함수 프로시저

자바 강의실

목차

1. 저장 프로시저와 사용자 함수

- 저장 프로시저를 생성하기 위한 CREATE PROCEDURE 의 형식은 다음과 같습니다.

```
CREATE [OR REPLACE ] PROCEDURE prcedure_name
( argument1 [mode] data_taye,
  argument2 [mode] data_taye . . .
)
IS
local_variable declaration
BEGIN
statement1;
statement2;
. . .
END;
/
```

1. 저장 프로시저와 사용자 함수

- 저장 프로시저를 생성하려면 CREATE PROCEDURE 다음에 새롭게 생성하고자하는 프로시저 이름을 기술합니다.
- 이렇게 해서 생성한 저장 프로시저는 여러 번 반복해서 호출해서 사용할 수 있다는 장점이 있습니다.
- 생성된 저장 프로시저를 제거하기 위해서는 DROP PROCEDURE 다음에 제거하고자 하는 프로시저 이름을 기술합니다.
- OR REPLACE 옵션은 이미 학습한 대로 이미 같은 이름으로 저장 프로시저를 생성할 경우 기존 프로시저는 삭제하고 지금 새롭게 기술한 내용으로 재 생성하도록 하는 옵션입니다.

1. 저장 프로시저와 사용자 함수

- 프로시저는 어떤 값을 전달받아서 그 값에 의해서 서로 다른 결과물을 구하게 됩니다.
- 값을 프로시저에 전달하기 위해서 프로시저 이름 다음에 괄호로 둘러 싼 부분에 전달 받을 값을 저장할 변수를 기술합니다.
- 이를 ARGUMENT 우리나라 말로 매개 변수라 합니다.
- 프로시저는 매개 변수의 값에 따라 서로 다른 동작을 수행하게 됩니다.
- [MODE] 는 IN과 OUT, INOUT 세 가지를 기술할 수 있는데 IN 데이터를 전달 받을 때 쓰고 OUT은 수행된 결과를 받아갈 때 사용합니다. INOUT은 두 가지 목적에 모두 사용됩니다.

1. 저장 프로시저와 사용자 함수

예: 프로시저 생성

질의 : **EMPLOYEES_COPY** 테이블에 저장된 모든 사원을 삭제하는 프로시저를 작성하기

```
CREATE OR REPLACE PROCEDURE DEL_EMP_COPY
IS
BEGIN
DELETE FROM EMPLOYEES_COPY;
END;

EXECUTE DEL_EMP_COPY; -- 실행하기
```

1. 저장 프로시저와 사용자 함수

- '프로시저가 생성되었습니다.'란 메시지와 함께 한 번에 저장 프로시저를 성공적으로 생성할 수도 있지만, '경고: 컴파일 오류와 함께 프로시저가 생성되었습니다' 와 같은 메시지가 출력되면 저장 프로시저를 생성하는 과정에서 오류가 발생해서 저장 프로시저 생성에 실패하는 경우입니다.

```
PROCEDURE DEL_EMP_COPY이(가) 컴파일되었습니다.
Errors: check compiler log
```

- 이럴 경우에는 오류를 제거해서 다시 저장프로서지를 생성해야 하는데, 오류 메시지를 모른 채 오류를 수정하기란 쉽지 않습니다.

1. 저장 프로시저와 사용자 함수

- 오류가 발생할 경우 “SHOW ERROR” 명령어를 수행하면 오류가 발생한 원인을 알 수 있게 됩니다.
- 원인을 분석하여 오류를 수정한 후 다시 저장 프로시저를 생성을 시도하여 ‘프로시저가 생성되었습니다.’란 메시지가 출력되어 저장 프로시저가 성공적으로 생성될 때까지 오류 수정 작업을 반복해야 합니다.

1. 저장 프로시저와 사용자 함수

예: 프로시저 생성

질의 : 저장 프로시저를 작성시 오류가 발생하면 이를 수정하기 위한 방법 연습

```
CREATE OR REPLACE PROCEDURE DEL_EMP_COPY
IS

DELETE FROM EMPLOYEES_COPY;
END;


SHOW ERROR; --에러 확인 -- 내용을 수정한 후 프로시저를 재 생성하여 실행한다
```

1. 저장 프로시저와 사용자 함수

- 저장 프로시저를 작성한 후 사용자가 저장 프로시저가 생성되었는지 확인하려면 USER_SOURCE 살펴보면 됩니다.
- 다음은 USER_SOURCE의 구조입니다.

```
SELECT * FROM USER_SOURCE;
```

- USER_SOURCE의 내용을 조회하면 어떤 저장 프로시저가 생성되어 있는지와 해당 프로시저의 내용이 무엇인지 확인할 수 있습니다.

1. 저장 프로시저와 사용자 함수

- DEL_EMP_COPY 저장 프로시저는 사원 테이블의 모든 내용을 삭제합니다.
- 만일 특정 사원만을 삭제하려면 어떻게 해야 할까요?
- 저장 프로시저를 생성할 때 삭제하고자 하는 사원의 이름이나 사원 번호를 프로시저에 전달해 주어 이와 일치하는 사원을 삭제하면 됩니다.
- 저장 프로시저에 값을 전달해 주기 위해서 매개 변수를 사용됩니다.

1. 저장 프로시저와 사용자 함수

- 매개변수가 있는 저장프로시저는 다음과 같이 정의합니다.

```
CREATE OR REPLACE PROCEDURE
DEL_EMP_COPY_BYNAME(VENAME EMPLOYEES_COPY.EMPLOYEE_NAME%TYPE)
IS
BEGIN
DELETE FROM EMPLOYEES_COPY WHERE EMPLOYEE_NAME=VENAME;
END;
/
```

- 저장 프로시저 이름인 DEL_ENAME 다음에 ( )를 추가하여 그 안에 선언한 변수가 매개 변수입니다. 이 매개변수에 값은 프로시저를 호출할 때 전달해 줍니다.

```
EXECUTE DEL_EMP_COPY_BYNAME( 'Pat' );
```

1. 저장 프로시저와 사용자 함수

- CREATE PROCEDURE로 프로시저를 생성할 때 MODE를 지정하여 매개변수를 선언할 수 있는데 MODE 에 IN, OUT, INOUT 세 가지를 기술할 수 있습니다.
- IN 데이터를 전달 받을 때 쓰고 OUT은 수행된 결과를 받아갈 때 사용합니다.
- INOUT은 두 가지 목적에 모두 사용됩니다.

1. 저장 프로시저와 사용자 함수

예: 프로시저 삭제

```
DROP PROCEDURE  DEL_EMP_COPY;
```

1. 저장 프로시저와 사용자 함수

- 저장 함수는 저장 프로시저와 거의 유사한 용도로 사용합니다.
- 차이점이라곤 함수는 실행 결과를 되돌려 받을 수 있다는 점입니다.
- 다음은 저장 함수를 만드는 기본 형식입니다.

```
CREATE [OR REPLACE ] FUNCTION function_name
( argument1 [mode] data_taye,
  argument2 [mode] data_taye . . .
)
IS
RETURN data_type;
BEGIN
statement1;
statement2;
RETURN variable_name;
END;
```

1. 저장 프로시저와 사용자 함수

- 프로시저를 만들 때에는 PROCEDURE라고 기술하지만, 함수를 만들 때에는 FUNCTION이라고 기술합니다.
- 함수는 결과를 되돌려 받기 위해서 함수가 되돌려 받게 되는 자료 형과 되돌려 받을 값을 기술해야 합니다.
- 저장 함수는 결과를 얻어오기 위해서 호출 방식에 있어서도 저장 프로서저와 차이점이 있습니다.

```
EXECUTE :varable_name := function_name(argument_list);
```

1. 저장 프로시저와 사용자 함수

예: 저장함수 생성

질의 : 다음은 특별 보너스를 지급하기 위한 저장 함수를 작성해 봅시다.
보너스는 급여의 200%를 지급한다고 합시다.

```
CREATE OR REPLACE FUNCTION CAL_BONUS(
  VEMPID IN EMPLOYEES_COPY.EMPLOYEE_ID%TYPE  )
  RETURN NUMBER
IS
  VSAL NUMBER(7, 2);
BEGIN
  SELECT SALARY INTO VSAL
  FROM EMPLOYEES_COPY
  WHERE EMPLOYEE_ID= VEMPID;

  RETURN (VSAL * 2);
END;
/

SHOW ERROR; -- 오류 확인
```

1. 저장 프로시저와 사용자 함수

예: 저장함수 실행

```
VARIABLE VAR_RES NUMBER; -- PL/SQL
EXECUTE :VAR_RES := CAL_BONUS(100);
PRINT VAR_RES;
```

1. 저장 프로시저와 사용자 함수

예: 저장함수 삭제

```
DROP FUNCTION CAL_BONUS;
```

2. 패키지

- 오라클 데이터베이스에 저장된 프로시저, 함수 뿐만 아니라 변수 등을 하나로 묶은 캡슐화된 객체
- 처리하는 작업의 성격이 비슷한 함수나 프로시저를 하나로 묶어 놓은 오라클 객체
- 패키지의 장점 (객체지향적 성격)
  - 애플리케이션을 좀 더 효율적으로 개발할 수 있게 도와준다. (모듈화)
  - 관련된 스키마 오브젝트들을 재 컴파일할 필요 없이 수정이 가능하다. (종속성)
  - 한 번에 여러 개의 패키지 오브젝트들을 메모리에 로드할 수 있
  - 프로시저나 함수들의 오버로딩이 가능하다.
  - 패키지 내의 모든 타입, 항목, 서브프로그램들을 PUBLIC 이나 PRIVATE 으로 선언해서 사용할 수 있다. (정보은닉)

3. 트리거

- 트리거
  - 명시된 이벤트(데이터베이스의 갱신)가 발생할 때마다 DBMS가 자동적으로 수행하는, 사용자가 정의하는 문(프로시저)
  - 트리거는 데이터베이스의 무결성을 유지하기 위한 일반적이고 강력한 도구
  - 제약조건이 트리거보다 성능이 우수하고, 코딩이 불필요하기 때문에 선언하고 수정하기가 용이하므로 가능하면 제약조건을 사용하는 것이 좋음
  - 트리거는 테이블 정의시 표현할 수 없는 기업의 비즈니스 규칙들을 시행하는 역할
  - 트리거를 명시하려면 트리거를 활성화시키는 사건인 이벤트, 트리거가 활성화되었을 때 수행되는 테스트인 조건, 트리거가 활성화되고 조건이 참일 때 수행되는 문(프로시저)인 동작을 표현해야 함
  - 트리거를 이벤트-조건-동작(ECA) 규칙이라고도 부름
  - E는 Event, C는 Condition, A는 Action을 의미
  - SQL3 표준에 포함되었으며 대부분의 상용 관계 DBMS에서 제공됨

3. 트리거

- 트리거(계속)
  - 트리거의 형식

```
CREATE TRIGGER <트리거이름>
AFTER          <트리거를 유발하는 이벤트들이 OR로 연결된 리스트> ON
                    <릴레이션>
[WHEN  <조건>]
BEGIN  <SQL문(들)> END
```

  - 이벤트의 가능한 예로는 테이블에 투플 삽입, 테이블로부터 투플 삭제, 테이블의 투플 수정 등이 있음
  - 조건은 임의의 형태의 프레디키트
  - 동작은 데이터베이스에 대한 임의의 갱신
  - 트리거가 제약조건과 유사하지만 어떤 이벤트가 발생했을 때 조건이 참이 되면 트리거와 연관된 동작이 수행되고, 그렇지 않으면 아무 동작도 수행되지 않음
  - 삽입, 삭제, 수정 등이 일어나기 전(before)에 동작하는 트리거와 일어난 후(after)에 동작하는 트리거로 구분

3. 트리거

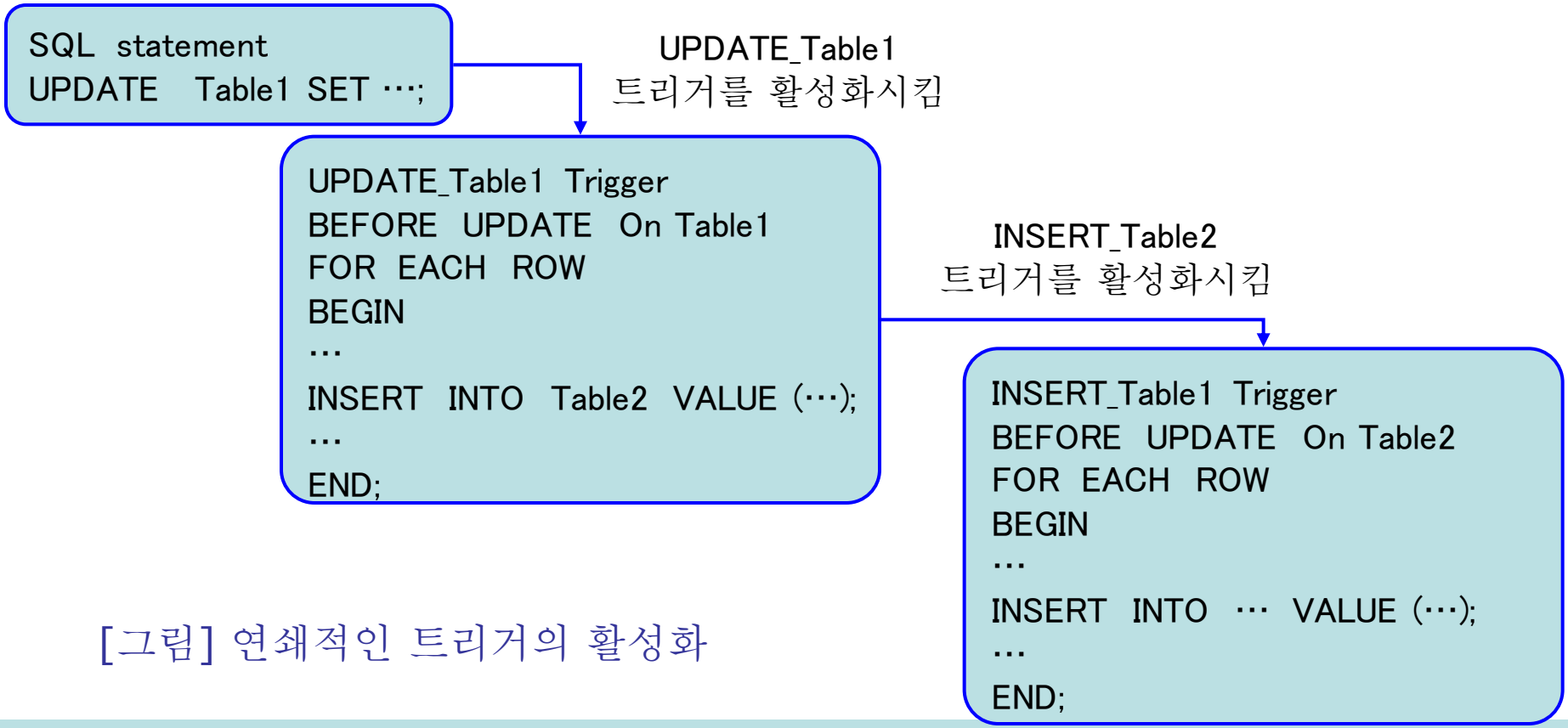
예: 트리거

새로운 사원이 입사할 때마다 사원의 급여가 10000미만인 경우에는 급여를 10% 인상하는 트리거를 작성하라. 여기서 이벤트는 새로운 사원 튜플이 삽입될 때, 조건은 급여 < 1500000, 동작은 급여를 10% 인상하는 것이다.

```
CREATE   TRIGGER   RAISE_SALARY
AFTER   INSERT   ON  EMPLOYEES
REFERENCING   NEW   AS   newEmployee
FOR   EACH   ROW
WHEN          (newEmployee.SALARY < 10000)
UPDATE         EMPLOYEES
SET           SALARY = SALARY * 1.1
WHERE         EMPLOYEE_ID= newEmployee.EMPLOYEE_ID ;
```

3. 트리거

- 연쇄적으로 활성화되는 트리거
  - 하나의 트리거가 활성화되어 이 트리거 내의 한 SQL문이 수행되고, 그 결과로 다른 트리거를 활성화하여 그 트리거 내의 SQL문이 수행될 수 있음



[그림] 연쇄적인 트리거의 활성화