' || d_ename);
EXCEPTION
        WHEN  TOO_MANY_ROWS THEN
                DBMS_OUTPUT.PUT_LINE('많은 줄이 결과로 나옵니다.');
        WHEN  NO_DATA_FOUND THEN
                DBMS_OUTPUT.PUT_LINE('데이터가 하나도 없습니다.');
        WHEN OTHERS THEN
                DBMS_OUTPUT.PUT_LINE('나도 모르는 문제가 발생!');
END;
/
```

실행

```
execute Proc01(10);
execute Proc01(50);
```

예제]

emp 테이블에 새로운 사원을 입력하는 프로시저를 만들어보자
단, 사원이름과 부서번호만 입력하도록 한다.
==> 그러면 사원번호가 입력되지 않아서 에러가 발생한다.
이 예외는 정의 되지 않은 예외이다.

프로시저 작성

```
CREATE OR REPLACE PROCEDURE Proc02
        (v_ename         IN emp.ename%TYPE,
        v_deptno         IN emp.deptno%TYPE)
IS
        --       정의되지 않은 예외를 사용하기 위해서는
        --       먼저 정의를 한 후 사용해야 한다.
        --       1.       예외에 사용할 이름을 정한다.
        --                형식>    적당한이름         EXCEPTION;
        not_null_test    EXCEPTION;

        --       2.       이 이름과 실제 발생할 코드값과 연결한다.
        --                형식>    PRAGMA EXCEPTION_INIT(이름, 코드값);
        PRAGMA EXCEPTION_INIT(not_null_test, -1400);
BEGIN
        INSERT INTO emp(ename, deptno)
        VALUES(v_ename, v_deptno);
EXCEPTION
        WHEN not_null_test THEN
                DBMS_OUTPUT.PUT_LINE('사원번호가 빠졌네요');
END;
/
```

실행

```
execute Proc02('jennie', 10);
```

예제]

부서 번호를 알려주면 해당 부서에 속한 사원수를
알려주는 프로시저를 만드세요.
단, 사원수가 3명 미만이면 사원수가 부족하다는 메시지를
예외처리를 이용해서 하세요.

프로시저 작성

```
CREATE OR REPLACE PROCEDURE Proc03
        (v_deptno IN emp.deptno%TYPE)
IS
        --      사용할 예외의 이름을 정한다.
        --      이름을 정하는 방법은 2번과 동일하다.
        user_error                  EXCEPTION;

        cnt             NUMBER;
BEGIN
        SELECT COUNT(ename)
        INTO            cnt
        FROM            emp
        WHERE           deptno = v_deptno;

        IF cnt <= 3 THEN
                --      나는 이 경우에 강제로 예외로 인정하고싶다.
                --      이처럼 강제로 예외를 처리할 경우
                --      형식>           RAISE   예외이름;
                RAISE   user_error;
        END IF;
        DBMS_OUTPUT.PUT_LINE('사원수 = ' || cnt);
EXCEPTION
        WHEN user_error THEN
                DBMS_OUTPUT.PUT_LINE('사원수가 부족하니 인원을 채용하
세요');
END;
/
```

실행

```
execute Proc03(10);
execute Proc03(20);
```

트리거

==> DML 질의명령이 발생하면
자동적으로 다른 처리를 하고자 할 경우에 사용하는 프로시저의 일종이다.

예를들어 한 테이블에 데이터가 입력되면
관련된 다른테이블에도 원하는 데이터를 입력하도록
하고자 할 경우에 사용되는 기능이다.

회원정보를 가입하면 이 회원의 등급을 만들어 주어야한다.
회원정보를 입력할 테이블과 등급을 관리하는 테이블이 다르면
두번 INSERT 명령을 내려야 한다.
이처럼 우리가 작업을 하다보면 관련된(서로 연결된) 작업이
진행되어야 하는 경우가 많이 발생한다.
이때 한가지 작업만 하면 다른 작업이 자동적으로 처리되도록
하고자 할 경우 사용하는 기법

테이블 준비

```
DROP Table Memb;
DROP Table Grade;

CREATE TABLE Memb
(
        m_id            varchar2(50),
        m_pass  varchar2(50)
);
CREATE TABLE Grade
(
        g_id            varchar2(50),
        g_grade varchar2(50)
);
```

만드는 형식

```
CREATE OR REPLACE TRIGGER 적당한이름
        BEFORE | AFTER
        --          DML 명령이          일어나기 전에 실행할지
        —           일어난 이후에 실행할지를 지정한다.
        INSERT[ or UPDATE or DELETE] ON 테이블이름
        --          트리거가 발행할 DML 명령이 있을때 트리거를 처리할지
                    지정한다.
        [FOR EACH ROW]
        --          생략하면 DML명령 회수에 따라 트리거를 실행한다.
        —           즉, DML 명령이 한번이면 트리거도 한번만 실행한다.
        —           사용하면 DML명령에 의해서 영향받은
                    레코드의  개수만큼 트리거가 실행한다.
        [WHEN
                    조건식]
        --          트리거가 발생해야 하는 조건식을 지정할 수 있다.
                    예 ]        WHEN
                                        deptno = 10
                                이라고 하면 부서번호가 10번인 내용이 변경되면
                                그때만 트리거를 실행하라는 의미가 된다.
BEGIN
        트리거에 사용할 질의 명령
END;
/
```

참고]

트리거가 발생하면 자동적으로 변수 두개가 발생한다.

:old **이전 데이터를 기억하고**

:new **이후 데이터를 기억한다.**

--> 이것은 **ROWTYPE 변수로 해당 테이블의 필드를 멤버 변수가 가지는 변수**이다.

예] :old.ename

==> DML 명령 이전의 ename 데이터를 기억한다.

:new.ename

==> DML 명령 이후의 ename 데이터를 기억한다.

예제]

누군가가 Member 테이블에 새로운 회원을 가입하면
자동적으로 Grade 테이블에 그 회원의 등급이 '새내기'
라고 입력되도록 해보자.

트리거 작성

```
CREATE OR REPLACE TRIGGER trigger_test
        BEFORE          --          Insert 이후에 트리거가 실행해야 하므로
        INSERT          on Member
        FOR EACH ROW
BEGIN
        INSERT
        INTO
                Grade
        VALUES
                (:new.m_id, '새내기');
END;
/
```

테스트

```
INSERT INTO Member VALUES('hong', '1234');
```

참고]

트리거는 실행되는 DML 종류에 따라서 다른 처리를 할 수 있다.

```
CREATE OR REPLACE TRIGGER 이름
        BEFORE
        INSERT OR DELETE OR UPDATE ON 테이블이름
        FOR EACH ROW
BEGIN
        이때 DML 종류에 따라 구분하는 방법
        IF INSERTING THEN
                insert가 발생하고 해야할 일
        ELSIF DELETING THEN
                delete가 발생하고 해야할 일
        ELSIF UPDATING THEN
                update가 발생하고 해야할 일
        END IF;
END;
/
```

예제]

앞의 Member에 각종 DML를 사용하여 트리거가 각각 실행되는지를 확인하자.

트리거 작성

```
CREATE OR REPLACE TRIGGER test01
        BEFORE
        INSERT OR UPDATE OR DELETE ON Member
        FOR EACH ROW
BEGIN
        IF INSERTING THEN
                DBMS_OUTPUT.PUT_LINE('인써트 질의 후 실행');
        ELSIF DELETING THEN
                DBMS_OUTPUT.PUT_LINE('딜리트 질의 후 실행');
        ELSIF UPDATING THEN
                DBMS_OUTPUT.PUT_LINE('업데이트 질의 후 실행');
        END IF;
END;
/
```

테스트

```
INSERT INTO Member VALUES('park', '1234');
UPDATE Member SET m_pass = '5678';
DELETE FROM Member;
```

참고]

트리거에서 사용하는 DML 명령은 자동 커밋되며
(즉 트랜젝션이 발생되며)
따라서 commit; rollback을 사용할 수 없다.

참고]

만약 FOR EACH ROW를 생략하면
:new와 :old를 사용할 수 없다.

데이터가 몇개 변경될 수 있는지 모르므로
무슨 데이터가 사용할 지 모르기 때문에.....

문제

실습준비

```
DROP TABLE GoodsInfo;   --      상품 정보
DROP TABLE Stock;       --      재고관리
DROP TABLE io;                  --      입출고 관리
CREATE TABLE GoodsInfo
(
        g_Code              varchar2(50),
        g_name      varchar2(50)
);

CREATE TABLE Stock
(
        g_Code              varchar2(50),
        g_stock     number(6)
);

CREATE TABLE io
(
        g_Code              varchar2(50),
        g_count     number(6),          --      개수
        g_kind      char(1)             --      약속    I : 입고,  O  : 출고
);
```

문제

GoodsInfo테이블에 새로운 상품이 등록되면
자동적으로 Stock에 그 상품의 제고 관리를 위한
등록을 재고 0으로 하여 등록하는 트리거를 만들어라.

문제

io테이블에 상품의 입고, 출고를 등록하면
즉각 재고 테이블의 재고가 변경되도록 하는
트리거를 만들어 보자.

```
:new.g_kind   =>      I : 입고, O : 출고

SET
        g_stock = g_stock + :new.g_count
```

테스트

```
INSERT INTO GoodInfo VALUES('1234', '냉장고');
INSERT INTO IO VALUES('1234', 10, 'I');
INSERT INTO IO VALUES('1234', 3, 'O');
```

스케줄러

==> 특정 시간이 되면 원하는 질의가 자동 실행되도록 하는 기능을 말한다.
즉, 오라클에게 스케줄을 알려주어서 그 스케줄대로
작업을 실행한다고 하여 스케줄러 라고 이름을 부여한 것이다.

작업 순서

1. **실행할 질의명령을 포함하는 프로시저를 만들어 놓는다.**
2. **프로그램 등록**

==> 스케줄로가 실행할 프로그램(1번에서 만든 프로시저)를
등록해 놓아야 한다.

3. **스케줄 등록**

==> 언제 프로시저가 실행될지를 지정하는 내용을
만들어 놓는 것

4. **잡 등록**

==> 2번의 내용과 3번 내용을 합쳐서 하나의 스케줄 작업을
완성하는 단계를 말한다.

예제]

1분에 한번씩 자동적으로 내용이 입력되도록 하고싶다.

테이블 준비

```
DROP TABLE Test02;
CREATE TABLE Test02
(
        t_no            NUMBER(6),
        t_date          date
);
```

1. 프로시저를 만든다.

```
CREATE OR REPLACE PROCEDURE Proc01
IS
BEGIN
        INSERT
        INTO
                Test02
        VALUES
                ((SELECT NVL(MAX(t_no), 0) + 1 FROM Test02), SYSDATE);
        COMMIT;
END;
/
```

2. 프로그램 등록

형식]

```
BEGIN
        DBMS_SCHEDULER.CREATE_PROGRAM
        (
                program_name => '적당한이름',
                progarm_action => '사용할 프로시저이름',
                --          프로시저를 이용하지 않고
                --          직접 질의 명령을 만들어도 상관이 없다.
                program_type = 'STORED_PROCEDURE',
                --          만약 위에서 직접 질의명령으로 등록하면
                --          STORED_PROCEDURE대신
                —           EXECUTABLE 이라고 하면된다.
                comments => '적당한 설명',
                enabled => 'TRUE'
                --          만드는 순간 활성화 하세요
        );
END;
/
```

예제의 경우

```
BEGIN
        DBMS_SCHEDULER.CREATE_PROGRAM(
                program_name => 'BATCH_PROGRAM',
                program_action => 'Proc01',
                program_type => 'STORED_PROCEDURE',
                comments => '날짜입력',
                enabled => TRUE);
END;
/
```

3. 스케줄 등록

==> 언제 실행될지를 미리 지정하는 것

형식]

```
BEGIN
        DBMS_SCHEDULER.CREATE_SCHEDULE
        (
                schedule_name =>        '적당한 이름',
                start_date =>       시작 날짜,
                end_date =>       종료 날짜,
                repeat_interval => 반복 주기,
                comments => '설명'
        );
END;
/
```

참고]

start_date, end_date는 생략해도 상관이 없다.
start_date를 생략하면 만드는 순간부터 스케줄이 가동한다.
end_date를 무한정 반복하게 된다.

예]
오늘밤 12시 부터 스케줄을 가동하고자 하면
start_date => (TRUNC(SYSDATE) + 0 / 24) + 1,
0 이 시작 시간
1 이 오늘 이후의 시작 날짜

참고]

repeat_interval은 반복 주기를 지정하는 부분으로
repeat_interval => FREQ=?1?;INTERVAL=?2? 의 형식으로 지정한다.

?1? 간격을 의미한다.
예> HOURLY 시간 간격
MINUTELY 분간격
DAILY 일간격
?2? 간격의 주기

repeat_interval => FREQ=MINUTELY;INTERVAL=1

예제의 경우

```
BEGIN
DBMS_SCHEDULER.CREATE_SCHEDULE(
        schedule_name => 'DAILY_1_MINUTE',
        repeat_interval => 'FREQ=MINUTELY;INTERVAL=1',
        comments => 'Every AM 03 HOUR');
END;
/
```

4. 위에서 만든 두가지를 합쳐서 일단위로 만든다.

형식]

```
BEGIN
        DBMS_SCHEDULER.CREATE_JOB
        (
                job_name =>'잡의 이름',
                program_name =>'BATCH_PROGRAM',
                schedule_name =>'DAILY_1_MINUTE',
                comments => 'Service desk stats main batch program',
                enabled =>TRUE
        );
END;
/
```

예제의 경우

```
BEGIN
        DBMS_SCHEDULER.CREATE_JOB (
        job_name =>'BATCH_JOB',
        program_name =>'BATCH_PROGRAM',
        schedule_name =>'DAILY_1_MINUTE',
        comments => 'Service desk stats main batch program',
        enabled =>TRUE);
END;
/
```

테스트

```
select t_no, TO_CHAR(t_date, 'yyyy-mm-dd hh:mi:ss') FROM Test02;
```

삭제하기

```
execute dbms_scheduler.drop_job('BATCH_JOB',false);
execute dbms_scheduler.drop_program('BATCH_PROGRAM',false);
execute dbms_scheduler.drop_schedule('DAILY_1_MINUTE',false);
```