데이터베이스: 강의노트 05

A. Silberschatz, H. Korth, S. Sudarshan
Database System Concepts,
Fourth Edition, McGraw-Hill, 2002.

Part II. Relational Databases

6 Integrity and Security

6.1 도메인 제약 조건

- 모든 속성은 도메인을 가진다. 따라서 속성은 지정된 도메인에 의해 값이 제한된다.
- 도메인 제약 조건은 가장 기초적인 무결성 제약조건이다.
- 고객명 속성과 직원명 속성의 도메인 같지만 잔액 속성과 지점명 속성의 도메인은 다르다.
  - 고객명 속성과 지점명 속성의 도메인은 물리적 레벨에서는 같지만 논리적 레벨에서는 다르다.
- 새 도메인의 정의: create domain 절 이용

  create domain *Dollars* numeric(12,2)

  create domain *Pounds* numeric(12,2)

  두 개의 새로운 도메인을 정의하였을 때 도메인이 *Dollars*인 속성에 도메인이 *Pounds*인 변수의 값을 할당하면 비록 물리적으로는 같은 타입이지만 오류가 발생한다.
- 한 도메인의 값을 다른 도메인으로 변경: cast 이용

  cast *r.A* as *Pounds*

  *r.A*가 *Dollars* 타입일 때 이것을 *Pounds*로 변경할 때 위와 같은 cast 명령을 사용한다.
- check 절을 이용하여 도메인을 제한할 수 있다.

  create domain 시간수당 int
  constraint 수당최소값
  check(value $>= 2000$)

  여기서 '수당최소값'은 나중에 갱신 위배가 있을 때 어떤 위배인지를 나타내기 위해 사용된다.
- 예1) 널 값을 가질 수 없도록 제한하는 방법

  create domain 계좌번호 char(10)
  constraint 계좌번호널검사
  check(value not null)

- 예2) 특정한 값만을 가지도록 제한하는 방법

  create domain 계좌종류 char(10)
  constraint 계좌종류검사
  check(value in('당좌예금','저축예금'))

6.2 참조 무결성

- 참조 무결성이란 한 관계의 어떤 속성에 등장하는 값을 다른 관계의 특정 속성에 등장하는 값으로 제한하는 것을 말한다.

6.2.1 기본 개념

- **허상 투플(dangling tuple)**: 두 관계 r과 s를 조인하였을 때 r에 있는 투플 중 조인에 참여하지 못하는 투플
- 조인할 때 허상 투플 문제를 해결하기 위해 외부조인 방법을 사용할 수 있다.
- 이 절에서는 허상 투플의 허용 여부에 관한 제한조건을 살펴본다.
- 다음과 같은 허상 투플은 바람직하지 않다.
  - *account* 관계에는 지점명이 "강남점"인 계좌가 있지만 branch 관계에는 "강남점"에 관한 지점 정보가 없다.
  - 그러나 반대의 경우는 허용해야 한다.
- 위 문제를 구체적으로 살펴보면 *account* 관계에서 지점명은 외부키이지만 *branch* 관계에서 지점명은 외부키가 아니다.
- $r_1(R_1)$과 $r_2(R_2)$의 주키가 각각 K_1과 K_2인 두 관계가 있다고 하자. $\alpha \subset R_2$가 다음을 만족하면 α는 관계 r_1의 주키 K_1을 참조하는 외부키라 한다.

  r_2에 있는 모든 투플 t_2에 대해 r_1에 다음을 만족하는 투플 t_1이 반드시 존재해야 한다.

  $$t_1[K_1] = t_2[\alpha]$$

- 이런 요구사항을 **참조 무결성 제약조건(referential integrity constraints)** 또는 **부분집합 종속성(subset dependency)**이라 한다.
- 참조 무결성 제약조건은 다음과 같이 요약할 수 있다.

  $$\Pi_\alpha(r_2) \subseteq \Pi_K(r_1)$$

- E-R 모델에서 관계 집합과 약한 관계 집합을 테이블로 표현하면 반드시 외부키를 가지게 된다.

6.2.2 데이터베이스 변경

- 데이터베이스를 변경하면 참조 무결성을 위배할 수 있다.
- r_2가 r_1을 참조하는 경우에 가능한 참조 무결성 위배
  - t_2를 r_2에 삽입: 다음이 반드시 성립해야 한다.

    $$t_2[\alpha] \in \Pi_K(r_1)$$

- 예1) r_1이 *branch* 관계이고, r_2과 *account* 관계일 때 r_2에 지점이 천안점인 계좌를 추가하면 r_1에 천안점이 주키인 투플이 있어야 한다.
- t_1를 r_1에서 삭제: 다음을 계산하여

$$\sigma_{\alpha=t_1[K]}(r_2)$$

결과 관계가 비어있지 않으면 삭제를 거부하거나 r_2에서 t_1을 참조하는 투플을 함께 모두 삭제해야 한다.

- 예2) r_1에 지점이 강남점인 튜플을 삭제하면 r_2에 지점이 강남점인 계좌가 있는지 살펴보고 있으면 거부하거나 지점이 강남점인 모든 계좌를 삭제한다.
- r_2에 있는 t_2의 α에 포함된 속성을 갱신: t_2'가 t_2를 갱신한 투플이면 다음이 반드시 성립해야 한다.

$$t_2'[\alpha] \in \Pi_K(r_1)$$

- 예3) r_2에 지점이 강남점인 계좌를 천안점으로 갱신하면 r_1에 천안점이 주키인 투플이 있어야 한다.
- r_1에 있는 t_1의 주키의 내용을 갱신: 다음을 계산하여

$$\sigma_{\alpha=t_1[K]}(r_2)$$

결과 관계가 비어있지 않으면 갱신을 거부하거나 r_2에서 t_1을 참조하는 투플을 함께 모두 갱신해야 한다.

- 예4) r_1에 강남점이라는 지점이 지점명을 강남1점으로 바꾸면 r_2에 지점이 강남점인 계좌가 있는지 살펴보고 있으면 거부하거나 지점이 강남점인 모든 계좌의 지점명을 강남1점으로 바꾼다.

6.2.3 SQL에서 참조 무결성

- SQL에서는 스키마를 정의할 때 외부키를 명시할 수 있다.
  - 예) *account* 관계의 지점명 속성의 정의
    지점명 char(15) references *branch*
- SQL에서 참조 무결성이 위배되면 기본적으로는 그 연산은 거부된다. 거부하지 않고 관련 투플을 삭제 또는 갱신하고자 하면 다음과 같이 속성을 정의한다.

```
create table account (
    계좌번호 char(10),
    지점명 char(15),
    잔액 integer,
    primary key (계좌번호),
    foreign key (지점명) references branch
        on delete cascade
        on update cascade,
    check(잔액 >= 0))
```

- cascade 대신에 다음을 사용할 수 있다.
  - set null: 삭제하는 대신에 참조하는 필드를 null 값으로 바꾼다.
  - set default: 도메인의 디폴트 값으로 필드를 바꾼다.
- null 값은 참조 무결성을 복잡하게 한다. 일반적으로 외부키도 널 값을 가질 수 없도록 하는 것이 바람직하다.
- 참조 무결성은 연산 단위로 검사하지 않고 트랜잭션 단위로 검사한다.
  - 예) 결혼 관계

6.3 Assertions

- **단정(assertion)**이란 데이터베이스가 항상 만족해야 하는 조건을 명시한 조건 술어(predicate)를 말한다.
- 도메인 제약조건과 참조 무결성 제약조건은 단정의 한 특수한 형태이다.
- SQL에서 단정문의 형태

```
create assertion <assertion-name>
        check <predicate>
```

- SQL에는 “for all X, P(X)”라는 구성이 없으므로 “not exists X such that not P(X)” 구성을 사용해야 한다.
- 예) 지점의 대출액 총액은 지점의 계좌잔액의 총액보다 적어야 한다.

```
create assertion sum-constraint check
    (not exists (select * from branch
        where (select sum(대출액) from loan
          where loan.지점명 = branch.지점명)
          >= (select sum(잔액) from account
          where account.지점명 = branch.지점명)))
```

- 단정문을 선언하면 시스템은 먼저 단정문의 유효성을 검사한다. 단정문이 유효하면 이후 데이터베이스 변경은 단정문을 위배하지 않는 경우에만 허용된다.

6.4 트리거

- **트리거(trigger)**는 데이터베이스에 대한 변경의 side effect로 자동으로 수행되는 문장을 말한다.
- 트리거 메커니즘의 구성
  - 트리거가 실행될 조건: 조건의 지정은 **이벤트**와 이벤트가 발생하였을 때 만족해야 하는 **조건** 두 가지를 지정한다.
  - 트리거가 실행되었을 때 취할 **행동**

이런 트리거 모델을 **event-condition-action** 모델이라 한다.

```
create trigger 자동대출 after update on account
referencing new row as nrow
for each row
when nrow.잔액 < 0
begin atomic
    insert into borrower
        (select 고객명, 계좌번호
        from depositor
        where nrow.계좌번호 = depositor.계좌번호);
    insert into into loan values
        (nrow.계좌번호,nrow.지점명,−nrow.잔액);
    update account
        set balance = 0
        where account.계좌번호 = nrow.계좌번호
end
```

<그림 6.1> 트리거 예제

6.4.1 트리거의 필요성

- 트리거는 사용자에게 어떤 문제 상황을 알리는 수단으로 또는 어떤 조건이 만족되었을 때 자동으로 어떤 작업을 수행하는 수단으로 유용하다.
- 예1) 계좌의 잔액이 음의 값을 가지지 못하도록 하는 대신에 음의 값을 가지게 되면 그것에 해당하는 대출 계좌를 만들도록 할 수 있다.
  - 사건: 인출
  - 조건: 인출 후 잔액이 음수가 되면
  - 행동: 1) *loan* 관계에 새 투플 추가, 2) *borrwer* 관계에 새 투플 추가, 3) 잔액을 0으로 설정
- SQL 예) 그림 6.1
  - *account* 관계에 대한 갱신이 있을 때마다 이 트리거가 조건을 검사한다.
  - 한 번에 여러 행이 갱신될 수 있으므로 갱신되는 각 행마다 조건을 검사해야 한다. 이 때 각 행이 nrow가 된다.
- SQL에서 트리거 작성할 때 사용할 수 있는 선택사항
  - 가능한 사건: update, insert, delete
  - 갱신에 대한 트리거의 경우에는 특정 속성을 지정할 수 있다. 속성을 지정하면 이 속성이 갱신될 때에는 트리거가 실행된다.

    create trigger 자동대출
    after update of 잔액 on *account*
  - referencing new row as는 갱신과 삽입할 때 사용되며, 삭제할 때에는 referencing old row as를 사용한다. 또한 갱신할 때도 새 값을 비교하는 것이 아니라 갱신 전 값에 따라 트리거를 실행해야 하는 경우에는 referencing old row as를 사용한다.
  - 트리거는 사건이 발생한 후뿐만 아니라 발생하기 전에 사용할 수 있다. 이 때는 after 대신에 before를 사용한다.
  - 변경이 적용된 각 투플에 대해 트리거를 실행하지 않고 한 SQL 문장에 대해 단일 행동을 하도록 할 수 있다. 이 때는 for each row 대신에 for each statement를 사용한다. 전자는 행 수준 트리거라 하며, 후자는 문장 수준 트리거라 한다. 하나의 SQL 문장으로 여러 행을 수정하였으면 행 수준 트리거는 각 행마다 실행된다. 반면에 문장 수준 트리거는 한 번만 실행된다.
  - for each statement을 사용하면 referencing old table as 또는 referencing new table as를 사용할 수 있다. 이 임시 테이블은 영향을 받는 모든 투플로 구성된 테이블이며, 이행 테이블(transition table)이라 한다. 이행 테이블을 사용할 경우에는 after만 사용할 수 있다.
  - 예2)

    create trigger abc
    after update on *account*
    referencing
        old table as oldTab
        new table as newTab
    for each statement
    ...

    여기서 oldTab은 갱신된 투플의 옛 버전으로 구성된 테이블이고, newTab은 갱신된 투플들로만 구성된 테이블이다.

6.5 보안과 인가

6.5.1 보안 위협

- 악의적인 접근의 종류
  - 비인가된 데이터 읽기
  - 비인가된 데이터 수정
  - 비인가된 데이터 파괴
- 데이터베이스 보안은 이와 같은 악의적인 접근으로부터 데이터베이스를 보호하는 것이다.
- 데이터베이스를 보호하기 위해서는 여러 레벨에서 보안 수단을 강구해야 한다.
  - 데이터베이스 시스템: 사용자의 인가(authorization) 제약조건의 위배되지 않도록 해야 한다.
  - 운영체제: 데이터베이스 시스템이 안전하여도 운영체제가 안전하지 않으면 데이터베이스를 우회하여 비인가된 접근을 할 수 있다.
  - 네트워크: 대부분의 데이터베이스는 원격에 있는 클라이언트로부터 접속을 허용한다. 따라서 이에 대한 보안이 필요하다.

  - 물리적: 컴퓨터가 설치된 장소에 대한 보안을 말한다.
  - 사용자: 내부 사용자 공격에 대해 방어하는 것이 가장 어렵다.

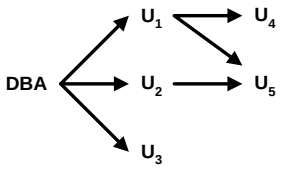
6.5.2 인가

- 데이터 조작과 관련하여 사용자에게 위임할 수 있는 권한의 종류
  - 읽기 권한: 데이터의 읽기는 가능하지만 수정은 할 수 없다.
  - 삽입 권한: 새 데이터의 삽입은 가능하지만 수정은 할 수 없다.
  - 갱신 권한: 수정은 할 수 있지만 데이터를 삭제할 수 없다.
  - 삭제 권한: 데이터를 삭제할 수 있다.

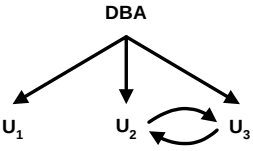
  사용자에게 이런 모든 권한을 줄 수 있고, 일부만 줄 수 있다.
- 스키마 조작과 관련하여 사용자에게 위임할 수 있는 권한의 종류
  - 색인 권한: 색인의 생성과 삭제가 가능하다.
  - 자원 권한: 새 관계의 생성이 가능하다.
  - 변경 권한: 관계에 새 속성을 추가하거나 관계의 기존 속성을 삭제할 수 있다.
  - 삭제 권한: 관계를 삭제할 수 있다.
- 자원 권한을 가지고 있는 사람이 새 관계를 생성하면 그 사람으로 자동으로 그 관계와 관련된 모든 권한을 받는다.

6.5.3 인가와 뷰

- 뷰를 사용하는 주 목적 중 하나는 사용자가 볼 필요가 없는 데이터를 사용자로부터 숨기는 것이다.
- 사용자가 관계를 접근할 직접적인 권한이 없어도 그 관계로부터 만든 뷰에 대한 접근 권한은 가질 수 있다.
- 자원 권한을 가지고 있지 않아도 뷰를 생성할 수 있다.
- 뷰를 생성한 사용자는 기본적으로 기존에 가지고 있는 권한에 해당하는 권한만 받는다.
  - r 관계에 대한 읽기 권한이 있는 사용자가 r 관계를 이용하여 뷰 v를 생성하면 이 사용자는 v에 대해 읽기 권한만 획득한다.
  - r 관계에 대한 어떤 권한도 없는 사용자가 r 관계를 이용하여 뷰 v를 생성하고자 하면 거부된다.



<그림 6.2> 권한 위임 그래프



<그림 6.3> 권한 철회에 대한 공격

6.5.4 권한 위임

- 어떤 권한을 가지고 있는 사용자는 다른 사용자에게 권한을 위임할 수 있다.
- 권한의 위임은 **권한 그래프(authorization graph)**로 표현할 수 있다. 이 그래프의 노드는 사용자를 나타내며, 화살표(edge)는 권한을 위임한 사용자와 위임받은 사용자 간에 관계를 나타낸다. 데이터베이스 관리자가 그래프의 루트가 된다. 루트에서 사용자까지 경로(path)가 있어야지만 사용자는 그 권한을 가진다.
  - 예) 그림 6.2

    이 그래프에서 데이터베이스 관리자(DBA)가 U_1의 권한을 철회(revocation)하면 U_4의 권한도 자동으로 철회된다. 하지만 U_5는 U_2로부터도 권한을 받았으므로 권한을 유지한다.
- 권한 철회를 우회하기 위해 두 사용자가 서로에게 권한을 줄 수 있다.

6.5.5 역할에 대한 개념

- 데이터베이스에 권한과 관련된 역할(role)을 정의하고, 사용자 대신에 역할에 권한을 위임할 수 있다.
- 새 사용자에게는 개별 권한을 위임하기 보다는 역할을 부여하여 기존 다른 사용자와 동일한 권한을 가지도록 할 수 있다.
- 같은 역할을 하는 사용자들에게 같은 식별자를 부여하고 모두 같은 식별자로 접속하도록 할 수 있다. 그러나 이 경우에는 나중에 어떤 사용자가 어떤 행위를 하였는지 구분할 수 없다.

6.5.6 감사 추적

- 데이터베이스는 **감사 추적(audit trail)**을 유지한다. 감사 추적이란 데이터베이스 변경에 대한 기록(행위,사용자,시간 등)을 유지하는 로그이다.

- 감사 추적과 보안: 문제가 발생하였을 때 발생 원인과 그것을 일으킨 사용자를 알아낼 수 있다.

6.6 SQL에서 인가

6.6.1 SQL에서 권한

- 표준 SQL에서 제공되는 권한의 종류: delete, insert, select, update, references, grant
  - select 권한: 읽기 권한에 해당
  - references 권한: 관계를 만들 때 외부키를 사용할 수 있는 권한을 말한다.
  - grant 권한: 권한을 위임할 수 있는 권한을 말한다. 사용자는 기본적으로 권한을 다른 사용자나 역할에게 위임할 수 없다.
- 권한 위임: grant 명령

  grant $<$ 권한 목록 $>$
  on $<$ 관계 이름 또는 뷰 이름 $>$
  to $<$ 사용자 또는 역할 목록 $>$

- 예) 사용자 U_1, U_2, U_3에게 *account* 관계에 대한 select 권한을 위임하라.

  grant select on *account* to U_1, U_2, U_3

- update 권한을 위임할 때에는 수정할 수 있는 속성을 지정할 수 있다.

  예) 사용자 U_1, U_2, U_3에게 *loan* 관계 중 대출액 속성에 대한 update 권한을 위임하라.

  grant update (대출액) on *loan* to U_1, U_2, U_3

- references 권한은 참조할 수 있는 주키를 지정해야 한다.

  예) 사용자 U_1에게 *branch* 관계의 주키인 지점명을 참조할 수 있는 권한을 위임하라.

  grant references (지점명) on *branch* to U_1

  references 권한이 필요한 이유는 참조 무결성에서 살펴본 바와 같이 외부키를 사용하면 삭제와 갱신 연산에 영향을 주기 때문이다.

- 모든 권한을 위임할 때에는 all privileges을 사용할 수 있다.
- 현재 사용자 뿐만 아니라 미래의 사용자에게도 모두 권한을 줄 때에는 사용자 식별자 대신에 public을 사용할 수 있다.
- 역할의 생성은 다음과 같이 한다.

  create role *teller*

  이 역할에 권한 위임은 grant 명령을 이용한다.

  grant select on *account* to *teller*

  어떤 사용자에게 역할을 할당할 때에도 grant 명령을 이용한다.

  grant *teller* to john

- grant 권한의 위임: with grant option 사용

  grant select on *account* to U_1 with grant option

- 권한의 철회: revoke 명령 사용

  revoke $<$ 권한 목록 $>$
  on $<$ 관계 이름 또는 뷰 이름 $>$
  from $<$ 사용자 또는 역할 목록 $>$
  [restrict|cascade]

  cascade란 권한의 철회로 다른 사용자의 권한까지 자동(연쇄 철회)으로 철회되는 것을 말한다. cascade가 기본이므로 이것을 원하면 생략할 수 있다. restrict를 사용하면 먼저 이것과 연관하여 자동으로 철회할 것이 있는지 검사하고 있으면 거부한다.

6.6.2 SQL 인가의 한계

- 개별 투플에 대한 권한을 위임할 수 없다.
- 웹의 경우에는 개별 식별자 대신에 보통은 하나의 식별자를 통해 접근하도록 한다. 따라서 SQL에서 권한을 검사하기 보다는 응용 서버에서 권한을 검사한다. 그러나 이렇게 하면 허점이 존재할 확률이 많다.