Writing Control Structures

1. IF 문

1-1. 개요

```
IF      condition      THEN
        statements;
[ELSIF condition       THEN
        statements;]
[ELSE
        statements;]
END IF;
```

- 만족되는 조건에 따라 선택적으로 작업 수행
- **ELSIF는 한 단어, END IF는 두 단어**
- 부울 제어 조건이 **TURE**면 연관된 명령문의 시퀀스가 실행되고, **FALSE** 또는 **NULL**이면 연관된 명령문의 시퀀스가 실행되지 않음
- **ELSIF** 절 사용에 대한 제한은 없음
- **ELSE** 절은 최대 한 개만 사용
- 조건적으로 실행되는 명령문은 알아보기 쉽도록 들여쓰기

1-2. 단순 IF 문

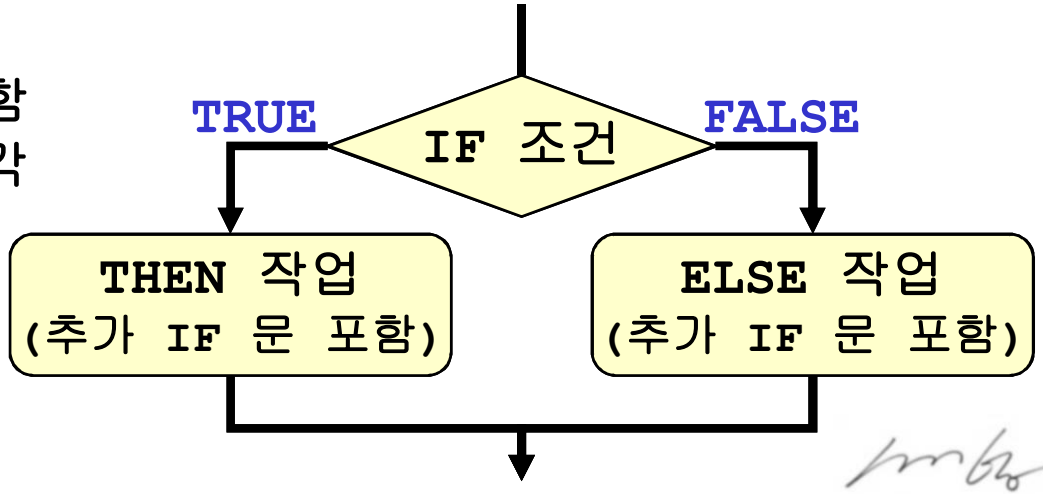
```
...
IF      v_ename         = 'MILLER'   THEN
        v_job           := 'SALESMAN';
        v_deptno        := 35;
        v_new_comm      := sal * 0.20;
END IF;
...
```

- 단순 IF 문장은 PL/SQL 조건이 TRUE인 경우에만 수행하고, 조건이 FALSE나 NULL이면 무시
- 두 경우(TRUE, FALSE나 NULL) 모두 프로그램의 END IF 다음에 오는 명령문에서 제어가 재개 됨

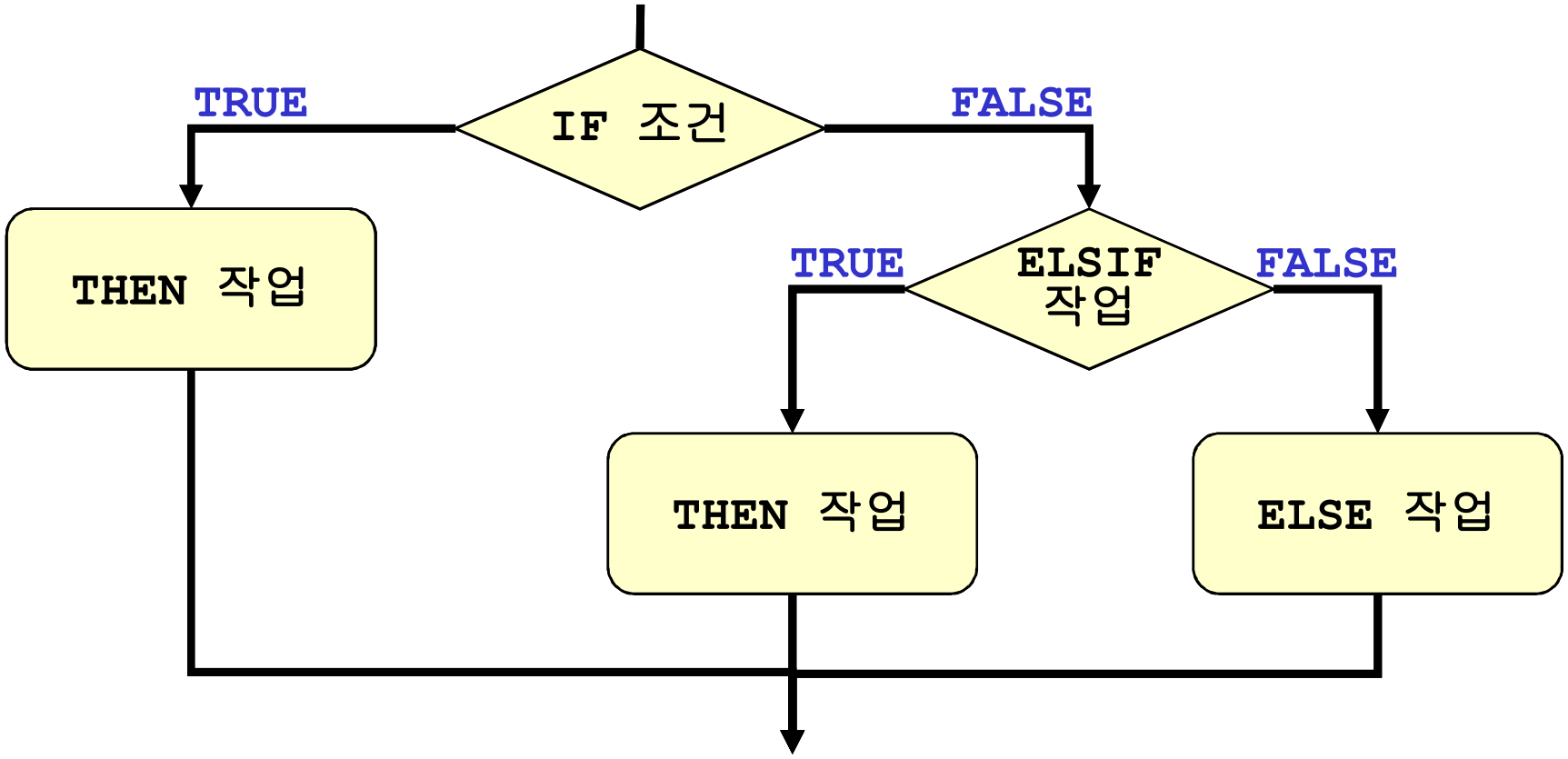
1-3. IF-THEN-ELSE 문

```
...
IF      v_shipdate - v_orderdate < 5 THEN
        v_ship_flag := 'Acceptable';
ELSE
        v_ship_flag := 'Unacceptable' ;
END IF;
...
```

- 조건이 FALSE나 NULL이면 ELSE 절을 사용하여 다른 작업 수행
- THEN과 ELSE 절은 IF 문을 포함할 수 있으며 중첩 IF 문은 각각 해당 END IF 로 종료



1-4. IF-THEN-ELSIF 문 (1)



1-5. IF-THEN-ELSIF 문 (2)

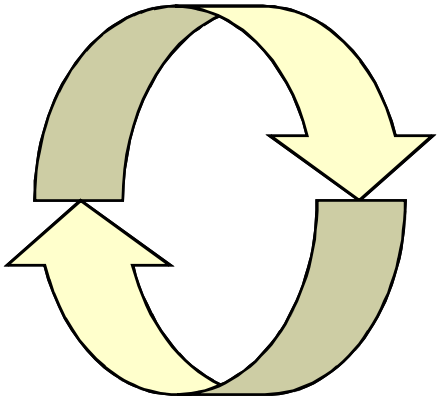
```
...
IF      v_deptno  = 10        THEN
        v_comm    := 5000;
ELSIF v_deptno  = 20        THEN
        v_comm    := 7500;
ELSE
        v_comm    := 2000;
END IF;
...
```

- **ELSIF** 절의 코드는 중첩 **IF** 문의 코드보다 쉽게 읽고 이해할 수 있음
- **ELSE** 절의 작업이 별개의 **IF** 문으로 구성된 경우 **ELSIF** 절을 사용하는 것이 편리
- **ELSIF** 절을 사용하면 추가된 각 조건 및 작업의 끝에 중첩 **END IF**를 사용하지 않아도 됨

2. 루프(LOOP) 문

2-1. 개요

- LOOP는 명령문이나 명령문의 시퀀스를 반복 실행
- LOOP 문의 세가지 유형
  - 기본 LOOP
  - FOR LOOP
  - WHILE LOOP



- 기본 LOOP는 전체적인 조건 없이 반복 작업을 수행
- FOR LOOP는 횟수를 기준으로 해서 반복 제어 작업을 수행
- WHILE LOOP는 조건을 기준으로 해서 반복 제어 작업을 수행
- EXIT 문은 LOOP를 종료

2-2. 기본 LOOP (1)

```
LOOP
    statement1 ;
    •••
    EXIT [WHEN condition]
END LOOP;
```

- LOOP 문의 가장 단순한 형태는 기본 LOOP 또는 무한 LOOP
- 실행 흐름이 END LOOP 문에 도달할 때마다 위에 있는 해당 LOOP문으로 반환
- LOOP를 시작할 때 이미 조건이 만족되었더라도 해당 명령문을 적어도 한 번 이상 실행
- EXIT 문을 사용하여 LOOP를 종료
- EXIT 문이 없으면 LOOP가 끝없이 실행

2-3. 기본 LOOP (2)

```
DECLARE
     v_empno          emp.empno%TYPE := 1000;
     v_counter        NUMBER(2) := 1;
BEGIN
     LOOP
        INSERT INTO  emp (empno, mgr)
        VALUES (v_empno, v_counter);
        v_counter  :=  v_counter + 1;
        v_empno := v_empno + 1;
        EXIT  WHEN  v_counter > 10;
     END LOOP;
END;
```

2-4. FOR LOOP (1)

```
FOR counter in [REVERSE]
    lower_bound..upper_bound  LOOP
  statement1;
  statement2;
  ...
END LOOP;
```

- FOR LOOP를 사용하면 PL/SQL이 수행하는 반복 횟수를 결정
- 카운터(counter)는 암시적으로 정수로 선언되었으므로 선언하지 않음
- LOOP 범위의 상한 및 하한에는 리터럴, 변수, 표현식이 올 수 있지만 반드시 정수여야 함
- LOOP 범위의 하한이 상한보다 큰 정수면 명령문의 시퀀스가 실행되지 않음

2-5. FOR LOOP (2)

```
DECLARE
      v_empno          emp.empno%TYPE := 700;
BEGIN
      FOR i IN 1..10 LOOP
           INSERT INTO emp (empno, mgr)
           VALUES (v_empno, i);
           v_empno := v_empno + 1;
      END LOOP;
END;
```

- 카운터(counter)는 LOOP 내에서만 참조할 수 있으며 LOOP 밖에서는 정의되지 않음
- 카운터의 기존 값을 참조하려면 표현식을 사용
- 할당 대상으로서 카운터를 사용하면 안됨

2-6. WHILE LOOP (1)

```
WHILE  condition  LOOP
   statement1;
   statement2;
   •••
END LOOP ;
```

- 제어 조건이 **TRUE**인 동안 명령문의 시퀀스를 반복
- 반복이 시작될 때마다 조건 평가
- 조건이 **FALSE**면 **LOOP**가 종료
- **LOOP**를 시작할 때 조건이 **FALSE**면 반복 작업이 더 이상 실행되지 않음
- 조건에 포함된 변수가 **LOOP** 본문 내에서 변경되지 않으면 조건이 **TRUE**로 유지되므로 **LOOP**가 종료되지 않음
- 조건의 결과가 **NULL**이면 제어는 **LOOP**를 통과하여 다음 명령문으로 전달

2-7. WHILE LOOP (2)

```
ACCEPT p_new_empno PROMPT 'Enter the employee number: '
ACCEPT p_mgr PROMPT 'Enter the number of mgr: '

DECLARE
     v_count         NUMBER(2) := 1;
BEGIN
     WHILE v_count <= &p_mgr LOOP
         INSERT INTO emp (empno, mgr)
         VALUES (&p_new_empno, v_count);
         v_count := v_count + 1;
     END  LOOP;
     COMMIT;
END;
/
```