<데이터베이스 실습 복습 정리>

161123 to 161130 (Exception to Subprogram)

이제 기말고사도 막바지에 다가오고 있다. 공부하느라고 다들 고생이 많다. 이번 차시가 PL/SQL의 마지막 부분이니 좀만 더 힘내도록 하자. 이번 차시에는 프로그래밍 언어의 꽃이라고 불리는 예외, C언어든 Java이든 모든 프로그래밍 언어에 존재하는 함수(메소드)의 개념과 같은 PL/SQL의 서브프로그램에 대해서 공부를 해보도록 하자.

[요약 중점 사항]

- PL/SQL의 Exception부에 대해 자세히 공부해 본다.
- 사용자가 예외에 대해서 직접 다룸으로서 실행 제어를 정확히 할 수 있고, 예외를 통해서 생기는 오류들에 대해 상황 조치를 할 수 있다.
- Subprogram의 올바른 이용 방법을 숙달할 수 있다.
- Subprogram을 이용하여 실제 SQL 문장에서도 활용 할 수 있다.

---

9. Exception at 161123

오늘도 알바가 있는 날이다. 그렇지만 필자가 지난번에 천식으로 인하여 새벽에 응급실에 다녀온 적이 있었다... 그래서 이런 날엔 필자는 알바를 할 수 없으니 다른 인원들을 보충해서 땜빵을 하는 방법이 있다. 이처럼 프로그래밍에도 문장을 작성하는데 있어서 오류가 걸리는 부분에 대해서는 상황조치를 할 필요가 있다. 그래서 PL/SQL에서 예외를 어떻게 처리하는지에 대해 자세히 알아보도록 하자.

1) 예외부(Exception)

예외부는 문장을 작성하는데 있어서 중간에 오류가 걸리게 되면 예외처리하는 부분에 대해서 문장을 작성해주면 되는데 어떻게 구성이 되어있는지에 대해 간략히 살펴보고 넘어가자.

```
Exception
  When 예외1[OR 예외2 ...] Then
    실행문;
  When 예외3 Then
    실행문;
  When Others Then
    실행문;
```

이처럼 When을 통해서 각종 예외에 대해서 상황을 조치하게끔 작성을 하면 된다. 허나 중요한 것은 예외에도 OR라는 관계 키워드를 이용할 수 있는데 말 그대로 예외 1이 걸리거나 예외 2가 걸리거나 그런 뜻맞다. 허나 주의할 점은 예외에 대해서는 다양하게 존재하기 때문에 When 절에 상호배타적인 예외로 구분을 해야 되니 주의해서 작성하도록 하자.

2) 예외의 종류

예외는 문장을 작성할 때 오타 등으로 인하여 발생하여 컴파일 에러(Compilation Error)가 걸리게 된다. 그렇지만 여러분들이 올바르게 작성을 했는데도 불구하고 오류가 걸린다면 그것은 런타임 에러(Runtime Error)가 걸리게 된다. 런타임 에러에 대해 상황을 조치하는 개념이 바로 예외이다.

예외에 대해서는 크게 2가지로 구분을 할 수 있는데 오라클에서 만든 Oracle-Exception과 사용자가 심심할 때 이러한 예외에 대해 처리할 수 있도록 만든 사용자 정의 예외가 있다.

3) Oracle-Exception(오라클 예외)

오라클 예외는 진짜 오라클 회사에서 예외에 대해서 처리를 올바르게 하도록 만든 예외이다. 오라클 예외에도 예외 명을 정의한 정의된 오라클 예외(Predefined Oracle Exception)가 있고, 예외 명이 없는 정의 되지 않은 오라클 예외(Non-Predefined Oracle Exception)가 있다. 정의된 오라클 예외에 대해서 간략히 살펴보도록 하자.

<Predefined Oracle Exception의 종류>

정의된 오라클 예외에도 부지기수하게 많다. 그렇지만 이에 대해서는 코드와 이름에 대해서는 굳이 암기할 필요까지는 없다. 이것은 필자 친구 중에 포항공대 다니는 녀석도 자세히 못 외운다. 허나 필자가 강요하고 싶은 것은 예외 명을 읽어 보면서 어떤 의미인지에 대해서 읽어보면 좋겠다.

ACCESS_INTO_NULL(ORA-06530)

정의되지 않은 오브젝트 속성, 즉 복합 변수에서 정의되지 않은 변수에 대해서 할당을 할 때 나오는 에러이다.

CASE_NOT_FOUND(ORA-06592)

Case 문 뒤에 When 문에 조건을 항상 써야 하는 것이 옳다. 그러나 이를 안 쓰면 이 에러가 나오게 된다.

COLLECTION_IS_NULL(ORA-06531)

컬렉션은 차후에 서브프로그램에서 보게 되지만, 선언되지 않은 컬렉션에 exists 메소드 이외를 사용하면 발생한다.

CURSOR_ALREADY_OPEN(ORA-06511)

커서가 이미 열려 있을 때. 즉 커서는 마지막에 이용한 사람이 불 끄고 집 가듯이 커서가 필요 없게 되면 꼭 닫아줘야 한다.

DUP_VAL_ON_INDEX(ORA-00001)

DUP은 duplication으로서 이중성의 뜻을 가진다. 유일한 값들에 대한 인덱스에 중복된 값이 있으면 뜨는 예외이다.

INVAILD_CURSOR(ORA-01001)

커서도 사람이다. 사람. 그래서 커서의 조작이 잘 못된 경우에 이 에러를 반환하게 된다. 간혹 반복문에서 마지막 세부 정보들에 대해 값을 할당하고 그 커서에 대해 또 다시 할당하려고 하면 나오는 에러로 보면 쉽다.

INVAILD_NUMBER(ORA-01722)

문자->숫자로 변환할 때 실패하는 예외인데 우리가 실상 쓰는 아스키코드로 사례를 든다면 알파벳 a를 97로 바꾸는 것은 C언어나 자바에서 가능하지만, PL/SQL에서 가능하다고 생각하면 큰 오산이다.

LOGIN_DENIED(ORA-01017)

deny는 거절당하다의 의미로서, 사용자가 잘 못된 비밀번호나 없는 사용자에 대해 입력할 때 발생하는 에러이다.

NO_DATA_FOUND(ORA-01403)

Select문을 실행하는데 아무런 값이 존재하지 않으면 발생하는 에러이다. 문맥만 봐도 이미 짐작했을 것이다.

NOT_LOGGED_ON(ORA-01012)

오라클 서버에서 접속을 안 한 상태로 PL/SQL을 작성하고 실행하면 발생하는 에러이다. 그니깐 접속을 꼭 확인하고 실행하도록 하자.

PROGRAM_ERROR(ORA-06501)

실제로 프로그램 내부에 문제가 생기면 뭐 같은 상황이 벌어지듯이, PL/SQL 내부에 문제가 생기면 발생하는 예외이다.

ROWTYPE_MISMATCH(ORA-06504)
fetch 문에서 잘 못된 데이터 형식에 대해서 할당을 할 경우에 나오는 예외이다. 그러니 할당을 하는 경우에는 데이터의 형식을 잘 맞춰서 작성하는 점을 목표로 삼아야겠다.
STORAGE_ERROR(ORA-06500)
Storage는 저장 공간이란 뜻이라는 건 다 알 것이다. 말 그대로 저장 공간에 에러가 있다면 발생하는 예외인데 저장 공간이 꽉 찼거나 저장 공간에 이상한 버그가 있을 때 나오는 예외로 보면 되겠다.
SUBSCRIPT_BEYOND_COUNT(ORA-06533)
중첩 테이블(join을 통한 확장 테이블)이나 Varray의 요소 값에 접근할 때 컬렉션의 요소 개수보다 더 큰 첨자 값으로 참조한 경우 발생하는 예외이다. 예를 들어 우리가 테이블 타입의 복합 변수를 선언한다고 가정을 하자. 그니깐 우리는 인덱스를 참조할 때 있어서 범위를 제대로 활용할 필요가 있다. 범위가 양수로만 정의된다고 가정을 하면 여태껏 테이블 타입에 추가된 인덱스가 55인데 57를 찾게 된다면 뜰 수밖에 없는 예외로 보면 되겠다.
SUBSCRIPT_OUTSIDE_LIMIT(ORA-06532)
중첩 테이블(join을 통한 확장 테이블)이나 Varray의 요소 값에 접근할 때 컬렉션의 첨자의 한계를 벗어난 참조가 일어날 때 발생하는 예외이다. 가령 테이블 타입 복합 변수를 선언할 때 12간지를 기준으로 자, 축, 인, 묘, … 술, 해까지 저장되어 있는데 찾을 인덱스 값을 토끼를 뜻하는 토(兎)를 입력한 경우에 첨자의 범위를 벗어나게 되고, 음수로도 인덱스를 설정해도 되는데 인덱스가 최종 설정된 값에 대해서 더 작은 값을 입력하면 뜨는 에러로 살펴볼 수가 있다.
SYS_INVALID_ROWID(ORA-01410)
문자열을 ROWID로 변환할 때, 무효한 문자열의 표현인 경우에 발생한다. ROWID는 알파벳 18개로 표현이 되는데 여기서 중간에 알파벳이 아닌 숫자를 입력하게 되면… 변환하는데 오류가 걸리겠죠…??
TIMEOUT_ON_RESOURCE(ORA-00051)
자원에 대한 대기 시간이 초과하였을 경우에 발생하는 예외. 그니깐 방금 배웠던 Update For문에서 Wait n을 이용해서 n초 동안 다른 섹션에서 트랜젝션을 안 해주면 결국 이 예외가 떠서 오류가 걸리게 된다.
TOO_MANY_ROWS(ORA-01422)
many와 much의 차이는 분명히 있다. 먹을 것으로 쉽게 설명하면 many는 눈깔사탕, 달콤하지만 새콤한 맛이 더 나는 츄잉 등등 셀 수 있는 것들이 많을 때 쓰는 어휘이고, much는 짜장면, 초코우유 등등 짜장면의 면발을 일일이 세지 않거나 초코우유에 들어 있는 분자를 일일이 세지 않는 이상 셀 수 없는 것들에 대해서 많을 때 쓰는 어휘이다. 이것이 중요한 게 아니라 우리가 PL/SQL에서 묵시적 커서를 이용하면 하나의 데이터로만 다룰 수 있다고 배웠다. 그렇지만 묵시적 커서에서 두 개 이상의 데이터를 이용하게 되면 결국에 이 예외가 발생해서 에러가 난다.
VALUE_ERROR(ORA-06502)
우리가 Number(7)로 정의된 변수에 999만 9999에 1를 더한다고 가정을 하자. 그러면 ########이 나오게 되어 범위를 초과하게 된다. 또한 날짜에 대한 문자열을 저장하는 문자열 변수가 본래 Varchar2(15)로 정의되어 있는데 갑자기 날짜를 2016년 x월 y일 HH시 MI분 SS초로 문자열로 변환해서 문자열 변수에 저장한다고 치면 범위를 벗어나게 되어 저장을 못 하게 된다. 이처럼 산술, 변환, 절삭, 크기 제약에 에러가 생길 때 발생하는 에러로 보면 된다.
ZERO_DEVIDE(ORA-01476)
수학을 열심히 공부해본 사람들은 알겠지만 1에서 0을 나누면 무한대(∞)로 생각하는 사람들이 더러 있겠지만, 수학자들의 시점으로 바라보면 수학적으로 존재하지 않는 결과가 나온다. 이처럼 SQL이 됐든 자바가 됐든 C언어가 됐든 간 0으로 나누는 것은 인터럽트로 0으로 나눌 수 없다는 예외를 발생하게 된다. 그래서 0으로 나누려고 시도를 하면 이 예외를 통해 에러를 발생시킨다.
(참고로 밑줄 친 부분은 우리가 주로 다룰 예외를 적어뒀습니다.)

4) 예외를 이용해봅시다

말 그대로이다. 예외들에 대해서 공부를 했다면 어떻게 이용하는지에 대해서 알아야 하는 것이 중요하다. 예외를 사용하는 방법에 대해서는 4가지로 나뉘어서 살펴볼 것인데 오라클 예외를 사용하는 방법, 특정 에러 코드에 대한 예외를 사용자가 직접 정의한 방법, Raise를 이용한 방법, RAISE_APPLICATION_ERROR 프로시저 이용하는 방법으로 나누어서 살펴보겠다.

(1) Oracle 예외 사용
우리가 종종 다루게 될 TOO_MANY_ROWS를 사례로 실험해 보겠다. 다음과 같이 노래 테이블(Musics)에서 노래제목(Title)을 뽑아오는 경우에 대해서 PL/SQL 문장을 작성해 보겠다.

```
Declare
  v_title Musics%Title%type;
Begin
  Select title into v_title
  from Musics
  where title Like '하%';
  DBMS_OUTPUT.PUT_LINE(v_title);
Exception
  When TOO_MANY_ROWS then
    DBMS_OUTPUT.PUT_LINE('노래가 두 곡 이상 입니다.');
End;
/
```

이처럼 작성을 하게 된다면 만일 Musics 테이블에 하로 시작하는 노래들 중에서 서태지와 아이들의 하여가, 김범수의 하루 이렇게 2곡이 있다고 가정을 하면 묵시적 커서에서는 하나의 데이터에 대해만 제어를 할 수 밖에 없으니 결국엔 예외를 발생하게 된다. 에러가 바로 발생하게 되면 예외 문장으로 바로 넘어가게 되니깐 에러가 발생할 것 같은 예외들에 대해서는 오라클 예외를 참조해서 작성해주면 되겠다.

(2) 특정 에러 코드에 대해 사용자가 직접 정의하는 방법
이에 대해서는 PRAGMA EXCEPTION_UNIT 함수를 이용을 하지만 이를 이용하기 위해서는 오라클 오류 번호에 대해서 연관을 짓도록 해야 한다. 무엇보다 중요한 것은 PRAGMA의 명령문은 PL/SQL 실행 시 처리되는 것이 아니라 블록 내의 모든 예외 이름에 대해 PL/SQL 컴파일러에 지시를 하는 역할을 한다. 그래서 다음과 같이 문장을 작성해 보겠다.(여기서 Musics 테이블의 Title은 Unique 제한 조건이 걸려져 있다고 생각하고 살펴보자.)

```
Set ServerOutput ON;
Declare
  new_exp Exception;
  PRAGMA EXCEPTION_INIT(new_exp, -1); -- 1
```

```
Begin
  insert into Musics
  Values('난 알아요'); -- 2
Exception
  When new_exp Then
  DBMS_OUTPUT.PUT_LINE('중복 저장입니다...');
End;
/
```

1 : 우선 예외를 이용하기 위해서 예외 객체인 new_exp에 대해서 선언을 해줘야 한다. 그래야지 Exception_Init에 그 예외에 대한 오라클 예외 번호를 줄 수 있게 되어서 3번과 같이 예외를 처리할 수 있기 때문이다.
2 : 물론 Musics의 Title 테이블이 중복된 값들에 대해서 추가를 하게 되면 예외를 발생하게끔 한다.
여기서 예외 번호가 -1인 것으로 봐서 Dup_Val_On_Index로 알 수 있는데 이러한 긴 예외 이름을 쓰다기 보다 사용자가 알아서 오라클 예외 번호를 통해서 문장을 함축해서 작성하는 것이 약이기 때문에 쓰는 것으로 생각하면 된다. 그래서 사용자가 오라클 예외들에 대해서 길게 쓰기 싫으면 이를 써야 되지만, 애석하게도 사용자는 오라클 예외의 번호에 대해서 알아야 가능하니 작성할 때 참조하면서 써보도록 하자. 또한 예외 번호는 음수로 작성해야 한다.

(3) Raise 함수 이용
Raise는 무언가를 일으키다, 사건을 발생시키는 뜻으로 되겠는데 이 키워드를 이용하게 되어 조건을 만족하지 않게 되면 이 예외를 발생하게끔 해 준다. 무엇보다 중요한 점은 어느 조건에서 오류가 발생할 것 같은 부분에 이 키워드를 통해서 자동적으로 예외부로 보내는 점을 역할을 한다. 쉽게 말해 자바에서 예외를 처리할 때 throws와 같은 역할로 생각하면 되겠다. 허나 이러한 예외에 대해서는 조건문에 걸리는 특정한 상황인 경우에만 주로 쓸 수 있으니 참조만 하고 넘어가도 될 듯하다.

```
Declare
  no_sel Exception;
Begin
  select *
  from employees
  where employee_id=&id;
  if SQL%NOTFOUND then
    Raise no_sel; End if;
Exception
  When no_sel then
    DBMS_OUTPUT.PUT_LINE('존재하지 않는 사원번호입니다.');
End;
/
```

(4) RAISE_APPLICATION_ERROR 프로시저
프로시저는 다음에 배울 서브프로그램에서 자세히 배우게 되겠지만 일단 메소드나 함수와 같다고 생각해두면 된다. 이번에는 사용자도 예외를 정의할 수 있는 찬스가 있는 것이다. 사용자가 정의 가능한 예외 번호는 20000부터 20999까지라서 1000개만 설정이 가능하다니, 안습이다. 위의 문장에서 Exception 문을 없애보고 Raise부분부터 약간만 리터칭해보자.

```
Begin
  Update employees
  Set salary=3000
  where employee_id=&id;
  if SQL%NOTFOUND then
    Raise_APPLICATION_ERROR(-20001, '존재 않는 사원번호'); End if;
End;
/
```

이처럼 쉽게 작성을 하게 되면 결국 존재하지 않는 데이터에 대해서 사용자가 만든 -20001번 예외를 통해서 처리를 하게 되어 예외 메시지를 다음과 같이 출력을 할 수 있게끔 한다.
ORA-20001 : 존재 않는 사원번호
추가적으로 select~into~는 값을 할당하는 경우에 대하여 -1403번의 예외를 정의하게 되니 사용자 정의 예외로 하기보다 NO_DATA_FOUND을 적극적으로 이용해보길 바란다. 또한 이는 트리거에서 많이 다루기 때문에 미리 숙지해두면 좋겠다.

이처럼 오라클에서 이미 만든 예외를 이용하고, 사용자가 예외에 대해서 제어를 하고, 새로 정의할 수도 있고, 예외에 대해서도 많은 방법이 있는지에 대해서 깨달을 수 있었다. 예외는 어느 프로그래밍 언어에 다 있기 마련이라서 공부하기 힘들더라도, 다 알아둘 필요는 없더라도, 집에서 라면에 밥 말아먹는 기분으로 읽어보면 도움이 된다.

---

10. SUBPROGRAM(1) at 161128

이제 PL/SQL에도 막바지가 다가가고 있다. 허나 우리가 공부한 것들은 여태껏 PL/SQL의 기본들만 다룰 뿐 실질적으로 중요한 부분이 바로 Subprogram이 되시겠다. 마치 C언어의 함수, 자바의 메소드 역할을 하는 PL/SQL의 오야붕급이 되시겠는데 이를 잘 활용하면 언젠가는 큰 도움이 되고, SQL 문장 이내에서도 또한 활용할 수 있으니 자세히 공부할 필요가 있다.

1) SUBPROGRAM VS PL/SQL문

우리가 PL/SQL의 블록 구조들 중에서 3가지를 공부하였을 때 생각이 났을 것이다. Anonymous Block(익명 블록), Stored Block(저장된 블록), Nested Block(중첩 블록) 3가지에 대해서 알아보았는데, 충격적인 사실을 알려드리면 우리가 여태껏 PL/SQL에서 써봤던 문장들은 전부 Anonymous Block으로 작성을 해왔었다. 그래서 왠지 PL/SQL 블록에 정의된 이름이 따로 없었고, 재사용도 못한 것 대해 눈치 채신 분들이 있을 것이다. 일반적으로 작성하는 PL/SQL문과 Subprogram의 차이점을 깨닫고 넘어가보도록 하자.

- PL/SQL에서는 정적 SQL을 이용해서 작성을 하여왔지만, Subprogram은 DDL, DCL 등 동적 SQL를 이용해서 작성을 할 수 있다.
- Subprogram은 아시다시피 이름이 지정되어있어서 마치 함수나 메소드를 연상하게끔 해준다.
- PL/SQL에서는 결과 값에 대해서 반환하지 않았었는데, Subprogram을 쓰면 결과 값에 대해 반환이 가능하다. 물론 하나만 가능하다. 여러 개가 가능하다고 생각하였다면 큰 오산이다.
- Subprogram은 데이터베이스에 저장이 된다. 마치 함수나 메소드도 작성을 하면 Stack 메모리에 누적이 되는 원리로 생각하면 된다. 이로써 다른 응용프로그램(Pro*c, JDBC 등등)에서도 불러올 수 있다는 사실도 알게 된다.
- PL/SQL은 매번 사용 시에 컴파일이 되지만, Subprogram은 새로 정의하고 난 뒤에 한 번만 컴파일이 되어 메모리 절약에 큰 영향을 주게 된다.

- Subprogram은 물론 파라미터(흔히 말하는 매개변수)를 활용해서 값들에 대해 제어를 할 수 있어 좋은 영향을 주게 된다.

이로써 Subprogram과 PL/SQL에 대해 간략히 비교를 해봤는데 쉽게 생각하는 방법으로서 PL/SQL은 main 메소드, Subprogram은 사용자가 정의한 메소드로 생각하면 쉽게 생각이 가능하다.

2) Procedure

procedure은 영어사전 뜻대로 절차, 과정이라는 뜻이다. Procedure는 단독으로 실행이 되거나 다른 툴, 프로시저, 환경 등에서도 호출되어 이용할 수 있다. 이는 서브프로그램의 일종으로서 p-code를 통하여 한 번만 컴파일 되고 계속 이용할 수 있어서 컴파일을 하는데 있어서 용량을 절약하는데 도움을 준다. 마치 놀이동산을 입장하는데 있어서 영유아를 지닌 가족들을 위해 한 번만 입장하는 자유이용권이 아닌 연간 회원권으로 1년 내내 무료로 이용할 수 있는 원리로 생각하면 쉽다.

2-1) Procedure 생성하기

Create [Or Replace] Procedure *procedure_name*
[매개변수_이름 [Mode] 매개변수_타입]
  IS/AS
    PL/SQL 블록;

이러한 구조는 프로시저 뿐만 아니라 Function에서도 적용이 되니 공부하면서 참조하면 되겠다. 그래서 각 구조들에 대해서 자세히 숙지하고 넘어가보도록 하자.
Or Replace : 이는 뷰(View)에서 많이 봤을 것이다. 기존에 같은 프로시저가 존재하는 경우에 이를 작성해 주면 새로운 프로시저로 정의를 때려주는 옵션이다. 역시 Alter Procedure를 이용할 필요 없이 같은 이름으로 Or Replace를 이용해서 프로시저를 수정하는 것을 추천하고 싶다.
매개변수들 : 그렇다. 우리가 방금 커서에서 살펴본 매개변수 이용하는 방법과 똑같이 이용하면 된다. 가령 사원들의 부서ID를 매개변수로 하면 v_depID Employees.department_id%type을 이용하면 되겠다. 여기서 조금 추가된 개념인 mode에 대해서 간략히 설명하면 아래와 같이 살펴볼 수 있다.

<Mode의 종류>
IN : 따로 Mode에 대해서 정의를 안 하게 되어도 이로 활용을 하기 때문에 굳이 이에 대해서는 정의 안 해도 좋다. 이를 활용하게 되면 형식 파라미터가 상수로 정의되어서 서브프로그램에 값을 전달하는 원리로 이용이 되고, 기본 값에 대해 Default로 초기화 할 수 있다. 가령 다음과 같이 정의를 하게 되면 매개변수를 따로 설정하지 않아도 v_depID는 30으로 적용을 해서 select로 검색을 하든 update로 봉급을 갱신하든 할 수 있게 되니 참조하자.
v_depID Employees.department_id%type Default 30

OUT : Mode에서 정의를 해줘야 하는 건 당연한 이야기이고, 값이 호출 환경에 반환되기 때문에 애석하게도 초기화된 변수들에 대해서 적용할 수가 없게 된다. 따라서 기본 값에 대해 할당이 불가능하다고 생각하면 된다. 이용방법은 다음과 같이 작성하면 되겠다.
v_depID OUT Employees.department_id%type

IN OUT : IN과 OUT를 믹스한 것과 같은 개념으로 생각할 수 있다. 그래서 초기화한 변수에 대해서 적용이 가능하고, 서브프로그램에도 전달이 되고 호출 환경에도 반환이 된다. 하지만 IN처럼 기본 값에 대해 할당할 수가 없으니...
v_depID IN OUT Employees.department_id%type

2-2) Procedure 확인하기
프로시저들에 대해서 확인하고 싶다면 user_source 딕셔너리를 활용하면 된다.

```
Select *
from user_source
where type='PROCEDURE';
```

이처럼 작성하게 되면 프로시저 이름과 라인 별 작성된 문장들에 대해서 출력을 하니 참조해보는 것도 나쁘지 않겠다.

2-3) Procedure 활용하기
프로시저를 활용해서 부서 번호를 매개변수로 받아서 일상에서는 떡값을 받지 않는 이상 있을 수가 없겠지만 봉급을 1.5배로 변경하는 문장을 작성해 보겠다.

```
Create Or Procedure rice_salary
(v_depID employees.department_id%type) is
Begin
  Update Employees
  Set salary=salary*1.5
  where department_id=v_depID;
End;
/
```

이를 작성하고 난 뒤에 아래와 같은 문장을 통해서 부서번호를 30으로 입력해서 떡값을 받을 수 있게 문장을 실행해 보겠다.

```
Exec Rice_Salary(30);
```

일반 프로시저를 실행하려면 exec 키워드를 이용해야 된다. 우리가 옛날에 컴퓨터실에서 공튀기기나 건물부수기를 할 때나 설치를 실행하는 등등 실행하는 파일의 확장자 중에서 .exe를 많이 봤을 것이다. 이는 execute의 줄임말로서 실행이라는 뜻이다. 그렇지만 SQL에서는 exec로 줄였기 때문에 그 뜻을 연상해서 공부하면 도움이 되겠다.

2-4) Procedure 삭제 및 수정
Procedure는 Drop Procedure, Alter Procedure 등으로 작성하면 되는데 왜 그럴까? Create 문으로 작성하면 새삼스럽게 DDL로 작성된 문장들에 대해서는 언제나 Create, Drop, Alter로 설정을 할 수 있다는 사실을 알 것이다. 방금 만든 떡값 프로시저를 삭제한다면 다음과 같이 이용하면 되겠다!!!

```
Drop Procedure Rice_Salary;
```

(참고) Alter View, Alter 서브프로그램
우리가 지난번에 배웠던 View에서 Alter View라는 녀석을 들어본 적이 없었다. 본래 view에도 Alter 문장이 있지만, 데이터들에 대해 변경을 하거나 view의 이름을 수정하는데 이용한다고 생각하면 쉽지만, 데이터들을 변경하는데 있어서 굳이 복잡하게 Add Query 같이 복잡한 문장들에 대해 접근하지 않고 or Replace를 이용하면 간단하게 정의가 되니깐 데이터들을 변경하려면 or Replace를, 그 외에 객체의 이름을 바꾸는(Rename to 바꿀_이름) 작업은 Alter View를 이용해보도록 하자.

3) Function

우리가 흔히 걸그룹 중에서 f(x)를 들어봤을 것이다. f가 바로 function을 뜻하는 f에서 따온 것이고, (x)는

수학에서는 정의역(매개변수, 파라미터)이 되어 결국 나오는 값 f(x)는 공역(반환 값)이 되는 것이다. 이런 소리를 왜 하는지에 대해서는 쉽게 설명하자면 방금 Procedure에서는 수학적인 함수에서 정의역을 때렸을 때, 값이 안 나오는 방식으로 생각을 할 수 있듯이 반환되는 값들에 대해서 존재를 하지 않았다. 그러나 Function을 이용해보면 수학적으로 함수를 작성하면 공역이 튀어 나오듯이, 결과 값을 반환할 수 있는 좋은 역할을 하게 되어 실제로 SQL문에도 활용하는데 많이 쓰이니 참조하면 도움이 되겠다.

3-1) Function의 구조
Function은 아까 배운 Procedure와 마찬가지로 같은 구조로 살펴볼 수 있는데, 여기서 추가된 내용은 반환하는 값들에 대해서 정의해야 하는 점이 추가되었으니 참조하도록 하자.
Create [Or Replace] Function *function_name*
(매개변수_이름 [Mode] 매개변수_타입)
  return [데이터_타입]
    IS/AS
    PL/SQL 문장
return [데이터_타입]위에 부분은 Procedure와 별반 다를 게 없으니 return [데이터_타입]부터 상세히 알아보자.
return [데이터_타입] : 함수를 정의했으면 반환되는 값이 있어야 한다. 마치 삼겹살을 굽는데 기름이 나오는 원리라고 생각하면 되겠다. 또한 데이터 타입은 굳이 크기를 설정해 주지 않아도 된다. 걍 Number, Varchar2, Date 등등으로 작성해도 무관하고, 테이블 내의 컬럼을 통해서 %type으로 작성해도 무관하다.
PL/SQL 문장 : 여기에는 우리가 일반적으로 작성하는 문장을 쓰면 되는데 주의할 점은 반환할 변수에 대해서 Return 키워드를 꼬오오옥!!! 써줘야 한다.

3-2) Function을 어디에 쓸까?
Function에서 받아온 반환 값은 여러 곳에 쓰일 수 있는데, 어느 곳에 쓰이는지에 대해 자세히 알아둬야지 잘못된 부분에 쓰면 불상사가 일어나니 필자가 여기에 적어두겠다.
-> Query에 있는 Select 문. 쿼리는 메인 쿼리가 될 수 있고, 서브 쿼리가 될 수도 있다.
-> Where/Having 절의 조건식에도 적용이 가능하다.
-> Query의 Connect By, Start With, Order By, Group By 절에서 적용이 가능하다. Start With은 Sequence를 밥먹듯이 공부했다면 알 것으로 판단되지만, Connect By는 필자도 모르겠다. 그냥 그렇다고 알아두면 된다.
-> Insert 문의 Values 문. 그냥 Values에 변수 형식에 맞춰서 작성하면 된다. 그렇다고 last_name에 문자열 '10000원' 같은 값을 저장하면 좀 그렇겠죠...
-> Update 문의 Set 문. 물론 조건식에서 쓰인다면 우측 항에 써야겠다. 좌측 항은 테이블에 있는 컬럼을 위한 배려석이니깐^^

3-3) Function 활용하기
방금 Procedure와 다를 바가 없는 점이 반환하는 값만 추가된 것. 그게 다다. 그러니 공부하는데 큰 문제는 없을 것으로 판단이 된다. 예를 들어 808번 직원의 상사(Manager)의 풀네임을 출력하는 문장을 작성해보자. (단 first_name, last_name을 묶은 제약 조건에 대해서는 unique 조건이 걸려져 있다고 생각하자.)

```
Create Or Replace Function
  emp_mgr_func(v_empID
  Employees.employee_id%type)
  return Varchar2
IS
  v_mgrName Varchar2(30);
  Begin
   Select first_name||' '||last_name into
v_mgrName
    from employees
    where employee_id = (Select manager_id
from employees where employee_id=v_empID);
    return v_mgrName;
  End;
/

Select emp_mgr_func(808)
from dual;
```

이처럼 풀네임을 변수 v_mgrName에 할당을 하고 반환까지 완료를 하고 dual(가상 결과 전용 뷰)에 함수 결과를 실행해보면 808번 직원 상사의 풀네임이 나오게 된다. 이처럼 PL/SQL를 한 번만 컴파일해서 계속 이용하는 저장된 블록을 통해서 반환되는 값을 이용하면 컬럼이든, 조건문이든 접근이 가능하니 유용하게 활용하도록 하자. 또한 서브프로그램에서 IS 부분 뒤에 Declare(선언부)를 작성하는데 있어서 변수만 정의해도 된다. 즉 Declare 키워드는 생략하고 필요로 한 변수, 커서 등을 선언하면 된다.

3-4) Function 확인하기
```
Select *
from user_source
where type='FUNCTION';
```
방금 Procedure처럼 결과가 나올텐데, 줄마다 나오는 결과를 보여줄 수 있으니 함수 이름(Name)을 잘 기억해서 where문에 검색하면서 적용하면 되겠다.

4) Package
우리가 일리네어의 '가'를 들어 보면, 더콰 형님이 부른 가사 중에 다음 날 눈을 뜨면 '새 지폐들이 내 문을 두드려, 여자들은 Package~' 라는 가사를 들어봤을 것이다. 가사의 뜻대로 무언가에 연관성이 있다면 따라 붙는 개념이 바로 패키지이다. 이처럼 실제로 연관된 프로시저나 함수들에 대해서 하나로 묶어서 쓰는 개념을 Package라고 칭한다.

4-1) Package 구성요소
패키지는 2가지로 나뉘는데 패키지 선언부(Spec), 패키지 몸체부(Body)로 구성이 된다. 패키지 선언부에서는 패키지 외부에서도 쓸 수 있는 Public 변수(그니깐 전역 변수)들에 대해 선언이 가능하고, 패키지 몸체부에서는 패키지 내부에 쓰이는 Private 변수들에 대해 선언이 가능하다. 그래서 패키지의 구성요소에 대해 세부적으로 숙지할 필요가 있다.

4-2) Package 선언부(Spec)
우리가 흔히 취업할 때 스펙, Spec 이러는 이야기를 들어본 적이 있을 것이다. 그러나 여기서 쓰이는 spec은 실제 Package에 필요로 한 변수 및 서브프로그램의 대가리를 쓰는 것으로 살펴보면 되겠다. 어떻

게 구성이 되는지 살펴보도록 하자.
Create [Or Replace] Package *package_name*
IS/AS
  (1) public type and item
  (2) Subprogram Heads
End *package_name* ;
(1) 우리가 자바에서 public이라는 녀석을 많이 봤을 것이다. 패키지 외부에서도 쓰일 수 있는 변수, 커서, 사용자 예외, Pragma 등등 public으로 정의가 된다.
(2) Subprogram의 Head부분이다. 예를 들어 프로시저인 경우에는 Procedure 프로시저_이름(매개변수_있으면_포함해서) 이렇게 작성하면 되고, 함수는 Function 함수_이름(매개변수_있으면_포함해서) return 반환형 이렇게 작성하면 되겠다. 또한 여기에 쓰이는 프로시저와 함수는 HR에 연결되어 있다면 패키지 외부에서도 선언이 가능한 public 서브프로그램으로 생각하면 된다. 아래와 같이 패키지 선언부를 정의할 수 있다. 또한 두 말하면 잔소리이지만 패키지 선언부의 이름과 패키지 몸체부의 이름은 둘 다 같아야 되고, End 뒤에 패키지 이름을 써줌으로서 마무리를 져야 한다. 마치 서약서 같은 거 작성하면 본인 서명으로 종결을 짓 듯이.

```
Create Or Replace Package emp_pkg is
  Procedure emp_hire_older;
  Procedure rice_salary
  (v_dep employees.department_id%type);
  Function max_salary
  (v_dep employees.department_id%type)
  return Number;
End emp_pkg;
/
```

4-3) Package 몸체부
패키지의 대가리를 선언했다면 이제 패키지의 몸 부분을 만들 차례이다. 우리가 방금 패키지 선언부에서 정의한 것들은 Public이 될 수 있겠지만, 여기서 새로이 정의하면 Private가 되어서 패키지 내부에서만 적용이 가능하게 된다. 이로써 패키지 몸체부는 어떻게 작성이 되는지에 대해 알아보자.
Create Or Replace Body *package_name*
 IS/AS
 (1) Private type and item
 (2) Subprogram bodies
End *package_name* ;
(1) 자바에서는 private 변수가 인스턴스 변수로서 객체 내에서만 공유하는 변수로 적용을 하듯이, 패키지 내부에서도 따로 이용하는 변수뿐만이 아니라 사용자 예외, PRAGMA, 커서 등등 여러 가지를 적용하게 된다. 물론 Private이기 때문에 패키지 내부에서만 쓰이겠다는 것을 짐작할 수 있다.
(2) Subprogram의 구성에 대해 작성을 하면 되는데 여기서 편리를 줄 수 있는 점은 공용으로 이용할 섭프로그램에 대해서는 뒷부분에 작성을 하고, 패키지 내부에 쓸 섭프로그램에 대해서는 앞에서 선언해주면 편리하게 이용할 수 있다. 그래서 참조되는 변수가 되든 서브프로그램이든 참조하는 서브프로그램(그니깐 우리가 패키지 밖에서 쓰는 섭프로그램)보다 먼저 정의되면 올바른 패키지를 완성하는데 도움이 된다.

패키지는 어떻게 작성하면 될까? 방금 패키지를 선언한 부분에서 썼던 섭프로그램들을 모조리 이용해보도록 하겠다.

```
Create Or Replace Package Body emp_pkg IS
  Procedure emp_hire_older IS
   v_fullname Varchar2(40);
   Begin
    Select first_name||' '||last_name into
v_fullname
    from employees
    where hire_date=(select min(hire_date)
                     from employees);
    DBMS_OUTPUT.PUT_LINE(v_fullname);
   End emp_hire_older; -- 1

  Procedure rice_salary(v_dep
employees.department_id%type) IS
   Begin
    Update Employees
    set salary=salary*1.5
    where department_id=v_dep;
   End rice_salary; -- 2

  Function max_salary
  (v_dep employees.department_id%type)
  return Number IS
  out_salary Employees.salary%type;
  Begin
    Select max(salary) into out_salary
    from employees
    where department_id=v_dep
    group by department_id;
    return out_salary;
  End max_salary; -- 3
End emp_pkg;
/
```

1, 2, 3번과 같이 패키지 내에 있는 프로시저, 함수들의 정의가 끝났으면 각 객체들의 이름들을 항상 써줘야 정상적으로 작동이 된다. 1번 프로시저는 서브쿼리를 이용해서 근무 기간이 가장 오래된 직원을 뽑아내는 문장이고, 2번 프로시저는 방금 이야기한 떡값이고, 3번 함수는 부서 별 최대 봉급을 뽑아내는 문장이다. 이처럼 패키지를 이용해서 작성을 하면 프로시저들과 함수들을 관리하는데 있어서 큰 도움이 된다. 하지만 패키지에서 주의할 점은 각 프로시저, 함수, 패키지부분에는 제안서에 서명하듯이 항상 마지막에 그에 해당하는 이름을 꼭 작성해주고 끝내야 한다. 또한 패키지를 실행하기 위해서는 다음과 같이 실행해주면 되겠다.

(1) 프로시저 활용
```
Exec emp_pkg.rice_salary(60);
```

(2) 함수 활용
```
Select emp_pkg.max_Salary(30)
from dual;
```

패키지 이름 뒤에 프로시저 이름이나 함수 이름을 이용해서 늘 이용해왔듯이 잘 이용해 먹으면 된다.

---

11. SUBPROGRAM(2) at 161130

이제 서브프로그램의 최종적인 Trigger를 활용하는 방법에 대해 남았다. 다음에는 SQL 문장을 응용프로그램에서 어떻게 이용하는지에 대한 사례를 JDBC(그 손석희 나오는 그 방송국 아니다...)로 들게 되면 이번 학기에 데이터베이스 실습 과목이 끝나게 된다. 이번

기말고사가 끝나면 필자는 힐링을 목적으로 문산에 있는 임진각을 갈 예정이고, 동경 여행을 위해 막노동을 뛰어서 방학 기간에도 바쁠 것으로 예상된다. 그건 둘째 치고, Trigger를 어떻게 이용하는지에 대해 자세히 공부해보도록 하자.

5) Trigger

Trigger는 영어로 좀 잔인한 말이겠지만... 방아쇠라는 뜻이다. 그래서 우리가 노래로 접근하면 빅뱅의 Bang Bang Bang을 예로 들면 중간에 뱅~뱅~뱅~ 빵야~ 빵야~ 빵야~, 다 꼼짝 마라~ 오늘 밤 끝장 보자~ 부분이 있다. 왜 필자가 이 노래에 대해 언급했는지에 대해서는 트리거의 정의를 살펴보면서 살펴보도록 하자.

5-1) Trigger의 정의

우리가 예를 들어 어느 테이블이나 뷰에 대해서 DML 문장(Insert, Delete, Update)을 적용하게 되는데 이를 이벤트라고 한다. 노래 가사에서 뱅뱅뱅~ 빵야 빵야 빵야 부분으로 살펴볼 수 있는데, 총을 쏘게 되면 마치 총알이 나오게 되는 부분을 살펴볼 수 있는데 이 부분을 트리거에 걸린 이벤트(Event)로서 살펴볼 수가 있다.
그리고 사용자가 트리거에 정의된 예외에서 어긋난 행위를 하면 예외에 대해서 발생을 하게 되어 이벤트에 대해서 묵시적으로 자동실행(Fire)를 하게 되는데 방금 노래 가사에서 다 꼼짝 마라~ 오늘 밤 끝장 보자~ 부분처럼 상대방을 총알 하나로 제압한다는 뜻을 담긴 노래 가사의 의미를 살펴볼 수 있듯이, 사용자가 데이터에 대해 예외를 발생하게 한 경우에 이벤트를 통해서 사용자가 올바르게 서브프로그램을 사용할 수 있도록 제어를 한다고 비유를 할 수 있다.
수업 시간에도 교수님이 말씀드렸다만, 트리거가 많아지게 되면 관련 객체들끼리 복잡한 종속관계가 되어버리기 때문에 적당히 쓰는 것이 좋겠다. 마치 집 근처에 유흥거리가 많으면 친구들이랑 놀 때 좋겠다만, 너무 시끄러워서 무언가 집중할 수 없을 때를 연상하면 되겠다.

5-2) DML Trigger

DML Trigger에는 여러 가지가 존재하는데 우리는 대표적인 Before Trigger를 살펴보고, After, Instead에 대해서는 읽어보는 것으로 마무리하면 되겠다. 더 공부하고 싶다면 참조를 바라보자. 또한 트리거를 통해서 복잡하게 정의하는 혼합 트리거를 이용할 수 있겠지만, 거기까지 공부하는데 있어서 무리로 예상이 되기 때문에 생략하고 넘어가도록 하자.
Before Trigger : 테이블에서 DML이벤트를 트리거하기 전에 Trigger 본문을 실행하게 된다. 즉 사용자가 Insert, Delete, Update를 하기 전에 사용자가 입력한 데이터에 대해 Trigger 본문에서 확인을 하고 난 뒤에 예외를 잡든 데이터를 조작하든 한다.

<참조> After Trigger / Instead Of Trigger
After Trigger는 Before Trigger의 반대 개념으로 DML이벤트를 실행하고 난 뒤에 트리거 본문을 실행하는 개념으로 살펴보면 되겠고, Instead Of Trigger는 Instead라는 녀석을 보면 Trigger문 대신에 Trigger 본문을 실행하고, 수정이 불가능한 뷰에 사용을 한다고 살펴보면 되겠다. After Trigger, Instead Of Trigger는 우리 수업 시간에는 다루지 않으니 Before Trigger만 알아두고 넘어가도 장땡이다.

5-3) 문장 Trigger / 행 Trigger

트리거는 저장된 행들에 대해서 적용을 할 수 있고 행을 떠나서 문장에만 적용하는 경우가 있는데 트리거에 대해서 어떻게 나누는지에 대해서 문장 트리거, 행 트리거 2가지로 나뉘어 살펴볼 수가 있다.

-> 문장 트리거 : 트리거를 실행하는데 있어서 행이 몇 개든 간에 트리거 작업이 영향 받지 않는 행의 데이터나 이벤트 자체에서 제공하는 데이터에 종속되지 않은 경우에 쓰는 경우에 유용하다.

-> 행 트리거 : 테이블 내의 트리거 이벤트에 영향을 받는 경우에는 실행을 하지 않는데, 행의 데이터나 트리거 이벤트 자체에서 제공하는 데이터에 트리거 작업이 종속될 경우에 유용하게 쓰인다.

우리가 주로 쓰는 트리거는 행 트리거가 되겠는데, 행 트리거에 필요로 한 데이터 구조를 숙지하고 넘어가면 좋겠다. 물론 행 트리거에 대해서 어떻게 활용이 되는지 문장을 설명하면서 다시 설명하겠지만, 간략한 정의에 대해서만 살펴보도록 한다.
-> OLD : Trigger가 처리한 레코드의 본래 값을 저장한다고 보면 된다.
-> NEW : 새로운 값들을 포함한다.

(참조) DML문장 이외에도 DDL 이벤트 트리거, 데이터베이스 이벤트 트리거도 존재한다. 이러한 이벤트 트리거는 이럴 수 있구나라고 알아두고 넘어가자.
-> DDL 이벤트 트리거
트리거를 이용해서 이벤트가 걸리게 되면 테이블, 뷰, 시퀀스 등등을 만들 수 있는 역할을 해준다.
-> 데이터베이스 이벤트 트리거
데이터베이스 전체에 영향을 주는 트리거이다. 데이터베이스 내부에 존재하는 생기는 일들을 관리하는 역할을 하는데, 데이터베이스의 종료나 시작 등에 대해 이벤트가 걸리면 자동 fire를 할 수 있으며, 특정 오류에 따라서 여러 가지를 할 수 있다.
-> User 이벤트 트리거
사용자가 발생시키는 작업에 트리거를 생성하는데, 대표적으로 Create, Alter, Drop을 하는 권한을 줄 수 있도록 하는 트리거로 생각하면 된다.

5-4) Trigger 문법

우리가 여태껏 공부한 서브프로그램의 생성 문법이랑 엄연히 차이가 있기 때문에 좀 정리를 해둘 필요가 있다.
Create [Or Replace] Trigger *trigger_name*
[Before/After](1) [event1 [or event2 ...]](2)
[of 컬럼들](5) on [table/view/schema name](3)
For Each Row [When ...] (4)
[PL/SQL문]

(1) Before/After는 timing의 개념이다. (2)번에서 실행한 이벤트에 대해서 실행하기 이전, 실행하기 이후로 시점을 나눠서 정의를 할 수 있어서, 트리거가 특정 뷰에 대한 DML인 경우에 timing부분에 Instead Of를 적용한다. 그렇지만 우리가 흔히 쓰는 타이밍은 Before가 되시겠다.

(2) 이벤트 : 이벤트의 종류는 여러 가지가 있다. DML_Event, DDL_Event, DATABASE_Event로 나뉘는데 우리가 흔히 쓰는 DML_Event의 종류로서는 Insert, Delete, Update 3가지로 나뉠 수가 있다.

(3) ON 스키마 : 이러한 이벤트를 어느 객체에 대해서 이용하는가에 대해서 정의를 때려주는 것이다. 테이블, 뷰에 대표적으로 쓰이게 되는데, 마치 우리가 지난 번에 배웠던 DCL 문장 중의 일부인 Grant 이벤트 On HR.Employees를 살펴볼 수 있다.

(4) For Each Row는 테이블, 뷰의 트리거를 행 트리거로 명시하는 절이다. 본래 생략을 해도 이로 자동적으로 설정이 되지만, When 문에서는 행 트리거의 각 행에 대해 제약을 주는 절이다. 물론 When 문은 생략이 가능하다.

(5) 우리가 컬럼들에 대해서 각기 Old, New 변수를 나눠기 위해서 쓰는 문장이 바로 OF이다. Of 후반부로는 컬럼들의 이름이 나오게 된다.

5-5) Trigger 관리
우리가 트리거를 이용하면 언젠가는 쓰이지 않을 것이고, 테이블에 속한 트리거를 비활성화시킬 수 있다. 어떻게 관리를 하는지에 대해 다음 페이지에서 살펴보자.
-> 트리거 활성화/비활성화
Alter Trigger *trigger_name* (Disable | Enable)
아시다시피, 트리거 이름 옆에 Enable로 작성하면 트리거를 활성화하고, Disable로 작성하면 트리거를 비활성화하는 것을 볼 수 있다.
-> 특정 테이블의 트리거 활성화/비활성화
Alter Table 테이블_이름
(Disable | Enable) All Triggers;
특정 테이블에서 쓰이는 트리거에 대해 모두 비활성화를 할 수 있고, 모두 활성화 할 수 있다. 테이블 내의 모든 트리거들에 대한 관리는 이처럼 작성을 해야 가능하니 참조하자.
-> Trigger 수정 후 컴파일
Alter Trigger 트리거_이름 Compile;
지금 필자가 요약본을 만들고 난 뒤에는 저장을 해두고 컴퓨터를 꺼야지 제대로 저장이 된다. 이처럼 트리거를 Alter 문을 통해서 수정하였다면(물론 Or Replace를 이용하는 것이 훨 배 낫지만...) Compile 키워드를 추가해서 갱신된 트리거를 관리하자.
-> Trigger 삭제
Drop Trigger 트리거_이름
트리거도 DDL 문장을 통해서 만들어지기 때문에 새삼스럽게 Drop으로 지울 수 있는 건 당연한 이야기이다.
-> Trigger 조회
Select *
from User_Triggers;
트리거의 세부 정보를 조회하고 싶다면 User_Triggers 세션을 조회해보면 된다.

5-6) 트리거 권한 주기
트리거도 마찬가지로 권한을 줘야 생성이 가능하다. 간만에 DCL를 복습해보는 차원해서 필자가 다시 적었으니 참조하자.
Grant Create Trigger To 사용자;
Grant [Alter/Drop] Any Trigger To 사용자;
이러한 트리거를 제어하는 문장을 통해서 트리거를 생성하고, 수정하고, 삭제를 하는 명령에 대해서 DBA 사용자는 일반 사용자에게 권한을 줘야 사용이 가능하다는 사실만 알아두면 좋겠다.

5-7) Trigger 이용하기
우리가 트리거를 어떻게 이용하는지에 대해 간략한 개념으로만 살펴봤다. 그래서 필자가 다음과 같은 테이블을 정의해서 어떻게 이용을 하는지에 대해서 학생 테이블을 통해서 예시를 들겠다.
1) 시간제한 관련 이벤트 잡아내기(문장 Trigger)
```
Create Or Replace Trigger stu_trigger
 Before Insert On Student
 Begin
  IF(TO_CHAR(sysdate, 'HH24:MI') not Between '09:00' and '18:00') Then
  Raise_Application_Error(-20010, '교무처 업무시간이 끝났습니다.');
  End if;
 End;
/
```
우리가 이처럼 문장을 작성하게 되면 가령 교무처가 아침 9시부터 오후 6시까지 운영해서 학생 정보를 추가하는데 있어서 교무처의 업무가 끝나면 학생의 정보를 추가하지 못하게 사용자가 정의한 예외의 원리를 이용해서 만들 수 있다. 또한 Insert를 하게 되면 트리거에 작성된 PL/SQL 문장이 실행하게 되면서 자동 실행을 하게 된다.

2) OLD/NEW 변수 이용하기(행 Trigger)
우리가 OLD/NEW 변수를 정의만 해두고 어떻게 이용하는지에 대해서 제대로 설명을 안 했을 것이다. 이러한 문장들이 어떻게 활용하는지에 대해 문장을 작성해 보겠다.

우리가 학생 이메일을 추가하는데 있어서 교무처에서 @sofa.com 이메일을 보내면 제대로 전송이 안 되어서 이러한 이메일을 보냈을 때를 대비해서 다시 입력받도록 하는 트리거를 작성해 보겠다.
```
Create Or Replace Trigger email_trg
 Before Insert ON Student
 For Each Row
 Begin
  IF(:NEW.email Like '%@sofa.com') Then
   Raise_Application_Error(-20020, '이메일을 다시 입력해 주세요.');
  End if;
 End;
/
```
-> Insert 문 적용하게 되면...
```
Insert Into Student(stuID, Name, email)
Values(101, '언더테이커', 'wwe@sofa.com');
```
-> ORA-20020 : 이메일을 다시 입력해 주세요.

요즘 시대에 이메일을 보내게 되어도 잘 안 보내지는 경우는 없진 않은데 옛날에는 종종 그런 현상이 있어서 이러한 원리를 통해서 이메일을 다시 쓰도록 하는 경우가 더러 존재하였다. 그건 둘 째 치고, For Each Row를 작성해주게 되면 행 트리거를 형성할 수 있게 된다. 또한 주의해야 할 점은 PL/SQL 실행문 내에서 New 변수를 이용 할 때에는 앞에 :(콜롬, 혹은 땡땡)을 작성해줘야 한다. 하지만 For Each Row 후에 When 문에는 콜롬을 안 써야 된다. 이는 우리가 지난 번에 배웠던 바인드 변수의 원리를 이용해서 설명하면 외부에서 사용자가 값들을 추가하게 되는데 PL/SQL 문에서 콜롬을 찍을 때 이용되는 New/Old 변수가 곧 바인드 변수가 되기 때문이고, Trigger 생성문에 대해서는 PL/SQL 문이 아닌 엄연히 DDL이기

때문에 여기서는 콜롬을 찍어주지 않게 된다.
New 변수에 대해선 다음과 같은 경우에 써주면 되겠다.
- Insert 문에서 Values 부분에서 쓸 때
- Update 문에서 Set 부분의 우측에서 쓸 때
New 변수를 활용할 경우에 마침표(.)을 통해서 새로이 추가되는 값이나 갱신을 시도하는 값들의 컬럼을 불러오면 된다.
또한 Old 변수는 Update문, Delete문에서 주로 쓰이는 Where문에서 조건 중에 해당되는지에 대해서 확인을 할 때 작성이 된다.

(참조) Trigger에서 2가지 이상 DML문장에 적용시키기
트리거를 이용해서 2가지 이상의 DML 작업에 대해서도 처리를 할 수 있다. 이는 If 문 뒤에 inserting, updating, deleting 이렇게 작성하면 되겠는데 이는 책에 안 나와서 필자가 인터넷에서 구글링해서 추가한 개념이다. 이런 식으로 쓸 수가 있다는 사실만 알아도 될 거 같다.

```
Create Or Replace Trigger stud2_trg
  Before Insert Or Update On Student2
    For Each Row
    Begin
     IF Inserting then -- 1
      DBMS_OUTPUT.PUT_LINE(:new.stuID||'번의 '||:new.Name||' 학생이 추가됨.');
     End if;
     IF Updating then -- 2
      DBMS_OUTPUT.PUT_LINE('Before -> '||:old.deptID||' / After-> '||:new.deptID);
     End if;
   End;
/
```

1번과 같이 작성을 하게 되면 학생의 정보가 추가되는 경우에 번호와 학생의 이름을 같이 출력해주는 역할을 한다. 여기서 insert에서는 New 변수를 쓴다고 생각하면 된다. 그냥 모두 다. 2번과 같은 경우에는 학생의 학과 번호가 변경되는 경우에 변경되기 전과 변경된 후에 대해서 출력을 한 예시이다. 여기서 old 변수를 이용하면 학생의 학과 번호가 변경되기 이전의 값을 불러오게 되고, new는 학생의 학과 번호가 변경된 값을 불러와서 결국에 학과 번호 변경 이전/이후에 대해 출력을 할 수 있다. 이건 오로지 참조로만 살펴보고 넘어가면 되겠다. Update 문에서도 deptID만 바꾸는 것만이 아니기 때문에...

4) Trigger의 When 문을 이용해보자
마지막으로 트리거에서 When문이 어떻게 이용되는지에 대해서 알아둘 필요가 있다. 트리거의 When문에는 행 트리거에 쓰이는 변수(new, old 변수) 이외에 테이블 수준 변수를 쓰면 작동이 안 된다. 규정이 이러니 참조하자. 마지막이니만큼 Employees 테이블을 이용해서 해보겠다. 예를 들어 새로 입력 받는 부서번호가 50인 경우에 사원을 추가하는 기간에 대해서 제한을 두는 경우에 다음과 같은 문장을 통해서 만들 수 있다.

```
Create Or Replace Trigger emp_50_trg
  Before Insert ON Employees
  For Each Rows
  When (New.Department_id=50)
Begin
  IF (sysdate>TO_DATE('2016/12/10', 'YYYY/MM/DD')) Then
    Raise_Application_Error(-20030, '부서 번호 50번의 직원 추가 기간이 끝났습니다.');
  End if;
End;
/
```

이처럼 사용자가 추가하려는 값에 대해 어느 부서에 대해서 직원 추가 기간 동안 추가를 할 수 있도록 트리거를 설정을 함으로서, When 문에 조건만 새롭게 추가해주면, 그 조건에 해당되는 정보들에 대해서 트리거 내부의 PL/SQL 문장을 자동 실행하게 되면서 데이터의 올바른 길잡이를 잡아주는 큰 역할을 하는 데이터 계의 경찰서와 같은 역할이라고 보면 되겠다.

(참조) 시퀀스를 이용해서 트리거에 DML문장을 작성하는 경우에는 dual 가상 테이블을 통해서 불러와야지 정상적으로 이용이 가능하다. 11g에서 장점으로 시퀀스 값을 그대로 이용이 가능하게끔 한 점이 있었다만, 트리거 내에서 시퀀스를 그대로 적용하게 된다면... 2개 이상의 행에 대해서 업데이트가 되니 이러한 불상사를 피하기 위해서 dual 테이블을 이용해보면 좋겠다. 아래는 2014년 기말고사에서 나온 문제를 필자가 풀어본 문장이다. 참조하길 바란다.

```
Create OR Replace Trigger tr_gogak_hist
  Before Update ON Gogak
  For Each Row
Declare
  v_next Number(6);
Begin
  IF Updating then
    select gogak_his_no_seq.nextval into v_next
    from dual;
    Insert Into point_history
    Values(v_next, :OLD.GNO, :OLD.point,
     :NEW.point, sysdate);
  End if;
End;
/
```

-> 이제 PL/SQL문까지 해서 책에 있는 오라클 SQL, PL/SQL의 내용은 마치게 됩니다. 차주에 배우는 JDBC에 대해서 시험에도 출제가 되니깐 차주에 그 부분에 대해서 요약정리를 하게 된다면 따로 연락드리겠습니다.