SQL 기초2

INSERT, DELETE, UPDATE문

❑ INSERT문

- ✓ 기존의 릴레이션에 투플을 삽입
- ✓ 참조되는 릴레이션에 투플이 삽입되는 경우에는 참조 무결성 제약조건의 위배가 발생하지 않으나 참조하는 릴레이션에 투플이 삽입되는 경우에는 참조 무결성 제약조건을 위배할 수 있음
- ✓ 릴레이션에 한 번에 한 투플씩 삽입하는 것과 한 번에 여러 개의 투플들을 삽입할 수 있는 것으로 구분
- ✓ 릴레이션에 한 번에 한 투플씩 삽입하는 INSERT문

```
INSERT
INTO    릴레이션(애트리뷰트1, ..., 애트리뷰트n)
VALUES (값1, ..., 값n);
```

INSERT, DELETE, UPDATE문(계속)

예 : 한 개의 투플을 삽입

질의: DEPARTMENT 릴레이션에 (5, 연구, ∧) 투플을 삽입하는 INSERT문은 아래와 같다.

```
INSERT INTO DEPARTMENT
VALUES (5, '연구', '' );
```

DEPARTMENT

DEPTNO	DEPTNAME	FLOOR
1	영업	8
2	기획	10
3	개발	9
4	총무	7
5	연구	0

INSERT, DELETE, UPDATE문(계속)

❑ INSERT문(계속)

✓ 릴레이션에 한 번에 여러 개의 투플들을 삽입하는 INSERT문

```
INSERT
INTO    릴레이션(애트리뷰트1, ..., 애트리뷰트n)
SELECT  ...  FROM  ...  WHERE  ...;
```

예 : 여러 개의 투플을 삽입

질의: EMPLOYEE 릴레이션에서 급여가 3000000 이상인 사원들의 이름, 직급, 급여를 검색하여 HIGH_SALARY라는 릴레이션에 삽입하라. HIGH_SALARY 릴레이션은 이미 생성되어 있다고 가정한다.

```
INSERT    INTO HIGH_SALARY(ENAME, TITLE, SAL)
SELECT    EMPNAME, TITLE, SALARY
FROM      EMPLOYEE
WHERE     SALARY >= 3000000;
```

INSERT, DELETE, UPDATE문(계속)

- DELETE문
  - ✓ 한 릴레이션으로부터 한 개 이상의 투플들을 삭제함
  - ✓ 참조되는 릴레이션의 삭제 연산의 결과로 참조 무결성 제약조건이 위배될 수 있으나, 참조하는 릴레이션에서 투플을 삭제하면 참조 무결성 제약조건을 위배하지 않음
  - ✓ DELETE문의 구문

```
DELETE
FROM    릴레이션
WHERE   조건;
```

INSERT, DELETE, UPDATE문(계속)

예 : DELETE문

질의: DEPARTMENT 릴레이션에서 4번 부서를 삭제하라.

INSERT, DELETE, UPDATE문(계속)

❑ UPDATE문

- ✓ 한 릴레이션에 들어 있는 투플들의 애트리뷰트 값들을 수정
- ✓ 기본 키나 외래 키에 속하는 애트리뷰트의 값이 수정되면 참조 무결성 제약조건을 위배할 수 있음
- ✓ UPDATE문의 구문

```
UPDATE   릴레이션
SET      애트리뷰트 = 값 또는 식[, ...]
WHERE    조건;
```

예 : UPDATE문

질의: 사원번호가 2106인 사원의 소속 부서를 3번 부서로 옮기고, 급여를 5% 올려라.

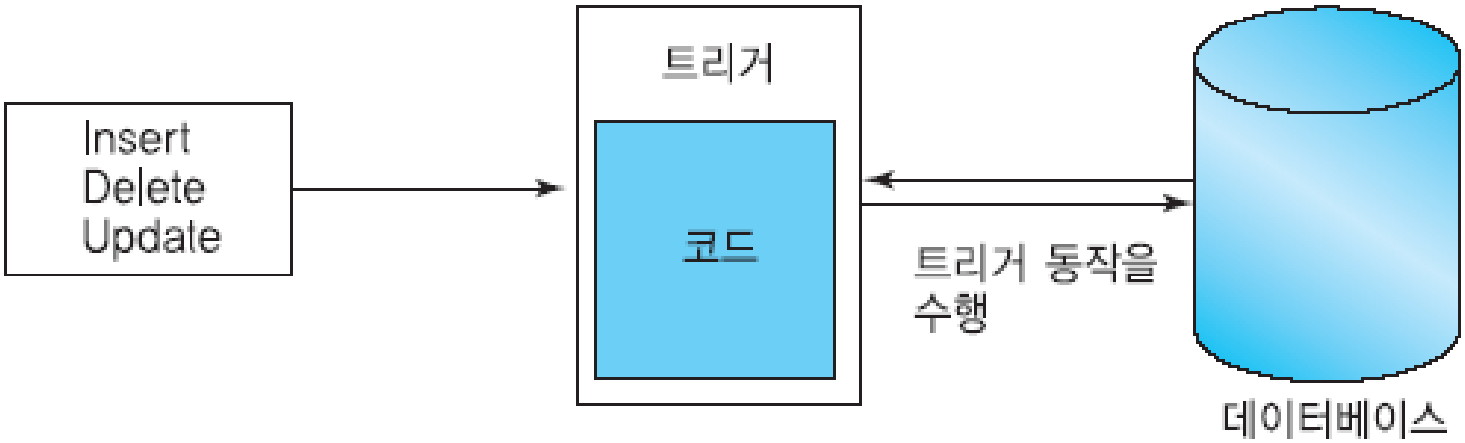
트리거(trigger)와 주장(assertion)

❑ 트리거

- ✓ 명시된 이벤트(데이터베이스의 갱신)가 발생할 때마다 DBMS가 자동적으로 수행하는, 사용자가 정의하는 문(프로시저)
- ✓ 데이터베이스의 무결성을 유지하기 위한 일반적이고 강력한 도구
- ✓ 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

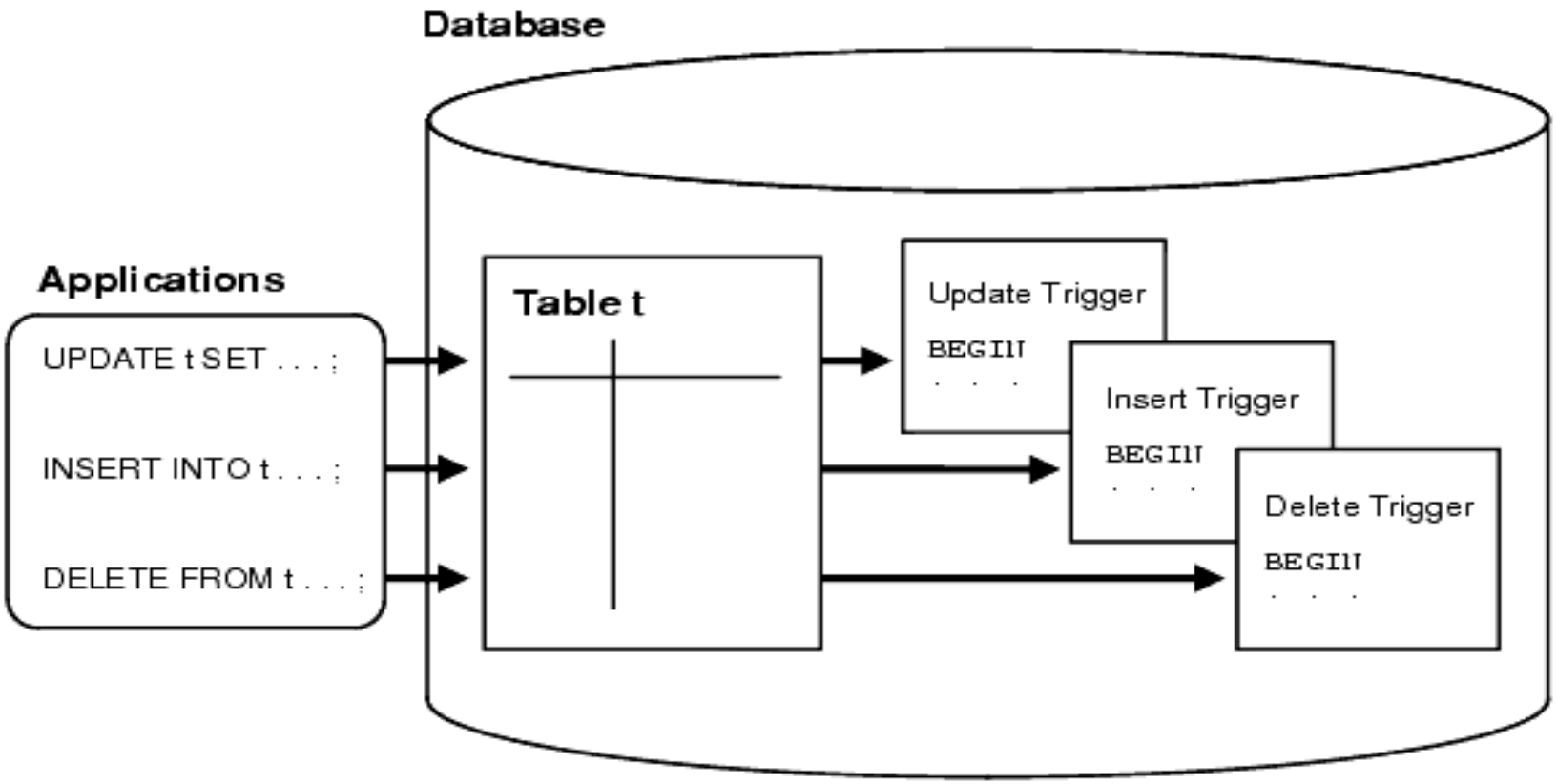
  E는 Event, C는 Condition, A는 Action을 의미
- ✓ SQL3 표준에 포함되었으며 대부분의 상용 관계 DBMS에서 제공됨

트리거와 주장(계속)



[그림 4.11] 트리거의 개념

트리거와 주장(계속)


Database
Applications
UPDATE t SET . . . ;
INSERT INTO t . . . ;
DELETE FROM t . . . ;
Table t
Update Trigger
BEGIN
. . .
Insert Trigger
BEGIN
. . .
Delete Trigger
BEGIN
. . .

트리거와 주장(계속)

❑ 트리거(계속)

✓ SQL3에서 트리거의 형식

```
CREATE TRIGGER 〈트리거 이름〉
AFTER 〈트리거를 유발하는 이벤트들이 OR로 연결된 리스트〉 ON 〈릴레이션〉 ← 이벤트
[WHEN 〈조건〉] ← 조건
BEGIN 〈SQL문(들)〉 END ← 동작
```

✓ 이벤트의 가능한 예로는 테이블에 투플 삽입, 테이블로부터 투플 삭제, 테이블의 투플 수정 등이 있음

✓ 조건은 임의의 형태의 프레디키트

✓ 동작은 데이터베이스에 대한 임의의 갱신

✓ 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

트리거와 주장(계속)

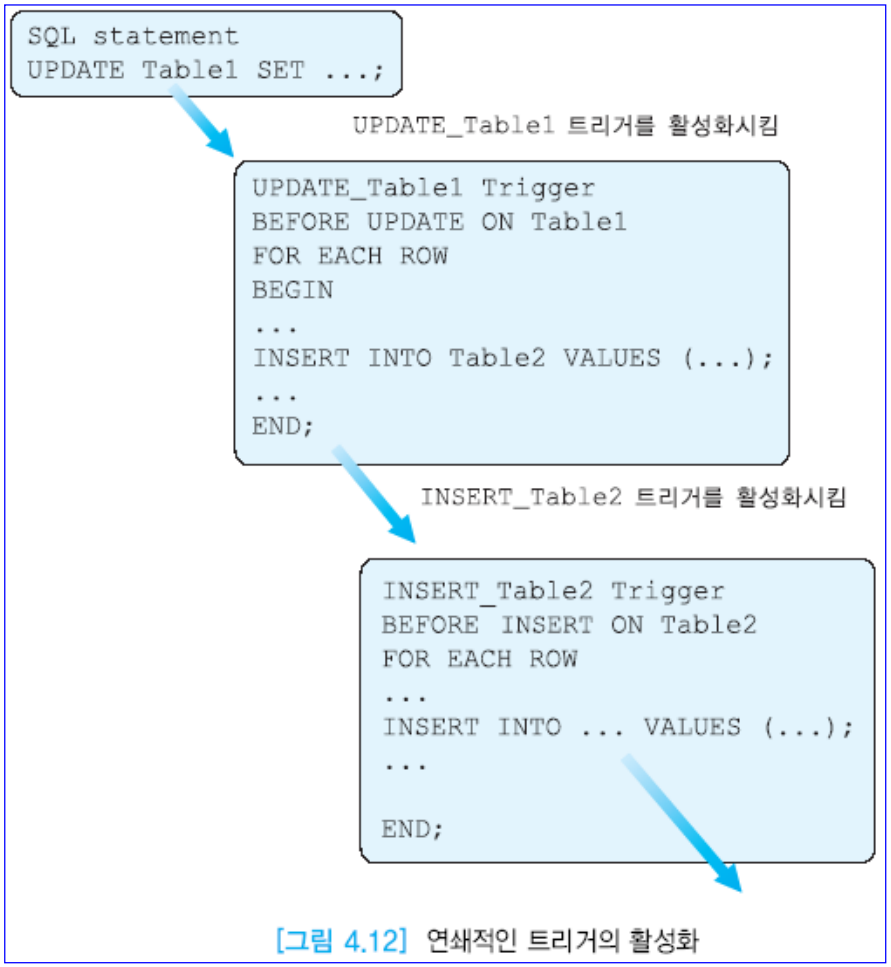
예 : 트리거

새로운 사원이 입사할 때마다, 사원의 급여가 1500000 미만인 경우에는 급여를 10% 인상하는 트리거를 작성하라. 여기서 이벤트는 새로운 사원 투플이 삽입될 때, 조건은 급여 < 1500000, 동작은 급여를 10% 인상하는 것이다. 오라클에서 트리거를 정의하는 문장은 SQL3의 트리거 정의문과 동일하지는 않다.

```
CREATE TRIGGER RAISE_SALARY
AFTER INSERT ON EMPLOYEE
REFERENCING NEW AS newEmployee
FOR EACH ROW
WHEN    (newEmployee.SALARY < 1500000)
UPDATE  EMPLOYEE
SET     newEmployee.SALARY = SALARY * 1.1
WHERE   EMPNO = newEmployee.EMPNO;
```

트리거와 주장(계속)

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



[그림 4.12] 연쇄적인 트리거의 활성화

트리거와 주장(계속)

- 주장
  - ✓ SQL3에 포함되어 있으나 대부분의 상용 관계 DBMS가 아직 지원하고 있지 않음
  - ✓ 트리거는 제약조건을 위반했을 때 수행할 동작을 명시하는 것이고, 주장은 제약조건을 위반하는 연산이 수행되지 않도록 함
  - ✓ 주장의 구문

```
CREATE ASSERTION 이름
CHECK  조건;
```

  - ✓ 트리거보다 좀더 일반적인 무결성 제약조건
  - ✓ DBMS는 주장의 프레디키트를 검사하여 만일 참이면 주장을 위배하지 않는 경우이므로 데이터베이스 수정이 허용됨
  - ✓ 일반적으로 두 개 이상의 테이블에 영향을 미치는 제약조건을 명시하기 위해 사용됨

트리거와 주장(계속)

예 : 주장

STUDENT(학생) 릴레이션과 ENROLL(수강) 릴레이션의 스키마가 아래와 같다. STUDENT 릴레이션의 기본 키는 STNO이다. ENROLL 릴레이션의 STNO는 STUDENT 릴레이션의 기본 키를 참조한다.

STUDENT(STNO, STNAME, EMAIL, ADDRESS, PHONE)
ENROLL(STNO, COURSENO, GRADE)

ENROLL 릴레이션에 들어 있는 STNO는 반드시 STUDENT 릴레이션에 들어 있는 어떤 학생의 STNO를 참조하도록 하는 주장을 정의하려 한다. 다시 말해서 STUDENT 릴레이션에 없는 어떤 학생의 학번이 ENROLL 릴레이션에 나타나는 것을 허용하지 않으려고 한다. 대부분의 주장은 아래의 예처럼 NOT EXISTS를 포함한다.

```
CREATE ASSERTION EnrollStudentIntegrity
CHECK   (NOT EXISTS
        (SELECT   *
         FROM     ENROLL
         WHERE    STNO NOT IN
                  (SELECT STNO FROM STUDENT)));
```