오라클의 PL/SQL

PL/SQL 의 이해

1. PL/SQL 의 개요

데이터베이스 내의 데이터를 조작하기 위해서 오라클이 제공해 주는 SQL 문을 사용했는데 SQL 의 장점은 쿼리문 하나로 원하는 데이터의 조회와 조작할 수 있다는 점이다.

SQL 자체는 비절차 언어이기에 몇 개의 쿼리문 사이에 어떠한 연결이나 절차적 방식을 사용하고자할 때는 사용할 수 없으므로 오라클 사에서 SQL 언어에 절차적인 프로그래밍 언어를 가미해 만든 것이 PL/SQL 이다.

PL/SQL 은 Oracle's Procedural Language extension to SQL 의 약자이다.

SQL 은 ANSI 표준 언어로 어떤 제품군도 사용할 수 있지만, PL/SQL 은 오라클의 고유한 언어이므로 다른 제품군에서 사용할 수 없다.

2. PL/SQL 의 특징

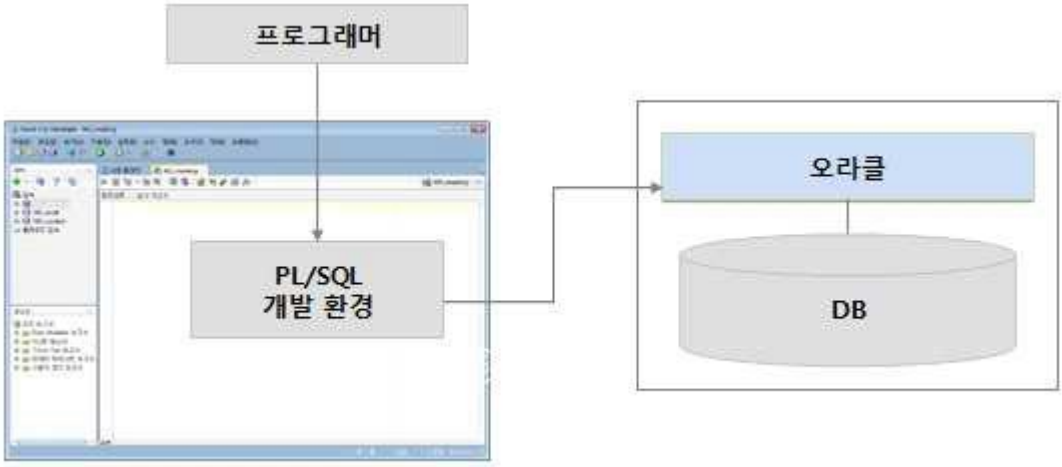
PL/SQL 은 C 언어와 유사한 문법을 사용하는 블록 구조의 언어로 sql 에 대한 Procedural Language 의 확장이며 컴파일한 후 결과를 실행한다.
PL/SQL 은 procedure, function, package 등을 만들 때 사용하며 sql data type 을 지원하며 변수 선언과 같은 표준 프로그래밍 구조를 제공한다.
PL/SQL 은 일반 프로그래밍 언어적인 요소를 거의 다 가지고 있어서 실무에서 요구되는 절차적인 데이터 처리를 다 할 수 있다.
PL/SQL 은 SQL 과 연동되어서 막강한 기능을 구현할 수 있다.
PL/SQL 은 데이터 트랜잭션 처리능력, 데이터에 대한 보안, 예외처리 등 데이터베이스와 관련된 중요한 모든 기능을 지원하기 때문에 데이터베이스 업무를 처리하기에 최적화된 언어이다.

3. PL/SQL 의 기능

PL/SQL 은 모듈 프로그램 생성이 가능하므로 프로그램 개발의 모듈화가 가능하다.
PL/SQL 은 식별자 선언 및 오류제어가 가능하다.
PL/SQL 은 성능향상, 이식성, 통합성, 효율성 기대 등을 가져올 수 있으며 절차적 언어 제어구조를 사용한 프로그램이 가능하다.

PL/SQL 에서 제공하는 절차적 프로그래밍 기능:

- 변수, 상수 등을 선언하여 SQL 과 절차형 언어에서 사용한다.
- if 문을 사용하여 조건에 따라 문들을 분기한다.
- loop 문을 사용하여 일련의 문을 반복적으로 실행한다.

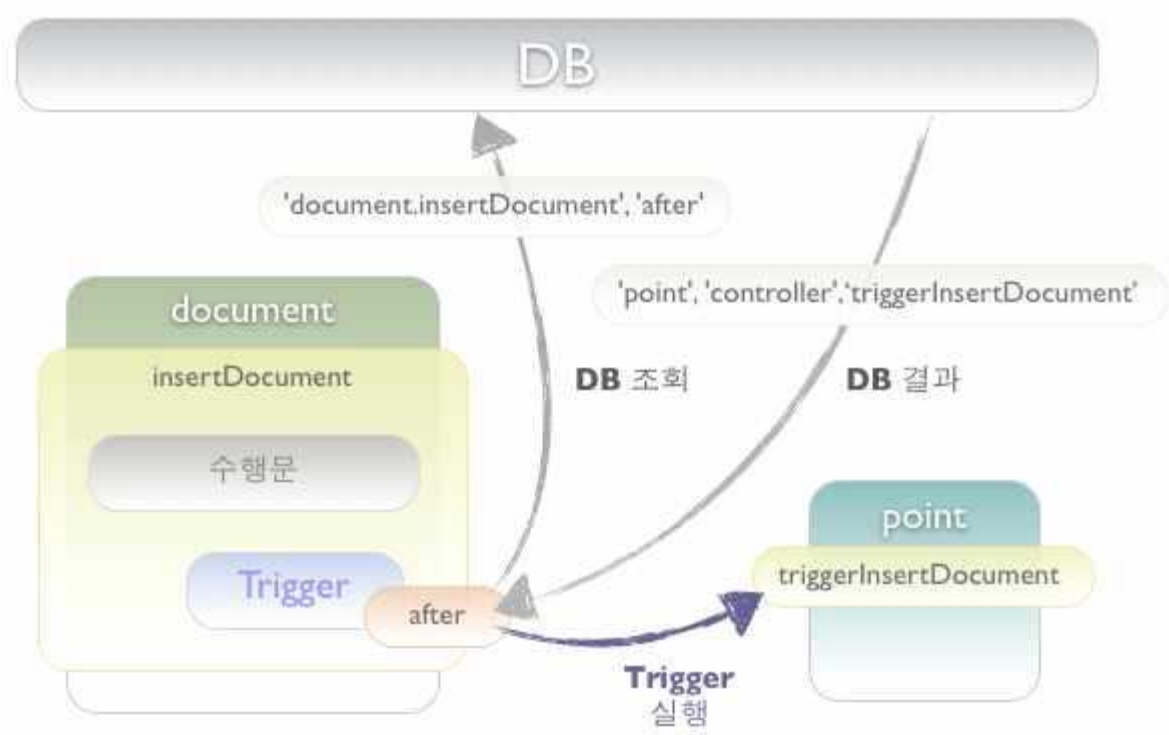


1) 패키지

패키지(package)는 관련된 stored procedure, function, cursor 와 변수들의 모임이다.
패키지는 이름이 있는 하나의 패키지로 번들(Bundle)이 되며 번들은 꾸러미로 비슷한 기능을 가진프로시저나 함수가 보관되어 있다.
PL/SQL 은 프로시저와 함수가 동시에 실행되기 때문에 패키지 내에서 선언한 변수는 프로시저와 함수 모두에 적용될 수 있다.

2) 트리거

데이터베이스 트리거(Trigger)는 테이블에 대한 이벤트에 반응해 자동으로 실행되는 작업을 의미한다.
트리거는 데이터 조작어의 데이터 상태 관리를 자동화하는 데 사용된다.
트리거를 사용하여 데이터 작업 제한, 작업 기록, 변경 작업 감사 등을 할 수 있다.


DB
'document.insertDocument', 'after'
'point', 'controller','triggerInsertDocument'
document
insertDocument
수행문
Trigger
after
DB 조회
DB 결과
point
triggerInsertDocument
Trigger
실행

3. PL/SQL 의 작성 규칙

PL/SQL 블록 내에서는 한 문이 종료할 때마다 명령의 종료 표시인 ;(세미콜론)을 사용한다.
begin … end 문 뒤에 세미콜론(;)을 사용하여 하나의 블록이 끝났다는 것을 명시한다.
PL/SQL 문은 여러 줄에 걸쳐질 수 있으나 명령어는 분리될 수 없다.
PL/SQL 블록의 내용을 읽기 쉽도록 공백문자를 사용하여 PL/SQL 문을 적절하게 분리함으로써 의미분석이 되도록 한다.
PL/SQL 문은 들여쓰기도 권장한다.
PL/SQL 블록 내의 오라클 함수를 사용할 수 있으나 그룹함수는 SQL 문에 포함되어야만 사용될 수 있다.
PL/SQL 의 프로시저 내에서는 단일행 함수만 사용할 수 있다.

PL/SQL 에서 시퀀스를 사용할 때 오라클 11g 이전 버전에서는 SQL 문을 이용하여 시퀀스를 변수에 할당한 후 해당 변수값을 사용했으나
오라클 11g 버전부터는 PL/SQL 문에서 바로 시퀀스를 사용할 수 있다.

PL/SQL 에서도 묵시적 데이터 타입 변환은 문자와 숫자, 문자와 날짜를 연산할 때 발생하며 이 부분 때문에 성능에 의도하지 않게 나쁜 영향을 줄 수 있으므로 데이터 타입의 일치에 항상 주의해야 한다.

1) 식별자

식별자는 PL/SQL 객체에 부여되는 이름으로 식별자명은 기본 오라클 명명 규칙을 준수한다.
식별자는 테이블 이름이나 변수명 등은 모두 식별자다.
오라클의 예약어는 식별자로 사용될 수 없다.
식별자명 중에 문자 리터럴과 날짜 리터럴은 '(싱글 쿼터)로 표시해야 하며 널 값은 NULL 상수로 기술한다.

식별자 중에서 다음과 같은 경우에 식별자를 '(싱글 쿼터)로 표시하여 사용하며 식별자를 연이어 사용할 때는 "(더블 쿼터)로 묶어야 하지만, "(더블 쿼터)로 묶은 식별자는 특별한 경우가 아니면 권장하지 않는다.

- 식별자의 대소문자 구분이 필요한 경우
- 공백과 같은 문자 포함할 경우
- 예약어를 사용해야 할 경우

2) 주석

주석은 직접 프로그래밍 코드의 성능에는 영향을 주지 않지만, 나중을 위해서 설명이나 해설 등을 기록해 두는 것이다.
주석의 표기에 따라 기능이 달라지며 주석의 표기는 다음과 같다.

- --(더블 하이픈) : 한 줄 주석 내용을 표시한다.
- /*(슬래시 아스터리스크) … */(아스터리스크 슬래시) : 여러 줄에 주석 내용을 표시한다.

3) SQL 문

PL/SQL 문 내에서의 SQL 문을 사용하기 위해서는 다음과 같은 점을 주의해야 한다.

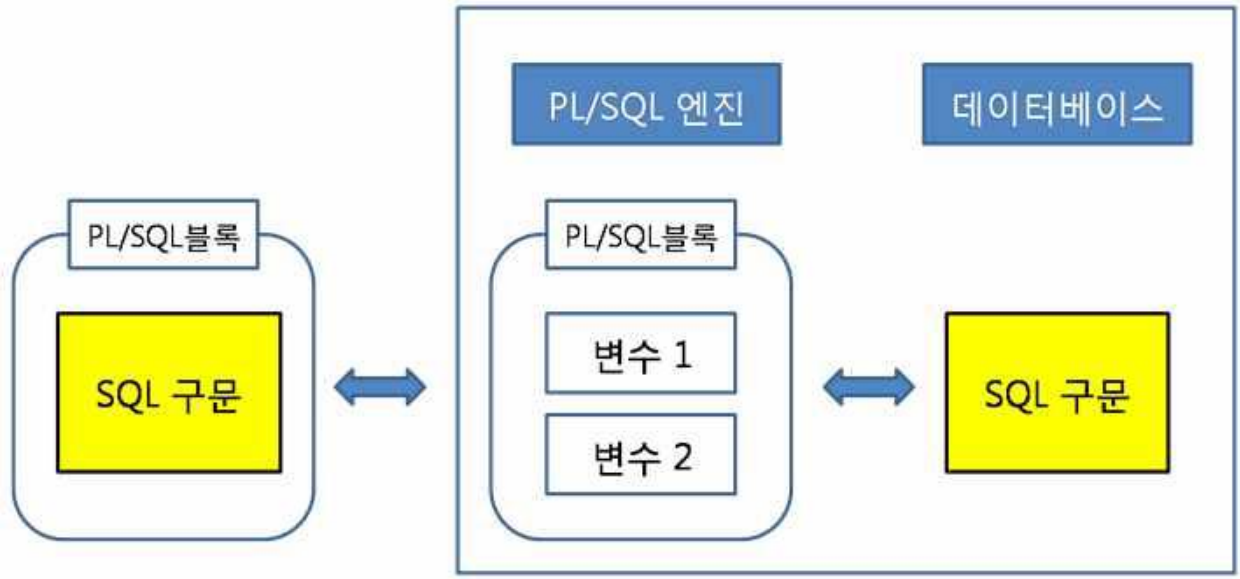
- begin … end 문의 end 명령어는 트랜잭션의 끝이 아니라 PL/SQL 블록의 끝을 나타낸다.
- PL/SQL 은 데이터 정의어와 데이터 제어어를 직접 지원하지 않지만, 동적 SQL 을 사용하면 PL/SQL 에서 데이터 정의어와 데이터 제어어를 실행할 수 있다.

실습:
간단한 메시지를 출력한다.

```
set serveroutput on;
begin
        dbms_output.put_line('안녕 PL/SQL');
end;
```

PL/SQL 의 구조

1. PL/SQL 의 블록



1) 블록의 구조

형식
[declare] 선언 블록
begin
문 1 ---------- 실행 블록
⋮
문 N
[exception] 예외 블록
end; ---------- 실행블록

(1) 선언 블록

선언 블록은 선택 사항이므로 필요할 때 declare 명령어를 선언한다.
선언 블록에는 변수, 상수, 커서 등을 선언하는 부분이다.
선언 블록에 선언한 변수와 상수의 끝에는 ;(세미콜론)을 표시해야 한다.

(2) 실행 블록

실행 블록은 필수 사항이므로 반드시 선언해야 한다.
실행 블록에는 PL/SQL 문이나 SQL 문이 위치하는 부분이다.
실행 블록에 선언한 PL/SQL 문이나 SQL 문의 끝에는 ;(세미콜론)을 표시해야 한다.
실행 블록은 begin 명령어로 시작하고 end 명령어로 끝나며 end 명령어의
끝에는 ;(세미콜론)을 표시해야 하며 실행문의 종료는 end 문을 만나야 종료된다.
begin 명령어와 end 명령어로 사이에는 제어문, SQL 문, CURSOR 문 등을 포함한다.
begin 명령어와 end 명령어 사이에는 SQL 문에는 데이터 질의어, 데이터 조작어, 데이터
처리어만 사용할 수 있으며 단락의 끝에는 ;(세미콜론)을 표시해야 한다.

(3) 예외 블록

예외 블록은 선택 사항이므로 필요할 때 exception 명령어를 선언한다.
예외 블록에는 예외처리문을 선언하는 부분이다.
예외 발생 시 사전에 정의한 대로 수행할 내용이 위치한다.

2) 블록의 형태

(1) anonymous 블록

형식:
[declare]
begin
문 1
⋮
문 N
[exception]
end;

anonymous 블록은 이름이 없는 익명블록으로 모든 PL/SQL 환경에서 사용할 수 있다.

(2) procedure 블록

형식:
create procedure 프로시저명
is
begin
문 1
⋮
문 N
[exception]
end;

procedure 블록은 반환값이 있을 수도 있고 없을 수도 있는 실행블록이다.
procedure 블록은 데이터베이스 내부에서 만들어지며 다른 애플리케이션에서도 사용할 수 있다.
procedure 블록은 매개변수를 받을 수 있으며 재사용이 가능하다.

(3) function 블록

형식:

```
create function 함수명
return 데이터 타입
is
begin
문 1
⋮
문 N
[exception]
end;
```

function 블록은 반환값을 반드시 반환하는 실행 블록이다.

function 블록은 데이터베이스 내부에서 만들어지며 다른 애플리케이션에서도 사용할 수 있다.

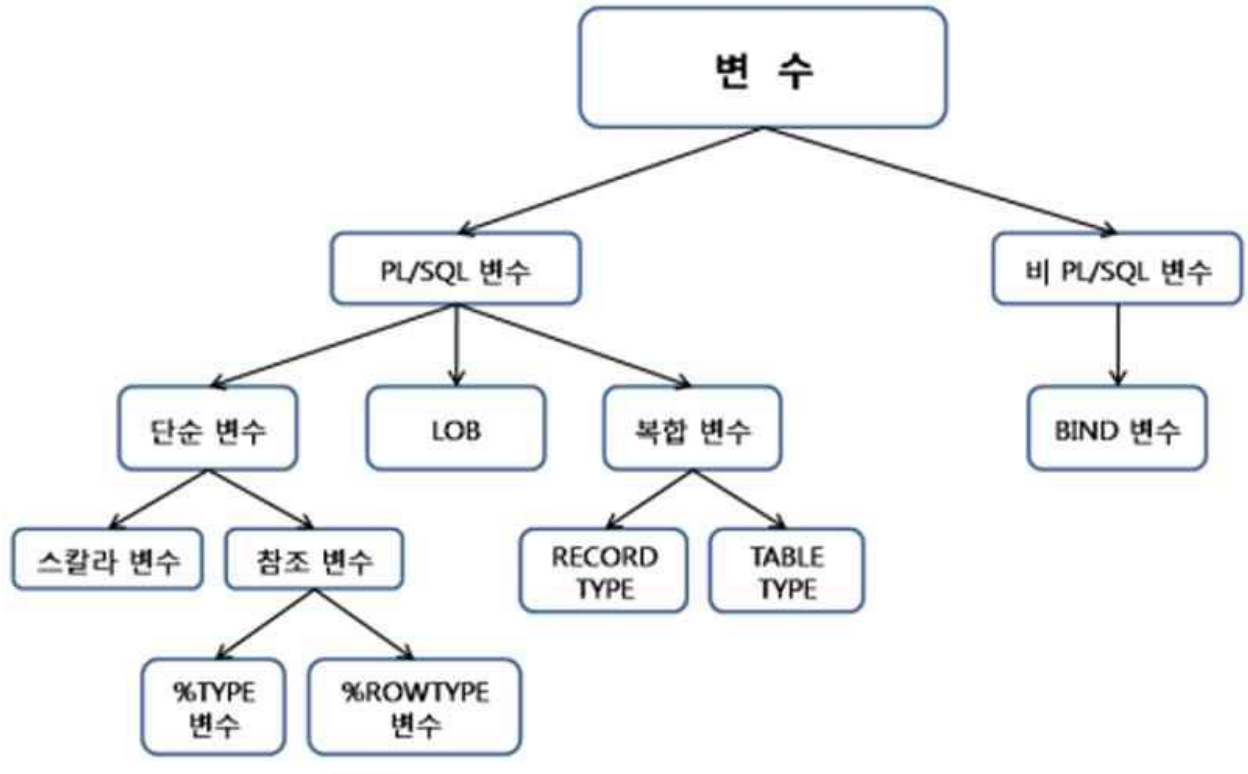
function 블록은 매개변수를 받을 수 있으며 재사용이 가능하다.

2. PL/SQL 의 변수

1) 변수의 개요

형식:
변수명 데이터 타입 [상수] [not null] [:= 초기값];

PL/SQL 블록 내에서 사용되는 모든 변수는 사용 전에 미리 선언되어야 한다.



결과값을 선언된 변수에 할당시키면 해당 변수는 세션이 종료될 때까지 사용이 가능한 상태가 된다.
PL/SQL 에서는 변수의 값을 지정하거나 재지정하기 위해서 :=(콜론이퀄)을 사용하며
콜론이퀄의 왼쪽에는 새 값을 받기 위한 변수를 선언하고 오른쪽에는 저장할 값을 선언한다.
변수에 할당되는 모든 값은 리터럴이다.

변수의 명명 규칙:

- 변수의 시작은 반드시 문자로 시작해야 한다.
- 변수는 특수 문자와 숫자를 포함할 수 있다.
- 변수는 예약어를 포함하면 안 된다.

2) 변수의 데이터 타입

(1) 숫자 데이터 타입
숫자 데이터 타입에는 number, integer, float 등이 있다.
숫자 데이터 타입은 숫자 리터럴에 사용되며 숫자 리터럴은 정수나 실수값을 나타낸다.

(2) 문자 데이터 타입
문자 데이터 타입에는 char, varchar2 등이 있다.
문자 데이터 타입은 문자 리터럴에 사용되며 문자 리터럴은 문자나 문자열 값을 나타낸다.

(3) 날짜 데이터 타입
날짜 데이터 타입에는 date 등이 있다.
날짜 데이터 타입은 날짜 리터럴에 사용되며 날짜 리터럴은 날짜 값을 나타낸다.

(4) 논리 데이터 타입
논리 데이터 타입에는 boolean 등이 있다.
논리 데이터 타입은 불리언 리터럴에 사용되며 불리언 리터럴은 불리언 변수에 할당된 값을 나타낸다.
불리언 리터럴의 값은 true 와 false 의 값이거나 null 값이다.

3) 변수의 형태

(1) 스칼라(scala) 변수
형식:
변수명 데이터 타입 [:= 초기값];

스칼라 변수는 스칼라값을 취할 수 있도록 속성이 정의된 변수이다.
스칼라는 하나의 수치만으로 완전히 표시되는 양을 의미하며 단순히 크기를 나타내는 값이라고 생각하면 된다.
오라클의 스칼라 변수는 SQL 에서 사용하던 데이터 타입과 거의 유사하다.

숫자 데이터 타입과 문자 데이터 타입으로 스칼라 변수를 선언하면 다음과 같다.
숫자 데이터 타입:

- 변수명 number;
- 변수명 number(5);
- 변수명 number := 20;

문자 데이터 타입:

- 변수명 varchar2(10);
- 변수명 varchar2(10) := '홍길동';

(2) 레퍼런스(reference) 변수

레퍼런스 변수는 변수의 주소값을 참조하는 속성이 정의된 변수이다.
오라클의 레퍼런스 변수는 테이블 컬럼의 데이터 타입을 참조하는 데이터 타입으로 선언한 변수이다.
개발자가 테이블에 정의된 컬럼의 데이터 타입과 크기를 모두 파악하고 있다면 문제가 없겠지만 대부분은 그렇지 못하기 때문에 오라클에서는 레퍼런스 변수를 제공한다.
오라클의 레퍼런스 변수는 컬럼의 데이터 타입이 변경되더라도 컬럼의 데이터 타입과 크기를 그대로 참조하기 때문에 굳이 레퍼런스 변수 선언을 수정할 필요가 없다는 장점이 있다.

%type 속성
형식:
변수명 테이블명.컬러명%type [:= 초기값];

%type 속성을 지정하면 컬럼 단위로 참조하는 레퍼런스 변수가 된다.
레퍼런스 변수는 컬럼에 맞추어 변수를 선언하기 위해 %type 속성을 사용한다.

%rowtype 속성
형식:
변수명 테이블명%rowtype;

%rowtype 속성을 지정하면 모든 컬럼을 참조하는 레퍼런스 변수가 된다.
레퍼런스 변수는 테이블의 모든 컬럼을 레코드로 선언하기 위하여 %rowtype 속성을 사용한다.

%rowtype 속성을 지정한 레퍼런스 변수의 특징:

- 테이블 내부의 컬럼 집합의 이름, 데이터 타입, 크기, 속성을 그대로 사용하여 선언한다.
- %rowtype 속성 앞에 오는 것은 테이블 이름이다.
- 지정된 테이블의 구조와 같은 구조를 갖는 변수를 선언한다.

%rowtype 속성을 지정한 레퍼런스 변수의 장점:

- 테이블 컬럼의 개수나 데이터 타입을 알지 못할 때 편리하다.
- 코딩 이후에 컬럼의 수나 데이터 타입이 변경될 경우 재수정이 필요 없게 된다.
- select 문을 이용하여 컬럼을 조회할 때 편리하다.

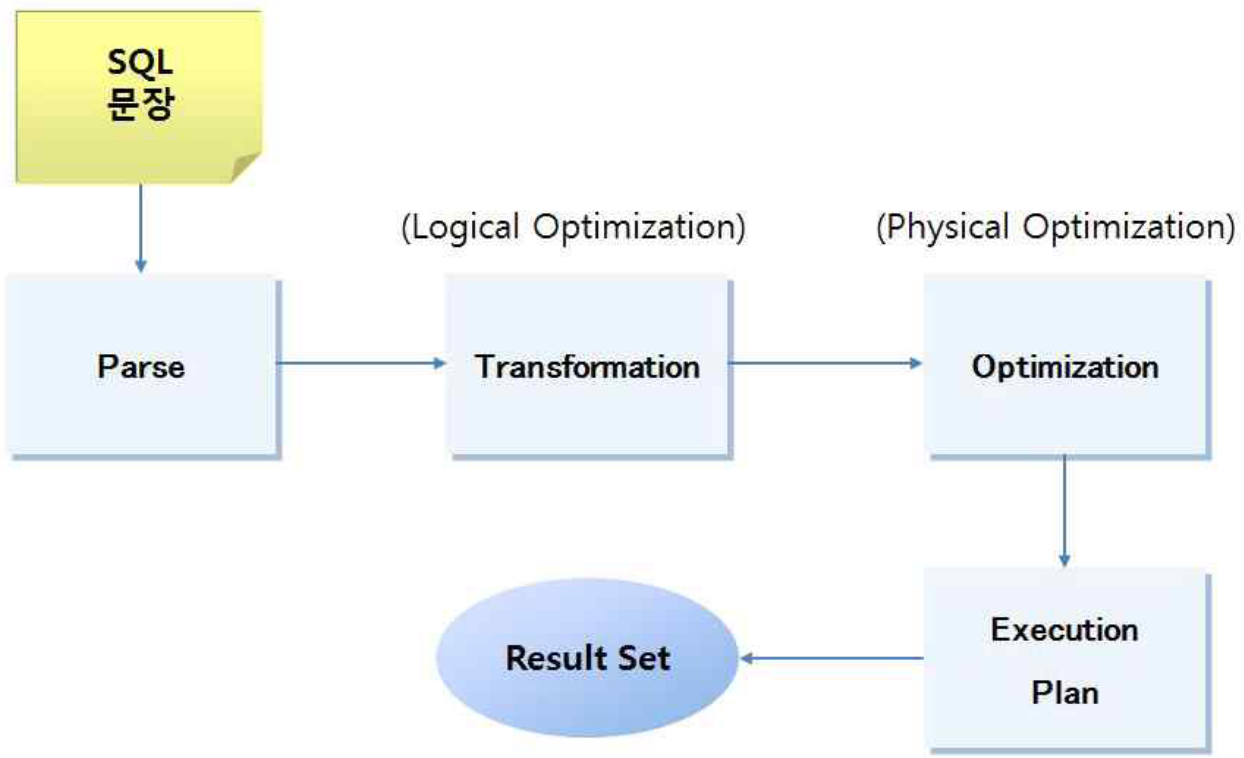
실습:

스칼라 변수를 선언하고 변수값을 조회한다.

```
-- 화면 출력기능을 활성화한다.
set serveroutput on;
-- 스칼라 변수를 선언한다.
declare
	sonno number(4);
	sonname varchar2(12);
-- 실행문을 시작한다.
begin
	sonno := 1001;
	sonname := '홍길동';
	dbms_output.put_line(' 사번 이름');
	dbms_output.put_line(' ---------------');
	dbms_output.put_line(' ' || sonno || ' ' || sonname);
-- 실행문을 종료한다.
end;
```

3. PL/SQL 의 데이터 질의어

데이터 질의어는 정보를 추출할 필요가 있을 때 SQL 을 사용한다.
PL/SQL 에서는 SQL 에서 사용하는 명령어를 그대로 사용할 수 있다.



테이블의 컬럼에서 질의 된 값을 변수에 할당시키기 위해 select 문을 사용한다.
SQL 문과 차이점은 PL/SQL 의 select 문은 into 문이 필요한데 into 문에는 데이터를 저장할 변수를 선언한다.

형식:
select 컬럼 1, 컬럼 2,…컬럼 N
into 변수 1, 변수 2,…변수 N
from 테이블명
where 조건식;

설명:

select 컬럼 1, 컬럼 2,…컬럼 N: 컬럼의 목록이다. 컬럼에는 행 함수, 그룹 함수, 표현식을 기술할 수 있다.

into 변수 1, 변수 2,…변수 N: 데이터를 저장할 변수의 목록이다. into 문은 값을 저장하기 위한 변수명을 선언하며 변수는 PL/SQL 의 레코드 변수이다.

from 테이블명: 조회할 테이블을 지정한다.

where 조건식; : 조회할 조건을 지정하며 조건을 충족하는 컬럼값으로 SQL 문을 제한한다.

조건식에는 PL/SQL 변수와 상수를 포함하여 컬럼명, 표현식, 비교 연산자로 구성된다.

조건식의 조건은 오직 하나의 값을 반환할 수 있는 조건이어야 한다.

조건식은 조회할 조건에 대한 조건이나 표현식으로 ;(세미콜론)으로 종료한다.

실습:

PL/SQL 의 select 문으로 emp 테이블에서 사원번호와 이름을 조회한다.

```
-- 화면 출력기능을 활성화한다.
set serveroutput on;
-- 레퍼런스 변수를 선언한다.
declare
        sonno emp.empno%type;
        sonname emp.ename%type;
-- 실행문을 시작한다.
begin
        select empno, ename into sonno, sonname
        from emp
        where ename = 'SMITH';
        -- 화면에 출력한다.
        dbms_output.put_line(' 사번 이름');
        dbms_output.put_line(' ---------------');
        dbms_output.put_line(' ' || sonno || ' ' || sonname);
-- 실행문을 종료한다.
end;
```

PL/SQL 의 제어문

1. 선택문

if 문은 조건을 제시해서 만족하느냐 하지 않느냐에 따라 문을 선택적으로 수행하기 때문에 선택문이라고 한다.

1) 단일 if ... then 문

형식:
if 조건문 then
조건에 만족할 경우 실행되는 문;
end if;

단일 if…then 문은 조건에 따라 어떤 명령을 선택적으로 처리하기 위해 사용하는 가장 대표적인 문이다.
if…then 문의 조건이 true 이면 then 문 이하의 문을 실행하고 조건이 false 이거나 null 이면 end if 문으로 수행을 종료한다.

실습:

emp 테이블에서 직원의 커미션을 구한다.

```
set serveroutput on;
-- 레퍼런스 변수를 선언한다.
declare
        sonemp emp%rowtype;
        sonsal number(7,2);
begin
        select * into sonemp
        from emp
        where ename='SMITH';
        -- 커미션이 null 일 경우를 조건에 지정하고 수행한다.
        if (sonemp.comm is null) then
        sonsal := sonemp.sal*12;
        -- 조건이 false 이거나 null 이면 수행을 종료한다.
        end if;
        dbms_output.put_line(' 사번 이름 커미션');
        dbms_output.put_line('-------------------------');
        dbms_output.put_line(' '||sonemp.empno||' '||sonemp.ename||' '||sonsal);
end;
```

실습:

emp 테이블에서 부서번호로 부서명을 확인한다.

```
set serveroutput on;
-- 레퍼런스 변수와 스칼라 변수를 선언한다.
declare
	sonno emp.empno%type;
	sonname emp.ename%type;
	sondeptno emp.deptno%type;
	-- sondname 변수를 null 로 초기화한다.
	sondname varchar2(20) := null;
begin
	select empno, ename, deptno
	into sonno, sonname, sondeptno
	from emp
	where empno=7369;
	-- sondeptno 변수가 10 일 경우를 조건에 지정하고 수행한다.
	if (sondeptno = 10) then
		sondname := 'accounting';
	end if;
	-- sondeptno 변수가 20 일 경우를 조건에 지정하고 수행한다.
	if (sondeptno = 20) then
		sondname := 'clerk';
	end if;
	-- sondeptno 변수가 30 일 경우를 조건에 지정하고 수행한다.
	if (sondeptno = 30) then
		sondname := 'sales';
	end if;
	-- sondeptno 변수가 40 일 경우를 조건에 지정하고 수행한다.
	if (sondeptno = 40) then
		sondname := 'operations';
	-- 조건이 false 이거나 null 이면 수행을 종료한다.
	end if;
	dbms_output.put_line(' 사번 이름 부서명');
	dbms_output.put_line(' --------------------------');
	dbms_output.put_line(' '|| sonno ||' '||sonname||' '||sondname);
end;
```

2) 이중 if...then...else 문

형식

if 조건문 then
조건에 만족할 경우 실행되는 문;
else
조건에 만족하지 않을 때 실행되는 문;
end if;

if…then 문 중에 가장 일반적으로 많이 사용되는 형식이 이중 if…then…else 문이다.
이중 if…then…else 문은 참일 때와 거짓일 때 각각 다른 문을 수행하도록 지정할 수 있다.
이중 if…then…else 문은 조건을 검사하고 그 결과가 참이면 조건에 만족하는 문을 수행하고 거짓이면 조건에 만족하지 않는 문을 수행한다.

실습:
emp 테이블에서 사원명으로 연봉을 조회한다.

```
set serveroutput on;
-- 레퍼런스 변수와 스칼라 변수를 선언한다.
declare
        sonemp emp%rowtype;
        sonsal number(7, 2);
begin
        select * into sonemp
        from emp
        where ename='SMITH';
        -- 커미션이 null 일 경우를 조건에 지정하고 수행한다.
        if (sonemp.comm is null) then
                sonsal := sonemp.sal*12;
        -- 커미션이 null 이 아닐 때 수행한다.
        else
                sonsal := sonemp.sal*12+sonemp.comm;
        -- 조건이 false 이거나 null 이면 수행을 종료한다.
        end if;
        dbms_output.put_line(' 사번 이름 연봉');
        dbms_output.put_line('-------------------------');
        dbms_output.put_line(' '||sonemp.empno||' '||sonemp.ename||' '||sonsal);
end;
```

3) 다중 if...then...elsif...else 문

형식:
if 조건문 then
조건에 만족할 경우 실행되는 문 1;
elsif 조건문 then
조건에 만족할 경우 실행되는 문 2;
⋮
elsif 조건문 then
조건에 만족할 경우 실행되는 문 N;
else
조건에 만족하지 않을 때 실행되는 문;
end if;

다중 if…then…elsif…else 문은 경우의 수가 둘이 아닌 셋 이상에서 하나를 선택해야 할 때 사용한다.

실습:

emp 테이블에서 부서번호로 부서명을 확인한다.

```
set serveroutput on;
-- 레퍼런스 변수와 스칼라 변수를 선언한다.
declare
        sonemp emp%rowtype;
        sondname varchar2(14);
begin
        select * into sonemp
        from emp
        where ename='SMITH';
        -- sondeptno 변수가 10 일 경우를 조건에 지정하고 수행한다.
        if (sonemp.deptno = 10) then
        sondname := 'accounting';
        -- sondeptno 변수가 20 일 경우를 조건에 지정하고 수행한다.
        elsif (sonemp.deptno = 20) then
                sondname := 'clerk;
        -- sondeptno 변수가 30 일 경우를 조건에 지정하고 수행한다.
        elsif (sonemp.deptno = 30) then
                sondname := 'sales';
        -- sondeptno 변수가 40 일 경우를 조건에 지정하고 수행한다.
        elsif (sonemp.deptno = 40) then
                sondname := 'operations';
        -- 조건이 false 이거나 null 이면 수행을 종료한다.
        end if;
        dbms_output.put_line(' 사번 이름 부서명');
        dbms_output.put_line(' --------------------------');
        dbms_output.put_line(' '|| sonemp.empno ||' '||sonemp.ename||'
        '||sondname);
end;
```

2. 반복문

반복문은 특정 조건에 따라서 특정 실행문을 반복적으로 수행한다.
반복문은 반복 횟수를 지정하는 것도 가능하고 무한정 반복 처리하는 무한 루프도 가능하다.
선택문이 결과에 중점을 둔다면 반복문은 목적에 중점을 둔다.

1) loop...end loop 문

형식:
loop
조건에 만족할 경우 실행되는 문 1;
조건에 만족할 경우 실행되는 문 2;
⋮
조건에 만족할 경우 실행되는 문 N;
exit [when 조건식];
end loop;

loop…end loop 문은 조건없이 반복 작업을 제공한다.
loop…end loop 문은 실행의 흐름이 end loop 문에 도달할 때마다 loop 문으로 제어권을 전달하는 이러한 루프를 무한 루프라 하고 여기서 빠져나가려면 exit 문을 사용한다.
loop 문은 루프에 들어갈 때 조건이 이미 일치했다 할지라도 적어도 한 번은 문이 실행된다.
exit 문을 이용하여 end loop 문 다음으로 제어권을 전달하기 때문에 루프를 종료할 수 있다.
조건에 따라 루프를 종료할 수 있도록 when 문을 덧붙일 수 있으며 when 문 다음의 조건식이 참이면 루프를 끝내고 루프 후의 다음 문으로 제어권을 전달한다.

실습:

loop…end loop 문으로 1 부터 5 까지 출력한다.

```
set serveroutput on;
declare
        num number := 1;
begin
        -- end loop 문에서 제어권을 전달받고 반복한다.
        loop
                dbms_output.put_line(num);
                -- num 변수에 1 을 더하여 num 변수에 할당하여 누적한다.
                num := num + 1;
                if (num > 5) then
                        exit;
                end if;
                -- 제어권을 loop 문으로 전달한다.
        end loop;
end;
```

2) for...in...end loop 문

for…in…end loop 문은 카운트를 기본으로 반복 작업을 제공한다.
for…in…end loop 문은 실행의 흐름이 end loop 문에 도달할 때마다 for…in 문의 카운트로 제어권을 전달하여 루프 한다.
for…in…end loop 문에서 사용되는 카운트는 정수로 자동 선언되므로 따로 선언할 필요가 없다.
for…in…end loop 문은 루프를 반복할 때마다 자동으로 1 씩 증가 또는 감소한다.

형식:
for counter in [reverse] lower..upper loop
조건에 만족할 경우 실행되는 문 1;
조건에 만족할 경우 실행되는 문 2;
⋮
조건에 만족할 경우 실행되는 문 N;
end loop;

(1) counter

upper 나 lower 에 도달할 때까지 루프를 반복함으로써 1 씩 자동으로 증가하거나 감소하는 값을 가진 암시적으로 선언된 정수이다.

(2) reverse

upper 에서 lower 까지 반복함으로써 인덱스가 1 씩 감소하도록 한다.

(3) lower..upper

lower..upper 는 lower 와 upper 사이를 ..(더블 도트)로 구분한다.
lower 는 counter 값의 범위에 대한 하단의 한계값을 지정한다.
upper 는 counter 값의 범위에 대한 상단의 한계값을 지정한다.

실습:

for…end loop 문으로 1 부터 5 까지 출력한다.

```
set serveroutput on;
declare
begin
        -- end loop 문에서 제어권을 전달받고 num 변수는 1 부터 5 까지 반복한다.
        for num in 1..5 loop
                dbms_output.put_line(num);
        -- 제어권을 for…in 문의 num 변수로 전달한다.
        end loop;
end;
```

3) while…loop…end loop 문

형식:

while 조건식 loop
조건에 만족할 경우 실행되는 문 1;
조건에 만족할 경우 실행되는 문 2;
⋮
조건에 만족할 경우 실행되는 문 N;
end loop;

while…loop…end loop 문은 조건을 기본으로 반복 작업을 제공한다.
while…loop…end loop 문은 실행의 흐름이 end loop 문에 도달할 때마다 while…loop 문의 조건식으로 제어권을 전달하여 루프 한다.
while…loop…end loop 문은 조건식이 참인 동안만 문을 반복한다.
while…loop…end loop 문의 조건식은 반복이 시작될 때 확인하므로 조건이 거짓이면 루프를 빠져나가므로 루프 내의 문이 한 번도 수행되지 않을 경우도 있다.

실습:

while…loop…end loop 문으로 1 부터 5 까지 출력한다.

```
set serveroutput on;
declare
        num number := 1;
begin
        -- end loop 문에서 제어권을 전달받고 num 변수는 5 이하까지 반복한다.
        while (num <= 5) loop
                dbms_output.put_line(num);
                -- num 변수에 1 을 더하여 num 변수에 할당하여 누적한다.
                num := num + 1;
        -- 제어권을 while…loop 문의 (num <= 5) 조건식에 전달한다.
        end loop;
end;
```

PL/SQL 을 통한 쿼리의 확장

1. 저장 프로시저

1) 저장 프로시저의 개요

저장 프로시저(Stored Procedure)는 개발자가 만든 PL/SQL 문을 데이터베이스에 저장하고 필요한 경우에 여러 번 호출하여 사용할 수 있다.
저장 프로시저는 자주 사용되는 쿼리문을 모듈화시켜서 필요할 때마다 호출하여 사용할 수 있다.
저장 프로시저를 사용하면 복잡한 데이터 조작어를 필요할 때마다 다시 입력할 필요 없이 간단하게 호출만 해서 실행 결과를 얻을 수 있다.
저장 프로시저를 사용하면 성능도 향상되고 호환성 문제도 해결된다.
저장 프로시저를 사용하면 여러 클라이언트가 업무 규칙을 공유할 수 있다.
저장 프로시저는 다양한 클라이언트 응용 프로그램에서 서버에 보낼 SQL 문들을 미리 모아서 서버에서 관리하는 데이터로 저장해 두므로 여러 클라이언트에서 서버에 저장해 둔 SQL 문을 함께 사용할 수 있다.
저장 프로시저는 업무처리 규칙이 바뀌었을 때도 모든 클라이언트 응용 프로그램이 서버에 바뀐 업무처리 규칙을 공유할 수 있으므로 클라이언트마다 따로 조치를 처리해 줄 필요가 없다.

저장 프로시저는 속도문제에 있어서 다음과 같은 장점:

- 저장 프로시저는 서버에서 SQL 구문 검사를 끝난 상태에서 대기하고 있다가 호출만 되면 바로 실행을 한다는 것인데 저장 프로시저로 처리하지 않으면 일일이 SQL 문을 보내서 구문이 실행될 때마다 SQL 구문 검사가 매번 실행되어 속도가 느려진다.
- 저장 프로시저는 네트워크에서 오고 가는 긴 SQL 문의 네트워크 트래픽도 줄일 수 있는데 복잡한 SQL 코딩은 그 라인이 몇백 라인도 될 수 있으며 많은 SQL 이 네트워크에서 오간다면 네트워크 트래픽을 무시하지 못하지만, 저장 프로시저는 저장 프로시저의 이름만 호출하기 위해 네트워크에서 왔다 갔다 하므로 비교적 훨씬 적은 데이터가 네트워크상에서 움직인다.

오라클에서 지원하는 저장 프로시저는 다른 프로그래밍 언어에서 사용하는 함수의 개념과 비슷하다.
저장 프로시저는 클라이언트 응용 프로그램에서 반복적으로 같은 처리를 할 때 미리 SQL 문을 서버에 저장해 두고 클라이언트에서는 단순히 저장 프로시저를 매개변수와 함께 호출만 해주면 서버에서 해당 저장 프로시저를 읽어서 곧바로 실행하게 된다.

PL/SQL 에서 바인드 변수를 사용하여 출력할 수 있다.

형식 :바인드 변수명

바인드 변수는 PL/SQL 외부에서도 사용할 수 있는 변수를 의미한다.
바인드 변수는 호스트 환경에서 생성되어 데이터를 저장하기 때문에 호스트 변수라고 한다.
variable 명령어를 사용하여 PL/SQL 블록에서도 사용할 수 있다.
바인드 변수는 PL/SQL 블록이 실행된 후에도 액세스할 수 있다.
바인드 변수는 print 명령어를 이용하여 출력할 수 있다.
바인드 변수는 :(콜론)을 덧붙여 사용한다.
저장 프로시저의 매개변수와 바인드 변수의 이름은 같을 필요는 없지만, 데이터 타입은 반드시 같아야 한다.

2) 저장 프로시저의 생성

형식:
create[or replace] procedure 프로시저명(인자 1 [mode] 데이터 타입,
인자 2 [mode] 데이터 타입,…인자 N [mode] 데이터 타입)
is
begin
조건에 만족할 경우 실행되는 문 1;
조건에 만족할 경우 실행되는 문 2;
⋮
조건에 만족할 경우 실행되는 문 N;
end;

저장 프로시저를 생성하려면 create procedure 문 다음에 새롭게 생성하고자 하는 프로시저명을 선언한다.
replace 명령어는 같은 이름으로 저장 프로시저를 생성할 때 기존 저장 프로시저를 삭제하고 새롭게 선언한 내용으로 재생성한다.
is 명령어는 저장 프로시저의 환경에 대한 존재를 확인하며 is 명령어를 생략하면 컴파일 오류가 발생한다.
PL/SQL 의 제어문을 통해서 저장 프로시저를 제어할 수가 있다.

drop procedure 문으로 저장 프로시저를 삭제하며 형식은 다음과 같다.
형식:
drop procedure 프로시저명;

생성된 저장 프로시저를 제거하기 위해서는 drop procedure 문 다음에 제거하고자 하는 프로시저명을 선언한다.
저장 프로시저는 어떤 값을 전달받아서 그 값에 의해서 서로 다른 결과물을 구하게 된다.
값을 저장 프로시저에 전달하기 위해서 프로시저명 다음에 ( )(퍼렌씨시스) 안에 전달받을 값을 저장할 인자인 매개변수를 기술한다.
저장 프로시저에 값을 전달해 주기 위해서 지정한 모드(mode)로 인자인 매개변수에 전달한다.
프로그래밍에서 모드는 특정한 방식을 의미한다.

저장 프로시저에서 사용하는 모드의 형태는 다음과 같다.

(1) in 모드

in 모드는 데이터를 전달받을 때 사용하며 매개변수에 입력한 값을 전달하는 것이다.
저장 프로시저에서 사용된 매개변수는 저장 프로시저를 호출할 때 값을 저장 프로시저 내부에서 받아서 사용할 수도 있는데 저장 프로시저 호출 시 넘겨준 값을 받아오기 위한 매개변수는 in 모드를 지정한다.

(2) out 모드

out 모드는 수행된 결과를 받아갈 때 사용하며 매개변수에 저장된 값을 호출하는 것이다.
저장 프로시저에 구한 결과값을 얻어 내기 위해서는 매개변수는 out 모드를 지정한다.

(3) inout 모드

inout 모드는 in 모드와 out 모드의 두 가지 목적에 모두 사용된다.

1) 저장 프로시저의 생성

실습 1:

저장 프로시저 생성을 위한 테스트 테이블을 복사한다.

```
-- 저장 프로시저에 사용할 테이블을 emp 테이블에서 복사한다.
create table empcopy
as
select * from emp;
```

실습 2:

생성한 테이블 정보를 조회한다.

```
-- 생성한 테이블의 정보를 조회한다.
select * from empcopy;
```

실습 3:

저장 프로시저를 생성한다.

```
-- 저장 프로시저를 생성한다.
create or replace procedure del_all
is
begin
        -- empcopy 테이블의 저장된 데이터를 삭제한다.
        delete from empcopy;
        -- 트랜잭션 작업이 성공하여 완료한다.
        commit;
end;
```

실습 4:

저장 프로시저를 실행한다.

```
-- 저장 프로시저를 실행하여 empcopy 테이블에 저장된 데이터를 삭제한다.
execute del_all;
```

실습 5:
저장 프로시저 실행을 확인한다.

```
-- 저장 프로시저에 대한 실행을 확인한다.
select * from empcopy;
```

2) 저장 프로시저의 조회

user_source 시스템 테이블인 데이터 사전으로 어떤 저장 프로시저가 생성되어 있는지와 해당 프로시저의 내용이 무엇인지 확인할 수 있다.

user_source 시스템 테이블의 컬럼 의미:

- name 컬럼 : 저장 프로시저의 프로시저명을 저장한다.
- type 컬럼 : 저장 프로시저의 타입을 저장한다.
- line 컬럼 : 행의 라인 번호를 저장한다.
- text 컬럼 : 저장 프로시저 코드를 저장한다.

실습 1:
저장 프로시저의 구조를 확인한다.

```
-- user_source 시스템 테이블의 구조를 확인한다.
desc user_source;
```

실습 2:
저장 프로시저의 코드를 확인한다.

```
-- 저장 프로시저의 코드를 확인한다.
select name, type, line, text from user_source;
```

3) 저장 프로시저의 매개변수

실습 1:
저장 프로시저 생성을 위한 테스트 테이블을 복사한다.

```
-- 저장 프로시저에 사용할 테이블을 emp 테이블에서 복사한다.
create table empcopy2
as
select * from emp;
```

실습 2:

생성한 테이블 정보를 조회한다.

```
-- 생성한 테이블의 정보를 조회한다.
select * from empcopy2;
```

실습 3:

매개변수가 존재하는 저장 프로시저를 생성한다.

```
-- 매개변수가 존재하는 저장 프로시저를 생성한다.
create or replace procedure del_ename(sonename empcopy2.ename%type)
is
begin
        -- empcopy 테이블의 저장된 데이터를 조건에 맞게 삭제한다.
        delete from empcopy2 where ename like sonename;
        -- 트랜잭션 작업이 성공하여 완료한다.
        commit;
end;
```

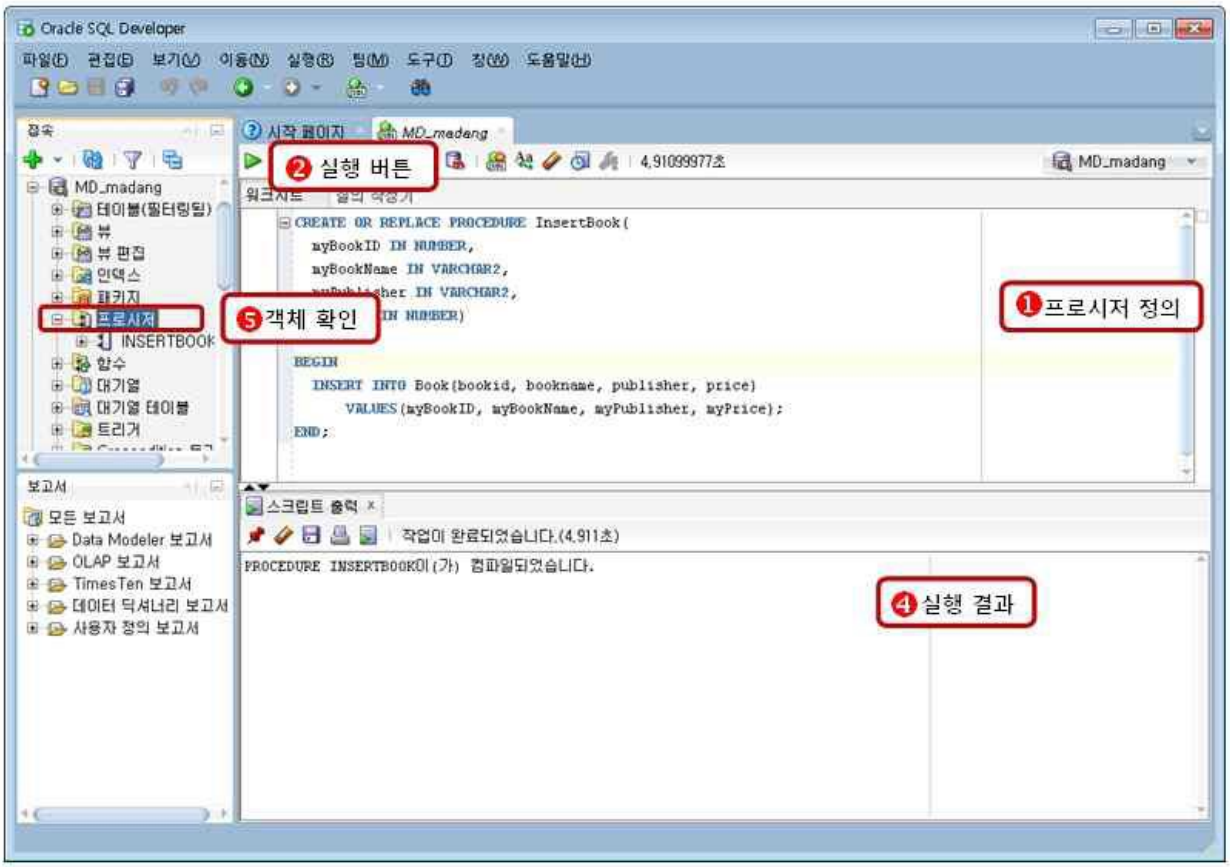
실습 4:

저장 프로시저를 실행한다.

```
-- 저장 프로시저를 실행하여 empcopy 테이블에 S 문자로 시작하는 사원의 데이터를
삭제한다.
execute del_ename('S%');
```

※ SQL Developer 에서 저장 프로시저 확인

–SQL Developer 에서 저장 프로시저를 실행하면 다음 그림과 같이 확인할 수 있다.


Oracle SQL Developer
파일(F) 편집(E) 보기(V) 이동(N) 실행(R) 팀(M) 도구(T) 정(W) 도움말(H)
접속
MD_madang
테이블(필터링됨)
뷰
뷰 편집
인덱스
패키지
프로시저
INSERTBOOK
함수
대기열
대기열 테이블
트리거
보고서
모든 보고서
Data Modeler 보고서
OLAP 보고서
TimesTen 보고서
데이터 딕셔너리 보고서
사용자 정의 보고서
시작 페이지
MD_madang
❷ 실행 버튼
4.91099977초
MD_madang
워크시트
CREATE OR REPLACE PROCEDURE InsertBook(
myBookID IN NUMBER,
myBookName IN VARCHAR2,
IN NUMBER)
BEGIN
INSERT INTO Book(bookid, bookname, publisher, price)
VALUES(myBookID, myBookName, myPublisher, myPrice);
END;
❶ 프로시저 정의
❺ 객체 확인
스크립트 출력
작업이 완료되었습니다.(4.911초)
PROCEDURE INSERTBOOK(이)가 컴파일되었습니다.
❹ 실행 결과

4) 바인드 변수를 통한 부서 정보 조회

실습 1:

부서 정보를 조회하는 저장 프로시저를 생성한다.

```
-- 매개변수가 존재하는 저장 프로시저를 생성한다.
create or replace procedure dept_select (
-- in 모드로 dept 테이블의 deptno 컬럼에 입력한 값을 vdeptno 매개변수에 전달한다.
vdeptno in dept.deptno%type,
-- out 모드로 dept 테이블의 dname 컬럼값을 받아와서 vdname 매개변수에서 호출한다.
vdname out dept.dname%type,
-- out 모드로 dept 테이블의 loc 컬럼값을 받아와서 vloc 매개변수에서 호출한다.
vloc out dept.loc%type)
is
begin
        -- select 문으로 수행한 결과값을 into 문 뒤에 선언한 변수에 저장한다.
        select dname, loc into vdname, vloc from dept
        where deptno=vdeptno;
end;
```

실습 2:

바인드 변수로 데이터를 저장하고 부서번호로 부서 정보를 조회한다.

```
/*
컬럼에 저장된 값을 호스트 환경에서 생성하여 데이터를 저장하기 위해 바인드 변수를
선언한다.
variable 명령어를 사용하여 PL/SQL 블록에서도 사용할 수 있다.
저장 프로시저의 매개변수와 바인드 변수의 이름은 같을 필요는 없지만, 데이터 타입은 반드시
같아야 한다.
*/
variable vdname varchar2(30);
variable vloc varchar2(30);
-- 바인드 변수를 호출하는 저장 프로시저를 실행하고 바인드 변수는 :(콜론)을 덧붙여
사용한다.
execute dept_select(40, :vdname, :vloc);
-- 바인드 변수는 print 명령어로 출력할 수 있으므로 print 명령어로 출력한다.
print vdname;
print vloc;
```

5) 바인드 변수를 통한 사원정보 조회

실습 1:

사원정보를 조회하는 저장 프로시저를 생성한다.

```
-- 매개변수가 존재하는 저장 프로시저를 생성한다.
create or replace procedure sel_empno (
sonempno in emp.empno%type,
sonename out emp.ename%type,
sonsal out emp.sal%type,
sonjob out emp.job%type)
is
begin
        select ename, sal, job into sonename, sonsal, sonjob
        from emp
        where empno=sonempno;
end;
```

실습 2:

바인드 변수로 데이터를 저장하고 사원 번호로 사원정보를 조회한다.

```
--컬럼에 저장된 값을 호스트 환경에서 생성하여 데이터를 저장하기 위해 바인드 변수를
선언한다.
variable var_ename varchar2(15);
variable var_sal number;
variable var_job varchar2(9);
-- 바인드 변수를 호출하는 저장 프로시저를 실행하고 바인드 변수는 :(콜론)을 덧붙여
사용한다.
execute sel_empno(7369, :var_ename, :var_sal, :var_job);
-- 바인드 변수는 print 명령어로 출력할 수 있으므로 print 명령어로 출력한다.
print var_ename;
print var_sal;
print var_job;
```

6) 저장 프로시저의 insert 데이터 조작어

실습 1:

dept 테이블에 데이터를 입력할 저장 프로시저를 생성한다.

```
create or replace procedure dept_insert(
-- in 모드로 dept 테이블의 deptno 컬럼에 입력한 값을 vdeptno 매개변수에 전달한다.
vdeptno in dept.deptno%type,
-- in 모드로 dept 테이블의 dname 컬럼에 입력한 값을 vdname 매개변수에 전달한다.
vdname in dept.dname%type,
-- in 모드로 dept 테이블의 loc 컬럼에 입력한 값을 vloc 매개변수에 전달한다.
vloc in dept.loc%type)
is
begin
        insert into dept values(vdeptno, vdname, vloc);
end;
```

실습 2:

dept 테이블에 데이터를 입력할 저장 프로시저를 실행한다.

```
-- 저장 프로시저로 데이터를 입력한다.
execute dept_insert(50, '기획실','서울');
```

실습 3:

dept 테이블에서 입력한 저장 프로시저 실행을 확인한다.

```
-- 입력한 저장 프로시저에 대한 실행을 확인한다.
select * from dept;
```

7) 저장 프로시저의 update 데이터 조작어

실습 1:

dept 테이블에 데이터를 수정할 저장 프로시저를 생성한다.

```
create or replace procedure dept_update(
-- in 모드로 dept 테이블의 deptno 컬럼에 입력한 값을 vdeptno 매개변수에 전달한다.
vdeptno in dept.deptno%type,
-- in 모드로 dept 테이블의 dname 컬럼에 입력한 값을 vdname 매개변수에 전달한다.
vdname in dept.dname%type,
-- in 모드로 dept 테이블의 loc 컬럼에 입력한 값을 vloc 매개변수에 전달한다.
vloc in dept.loc%type)
is
begin
        update dept set deptno= vdeptno, dname= vdname, loc=vloc
        where deptno = vdeptno;
end;
```

실습 2:

dept 테이블에 저장 프로시저로 데이터를 수정한다.

```
-- 저장 프로시저로 데이터를 수정한다.
execute dept_update(50, '기획실','부산');
```

실습 3:

dept 테이블에서 수정한 저장 프로시저 실행을 확인한다.

```
-- 수정한 저장 프로시저에 대한 실행을 확인한다.
select * from dept;
```

8) 저장 프로시저의 delete 데이터 조작어

실습 1:

dept 테이블에 데이터를 삭제할 저장 프로시저를 생성한다.

```
create or replace procedure dept_delete(
-- in 모드로 dept 테이블의 deptno 컬럼에 입력한 값을 vdeptno 매개변수에 전달한다.
vdeptno in dept.deptno%type)
is
begin
        delete from dept where deptno = vdeptno;
end;
```

실습 2:

dept 테이블에 저장 프로시저로 데이터를 삭제한다.

```
-- 저장 프로시저로 데이터를 삭제한다.
execute dept_delete(50);
```

실습 3:

dept 테이블에서 삭제한 저장 프로시저 실행을 확인한다.

```
-- 삭제한 저장 프로시저에 대한 실행을 확인한다.
select * from dept;
```

2. 저장 함수

형식
create[or replace] function 함수명(인자 1 [mode] 데이터 타입,
인자 2 [mode] 데이터 타입, … 인자 N [mode] 데이터 타입)
return 데이터 타입;
is
begin
조건에 만족할 경우 실행되는 문 1;
조건에 만족할 경우 실행되는 문 2;
⋮
조건에 만족할 경우 실행되는 문 N;
end;

저장 함수(Stored Function)는 자주 사용되는 쿼리문을 모듈화시켜서 필요할 때마다 호출하여 사용한다.
저장 함수는 저장 프로시저와 거의 유사한 용도로 사용하며 차이점은 저장 함수는 실행 결과를 반환할 수 있다는 점이다.
저장 함수를 생성하려면 create function 문 다음에 새롭게 생성하고자 하는 함수명을 선언한다.
replace 명령어는 같은 이름으로 저장 함수를 생성할 때 기존 저장 함수를 삭제하고 새롭게 선언한 내용으로 재생성한다.
return 명령어로 함수가 되돌려 받게 되는 데이터 타입과 되돌려 받을 값을 기술한다.
is 명령어는 저장 프로시저의 환경에 대한 존재를 확인하며 is 명령어를 생략하면 컴파일 오류가 발생한다.

저장 함수는 호출 결과를 얻어오기 위해서 호출 방식에서도 저장 프로시저와 차이점이 있으며 호출 형식은 다음과 같다.

형식:
execute :변수명 := 함수명(인자 리스트);

PL/SQL 의 제어문을 통해서 저장 프로시저를 제어할 수가 있다.

drop function 문으로 저장 함수를 삭제하며 형식은 다음과 같다.
형식:
drop function 함수명;

생성된 저장 함수를 제거하기 위해서는 drop function 문 다음에 제거하고자 하는 함수명을 선언한다.
값을 저장 함수에 전달하기 위해서 함수명 다음에 ( )(퍼렌씨시스) 안에 전달받을 값을 저장할 인자인 매개변수를 기술한다.
저장 함수에 값을 전달해 주기 위해서 지정한 모드(mode)로 인자인 매개변수에 전달한다.
프로그래밍에서 모드는 특정한 방식을 의미한다.

저장 함수에서 사용하는 모드의 형태는 다음과 같다.
(1) in 모드: 데이터를 전달받을 때 사용하며 매개변수에 입력한 값을 전달하는 것이다.
(2) out 모드: 수행된 결과를 받아갈 때 사용하며 매개변수에 저장된 값을 호출하는 것이다.
(3) inout 모드: in 모드와 out 모드의 두 가지 목적에 모두 사용된다.

1) 급여 200% 보너스 지급

실습 1:
사원의 급여에 200% 보너스를 지급하는 저장 함수를 생성한다.

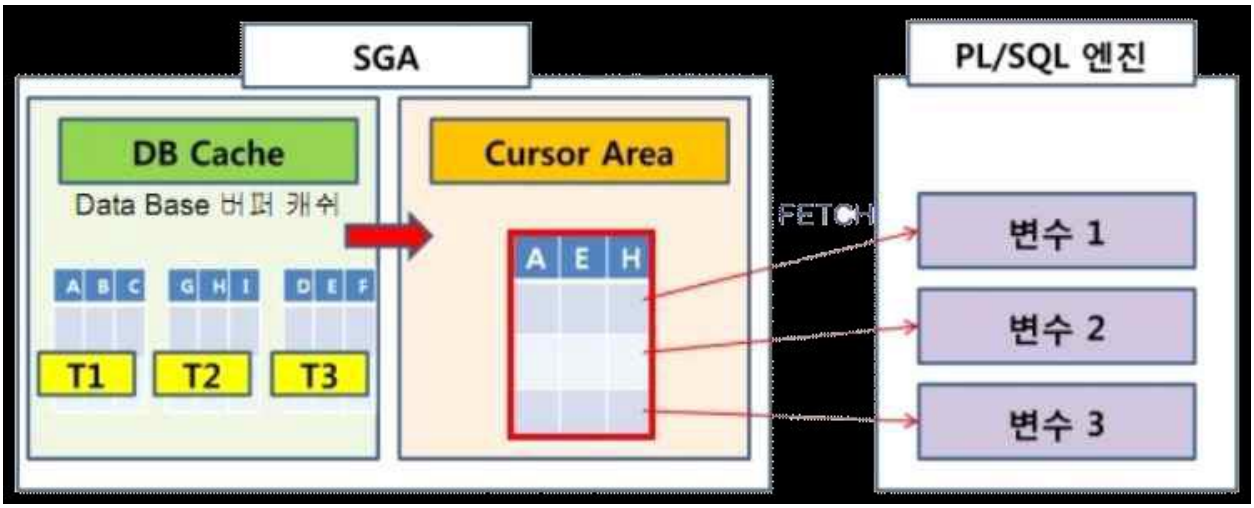
```
-- 매개변수가 존재하는 저장 함수를 생성한다.
create or replace function cal_bonus(
-- in 모드로 emp 테이블의 empno 컬럼에 입력한 값을 sonempno 매개변수에 전달한다.
sonempno in emp.empno%type)
        -- number 데이터 타입의 값을 반환한다.
        return number
is
        sonsal number(7, 2);
begin
        select sal into sonsal
        from emp
        where empno = sonempno;
        -- 조회 결과로 얻어진 급여로 200% 보너스를 구해서 함수의 결과값으로 반환한다.
        return (sonsal * 200);
end;
```

실습 2:
바인드 변수로 데이터를 저장하고 급여를 조회한다.

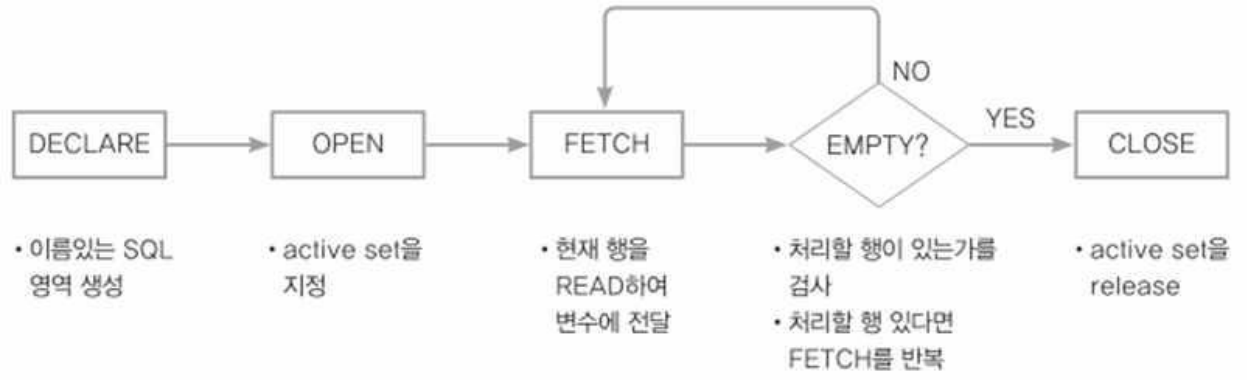
```
--컬럼에 저장된 값을 호스트 환경에서 생성하여 데이터를 저장하기 위해 바인드 변수를
선언한다.
variable var_res number;
-- 바인드 변수를 호출하는 저장 함수를 실행하고 바인드 변수는 :(콜론)을 덧붙여 사용한다.
execute :var_res := cal_bonus(7369);
-- 바인드 변수는 print 명령어로 출력할 수 있으므로 print 명령어로 출력한다.
print var_res;
```

3. 커서

데이터베이스의 커서(Cursor)는 일련의 데이터에 순차적으로 액세스할 때 검색과 현재 위치를 포함하는 데이터 요소이다.
일반적으로 데이터베이스는 같은 종류의 데이터를 많이 축적하고 있으며 사용자가 요청하는 데이터는 1 건 또는 여러 건이 될 수 있다.



커서가 오픈되면 DB 버퍼 캐시에서 필요한 컬럼만 빼서 Cursor Area 에 복사하여 가져다 놓는다.
커서를 통해 Cursor Area 에 있는 데이터 위치를 찾아가서 데이터를 꺼내고 꺼낸 데이터를 PL/SQL 변수에 담는다.
사용자가 요청하는 데이터가 여러 건일 때 커서를 사용하여 처리할 수 있다.
PL/SQL 에서 처리 결과가 여러 개의 행으로 구해지는 select 문을 처리하려면 커서를 이용한다.



오라클은 커서를 사용하여 여러 행을 조회하며 오라클 커서의 형태는 다음과 같다.

- 묵시적 커서 : 오라클에서 자동으로 선언해주는 SQL 커서다.
- 명시적 커서 : 프로그래머에 의해 선언되며 이름이 있는 커서다.

오라클에서는 커서의 속성을 통해 커서의 상태를 알려준다.

커서의 속성을 이용해서 커서를 제어해야 하는데 속성의 종류는 다음과 같다.

(1) %rowcount 속성

%rowcount 속성은 커서가 얻어 온 총 행의 개수이다.
가장 마지막 행이 몇 번째 행인지 카운트하고 해당 커서에서 실행한 총 행의 개수를 반환한다.

(2) %found 속성

%found 속성은 해당 커서 안에 아직 수행해야 할 데이터가 있을 때 true 를 반환하고 없을 때 false 를 반환한다.

(3) %notfound 속성

%notfound 속성은 해당 커서 안에 수행해야 할 데이터가 없을 때 true 를 반환하고
있을 때 false 를 반환한다.

(4) %isopen 속성

%isopen 속성은 현재 묵시적 커서가 메모리에 오픈되어 있을 때는 true 를 그렇지 않을 때는 false 를 반환한다.

형식:

```
cursor 커서명
is
조건에 만족할 경우 실행되는 문;
begin
        open 커서명;
        fetch 커서명 into 변수명;
        close 커서명;
end;
```

설명:

cursor 커서명: cursor 명령어 다음에 새롭게 생성하고자 하는 커서명을 선언한다.

is 명령어: 커서의 환경에 대한 존재를 확인하며 is 명령어를 생략하면 컴파일 오류가 발생한다.

open 커서명; : 생성한 커서를 연다. 질의를 수행하고 조회 조건을 충족하는 모든 행으로 구성된 결과 세트를 생성하기 위해 커서를 open 명령어로 커서를 연다.

fetch 커서명 into 변수명; : 구성된 결과 세트에서 컬럼 단위로 데이터를 읽어 들이고 인출 한 후에 커서는 결과 세트에서 다음 행으로 이동한다.

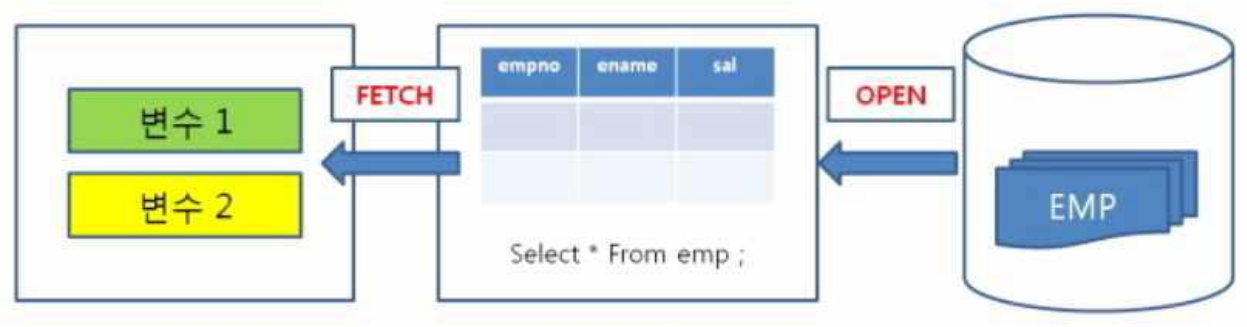
fetch 명령어는 행에 대한 정보를 얻어 와서 into 명령어 뒤에 선언한 변수에 저장한 후에 다음 행으로 이동한다.

얻어진 여러 개의 컬럼에 대한 결과값을 모두 처리하려면 반복문에 fetch 명령어를 기술해야 한다.

close 커서명; : 사용한 커서를 닫는다.

close 명령어는 커서를 사용할 수 없게 하고 결과 세트를 해제한다.

select 문이 처리되고 완성된 후에는 커서를 닫고 필요하다면 커서를 다시 열 수도 있다.



1) 명시적 커서로 dept 테이블의 모든 데이터 조회

실습 1:

dept 테이블의 모든 데이터를 조회할 프로시저를 생성한다.

```
create or replace procedure cursor_call
is
	-- %rowtype 속성으로 dept 테이블의 모든 컬럼을 참조하는 레퍼런스 sonemp
변수를 선언한다.
	sondept dept%rowtype;
	-- soncursor 커서를 생성한다.
	cursor soncursor
	is
	select * from dept;
begin
	dbms_output.put_line('부서번호 부서명 지역명');
	dbms_output.put_line('--------------------------------------');
	-- 생성한 soncursor 커서를 연다.
	open soncursor;
	-- end loop 문에서 제어권을 전달받고 반복한다.
	loop
		-- soncursor 커서에서 행을 정보를 획득하고 저장하고 다음 행으로 이동한다.
		fetch soncursor into sondept.deptno, sondept.dname, sondept.loc;
		-- 해당 커서 안에 수행해야 할 데이터가 없을 때 루프를 종료한다.
		exit when soncursor%notfound;
		dbms_output.put_line(sondept.deptno||' '||sondept.dname||'
	'||sondept.loc);
	-- 제어권을 loop 문으로 전달한다.
	end loop;
	-- 사용한 soncursor 커서를 닫는다.
	close soncursor;
end;
```

실습 2:
저장 프로시저 실행을 확인한다.

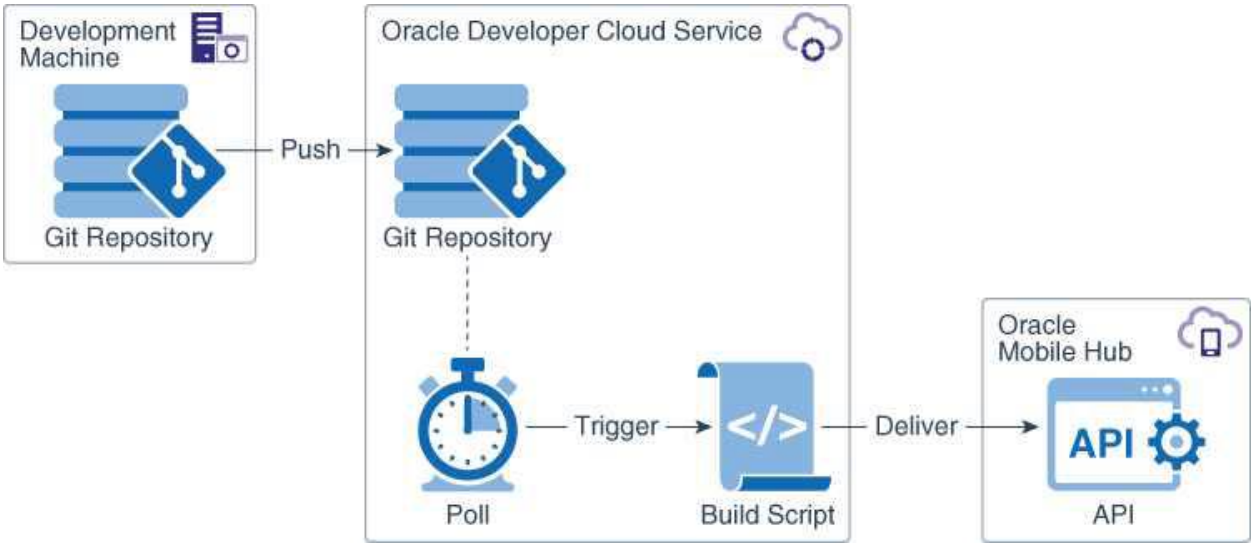
```
-- 화면 출력기능을 활성화한다.
set serveroutput on
-- 저장 프로시저를 실행하여 dept 테이블의 모든 컬럼을 출력한다.
execute cursor_call;
```

4. 트리거

형식:

```
create[or replace] trigger 트리거명
{before|after}
{insert|update|delete} on 테이블명
begin
        조건에 만족할 경우 실행되는 문;
end;
```

트리거(Trigger)는 데이터베이스가 미리 정해 놓은 조건을 만족하거나 어떤 동작이 수행되면 자동으로 수행되는 동작을 말한다.



오라클에서의 트리거는 어떤 이벤트가 발생하면 이벤트로 테이블이 자동으로 변경되게 하려고 사용한다.
트리거는 테이블의 데이터가 변경되면 자동으로 수행되므로 이 기능을 이용하며 여러 가지 작업을 할 수 있다.
트리거는 특정 동작을 이벤트로 하여 그 동작으로 인해서만 실행되는 저장 프로시저이다.
트리거를 생성하려면 create trigger 문 다음에 새롭게 생성하고자 하는 트리거명을 선언한다.
replace 명령어는 같은 이름으로 트리거를 생성할 때 기존 트리거를 삭제하고 새롭게 선언한 내용으로 재생성한다.

트리거는 자동으로 수행되므로 트리거를 사용자가 직접 실행시킬 수는 없다.
트리거는 연관된 작업을 처리하는 데 있어서 여러 번 프로시저를 호출해서 실행할 필요가 없으므로 복잡성을 줄일 수 있다는 장점이 있다.
프로젝트 수행 시 개발자들이 복잡한 업무를 숙지하지 않아도 되기 때문에 프로젝트를 안정적으로 수행할 수 있다는 장점도 있지만, 오류가 발생하면은 수정하기가 어려운 복잡성을 가지므로 주의해서 사용해야 한다.

트리거의 타이밍의 형태는 다음과 같다.

- before 타이밍 : insert, update, delete 문이 실행되기 전에 트리거를 실행한다.
- after 타이밍 : insert, update, delete 문이 실행되고 난 후에 전에 트리거를 실행한다.

1) 메시지 출력

실습 1:
트리거 사용을 위한 테이블을 생성한다.

```
-- emp_son 테이블을 생성한다.
create table emp_son(
        empno number(4) primary key,
        ename varchar2(12),
        job varchar2(21)
);
```

실습 2:
트리거를 생성한다.

```
-- 트리거를 생성한다.
create or replace trigger trg_son
-- insert 문이 실행되고 난 후에 트리거를 실행한다.
after
-- emp_son 테이블에 데이터가 입력할 때 이벤트를 발생시킨다.
insert on emp_son
begin
        dbms_output.put_line('신입사원이 입사했습니다.');
end;
```

실습 3:
트리거를 실행한다.

```
-- 화면 출력기능을 활성화한다.
set serveroutput on
insert into emp_son values(1, '홍길동', '대리');
```