<데이터베이스 실습 복습 정리>

161109 to 161123 (PL/SQL ‘s Prime Time)

오라클에서만 존재하는 SQL 계의 C언어/자바 급 프로그래밍 언어인 PL/SQL에 대해서 배웠다. 초반에는 PL/SQL 언어는 어떤 원리로 돌아가는지에 대해 설명하고, PL/SQL 언어에 대해 기초적인 문장부터 시작을 해 보겠다.

[요약 중점 사항]

- PL/SQL 문장의 원리를 이해하고 어떻게 적용하는지에 대해 공부한다.
- PL/SQL 문장의 기초적인 문장들을 공부할 수 있도록 공부한다.
- 커서를 이용하여 다중 데이터들을 PL/SQL에 적용할 수 있다.

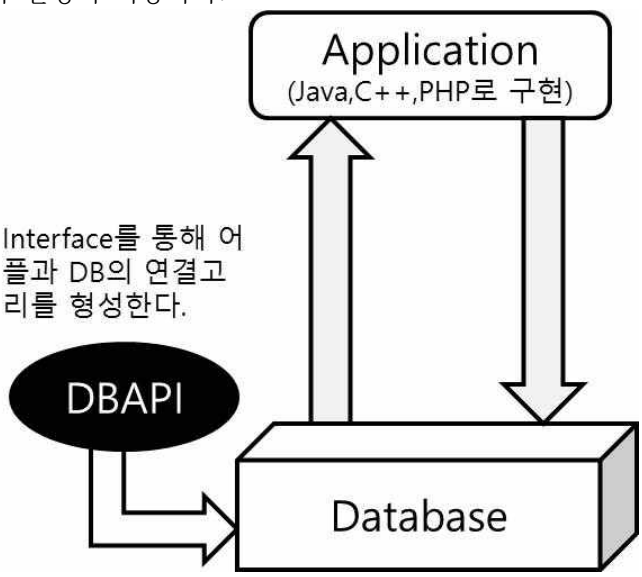
---

5. PL/SQL의 원리 at 161109

우리가 여태껏 작성한 SQL 문장을 이용해서 테이블 내에 있는 컬럼들의 값들을 DML로 통해서 제어를 하거나 뷰, 서브쿼리, join 등을 통해서 조작을 할 때 불편한 부분이 많았다. 그렇지만 SQL과 PL/SQL를 이용하면 효과적으로 DB에 접근하는데 큰 도움을 준다. 그럼 PL/SQL에 대해서 어떤 원리인지에 대해 공부를 해보자.

1) PL/SQL의 뜻??

절차 지향 프로그래밍 언어에는 C언어, 객체 지향 프로그래밍 언어에는 C++, Java가 있을 것이다. 이는 Oracle에서 특별히 제공하는 프로그래밍 언어이다. PL의 약자는 Procedural Language의 줄임말로서 절차적인 기능은 기본적으로 가지는 언어라고 생각하면 된다.(PL 이외의 함수는 Functional Language라고 칭한다.) 우선적으로 우리가 어플리케이션(폰에서 이용하는 그것도 되고, 프로그램이나 소프트웨어도 되겠다.)을 통해서 DB에 접근하기 위해서는 어떤 과정을 필요로 한지에 대해 간략한 그림으로는 다음과 같이 설명이 가능하다.



이처럼 DBAPI를 통해서 Java, C++ 등의 언어로 구현한 프로그램들에 대해 접근이 가능하다. 그래서 데이터베이스 내에 쓰이는 SQL은 표준화를 할 필요가 있다. 표준화를 할수록 Performance는 증가하게 되면서 네트워크를 타게 되면서 Traffic이 많아지게 된다. (물론 상황에 따라 달라질 수도 있다. 항상 그런 것만이 아니라는 뜻이다.)

그래서 SQL에 있는 데이터들을 어플리케이션에게 전달하기 위해서는 PL/SQL를 공부할 필요가 있다. 그래서 PL과 SQL이 결합하면 마치 짜장면과 짬뽕이 결합해 짬짜면으로 고민 해결을 하듯이 이를 통해 DB 관련 작업들을 수월하게 처리할 수 있다.

2) PL/SQL의 런타임 구조

런타임 구조는 개발자가 우선 PL/SQL 블록을 만들게 된다. 그리고 PL/SQL 엔진은 프로시저를 통해서 프로시저문 실행자가 실행할 수 있도록 번역을 해준다. 물론 PL/SQL 내에 있는 SQL 문장들은 PL/SQL엔진이 발견하게 되면 자동적으로 문맥 교환을 실행하고 오라클 서버에서 실행을 해서 결과를 반환하고 PL/SQL엔진에게 주는 원리로 작동이 된다. 물론 결과가 처참하면 사용자가 작성한 PL/SQL은 작동하지 않게 된다. 여기서 데이터베이스에서 처리된 데이터를 PL/SQL에 저장하기 위해서는 변수를 잘 선언해야 된다. 변수 선언에 대해서는 공부하면서 배워보겠다.

3) PL/SQL 기본 구조

Java에서는 변수, 제어문, 반복문, 함수, 예외 처리 등이 있었다. 이처럼 PL/SQL 기본 구조에도 이러한 문장들을 쓸 수 있도록 각 블록 별로 정리되어 있다는 점을 숙지하고 넘어가자.

Declare : 선언부. 변수나 상수들을 선언하는 부분이다. 물론 생략은 가능하지만 필요한 변수들이 있으면 유두리있게 정의하도록 하자.

Begin : 제어문, 반복문, 함수 등의 문장들을 선언하는 실행부이다. 다른 건 다 몰라도 실행부를 생략하게 되면 PL/SQL문이 작동하지 않으니 주의하자.

Exception : 결과들 중에서 예외 발생 시 해결하는 문장들을 작성한다. 물론 생략은 가능하지만 다음에 예외를 배우면서 자세히 알아보도록 하자.

간략히 3가지의 블록에 대해서 정의를 내렸지만, 이에 대해서 어떻게 작성하는지에 대해서는 다음에 예시로 들어보겠고, Block들의 종류는 무엇이 있는지에 대해 공부해보자.

Nested Block : 우리가 IF문도 중첩하고, For 문도 중첩을 할 수 있듯이, 블록 내부에도 블록을 넣을 수 있다. 이를 중첩 블록이라고 칭한다.

Anonymous Block : 우리가 인라인 뷰를 공부했을 때 일회성 뷰를 from 문에서 선언해서 이용하였다. 이처럼 일회성으로 많이 쓸 때 만드는 블록을 일회성 블록이라고 칭한다.

Stored Block : 서브 프로그램, 프로그램 단위라고도 하며 스키마를 구성하는 오브젝트로서 파싱된 뒤 오라클 서버 내부에 저장되거나 툴에 라이브 형태로 되어 있다. 그래서 주기적으로 반복해서 써먹을 때 많이 만드는 블록을 저장된 블록이라고 한다. 이러한 블록들은 Create문을 통해서 만들어도 되는데, 이는 서브 프로그램 부분부터 살펴보도록 하자.

4) PL/SQL문을 통해서 구조를 살피기

간략히 직원 번호가 123번인 직원에 대해 출력하는 문장에 대해 작성해 보겠다. 다음 페이지에서 필자가 작성한 문장을 통해서 공부를 해보자.

```
Set ServerOutput ON; -- 1

Declare
  out_empID employees.employee_ID%type;
  out_name Varchar2(20); -- 2
Begin
  Select employee_id, first_name||' '||last_name into out_empID, out_name -- 3
  from employees
  where employee_id=123;

DBMS_OUTPUT.Put_Line(out_empID||'----'||out_name); -- 4
End;
/ -- 5
```

1 : PL/SQL은 기본적으로 처리된 PL/SQL 문장의 결과를 화면에 출력하지 않는다. 그래서 결과를 출력하고 싶다면 ServerOutput를 ON으로 처리해두고 시작을 하자.

2 : %type는 테이블의 컬럼을 불러와서 이와 같은 데이터 형식으로 저장해주는 역할을 해서 굳이 테이블 내 컬럼의 데이터 형식에 대해 알아볼 필요 없는 좋은 연산자이다. 그리고 변수를 선언할 때 자바나 C언어에선 (데이터형식 이름 -> int num) 이런 식으로 정의했지만 PL/SQL에서는 (데이터이름 데이터형식)으로 한다는 점을 알아두길 바란다.

(예시) Data_Num Number(10)
Data_Name Varchar2(20)

3 : into는 Insert Into라는 문장에서 많이 봤을 것이다. 허나 Select에서 쓰이는 Into 문은 PL/SQL에서만 쓸 수 있다는 점을 알아두도록 하자.

4 : 여기서 DBMS_OUTPUT는 무엇이고 Put_Line은 무엇일까? 우리가 Java를 통해서 문장을 출력하는 문장에 대해서는 System.out.println을 이용했을 것이다. System.out에 있는 패키지 내의 클래스를 이용해 println이란 메소들 실행해 문장을 출력하듯이 DBMS_OUTPUT는 패키지 부분이고, Put_Line은 그 패키지 내에 있는 함수이다. 그래서 문장을 출력할 때는 DBMS_OUTPUT.PUT_LINE을 이용한다는 점을 알고 넘어가자.

5 : 이는 PL/SQL을 실행하기 위해서 마지막 라인에는 항상 /라는 녀석이 같이 다녀야 된다. 이를 써줘야 PL/SQL 내부의 문장이 올바르게 출력이 되지만, 이를 쓴 이유는 PL/SQL 문장이 끝났을 때 구분할 때 쓰게 된다. 나중에 서브프로그램에서 이를 안 써주고 실행하면 까다로워지니 마지막에 쓰면서 습관을 기르도록 하자.

5) PL/SQL 작성 시 기본 규칙 및 권장 사항

모든 프로그래밍에도 규칙이 있듯이 PL/SQL에도 있다. 대개 SQL 규칙과 비스무리 하지만 한 번 더 짚고 넘어가자.

- 문장은 여러 줄 가능. 키워드 분리하면 안 되는 건 알고 있을 것이다.
- 블록의 내용을 쉽게 구분하기 위해 들여쓰기를 잘 써먹자.
- 예약어(쉽게 말해 키워드)는 식별자용으로 못 쓴다. 그러나 Alias로 “Create”이런 식으로는 가능하다.
- 식별자는 테이블 이름, 변수 이름 정하는 거와 비스무리하다.
- 문자열/날짜는 당연히 싱글쿼트(‘ ’)로 구분해야 한다. 물론 null 값은 상수 NULL을 이용하면 된다.
- 주석은 --(단일 행 주석), /* */(복수행 주석)으로 이용하자.
- PL/SQL 블록 내에서 명령어에선 오라클 함수를 이용할 수 있지만, 그룹함수와 DECODE 함수에 대해서는 SQL 문장에 포함되어야만 사용이 가능하다. 다른 경우에 그룹 함수, Decode 함수를 이용하면 에러가 발생해서 예외로 처리되야 하고, SQLCODE 함수와 SQLERRM함수가 별도로 존재하기 때문에 궁금한 사람은 구글링이나 지식인에게 물어보면 되겠다.

---

6. PL/SQL 기본 문장 at 161114

PL/SQL의 원리에 대해서 지난주에 자세히 공부해 봤다. 비록 원리만 공부를 하게 되어 까다롭게 느끼겠지만, 외출 할 때 신발끈을 잘 묶어야 안 넘어지듯이 기본을 다스리는 것도 중요하다고 필자는 느낀다. 그래서 이번 시간에는 PL/SQL 내에서 쓰이는 개념들에 공부를 해보는 기회를 가져보자.

1) PL/SQL 내 SQL 문장 활용

PL/SQL 실행부(Begin 아래부로)에도 SQL 문장을 이용을 하는 경우가 많다. 그래서 이를 올바르게 이용하는 방법에 대해 설명할 필요가 있다. 우선 SQL에서 이용할 수 있는 문장들은 Select~into~은 물론 SQL에서 평소에 쓰는 DML(Update, Delete, Merge 문), TCL(commit, savepoint, rollback 등)에 대해 적용을 할 수 있다. 그렇지만 DML, TCL를 이용하는데 있어서 다음과 같은 주의사항을 숙지해야 한다.

- END 키워드를 commit로 생각하지 말자. 그니깐 END는 트렌잭션의 끝이 아니다. 오로지 PL/SQL의 끝이다.
- PL/SQL에서 DDL(Create, Alter, Drop 문 같은 거), DCL(Grant, Revoke 문)를 쓴다고 생각하는 점은 쉽게 해서 수업 시간인데 선생님 몰래 일어나서 춤추는 행위로 생각하면 된다. 왜냐면 DDL, DCL은 동적 SQL(Dynamic SQL)문으로서 PL/SQL 내부에 쓰지 말고 밖에서 써줘야 한다.

<참고> 동적 SQL은 런타임에 문자열을 작성되며 파라미터의 위치 표시자를 포함시켜야 이용이 가능하다. 그렇지만 현재로서 배우는 PL/SQL 내부에서는 데이터의 값들은 1가지로만 출력이 되기 때문에 차후에 커서, Subprogram 개념을 들어가게 되면 자세히 공부해보고 우선 PL/SQL의 기본적인 이용방법을 공부하는 점을 목표로 삼고 공부하자.

1-1) PL/SQL에서 Select 명령으로 조회하기

공부를 시작하기에 앞서서 1페이지의 4번 문장에서 구조를 간략히 살펴봤다. 위의 문장을 다시 공부한다는 개념으로 아래와 같은 문장을 살펴보도록 하자.

```
Set Serveroutput ON;
Declare
  p_name Varchar2(30);
  p_salary employees.salary%type; -- 3
Begin
  Select first_name||' '||last_name, salary into p_name, p_salary -- 1
```

```
  from employees
  where employee_id=123; -- 2
  DBMS_OUTPUT.PUT_LINE(p_name||'        -->
'||p_salary);
End;
/
```

1 : 그렇다. 방금 전에 Select ~ into ~에 대해 간략히 설명했을 것이다. 이는 오로지 PL/SQL 내부에서만 쓸 수 있다는 점은 당연한 이야기이고, 사용자가 Select에서 알아낼 컬럼들에 대해서 데이터 타입과 into 부에서 변수의 데이터 타입과 똑같은 건 당연한 이야기이고 개수도 똑같아야 한다. 마치 Insert Into에서 테이블의 내용을 통하여 데이터 형식을 맞춰서 값을 추가하듯이...

2 : where 문은 굳이 왜 썼을까? 이러한 문장을 작성하는 경우에는 어쩔 수 없이 1건의 데이터만 조회되어야 할 수 밖에 없기 때문이다. 여러 건을 대입하고 싶다면, Cursor에서 뵙겠습니다...
추가적으로 만일 Where employee_id=&id로 작성을 하게 되면 어떻게 될까? undefined 블라블라 해서 지난번에 설명한 것이 있는데 이는 7-중간 차시 요약정리를 살펴보면 원리를 이해하겠다만, 자바에서 Scanner와 같은 역할을 보면 되는데 이름은 변수 이름 설정의 제약 조건만 안 벗어나면 상관없지만, 이를 작성하게 되면 사용자에게 입력을 받을 수 있는 유용한 기능이니 참조하자.

3 : %type은 이따가 참조 변수에서 제대로 이야기 하겠지만 테이블 내의 컬럼과 데이터 타입을 같게 선언을 하도록 이를 참조하는 역할을 한다고 알면 되겠다.

(참고) PL/SQL에서도 대/소문자 구별 제대로 안 해도 호랑이가 안 잡아먹으니깐 이에 대해서는 걱정하지 않아도 된다. 다만 리터럴(문자열) 내부에서는 잘못하면 호랑이가 잡아먹을 수도 있으니 그 때는 상황에 맞춰서 구별하도록 하자.

1-2) PL/SQL에서 DML문장 이용하기
Insert, Delete, Update, Merge 문들을 PL/SQL에서 이용할 수 있다는 점을 알 수 있다. DML에 관련되어 숙지를 못한 사람들은 필자가 정리 해둔 5-6차시 퀴즈 복습 파일을 참조하길 바라면서 아래와 같은 테이블과 시퀀스를 만들었으니 이를 이용해서 어떠한 원리로 돌아가는지 알아보자.

```
Create Table Ramen
(ID number, Name Varchar2(20))
// 아시다 싶다면 필자는 라면을 잘 먹는 관계로 이번에는 라면에 대한 값을 추가시켜보겠다.
Create Sequence Ramen_SEQ;
// 이대로 작성하면 자동적으로 start with는 1, increment by 1, nocache, nocycle 설정이 되는 건 다들 알 것이라 믿는다. 최댓값은 물론 시퀀스가 추가될 수 있는 마지막 값으로 생각하면 된다. 그 수는 어마어마하고 쓸 일이 없으니 걍 패스하자.

<Insert 문>
Set Serveroutput ON;
Declare
  v_Name Varchar2(20) := '&name'; -- 1
Begin
  Insert Into Ramen
  Values(Ramen_SEQ.nextVal, v_Name); -- 2
End;
/
```

1 : 1번 문장에서 &name을 왜 싱글 쿼트 안에 박았을까? 만일 싱글 쿼트 없이 &name으로 작성하면 값들을 입력할 때 '진라면', '너구리'... 이런 식으로 써야 하는 뭐 같은 경우가 발생하게 된다. 그래서 사용자들의 편의를 제공하기 위해서 싱글 쿼트 안에 작성해주자.
또한 데이터베이스에서 :=를 이용해줘야 대입 연산자를 이용할 수 있다. 이는 연산자를 공부할 때 자세히 공부해보자.

2 : Sequence는 PL/SQL 내부에서 작성하게 된 시발점이 바로 11g이다. 우리가 쓰고 있는 11g에서는 무난하게 쓸 수 있지만 이전 버전을 이용하게 되면 dual 테이블을 이용해 귀찮은 방법 밖에 없었다.

```
<Update 문>
Set Serveroutput ON;
Declare
  v_Name Varchar2(20) := '&name';
Begin
  Update ramen
  Set v_Name
  where no=3;
End;
/
commit;
```

Update 문은 Insert 문 평소에 이용한 것처럼 쓰면 된다. 그래서 사용자에게 입력을 받아서 3번 라면의 이름을 짜파게티로 입력하면 바꿀 수 있으니 참조하자. 또한 이에 대해 적용하기 위해서 마지막에 commit을 꼭 써두도록 하자.

```
<Delete 문>
Set Serveroutput ON;
Declare
  v_ID Number := '&name';
Begin
  Delete from ramen
  where ID=v_ID;
End;
/
commit;
```

Delete 문도 또한 같은 방법으로 이용하면 된다. 사용자에게 번호를 입력받아서 그 번호에 해당되는 튜플을 삭제하는 문장으로 생각하면 된다. 또한 이를 적용하기 위해서 뒤에 commit를 쓰도록 하자.

```
<Merge 문>
```

Merge 문은 솔직히 어떻게 이용하는지에 대해서 거의 까먹었을 것이다. 그렇지만 방금 라면 테이블와 같은 구조로 Ramen1, Ramen2 테이블로서 Ramen1에 병합을 시키되 ID가 같다면 Ramen2의 정보를 저장시키도록 하는 문장을 작성하겠다.

```
Begin
  Merge into Ramen1 rm1
    Using Ramen2 rm2
    ON(rm1.ID=rm2.ID)
    When Matched Then
      Update Set rm1.name=rm2.name
    When Not Matched Then
      Insert Values(rm2.id, rm2.name);
End;
/
```

이처럼 작성을 하면 라면 테이블의 ID로서 기준을 잡아서 같은 값인 경우에 Ramen1의 Name을 Ramen2의 Name으로 적용시키고, Ramen2의 ID가 없는 경우에는 Ramen2의 정보를 Ramen1에 추가를 시키는 개념으로 생각하면 된다. 여기서는 딱히 입력 연산자(&)를 쓰지 않으니 참고만 하자.

2) PL/SQL에서 렉시칼

면도칼은 알겠지만, 렉시칼이 뭔지 모를 것이다. lexical은 사전적인, 사전적으로 라는 뜻으로서 PL/SQL 내부에서 사용되는 문자 집합을 뜻한다. 문자 집합은 식별자, 구분자, 리터럴, 주석 등으로 구분이 된다.

2-1) 식별자

식별자는 테이블, 뷰, 시퀀스, 제약조건 등에 이름을 부여하는 개념으로 생각하면 된다. 식별자는 객체 이름, 변수 이름 등등 모두가 속한다. 식별자에 큰 따옴표를 쓰는 경우가 있지만, 이는 대/소문자 구분, 공백문자 포함되거나, 예약어(키워드)를 시용하는 경우에 따로 쓰면 되는데 이는 웬만하면 쓰지 않는 게 약이다.

2-2) 구분자

구분자는 특별한 의미를 지닌 기호이지만, 흔히 쓰는 개념들과 비스무리하기 때문에 새롭게 배우게 되는 구분자들에 대해서 참조해 보도록 하겠다.
@ : 원격 엑세스 표시자. 이는 타인의 스키마를 참조할 때 쓰는 것이지만... 나중에 심심하면 구글링해보도록 하자.
;(세미콜론) : 늘 그래왔듯이 명령문 종료자이다. 이는 END 문이든 어느 문장에 명령이 완료된 후에 꼭 써줘야 된다. 평소에 SQL에서 안 써도 됐다고 생각하지만 PL/SQL에서는 각 문단 별로 구분이 필요로 하니 써줘야 된다.
:= : 할당 연산자. 이는 방금 전에 = 와 :=와 엄연히 다르다. 데이터베이스에서 =는 등호이고, :=는 대입연산자이다. 자바나 C언어에선 =가 대입 연산자이고, ==가 등호로 통했지만, 여기서는 :=를 이용해서 값들에 대해 적용을 시킨다.
/*, */, -- : 주석문.

2-3) 리터럴

변수에 할당되는 모든 값들은 리터럴이다. 그니깐 여기서는 모든 문자, 숫자, 부울, 날짜 값들이 리터럴이라는 그 뜻이다.

2-4) 주석

주석은 늘 그래왔듯이 -- 를 이용하면 단일 행 주석, /* */를 이용하면 다중 행 주석이니 참조하자. 데이터베이스를 작성하면서 까다로운 부분이나 설명하기 좋은 문장을 작성하기 위해서 주석을 생활화하자.

3) 중첩 PL/SQL 블록 작성하기

PL/SQL에서도 문장을 중첩해서 작성을 할 수 있다. 다음과 같이 작성되었다고 가정하자.

```
Set ServerOutput ON;
Declare
   v_value1 varchar2(10) := 'Seoul'; -- 1
Begin
   Declare
     v_value2 varchar2(10) := 'Daejeon'; -- 2
   Begin
     DBMS_OUTPUT.PUT_LINE(v_value1||' '||v_value2); -- 4
    End;
   DBMS_OUTPUT.PUT_LINE(v_value1); -- 3
   End;
   /
```

1 : 1번에서 정의한 변수들에 대해서는 밑줄 친 Begin 부분부터 계속 이용할 수 있다. 물론 마지막에도 쓸 수 있으니 참조하자.
2 : 2번에서 정의한 변수들에 대해서는 이태리체에 점선으로 그어진 부분에서만 쓸 수 있으니 밖에서 쓸 생각을 하지 않는 것이 좋다. 쉽게 이야기해서 3번 문장 대신에 4번 문장을 쓰게 되면 뭐된다고 생각하면 된다.

3) PL/SQL에서 이용되는 변수

PL/SQL에서 쓰이는 연산자(** 라고 있는데 이는 우리가 수학에서 항상 붙어 다니는 제곱 연산자로 보면 된다.)에 대한 설명은 SQL와 다를 바가 전혀 없기 때문에 굳이 작성하지 않겠다.
그렇지만 PL/SQL에서 변수는 왜 이용하는 것일까? 변수는 데이터를 임시로 저장해주는 역할을 하고, 저장된 값들의 조작하고 반복해서 재활용을 하기 위해 쓴다. 마치 자바나 C언어에서 쓰는 변수와 같은 개념인데 변수에 대한 이름은 테이블의 이름과 같이 결정하면 되니깐 굳이 설명은 따로 안하겠다. 그려면 변수는 각 PL/SQL 문단 별로 어떻게 이용을 하는지에 대해 서술해 보겠다.
-> 선언부 : := 나 Default로 특정 값으로 초기화를 할 수 있다. 변수는 참조되기 전에 선언되어야만 된다. 즉 자바로 쉽게 이야기하면 변수를 선언하지 않고 변수를 써버리면 컴파일 오류가 나듯이, 이를 선언하지 않고 써버리면 두 눈으로 봐도 아이러니할 것이다.
-> 실행부 : 값을 할당(혹은 대입)할 수 있다. 또한 차후에 배울 Subprogram의 개념에서 매개 변수로 전달이 된다.
-> 예외부 : 물론 선언부와 실행부를 통해서 이용한 변수들에 대해서도 예외부에 작성할 수 있다. 이러한 부분은 예외부에서 살펴보면 되겠다.

3-1) 변수 작동 원리

PL/SQL 변수는 기억 장소로서 메모리에 확보된다. 이 변수의 실행 가능한 범위는 실행이 종료될 때까지이다. 자세히 설명하면 문장이 실행되면 메모리에 차지를 하다가 차후에 종료가 되면 메모리를 반환한다는 뜻이다. 변수를 선언하게 되면 방금 중첩 PL/SQL 문에서 알 수 있듯이 해당 블록 내의 변수가 우선 참조되고, 그 블록을 벗어나게 되면 그림의 떡이 되어버리는 개념이 되는 지역 변수와 같은 원리로 생각하자. 그렇지만 모든 블록에서 사용할 전역변수를 선언하기 위해서는 패키지를 이용하면 되는데 차후에 배워보도록 하자.

3-2) 변수의 종류

변수의 종류는 PL/SQL 변수와 비 PL/SQL 변수로 나뉘게 된다. 우리가 대표적으로 알아볼 PL/SQL 변수는 스칼라 변수, 참조 변수, 복합 변수, 바인드 변수에 대해서 알아 볼 것이다.

3-3) 단순 변수 - Scalar 변수

우리가 물리 시간에 스칼라와 벡터에 대해 잠깐 공부

했을 것이다. 스칼라는 물체의 양에 대해서만 측정하고(속력, 길이, 질량 등등), 벡터는 물체의 양과 방향에 대해 둘 다 측정하는 개념(속도, 변위 등등)이다. 이처럼 Scalar 변수는 물리 시간에 배운 Scalar 개념처럼 단일 값을 가지는 변수의 데이터형을 직접 지정하는 변수이다.

스칼라 변수의 구조는 다음과 같이 볼 수 있다.
[변수이름(1)] [Constant(2)] datatype(3) [NOT NULL](4)
[:= / DEFAULT expr(5)]
(1) 지난번 요약정리에도 그랬듯이 데이터베이스에서는 변수 이름부터 먼저 정의가 된다고 설명했을 것이다. 변수 이름은 의미 있게 지어야 문장을 작성하는데 있어서 편의를 제공한다.

(2) Constant는 상수이다. 우리가 예를 들어 π(3.1415....), e(2.7블라블라), 중력 가속도(9.8블라블라 m/s2)처럼 정해진 값에 대해 이용을 하는 경우가 있다. 이럴 때 Constant를 이용해서 다음과 같이 정의를 할 수 있다. *Constraint와는 다르다!!! 주의하자!!!*
math_PI Constant Number := 3.1415;

(4) 물론 Not Null에 대해서 제약 조건을 추가할 수 있다. 이 변수는 값을 항상 값을 가질 수밖에 없을 때 제약을 주는 것이다.

(5) 변수에 부여할 기본 값이다. 물론 수식, 함수, 다른 변수로도 가능하다.

(3) 자, 이제 Scalar 변수의 데이터 타입이다. Create Table에서 공부했겠지만, 다시 공부한다는 의미해서 작성을 하겠다.

Char(길이) : 고정 길이의 문자열을 저장한다. 길이가 10인 경우에 문자열 'Dok2'를 써도 길이는 10으로 나오게 된다.
Varchar2(길이) : 가변 길이의 문자열을 저장한다. 길이가 10인 경우에 문자열 'Dok2'를 쓰면 길이는 4로 나오게 되어 데이터 용량 절약에 도움을 준다.
Number(자리 수, 소수점 이하 수) : 쉽게 말해 숫자와 실수, 모두를 저장하는 개념이다. 최대 자리 수는 38자리까지 가능하다.
Date : 그렇다. 날짜와 시간을 출력한다. 날짜의 범위는 단군 할아버지 시절에 대해서도 저장할 수 있고 9999년까지 저장이 가능하다.

(참고) Timestamp와 Boolean 타입
Timestamp : Date의 확장판이다. 단위 초(0.000001초)에 대해 확장으로 구현을 할 수 있다. 소수의 자릿수도 결정이 가능한데 기본 값은 6이다.
Boolean : 자바에서 쓰이는 Boolean과 똑같지만, 여기서는 Null도 포함이 된다. 왜냐면 그 값이 사실인지 그짓인지에 대해서 판별을 못 하면 Null로 지정을 해두기 때문이다.

여기까지는 데이터 타입의 기본적인 것들이다. 그렇지만, 아래 개념들에 대해서는 복합변수들에 대해 Index를 결정하는데 있어서 중요하니 개념들에 대해 간략히 숙지를 해두자.

Binary_Integer : 이 타입은 음수에서 양수까지의 정수를 저장하는 타입이다. 정수만 저장이 가능하다.

PLS_Integer : 이 타입 또한 마찬가지인데 이 녀석이 Binary_Integer보다 용량 절약을 하고, 속도도 빠르다. 그런데 같은 개념을 또 쓰는지에 대해서는 오라클의 사정이니 그런가 보다하고 넘어가자.

<참고1> TimeStamp With Time Zone / TimeStamp With Local Time Zone : TimeStamp의 확장판. 이 녀석은 UTC 시간과 로컬 시간의 차이를 이용해서 TimeStamp와 같은 개념을 쓰는 건데 이는 우리가 한국표준연구원에 입사할 생각이 없다면 그냥 넘어가자.
<참고2> Interval Year To Month / Interval Day To Second
Interval Year(precision) to Month는 연도와 월의 간격을 조정하는데 이용하는데 가령 년도가 2016년 11월이라고 치고, precision을 이용하면 16년 11월 이런 식으로 출력을 하게 된다.

물론 Interval Day(precision1) to Second(precision2)도 위와 같은 개념으로 이용하여 일/시/분/초 간격을 저장하거나 조작하는데 이용하게 된다. 자리 수는 변수나 상수로는 불가능하고 0~9 사이의 정수 리터럴을 이용한다.

3-4) Reference(참조) 변수
변수들 중에도 분명 참조를 역할을 하는 변수가 존재하기 나름이다. 예를 들어 우리가 데이터 타입을 정의해야 되는데 테이블 내에 있는 컬럼의 데이터 타입이 무엇인지에 대해 모를 경우에 %라는 연산자를 이용해서 구해올 수가 있다. 다음과 같이 2가지의 경우를 살펴보도록 하자.
v_first_name employees.first_name%type; // 1
v_row employees%Rowtype; // 2
1 : 그냥 type은 employees 테이블의 first_name 컬럼과 같은 데이터형을 불러와서 v_first_name도 이와 같이 선언을 할 수 있게끔 해준다.
2 : Rowtype은 employees 테이블에 있는 모든 컬럼들에 대한 정보를 추출하게끔 해준다. employee_id부터 해서 department_id까지 모든 정보를 v_row에게 짱 박히게끔 해준다고 생각하면 된다.

%type을 이용해서 작성은 앞에서도 많이 해봤으니 %rowtype를 이용해서 직원 정보를 출력하는 문장을 작성해보자.

```
Set ServerOutput On;
Declare
  v_emp Employees%Rowtype;
Begin
  Select * into v_emp
  from employees
  where employee_id=123;
DBMS_OUTPUT.PUT_LINE(v_emp.employee_id||'-'
||v_emp.last_name||'-'||v_emp.salary);
End;
/
```

이처럼 작성하게 되면 v_emp에 employees 내부 컬럼 정보들이 들어오게 되면서 차후에 PUT_LINE에서 v_emp에서 각 컬럼들 요소를 불러옴으로서 문장을 출력하게끔 해준다. %Type를 쓰기 귀찮은 사람들은 %Rowtype을 통해서 테이블에 있는 컬럼 요소들을 불러와서 출력을 해 보는 것도 나쁘지 않을까 생각한다.

---

7. 복합변수/비 PL/SQL 변수와 PL/SQL 제어문 at 161116

복합변수에 대해서는 본래 6번에서 같이 설명했어야 되는데 요약정리를 구분해서 하는 이유는 수업 시간에 배웠던 개념들을 기준으로 재정의하는 것이니 이에 대해서는 양해를 구한다. 또한 PL/SQL문에서도 C언어나 자바처럼 IF문, for문, while문 등등이 있다는 놀라운 사실을 곧바로 알게 될 것이다. 우리가 처음에 시퀀스를 공부했을 당시에 기본 프로그래밍 언어를 공부했을 때 증감 연산자에 대해 생각해봤을 것이다. 이처럼 PL/SQL도 기본 프로그래밍의 개념들에 대해서 다시 생각해 보면서 공부를 하게 되면 큰 도움이 되니 이를 연계해서 금상첨화의 효과를 거두길 바란다.

3-5) 복합 변수
우리가 C언어에서 Struct 개념을 공부했을 때 우리가 흔히 쓰는 Employees 테이블 내에서 직원들의 정보에 대해서는 각양각색으로 저장이 되어 있지만, 그 중에서 직원 번호와 직원 이름(성/명) 들 해서 3개만 뽑아올 수 있도록 하는 View 원리와 같은 원리로 복합변수에 대해 공부를 해두면 테이블 내에 있는 컬럼들 중 일부만 뽑아오면 되기 때문에 PL/SQL 내에서 필요로 한 변수들에 대해서만 정의를 할 수 있으니 알아둘 필요가 있다.

1) 복합 변수의 종류
복합 변수에는 Record Type 변수, Table Type 변수 2가지로 나눌 수 있는데, Record Type은 방금 View와 같은 원리로 써먹으면 되고, Table Type은 인덱스를 정수형 변수나 문자열 변수를 통해서 배열과 같은 원리로 쓰게 된다.

2) Record Type 변수
Record Type 변수 형에 대해 공부를 해보자.
구조는 다음과 같이 정의가 가능하다.
Type [변수형_이름] is Record // 1
(각종 변수들) // 2

1 : Declare(선언부)에서 선언을 해야 제대로 이용을 할 수 있는데, Type으로 작성한 것으로 봐서 이는 또 하나의 변수의 형태로 생각을 할 수 있다. 가령 Timestamp, Inerval Month to Year 등등의 확장형 변수들도 이러한 원리를 통해서 만들었다고 생각해 볼 수가 있다. 이를 선언하고 난 뒤에는 2번처럼 변수들의 목록을 적어둬야 한다. 마치 Create Table 문과 비스듬하다고 생각을 할 수 있다.
2 : 각종 변수들은 방금 변수를 선언하듯이 하면 된다. 늘 그랬듯이 순서는 변수 이름, 변수의 종류 이처럼 작성을 해주고 Create Table 문과 다른 점은 변수를 다 선언하고 난 뒤에 세미콜론을 박아두도록 하자. 왜냐... PL/SQL 형님이 하라는데로 하자.

이를 통해서 직원 번호와 직원 이름들을 사례로 다음과 만들어 보겠다.

```
Type emp_simp_info is RECORD
(vempid employees.employee_id%type,
 vf_name employees.first_name%type,
 vl_name employees.last_name%type);
```

이처럼 선언을 해주고 난 뒤에 그냥 쓰면 만들나 마나이다. 그래서 이를 통해서 다음과 같이 변수를 선언해줘야 다음과 같이 이용이 가능하다.

```
Declare
Type emp_simp_info is RECORD
(vempid employees.employee_id%type,
 vf_name employees.first_name%type,
 vl_name employees.last_name%type);
v_empinfo EMP_SIMP_INFO;
Begin
  Select employee_id, first_name, last_name into v_empinfo
  from employees
  where employee_id=123;

DBMS_OUTPUT.PUT_LINE(v_empinfo.vempid||'-'||v_empinfo.vl_name);
End;
/
```

밑줄 친 부분에 왜 %type을 안 써먹을까? EMP_SIMP_INFO가 쉽게 말해서 이미 변수 형이 되어버려서 굳이 쓰지 않는다. 그러나 방금 전에 살펴본 %Rowtype과 매우 유사하게 이용할 수 있는데 차이점이 무엇인지 두 눈으로 똑똑히 볼 수 있다. 차이점은 대략 다음과 같다.
-> Record Type은 필요한 컬럼들로만 구성이 가능하고, %Rowtype은 테이블 내의 모든 컬럼을 이용해야 하니...
-> Record Type은 필드(변수)를 정의하고 정의한 이름으로만 쓰면 되는데 %Rowtype은 테이블 내에서 정해진 컬럼의 이름으로 써야 한다.
-> Record Type은 static 한 정적 변수이지만, %Rowtype은 다이나믹한 동적 변수이다. 그래서 Alter Table로 컬럼의 형태를 변경해도 %Rowtype은 최신 테이블 내 컬럼의 형태로 써먹을 수 있는데, Record Type은 변경하기 전에 썼던 데이터 형식으로 밖에 못 쓴다.

3) Table Type 변수(컬렉션)
Table Type의 종류는 3가지로도 나뉘는데 대표적인 연관 배열에 대해서만 알아두면 되겠다. Table Type의 데이터 형태는 다음과 같이 Key, 컬럼 데이터 형식 2가지로 쓰인다. 마치 배열이나 Map 구조와 비스무리하게 생겼는데 Key에 대해서는 2가지 방법으로 정의를 할 수 있다.

- 숫자형 변수
늘 그래왔듯이 Number를 쓴다고 생각하면 처리 속도를 보장을 못하니 방금 전에 공부해본 PLS_INTEGER, BINARY_INTEGER 2가지를 이용해보자. 속도와 용량 면에서는 PLS_INTEGER가 유용하지만 대표적으로는 BINARY_INTEGER를 이용한다.

- 문자형 변수
문자로도 물론 정의할 수 있다. Varchar2를 많이 쓰게 되는데 Char로 쓰면 글자 수를 결정하는데 있어서 에바이기 때문이다.

숫자형 변수와 문자형 변수를 이용하면 다음 페이지에 있는 스폰지밥 등장 캐릭터와 우리가 흔히 볼 수 있는 동물들을 통해 연도를 정하는 조상들의 지혜를 엿볼 수 있는 12간지를 예로 들 수 있다.

Key	Value
1	스폰지밥
2	뚱이
3	징징이
4	집게사장
5	핑핑이

Key	Value
자	쥐
축	소
인	호랑이
묘	토끼
진	용

이처럼 데이터에 인덱스나 특정 문을 넣어주는 Table Type 변수는 어떻게 이용할까? 아래의 문장을 살펴보도록 하자.

```
Type tablo_stu_name is TABLE OF
Student.name%type // 1
Index By BINARY_INTEGER; // 2
v_name TABLO_STU_NAME; // 3
```

1 : Table type은 애석하게도 하나의 데이터만 허용이 된다. 왜냐면 동일한 유형의 데이터들을 하나의 연속적인 메모리 공간을 확보하기 위해 쓰기 때문이다. 그래서 Table 뒤에 of가 붙는구나 라고 생각하면 된다.
2 : 인덱스에 대해서는 방금 숫자형 변수/문자형 변수에서 설명한 부분을 한 번 더 읽어보자. 그리고 Table type은 인덱스를 설정하고 난 뒤에 세미콜론을 박아야 한다. 아까 설명한 것처럼 PL/SQL 누님이 그렇다면 그렇다고 생각하자.
3 : 방금 공부한 Record Type과 같이 Table Type은 일종의 변수 형태로 되어 변수를 만들 때처럼 이용하면 유용하게 이용할 수 있다.

물론 정의하는 방법에 대해 배웠으니, 어떻게 쓰이는지 사례를 봐야 하는 게 인지상정이다. 위 문장을 토대로 활용해 보겠다.

```
Declare
  t_name student.name%type; // *
  Type tablo_stu_name is Table Of
  student.name%type
  Index By Binary_Integer;
  v_name TABLO_STU_NAME;
Begin
  Select name into t_name // *
  from student
  where stu_id=808;
  v_name(0) := t_name; // *
  DBMS_OUTPUT.PUT_LINE(v_name(0));
End;
/
```

// *에서 살펴볼 수 있듯이 t_name으로 우선 학생의 정보를 받아들이고 난 뒤에 v_name 안에 넣어줘야 된다. 왜냐면 v_name은 포인터처럼 배열로 가리키는 원리로 쓰지 않기 때문에 t_name으로 정보를 얻어와서 v_name에 0번째 요소에 저장을 할 수밖에 없다.

3-6) 바인드 변수

바인드 변수는 비 PL/SQL 변수이다. 비는 태양을 피하는 그 분이 아니고, 아닐 비(非)이다. 이는 우리가 흔히 테이블 내 정보들을 살펴볼 수 있는 desc와 같이 호스트 환경에서 생성하고 활용할 수 있다. 그래서 이를 호스트 변수라고도 한다.
Declare로 이용해서 정의하면 안 된다. Variable 키워드로 정의를 해야 한다. 물론 PL/SQL문 내에서도 응용이 가능하다. 허나 사용자에게 입력 받아 이용하는 치환 변수와 구분을 할 필요가 있다. 바인드 변수는 어떻게 이용할까? 우측 상단부의 문장을 살펴보도록 하자.

```
Variable v_bind Varchar2(25);
Begin
  Select last_name into :v_bind
  from employees
  where employee_id=808;
End;
/
Print v_bind;
```

우리가 중요한 시점을 봐야 하는 점은 밑줄 친 부분이다. 물론 바인드 변수는 변수를 선언하는 방법과 똑같이 이용하면 되지만, into 문에서는 : (콜롬. 흔한 말로 ~는, 땡땡)으로 해줘야 v_bind에 직원 번호가 808인 녀석이 v_bind에 저장이 된다. PL/SQL에서 벗어나면 Print라는 호스트 함수를 통해서 v_bind에 대해 어떤 값이 저장되어 있는지 출력을 할 수 있다.

---

1) PL/SQL 제어문/반복문 활용하기

그렇다. 제어문과 반복문은 우리가 생각한 if문, for문, while문처럼 이용이 가능하다. PL/SQL은 오라클에서만 된다고 설명한 적이 있었다. 그래서 오라클이 Java의 영향을 받아서 PL/SQL를 만들었다고 생각하면 되겠다. 솔까놓고 필자도 Java와 PL/SQL의 관계에 대해서는 자알... 모르겠다. 아무튼 이를 어떻게 이용하는지에 대해 제어문부터 알고 넘어가자.

1-1) 제어문

제어문에는 자바나 C언어에서 많이 봤던 if문, 중간고사 때 커트라인을 통해 계급을 뽑아내는 Case문 2가지로 구성이 된다. if문은 괄호로 묶는 대신 Then이라는 키워드로 쓰면 되겠고, Case 문은 우리가 늘 그래왔듯이... Case When 블라블라 Then 이처럼 써먹으면 된다. 우선 if문을 먼저 살펴보고, Case문을 예제 문제로 살펴보면서 공부해보자.
if문> 우선 if문에 대해서는 다양하게 쓸 수 있다.
이용하는 2가지 방법으로 살펴보도록 하자.

IF (조건) then
  실행문;
ElsIF (조건) then
  실행문;
Else
  실행문;
END IF;
	if(조건){
	}
	else if(조건){
	}
	else{
	}

IF (조건) then
  실행문;
End IF;
	if(조건){
	}

좌측은 PL/SQL에서 쓰는 if문, 우측은 C나 자바에서 쓰는 if문이다. 그렇지만 차이점이 있다. 우리가 자바에서 else if를 쓸 때 PL/SQL에서는 ELSIF로 한다. Else일 때 빼고 모든 조건 뒤에는 then이라는 키워드가 붙게 된다. 마치 Case When 다음에 Then을 쓴 거와 같다고 보면 된다. 또한 조건문이 끝나면 언제나 그랬듯이 END IF를 꼭 써줘야 된다. 그래야 정상적으로 반복문이 돌아가게 된다.
이를 이용해서 다음과 같은 문장을 살펴볼 수 있겠다.

```
Declare
  v_salary employees.salary%type;
  v_last employees.last_name%type;
  v_level Varchar2(20);
Begin
  Select last_name, salary into v_last, v_salary
  from employees
  where employee_id=123;
```

```
  if(v_salary>10000) then
    v_level:='하이퍼사원';
  elsif(v_salary between 8000 and 10000) then
    v_level:='된사원';
  elsif(v_salary between 6000 and 7999) then
    v_level:='말년사원';
  else
    v_level:='신입사원';
  end if;

  DBMS_OUTPUT.PUT_LINE(v_last||' '||v_level);
End;
/
```

v_level에는 123번 직원의 봉급 별로 계급을 나누는 구현을 IF문으로 해봤다. Case~ When~ 도 이처럼 작성할 수 있으니 Case~ When~ 문을 참조해보자.

Case문> Case~ When~ 문은 3-4차시 퀴즈 복습 요약정리에 중간에 있을 것이다. 이의 구조에 대해서 복습을 하기 위해 아래의 문장을 살펴보자.

```
Set serveroutput on;
Declare
  v_depID employees.department_id%type;
  v_last employees.last_name%type;
  v_depname Varchar2(20);
Begin
  Select last_name, department_id into v_last, v_depID
  from employees
  where employee_id=123;

  v_depname :=
    Case When v_depID=10 then '영업부'
         When v_depID=20 then '인사부'
         When v_depID=30 then '재무부'
         Else '비비디바비디부' End;
  DBMS_OUTPUT.PUT_LINE(v_last||' '||v_depName);
End;
/
```

우리가 늘 그래왔듯이 Case When (조건) then 해당단어, 이런 방법으로 하는 방법이 있고, Case 확인_변수 When 확인_변수_해당값 then 해당단어, 이런 방식으로 하는 방법이 있다. 전자는 해당 값이 불규칙적일 때 하고, 후자는 해당 값이 딱딱 정해져 있으면 하도록 하자.

1-2) 반복문

반복문은 자바나 C언어에서 공부한 While문, for문, do~while 문 3개로 나뉘겠지만, PL/SQL에서도 이처럼 나뉘어서 개념을 숙지하면 되니깐 어렵지 않다.

Loop문>

Loop 문은 어떻게 쓸까? 이는 Do~While문처럼 이용하면 된다. While 문이 끝나는 경우에 조건을 확인해서 반복을 하는 원리인데 Loop문도 마지막에 조건을 확인해서 조건을 만족하게 되면 반복문을 빠져 나가게 된다. 어떤 구조인지 살펴보자.

```
Set serveroutput on;
Declare
  v_num Number := 0;
Begin
  Loop
    DBMS_OUTPUT.PUT_LINE(v_num);
    v_num := v_num+2; // 1
    exit when v_num>10; // 2
  End Loop;
End;
/
```

Loop 문을 돌기 위해서는 밑줄 친 부분이 빠지면 절대 안 되겠다. 1번은 증감 연산자로서, 이가 없으면 무한 루프를 돌 수 밖이다. 그리고 2번 문장이 없는 경우에는 범위를 알 수 없게 된다. 그래서 이 문장을 돌게 되면 결과는 0, 2, 4, 6, ... , 10이 나오게 된다.

While 문>

방금 배운 Loop 문과 크게 다를 바가 없지만, 검사를 하는 시점이 달라진다. 그래서 방금 작성한 범위를 그대로 써보겠다. 아래 문장을 참조하자.

```
Declare
  v_num Number := 0;
Begin
  While v_num<10 Loop
    DBMS_OUTPUT.PUT_LINE(v_num);
    v_num := v_num+2;
  End Loop;
End;
/
```

이처럼 작성할 시에 늘 그래왔듯이, 0, 2, ..., 8 까지만 나오게 된다. 왜 그럴까? 앞에서 참조를 하기 때문인데, 만일 loop문처럼 exit when 문을 썼었다면 10까지 나왔지만, while 문 조건이 벗어나게 되면 문장 자체를 실행하지 않게 되기 때문이다. 일반적으로 우리가 알고 있는 while문과 매우 유사한데 조건을 쓰고 난 뒤에 Loop -> { 과 End Loop -> } 을 중괄호로 연상해서 외우면 무리 없을 것이다.

For 문>

필자가 제일 좋아하는 반복문 중에 하나이다. 왜냐면 이는 while 문처럼 일정한 범위가 아닌 딱 정해진 숫자만 반복을 돌기 때문에 필자가 집에서 쌍화탕 끓여 먹듯이 써먹는 반복문이다. 본래 for 문의 반복문은 대표적으로 for(int k=0;k<10;k++) 이런 식인데, 여기서는 좀 햇깔린다. 다음 문장을 살펴보자.

```
Set serveroutput on;
Declare
  v_num Number := 0;
Begin
  For i in 0..10 [Reverse] Loop
    DBMS_OUTPUT.PUT_LINE(i);
  End Loop;
End;
/
```

이처럼 작성하게 되면 밑줄 친 부분이 바로 범위인데 이는 in으로 시작점과 끝점 숫자를 쓰고 그 중간에 점 2개를 넣어야 한다. 혹시 Java에서 배열 반복문을 쓸 때 (int ... k) 이걸로 생각한 사람들이 많아서 점이 3개라고 생각했다면 큰일 난다. 물론 범위 뒤에 Reverse를 써서 해도 되지만 범위에 대해서는 유두리 있게 10..0으로 써줘야 한다. 숫자 이외에 문자열, 커서 등 여러 가지가 들어 갈 수도 있다.

<참고> Set Verify Off라는 것은...?? 우리가 대체 변수를 통해 입력을 받게 되는데 입력받은 값에 대한 세부 정보들에 대해서 자세히 출력하고 싶다면 Set Verify On을 하면 된다. 허나 이를 생략하고 넘어가고 싶다면 Set Verify Off으로 설정해주면 된다.

---

8. Cursor At 161121

PL/SQL 쪽은 코딩 설명과 개념을 같이 하다 보니 요약본 페이지 숫자가 어느덧 2자리 수를 넘기는 건 처음이다. 커서는 마우스로 왔다리 갔다리 하는 그거라는 그 뜻도 있지만 요즘 김장철로 개념 설명을 접목하면 우리가 김치를 50포기 정도 담그면 5포기 정도는 이웃에게 나눠주고 5포기는 새 김치로 보쌈이나 겉절이로 먹을 것이고 나머지 40포기는 두루치기로 숙성해서 나중에 신 김치로 먹기 위해 김치냉장고를 이용해서 보관하게 된다. 이처럼 김치를 데이터로 접목한다면, 사용자가 다룰 데이터들을 김치냉장고 격인 명시적 커서를 통하여 PL/SQL 문에서 필요로 한 데이터들을 커서에 보관하다가 실행부에서 쓰면 된다. 이처럼 사용자가 이를 활용하면 여러 데이터들을 손쉽게 다루게 되어 반복문을 활용하는데 있어서 큰 도움이 된다. 그래서 우리는 커서라는 녀석을 자세히 공부할 필요가 있다.

1) Cursor의 간단한 원리

커서를 공부를 하기 앞서 어떤 원리로 돌아가는지에 대해서 숙지를 할 필요가 있다. SQL를 실행할 때 처리를 위한 메모리 공간을 사용하는 것은 우리가 이미 알고 있을 것이다. 버퍼 캐시(그 현금 아닙니다...)의 원리를 이용하여 그 부분에 커서를 복사하고 커서에서 원하는 데이터를 추출해 후속 작업을 하는 것이 간단한 원리로 살펴볼 수 있다. 그 메모리 공간을 Private SQL Area라고 칭하게 되면서 오라클의 서버 프로세스 구성이 개인 작업을 위한 서버인 Dedicated Server 환경이나 여러 유저들이 작업하는 환경 서버인 Multi-Threaded Server이냐에 따라서 위치되는 곳이 다르다.

2) Cursor의 종류

커서는 2가지로 나뉘어서 살펴보게 되는데 묵시적 커서, 명시적 커서 2가지로 나뉠 수가 있다. 묵시적 커서는 오라클이 알아서 정리해주고 생성을 시켜주는데 명시적 커서를 이용하게 되면 마치 마지막에 남은 인원이 불을 끄고 집 가듯이 정의한 사용자가 알아서 정의하고 Clean-Up을 해야 된다.

3) 묵시적 커서(Implicit Cursor)

묵시적 커서는 말 그대로 오라클에서 자동적으로 선언해주는 SQL 커서여서 사용자는 이게 있는지도 몰랐을 정도로 미스터리한 커서이다. 우리가 쓰는 Select, Insert, Delete, Update 문 등등이 실행되면 묵시적 커서가 선언이 된다. 묵시적 커서의 주의사항은 다음과 같이 요약할 수 있다.

- 세션(Session) 내에 단 한 개만이 선언되어 사용되다가 문장이 종료됨과 동시에 정리가 된다. 원래 명시적 커서처럼 데이터들에 대해서 반복적으로 뽑아오지 않는 이상 실행부를 시작하자 말자 종료하면 자동적으로 사라진다고 생각하면 되겠다.
- 묵시적 커서 데이터는 1행만 가능하다. 그래서 PL/SQL를 처음 도입했을 때 데이터를 다룰 때 한 행만 되는 이유를 눈치챌 수 있을 것이다. 물론 For 문을 이용해서 유두리 있게 사용해도 상관은 없지만, 반복문의 변수를 변경하는 번거로움을 없애기 위해서 명시적 커서를 이용하게 된다.

4) 커서의 속성(Attribute)

나는 수원 근처에 살고, 쌍화차 끓여먹는 것을 좋아하고, 모형 수집과 음악 감상을 취미로 갖고 있다. 이처럼 개개인 별로 속성이 있듯이 커서에 대해서도 속성이 존재한다. 커서에는 어떠한 속성이 있는지에 대해 살펴볼 필요가 있고, 묵시적 커서든 명시적 커서든 원리는 똑같으니 살펴보도록 하자. 묵시적 커서이든 명시적 커서 이름 뒤에는 다음과 같은 키워드로 살펴볼 수 있으니 참고하도록 하자.

<> 묵시적 커서는 명시적 커서 이름을 쓰는 대신에 SQL로 작성하면 되겠다.

- %ROWCOUNT

우리가 인덱스에서 ROWID를 통하여 세부 정보의 ID를 얻을 수 있듯이 ROW뒤에 Count가 붙는 거로 봐서 해당 커서에 실행한 총 행의 개수로 파악을 할 수 있다. 쉽게 말해서 마지막 세부 정보의 행번호를 뜯어온다고 생각하면 될까..??

- %FOUND

커서 내부에서 수행해야 할 데이터가 있는 경우는 참(True)을 반환, 아닌 경우에는 그짓(False)을 반환한다. 명시적 커서인 경우에는 Fetch 문을 통해서 값을 불러올 텐데 읽혀진 행이 있으면 참(True), 없음 그짓(False)을 반환한다.

- %NOTFOUND

고딩 때 생물 선생님의 명대사가 있다. 한 개념이 정의되어 있다면 그 개념의 반대의 정의는 반대로 따르니 한 개념만 공부하라고 말이다. 이처럼 NOTFOUND는 데이터가 없는 경우에 참(True), 존재하는 경우 그짓(False)을 반환한다. 명시적 커서도 마찬가지다. 그러니 %Found만 알아둬도 그것의 반대로 생각하면 되니 자세히 설명하진 않겠다.

- %ISOPEN

묵시적 커서가 됐든 명시적 커서가 됐든 간에 메모리에 Open되어 있으면 참(True), 아니면 그짓(False)을 반환하는 개념이다. 이는 명시적 커서를 이용한 반복문에서 자주 만나게 될테니 숙지하고 넘어가자.

5) 명시적 커서(Explict Cursor)

묵시적 커서는 사용자가 정의를 안 해도 알아서 쓰였지만 이는 사용자가 알아서 일일이 정의하고, PL/SQL 실행부에 Open, Close를 해야 되고, 변수에 알아서 할당(Fetch)해줘야 한다. 명시적 커서는 한 행이 아닌 여러 행들을 다루는 반복문의 영원한 동반자이다. 묵시적 커서로 여러 행들을 다루게 되면 TOO_MANY_ROWS 예외하게 되는 단점에 대해 보완을 해줘서 좋은 역할을 한다. 물론 여러 개의 명시적 커서를 선언하고 이용할 수 있고 커서 속성 변수는 위에서 설명했듯이 사용자가 정의한 명시적 커서의 이름으로 접두어를 이용하여 사용이 가능하다.

<명시적 커서 단계>

Declare(명시) - Open(열기) - Fetch(할당) - Close(종료) 4단계로 나뉠 수 있다. 각 단계 별로 꼼꼼히 살펴보도록 하자.

Phase 1) Declaration(커서 선언)

커서는 선언부(Declare)에서 선언을 해주고 실행부(Begin)에서 여닫고, 할당을 할 수 있는데 선언은 어떻게 할까? 다음 쪽 문장을 참조하자.

```
Cursor [커서_이름] IS
  [서브쿼리];
```

생각보다 단순하다. 서브 쿼리는 우리가 불러오고 싶은 데이터 값들에 대해서 담은 것이다. 예를 들어 사원들 중에서 봉급이 3000이상이고, 부서 번호가 30번에서 50번까지인 직원에 대해서 직원 번호, 성, 봉급을 뽑아오는 것으로 사례를 들어보겠다.

```
Cursor emp_cursor IS
  Select employee_id, last_name, salary
  from employees
  where department_id Between 30 and 50
  and salary>=3000;
```
이처럼 작성하면 무난하게 커서를 선언할 수 있는데 문제는 이를 변수를 선언하듯이 쓸 수 있긴 한데 %Rowtype을 이용하면 되지만 이는 필자가 또 정리한 파일을 참조하고 공부해두면 좋겠다. 명시적 커서를 제대로 이용하는 방법은 실행부부터 참고하자.

Phase 2) Open(커서를 엽시다.)
선언문에서 만든 커서를 불러오는 문장이 바로 Open 이다. 커서를 선언할 때 메모리 공간을 차지한다고 생각을 하지만, 실질적으로는 여기서부터 메모리가 차지된다고 봐야 된다. 이를 작성해 주면 저장된 데이터의 첫 번째 행을 가리키는 포인터와 같은 역할을 해준다.
```
Open emp_cursor;
```
물론 만든 커서 이름으로 불러오는 것으로 쳐줘야지 실질적으로 실행하는 경우에 메모리에 적재된다.

Phase 3) Fetch(할당합시다)
PL/SQL에서 유일하게 허락하는 Select 문의 into 문을 통해서 변수에 각 값들을 할당하였다. 이처럼 Fetch 커서_이름 into 변수들로 하면 손쉽게 값들을 할당하는데 큰 문제가 없다. 필자가 선언한 데이터가 v_empid, v_name, v_sal이라고 하면 다음과 같이 작성을 할 수 있다.
```
Fetch emp_cursor into v_empID, v_name, v_sal
```
여기서 밑줄 친 부분을 참조하자. 물론 위에 밑줄 친 부분도 참조하자. 섭쿼리에서 선언한 컬럼들의 데이터 형식으로 변수를 같이 선언해야 쓰는데 지장이 없지, 만일 순서를 어긋나게 했다간 무용지물이 되어 버린다. 물론 선언문에 복합변수를 이용할 수 있는데 이는 ref Cursor를 불러와서 이용하는 방법이 있고, Record 복합 변수를 이용하는 방법이 있고, 명시적 커서 이름에 %Rowtype을 붙어줘서 변수로 새롭게 쓰는 방법이 있는데 이에 대해서는 새로운 파일에 재작성 할테니 그렇다고만 알아두자.
(파일 Cursor_Record_변수.hwp를 부록으로 보내겠다. 참조하면 큰 도움이 되겠다.)

Phase 4) Close(종료합시다.)
우리가 편의점에서 초코우유를 사면 분명 유통 기한 내로 마시거나 이미 지났으면 폐기 처분을 해야 한다. 이처럼 명시적 커서도 언젠가는 쓸모없게 된다면 버려지기 마련이다. 우리가 C언어에서 malloc라는 녀석을 들어봤는데 이를 free해줌으로서 메모리 공간을 절약해주듯이, 이를 작성해줘야 PL/SQL 문장을 수행하는데 있어서 방해가 되지 않는다. 아래 문장으로 항상 닫아주도록 하자.
```
Close emp_cursor;
```

6) 명시적 커서를 이용한 반복문 적용
아까 앞에서 설명했듯이 명시적 커서는 반복문의 동반자라는 엄청난 타이틀이라고 이야기를 했다. loop 문이랑 for문에서 어떻게 이용이 되는지 다음과 같은 문장을 참조해보자.

6-1) Loop문에서 커서 활용하기
Loop 문에서 커서를 쓰고 싶다면 선언하는 것은 기본으로, 일일이 여닫고, 할당을 해줘야 한다. 이를 어떻게 활용할 것인지에 대해서 아래 문장을 살펴보자.

```
Declare
  v_empID employees.employee_id%type;
  v_sal employees.salary%type;
  Cursor emp_cursor IS
    Select employee_id, salary
    From employees
    Where salary<7500;
Begin
  Open emp_cursor;
  Loop
    Fetch emp_cursor into v_empID, v_sal;
    exit when emp_cursor%NOTFOUND;
   DBMS_OUTPUT.PUT_LINE(v_empID||'-'||v_sal);
  End Loop;
  Close emp_cursor;
End;
/
```
이처럼 작성을 해주면 사원들 중에서 봉급이 7500 미만인 직원들에 대해서 값들을 출력하게끔 해준다. 필자가 공부하면서 제일 중요로 한 점은 fetch 문과 exit 문에 대해서 순서를 그대로 이용하라고 알리고 싶다. 만일 밑줄 친 문장의 순서를 서로 바꾸면 마지막 행의 값이 한 번 더 나오는 불상사가 일어나게 된다. 주의하도록 하자.

(정말 참고) 커서는 while문에서 왜 적용을 안 할까? while문으로 정말 이용하고 싶다면 첫 번째 데이터들에 대해서 분할을 실시하고 난 뒤에 while문에서 %Found라는 변수로 써야 되는 번거로움과 불편함이 있어서 그렇다. 우리가 공부한 loop문, for문에서 이용하는 커서 개념만 알아두고 넘어가는 편이 나을 듯하다.

6-2) For문에서 커서 활용하기
For 문은, 그렇다. 필자가 왜 For 반복문을 좋아하는지에 대해 눈치 챘을 것이다. 왜냐면 사용자가 별도의 단일변수, 복합변수를 선언할 필요도 없고, 여닫고 할당하는 문장을 굳이 안 써도 되어 개이득이기 때문이다. 그래서 For문은 어떻게 이용을 하는지에 대해 다음과 같이 작성할 수 있다.

```
For [레코드_변수_이름] IN [커서_이름] Loop
  실행문;
End Loop;
```

이처럼 작성을 해서 이용하면 되는데 레코드 변수 이름은 Declare 부에서 선언했으면 그거 이용하면 되지만 실상 for 문에 작성하면 자동적으로 형성되어 더 개이득이다!!!
대신에 주의할 사항은 레코드 변수 이름에서 정의를 하고 난 뒤 커서에서 정의한 컬럼들의 이름을 맞춰줘야 한다. 여태껏 for 문을 통해서 여닫는 문장, 할당 문장에 레코드 변수까지 자동적으로 만들어주면 이러한 대가를 따라야 된다고 생각하면 된다. 아래의 문장을 참고로 어떻게 이용하는지 알아보자.

```
Set ServerOutput ON;
Declare
  Cursor dep_cursor IS
    Select department_id, department_name
    from departments
    where location_id=1700;
Begin
```

```
  for v_dep in dep_cursor Loop
DBMS_OUTPUT.PUT_LINE(v_dep.department_id||'
'||v_dep.department_name);
  end loop;
End;
/
```

필자가 이야기한 것처럼 밑줄 친 부분을 주의해서 유두리있게 작성을 해줘야지 for 문을 이용해서 각 데이터 별로 값들을 출력할 수 있다. 또한 방금 작성한 Loop 문보다 짧아져서 좋은 효과를 얻을 수 있다. 물론 Cursor를 불러오지 않아도 인라인 뷰(Inline View)를 활용해서 다음과 같이도 작성이 가능하다.

```
Set ServerOutput ON;
Begin
  for v_dep in (Select department_id,
                       department_name
                       from departments
                       where location_id=1700)
  Loop
DBMS_OUTPUT.PUT_LINE(v_dep.department_id||'
'||v_dep.department_name);
  end Loop;
End;
/
```

인라인 뷰는 from 문 뒤에 필요로 한 컬럼들을 불러와서 조건에 해당되는 값들에 대해 테이블을 따로 불러오지 않아도 알아서 써먹는 개념인데 잘 모르겠다면 9-10차시 퀴즈 복습에서 참고하도록 하자. 물론 또한 인라인 뷰에서 쓰이는 컬럼의 이름을 이용해서 레코드 변수(v_dep)에 적용해야 하는 건 당연한 이야기이다.

7) 파라미터 명시적 커서

파라미터는 영어로 매개변수라는 뜻이다. 우리가 메소드나 함수에 매개변수를 받아서 값을 적용하듯이, 파라미터를 이용해서 커서에 대해 설정을 할 수 있다. 대표적으로 파라미터는 where문에 많이 쓰면 좋다. 어떻게 이용하는지에 대해 작성해 보겠다.

7-1) Loop문에서 적용

```
Set ServerOutput ON;
Declare
  Cursor emp_cursor(w_dep
employees.department_id%TYPE) IS
    Select employee_id, salary
    from employees
    where department_id=w_dep;
  v_empID employees.employee_id%type;
  v_salary employees.salary%type;
Begin
  Open emp_cursor(30);
  Loop
    fetch emp_cursor into v_empID, v_salary;
    Exit when emp_cursor%notFound;
    DBMS_OUTPUT.PUT_LINE(v_empID||'
'||v_salary);
  End Loop;
  close emp_cursor;
End;
/
```

파라미터 커서를 선언하고 난 뒤에 Open을 하는 경우에만 파라미터를 따로 입력을 해주고, 닫거나 할당하는 경우에는 명시적 커서의 이름만 써주면 된다. 즉 쉽게 이야기해서 그냥 밑줄 친 부분에만 파라미터를 써주면 되고, 점선으로 밑줄 친 부분은 파라미터를 굳이 안 써도 된다. 이는 Loop 문을 이용한 경우를 사례로 들었고 For문에서는 아래와 같이 간단하게 이용하면 된다.

7-2) For문을 이용해 적용하기

```
Set ServerOutput ON;
Declare
  Cursor emp_cursor(w_dep
employees.department_id%TYPE) IS
  Select employee_id, salary
  from employees
  where department_id=w_dep;
Begin
  For v_emp IN emp_cursor(30) Loop
  DBMS_OUTPUT.PUT_LINE(v_emp.employee_id||'
'||v_emp.salary);
  End Loop;
End;
/
```

파라미터를 for에서 선언할 때만 써주면 되기 때문에 큰 불편함 없이 이용할 수 있다. 결과는 물론 7-1와 같은 기준으로 작성했다. 두 문장들을 해석하면 부서번호가 30번인 직원들의 직원번호와 봉급을 출력하는 문장을 PL/SQL 반복문을 이용해서 커서로 만든 것이다. 또한 파라미터는 여러 개로도 가능하니 참조하도록 하자.

8) For Update 문장

어렸을 때 필자가 일요일에 즐겨봤던 예능 프로그램으로 사례를 들어서 설명을 해보겠다. 우리가 KBS에서 방송했던 여걸파이브(알지는 모르겠지만...)에서 했던 코너 중에 아름다운 만남이라고 있다. 이는 고정 맴버들 순서대로 한 도화지에 그림들을 수정하거나 추가해서 60초 안에 그림을 그려 특별 게스트가 그 그림에 대한 정답을 맞추는 게임인데 틀리면 물벼락을 맞게 되는 코너이다. 이 코너를 접목시켜서 설명하면 사용자들이 데이터들에 대해서 수정을 하는 경우가 대다수일 것이다. 왜냐면 도화지에 고정 맴버들이 그림을 그리거나 뜯어 고치듯이 MTS 환경에 따라서 그 값이 사용자들에 따라서 이상하게 변할 수가 있다는 뜻이다. 그래서 다수의 사용자가 UPDATE를 동시에 하는 것을 방지하는 개념이 바로 동시성 제어(Currency Control)라고 한다. 그래서 우리가 배우는 커서를 선언 할 때 커서에 있는 행들에 대해 현재 사용자들이 다른 섹션에서 임의로 값들을 변경하고 난 뒤에 트렌젝션을 해줌으로서 Lock을 확보할 때까지 대기하는 동안 현재 실행하려는 PL/SQL 문에 대해 오류를 보내주는 개념으로 보면 된다. 이에 대해서 이해하기 어려워서 최대한 쉽게 설명하였다. 아래의 문장을 통해 참고하면서 자세히 알아보도록 한다.

```
Cursor [커서_이름] IS
  [서브_쿼리]
  For Update [Of 컬럼_이름] // 1
  [NOwait / Wait k]; // 2
```

1 : for Update 문을 통해서 다른 섹션(그니깐 다른 사용자가...)에서 update문에 대해 작업을 하고 있을 때 그 사람이 rollback, commit(통들어 트렌젝션이라고 칭하겠다.)할 때까지 대기를 하는 것이다. 그냥 써주면 무한 대기이다. 또한 컬럼들을 통해서 그 컬럼에 대해서 제한을 줄 수 있다.

2 : NOwait는 한 마디로 다른 세션에서 작업 중인 경우에 해당 행에 대해 트렌젝션을 하든간 말든간에 Lock을 획득하지 못하고 PL/SQL 문을 실행하면 오류를 뜨게 하는 개념이다.
또한 Wait k는 k초 동안 다른 세션에서 작업하는 동안 트렌젝션을 안 해주는 경우에 Lock을 설정 안 하게 되어 에러를 발생하는 원리로 생각하면 된다.

-> 쉽게 연상하자 -> 우리가 흔히 먹는 것으로 사례를 든다면 Nowait는 타인이 짜장면을 먹고 있는데 내가 찝찝해서 못 뺏어먹는 경우(Lock을 못 획득한 경우)로 생각하면 되겠고, Wait k는 타인이 본래 달콤은 하지만 새콤한 맛이 강한 츄잉을 먹고 있는데 개수가 줄어들기 이전(타인이 트랜젝션을 하기 이전)에 뺏어먹을 수 있는 개념(트렌젝션 완료 후 Lock을 획득)으로 생각해보면 접근하기 쉬울 것이다.

또한 이러한 커서를 선언하고 난 후, 행을 lock을 하고 난 후에 다이렉트로 Update 문장 등에서 해당 커서에 대해 접근하기 위해서는 Where Current Of [커서이름]을 쓴다. 그래서 이 문장을 작성하게 되면 커서에서 For Update문을 통해서 문장을 선언을 해주고, Current Of 커서_이름으로 하게 되면 커서에서 선언한 서브 쿼리의 ROWID를 따로 숙지 안 해도 알아서 값들이 수정되도록 하는 개념이다. 이처럼 타 계정의 세션에서 Update문을 작성하고 있는 경우에 트랜젝션을 하기 이전까지 값들에 대한 트렌젝션을 제대로 처리를 못 했어도 조건에 해당되는 행을 잠근 후에 Current Of를 통해서 제대로 수정하게끔 해준다. 이러한 문장들은 아래 예시로 작성한 문장을 살펴보도록 하자.

```
Declare
  Cursor emp_updcur is
    select employee_id, salary
    from employees
    where department_id=40
    For Update NOwait;
Begin
  Open emp_updcur;
  for v_emp IN emp_updcur loop
    update employees
    set salary=salary*1.3
    where current of emp_updcur;
  end loop;
End;
/
```

이처럼 문장을 작성해 주면 타 세션에서 부서 번호가 40번인 직원들의 값들을 수정하는 동안에 타 세션에서 트랜젝션을 안 해주면 Lock를 못 얻게 되어 오류를 출력한다. 허나 밑줄 친 부분에 대해서 작성을 하게 되면 행을 잠그고 난 뒤 Update문에 접근을 하도록 도와주는 역할을 하니 알아두도록 하자.
