

# 7장 . 제약과 트리거

---

- ◆ **SQL**에서의 키
- ◆ 참조 무결성(**referential integrity**)과 외래 키
- ◆ 애트리뷰트 값에 대한 제약
- ◆ 전역 제약 (**global constraint**)
- ◆ 제약의 변경
- ◆ **SQL3**에서의 트리거(**trigger**)

# SQL의 제약과 트리거

---

- 능동적 요소(**active element**) : 적절한 시점에 자동적으로 실행하는 데이터베이스에 저장된 수식 또는 문장
  - 무결성 제약 (**Integrity Constraint: IC**) 과 트리거
- 잘못된 새로운 정보에 대한 문제 해결 방법
  - 모든 삽입, 삭제, 그리고 갱신 명령에 정확성을 보장하기 위해 필요한 검사들을 적용시켜서 응용 프로그램을 작성
    - » 정확성 조건들은 관련된 모든 응용 프로그램들에 적용되어야 한다.
  - ☞ **SQL2**는 무결성 제약을 데이터베이스 스키마의 일부로서 제공
- ◆ 트리거: 특정 릴레이션에 삽입과 같은 어떤 명시된 사건들에 의해 구동되어 적절한 조치를 할 수 있는 능동적 요소
  - 트리거는 **SQL3**에서 제공됨

# SQL에서의 키

---

## ◆ 키 선언: **PRIMARY KEY**

- 애틀리뷰트의 타입 선언 다음에 **PRIMARY KEY** 라는 키워드를 덧붙여 그 애틀리뷰트가 주 키임을 선언
- 괄호로 묶인 애틀리뷰트의 리스트 앞에 **PRIMARY KEY**라는 키워드를 사용하여 그 애틀리뷰트(들)이 주 키임을 선언
  - » 주 키가 하나 이상의 애틀리뷰트들로 이루어질 때 주로 사용

```
CREATE TABLE MovieStar (  
    name CHAR(30) PRIMARY KEY,  
    address VARCHAR(255),  
    gender CHAR(1),  
    birthdate DATE);
```

```
CREATE TABLE MovieStar(  
    name CHAR(30),  
    address VARCHAR(255),  
    gender CHAR(1),  
    birthdate DATE,  
    PRIMARY KEY(name));
```

# SQL에서의 키 (계속)

---

## ◆ 키 선언: UNIQUE

- **UNIQUE** 라는 키워드를 사용

```
CREATE TABLE MovieStar (  
    name          CHAR(30)  UNIQUE,  
    address       VARCHAR(255),  
    gender        CHAR(1),  
    birthdate     DATE );
```

```
CREATE TABLE MovieStar (  
    name          CHAR(30),  
    address       VARCHAR(255),  
    gender        CHAR(1),  
    birthdate     DATE,  
    UNIQUE (name));
```

- 하나 이상의 애트리뷰트들로 이루어질 경우

**PRIMARY KEY (title,year),    UNIQUE (title,year)**

- ☞ 키 제약 : **PRIMARY KEY** 나 **UNIQUE**로 선언되는 제약

# 주 키와 **UNIQUE** 애트리뷰트

---

- ◆ 주 키 선언은 **UNIQUE** 선언과 거의 동일하다. 차이점으로는
  - 주 키는 한 테이블에 단지 하나만 있을 수 있으나 **UNIQUE** 애트리뷰트는 여러 개가 있을 수 있다.
  - 외래 키(**foreign key**)는 릴레이션의 주 키만을 참조할 수 있다.
    - ☞ 다른 후보 키를 참조하는 외래 키를 허용하는 **ORACLE**과 같은 상용 제품들도 있다.
  - 실제 구현 시의 관점
    - » 주 키에 대해 항상 색인을 만든다.

# SQL에서의 키 (계속)

---

- 많은 **SQL** 제품들이 **UNIQUE** 키워드를 사용하는 색인 생성 문장을 제공한다

```
CREATE UNIQUE INDEX YearIndex ON Movie(year)
```

## ◆ 키 제약의 시행(enforcement)

- 테이블에 삽입이나 갱신이 일어날 때 키 제약을 검사
  - » 튜플의 삭제시에는 키 제약 검사 필요없음.
- 키 제약에 대한 검사 시, 그 키에 대한 색인이 있으면 아주 효율적이다.

# 참조 무결성과 외래 키

---

## ◆ 외래 키(**foreign key**)

- 다른 릴레이션의 어떤 애트리뷰트를 참조하는 애트리뷰트(들)

## ◆ 외래 키 선언의 의미

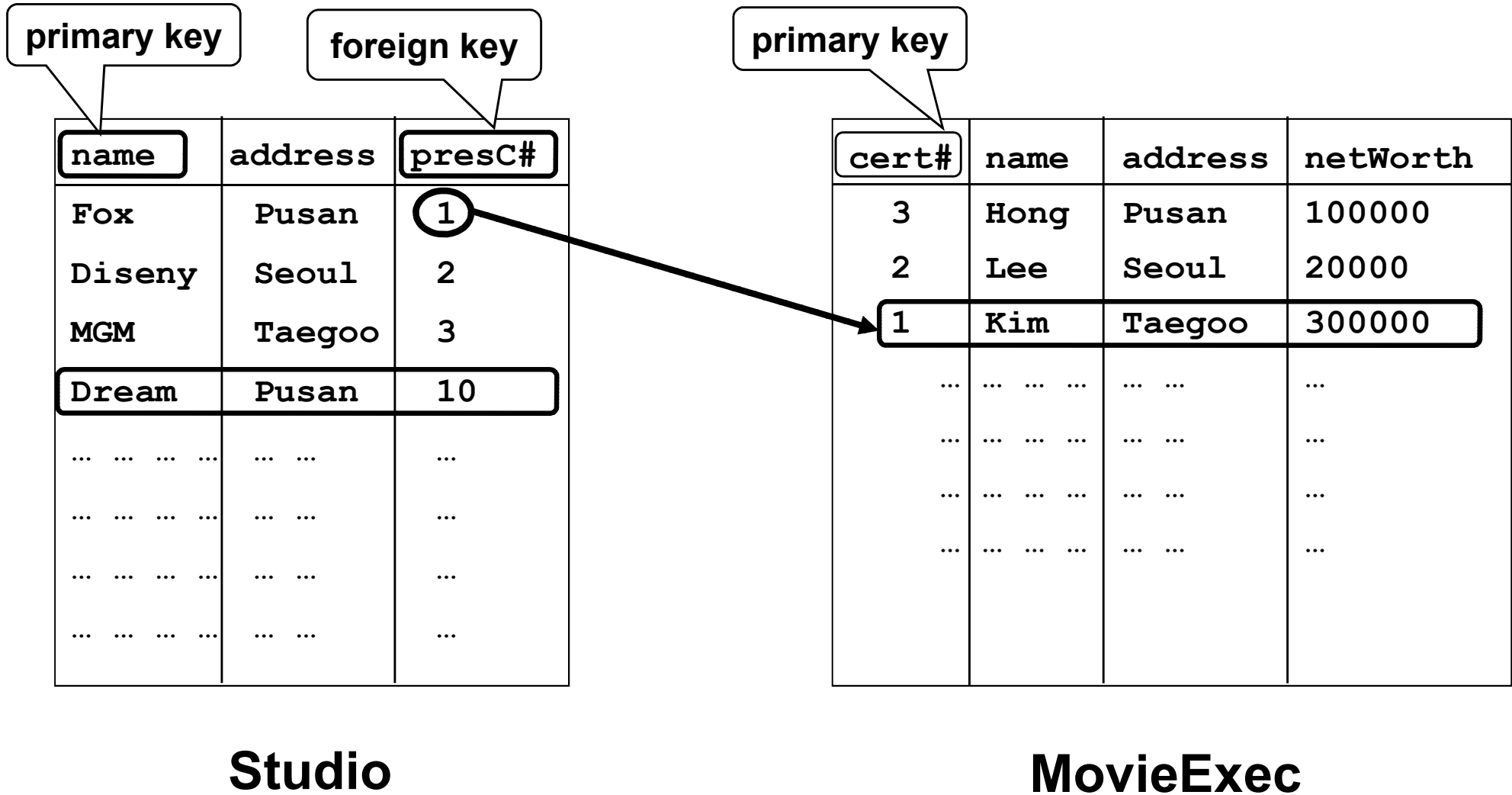
- 참조 무결성 제약(**referential integrity**) : 외래 키 제약

» 참조하는 릴레이션의 외래 키에 나타나는 애트리뷰트(들)의 값은 참조되는 릴레이션의 대응되는 애트리뷰트(들)에도 반드시 나타나야 한다.

- 참조되는 애트리뷰트(들)는 반드시 주 키여야 한다.

☞ 참조되는 애트리뷰트(들)가 후보 키(**CANDIDATE KEY**)인 것을 허용하는 상용 제품도 있다. (**Oracle**)

# 참조 무결성과 외래 키 (계속)





# 참조 무결성과 외래 키 (계속)

---

## ◆ 외래 키의 선언

- **<attr-name> <type> REFERENCES <table> (<attribute>)**
- **FOREIGN KEY <attributes> REFERENCES <table> (<attributes>)**

```
CREATE TABLE Studio (  
    name      CHAR(30)  PRIMARY KEY,  
    address   VARCHAR(255),  
    presC#    INT REFERENCES  
        MovieExec(cert#));
```

```
CREATE TABLE Studio (  
    name      CHAR(30)  PRIMARY KEY,  
    address   VARCHAR(255),  
    presC#    INT,  
    FOREIGN KEY presC# REFERENCES  
        MovieExec(cert#));
```

- ☞ 어떤 studio 튜플이 presC# 요소의 값으로 NULL을 갖더라도, NULL이 MovieExec 안의 cert# 요소에 나타날 필요는 없다.(그러나 사실상 주 키 애트리뷰트에 NULL값이 허용되지 않는다.)

# 참조 무결성의 유지

---

## ◆ 디폴트(default) 정책

### – 위반하는 변경을 거부

» 참조하는(referencing) 릴레이션 : 참조 무결성을 위반하는 삽입이나 갱신 거부

» 참조되는(referenced) 릴레이션 : 참조 무결성을 위반하는 삭제나 갱신 거부

[예1] Studio에 ('Dream', 'Pusan', 10)을 삽입시 MovieExec에 presC#이 10인 튜플이 존재하지 않으면 실패

[예2] MovieExec의 (1, 'Kim', 'Taegoo', 300000)을 삭제시 Studio의 어떤 튜플도 cert#로 1을 가지지 말아야 성공

## ◆ 연쇄(cascade) 정책

### – 연쇄 삭제(cascade deletion)

» 참조되는 튜플을 삭제하면 참조하는 튜플도 삭제

### – 연쇄 갱신(cascade update)

» 참조되는 튜플을 갱신하면 참조하는 튜플도 갱신



# 참조 무결성의 유지 (계속)

---

## ◆ 널-설정(set-null) 정책

- 참조되는 릴레이션에서 삭제나 갱신이 발생했을 때, 참조하는 애트리뷰트의 값을 **NULL**로 설정
  - [ON DELETE | ON UPDATE] [CASCADE | SET NULL]
    - 외래 키의 선언 시에 명시
    - ORACLE 10g에서는 ON DELETE CASCADE/SET NULL 만 제공

```
CREATE TABLE Studio (  
    name          CHAR(30)  PRIMARY KEY,  
    address       VARCHAR(255),  
    presC#       INT REFERENCES MovieExec(cert#)  
                ON DELETE SET NULL  
                ON UPDATE CASCADE  
);
```

# 허상 튜플과 변경 정책

---

## ◆ 허상(dangling) 튜플

- 참조되는 릴레이션에 나타나지 않는 외래 키 값을 가지는 튜플
- 조인에 참여하지 못하는 튜플
  - ☞ 만약 튜플의 외래 키 값이 참조되는 릴레이션에 없다면 그 튜플은 그 튜플이 속한 릴레이션과 참조되는 릴레이션의 조인에 참여하지 못한다.
- 참조 무결성을 위반하는 튜플 ◀
  - » 디폴트 정책 : 허상 튜플을 만들 경우 동작을 금지
  - » 연쇄 정책 : 모든 허상 튜플들을 삭제하거나 갱신
  - » 널-설정 정책 : 허상 튜플의 외래 키 값을 **NULL**로 설정

# 애트리뷰트 값에 대한 제약

---

## ◆ NOT NULL 제약

- 애트리뷰트가 **NULL** 인 튜플을 허용하지 않는다.
- **NOT NULL** 키워드를 사용해서 선언한다.

[예] Studio 릴레이션에서 **presC#**가 **NULL**이 아니어야 한다고 하자.

**presC# INT REFERENCES MovieExec(cert#) NOT NULL**

- » 어떤 튜플의 **presC#** 요소의 값을 **NULL**로 변경할 수 없다.
- » **presC#** 요소의 값이 **NULL**인 튜플을 삽입할 수 없다.
- » 널-설정 정책을 사용할 수 없다.

# 애트리뷰트 값에 대한 제약 (계속)

---

## ◆ 애트리뷰트-기반(attribute-based) CHECK 제약

- 애트리뷰트의 값에 대한 제한
- 키워드 **CHECK** 다음에 조건을 명시
- 튜플이 이 애트리뷰트에 대해 새로운 값을 가질 때마다 검사
  - » 애트리뷰트 값이 갱신되거나, 또는 새로운 튜플이 삽입될 때 검사
  - » 새로운 값에 의해 제약이 위반되면 그 변경은 거부

```
CREATE TABLE Studio (  
    name      CHAR(30)  PRIMARY KEY,  
    address   VARCHAR(255),  
    presC#    INT REFERENCES MovieExec(cert#)  
              CHECK (presC# >= 100000));  
  
gender CHAR(1) CHECK (gender IN ('F', 'M'))
```

# 애트리뷰트 값에 대한 제약 (계속)

---

## ◆ CHECK에 부질의(subquery)를 포함하는 조건

- 조건에는 그 릴레이션의 다른 애트리뷰트들이나 튜플들이 언급되어도 되며, 다른 릴레이션들도 언급될 수 있다.
  - » **ORACLE**을 비롯한 대부분의 **DBMS**는 **CHECK** 문장에 **subquery**를 포함하지 못한다.
- **WHERE** 뒤에 올 수 있는 어떤 것도 조건이 될 수 있다.
- ☞ 제약의 검사는 그 제약이 기술된 애트리뷰트에만 적용된다는 점에 주의해야 한다. 따라서 만일 검사되는 애트리뷰트 외의 다른 어떤 요소가 변경되면, 조건은 **FALSE**가 될 수 있다.


# 애트리뷰트 값에 대한 제약 (계속)

---

[예] 애트리뷰트-기반 CHECK 제약을 이용하여 “Studio 릴레이션의 presC# 애트리뷰트가 MovieExec 릴레이션의 cert#를 참조한다” 라는 참조 무결성 제약을 표현하려 한다고 하자.

```
CREATE TABLE Studio (  
    name          CHAR(30)  PRIMARY KEY,  
    address       VARCHAR(255),  
    presC#        INT CHECK  
                (presC# IN (SELECT cert# FROM MovieExec));
```

☞ 이 문장은 올바른 애트리뷰트 기반 **CHECK** 제약이다, 그러나 그 효과는 참조 무결성 제약과 정확히 같지는 않다.

– **MovieExec** 릴레이션의 튜플을 삭제할 때, 이 변경은 위의 **CHECK** 제약에는 보이지 않는다. 



# 애트리뷰트 값에 대한 제약 (계속)

---

## ◆ 도메인 제약

- 제약을 가진 도메인을 선언하고, 그 도메인을 애트리뷰트의 데이터 타입으로 선언함으로써 그 애트리뷰트 값을 제한
- 키워드 **VALUE** 를 사용

```
CREATE DOMAIN GenderDomain CHAR(1)
CHECK (VALUE IN ('F', 'M'));
```

```
CREATE TABLE MovieStar (
    name          CHAR(30),
    address       VARCHAR(255),
    gender        GenderDomain,
    birthdate     DATE );
```

```
CREATE DOMAIN CertDomain INT
CHECK (VALUE >= 100000);
```

# 제약이 검사되는 시점의 연기

---

- 참조하는 튜플을 삽입한 후, 참조되는 튜플을 삽입한다고 해 보자.

- 여러 관련된 변경들이 실행될 때, 하나가 위반을 일으키고 다른 것이 그 문제를 해결하는 경우가 있을 수 있다.

【예】 Studio 와 MovieExec 테이블에서 참조 무결성 제약을 생각해 보자. Studio의 presC#가 MovieExec의 cert# 를 참조하는 외래 키이다. 여기서 새로운 스튜디오와 그 스튜디오의 사장을 추가한다고 할 때 위와 같은 현상이 나타날 수 있다.

- ◆ **SQL**은 제약이 검사되는 시점을 연기(**DEFERRED**) 시킬 수 있는 기능을 제공한다.

- **DEFERRED**로 선언되면, 그 검사는 트랜잭션이 완료되는 시점에서 수행되게 된다.
  - **ORACLE**에서는 **DISABLE constraints**가 있으며 **ENABLE constraints**로 명시적으로 제약을 실행시킬 수 있다.

# 전역(Global) 제약

---

## ◆ 튜플-기반(tuple-based) CHECK 제약

- 한 릴레이션의 튜플들에 튜플 단위의 제약을 선언
- 검사되는 시점, 조건의 사용, 다른 릴레이션의 변경이 보이지 않는 것 등은 애트리뷰트-기반 CHECK 제약과 동일

[ 예 ] 남자 스타의 이름은 'Ms.' 로 시작해서는 안된다.

```
CREATE TABLE MovieStar (  
    name          CHAR(30) ,  
    address       VARCHAR(255) ,  
    gender        CHAR(1) ,  
    birthdate     DATE ,  
    CHECK (gender = 'F' OR name NOT LIKE 'Ms. %')  
);
```

# 전역 제약 (계속)

---

## ◆ 무결성 단정(assertion) : CREATE ASSERTION

**CREATE ASSERTION <name> CHECK (<condition>)**

- 한 릴레이션 전체와 관련된 제약
    - » 예를 들어, 한 열에 있는 값들의 합이나 다른 집단화에 대한 제약
  - 둘 이상의 릴레이션들과 관련된 제약
- ☞ 무결성 단정 안의 조건은 항상 참이어야 한다. 지금까지 다루었던 **CHECK** 제약들은 부질의를 포함할 경우, 어떤 상황에서는 위반될 수도 있다.
- ☞ 무결성 단정은 다른 제약들과는 달리 그 자체가 스키마의 요소이다.

# 전역 제약 (계속)

---

- 무결성 단정은 전체 릴레이션에 대한 것이며 그 조건은 **T**나 **F**값을 가져야 한다. 따라서 집단화 연산자를 사용하거나 **EXISTS** 나 **NOT EXISTS**와 같이 릴레이션에 적용되어 **TRUE**를 결과로 생성하는 연산자를 사용하는 것이 일반적이다.
  - ☞ 전체 릴레이션에 대한 가장 일반적인 제약의 형태는  $C = \phi$ 이다.

[예] 재산이 적어도 \$10,000,000이 안되면 스튜디오의 사장이 될 수 없다.

Studio (name, address, presC#),

MovieExec (name, address, cert#, netWorth)

```
CREATE ASSERTION RichPres CHECK
  (NOT EXISTS
    (SELECT *
     FROM Studio, MovieExec
     WHERE presC# = cert# AND netWorth < 10000000 ));
```

# 전역 제약 (계속)

---

[예] 튜플-기반 CHECK 제약을 사용한, 유사하지만 동일하지는 않은 제약.

```
CREATE TABLE Studio (  
    name          CHAR(30)  PRIMARY KEY,  
    address       VARCHAR(255),  
    presC#       INT REFERENCES MovieExec (cert#),  
    CHECK (presC# NOT IN  
          (SELECT cert# FROM MovieExec  
           WHERE networth < 10000000));
```

- 어떤 스튜디오 사장의 재산 수 없다.
- 따라서 무결성 단정과 동일 선언에 또 다른 제약을 추가

```
CREATE TABLE MovieExec (  
    cert#        INT PRIMARY KEY,  
    name         CHAR(30),  
    address      VARCHAR(255),  
    netWorth     INT,  
    CHECK ( (cert# IN  
            (SELECT presC# FROM Studio)) AND  
            netWorth >= 10000000 );
```

# 전역 제약 (계속)

---

[예] **Movie** (title, year, length, inColor, studioName, producer#)  
한 스튜디오에서 제작한 모든 영화들의 총 상영시간이 10,000분을  
초과해서는 안된다.

```
CREATE ASSERTION SumLength CHECK (10000 >= ALL
    (SELECT SUM(length) FROM Movie GROUP BY studioName));
```

- **Movie** 테이블에 튜플-기반 **CHECK** 제약으로 표현

```
CHECK (10000 >= ALL
    (SELECT SUM(length) FROM Movie GROUP BY studioName));
```

- 튜플-기반 **CHECK** 제약으로 구현되면 튜플의 삭제 시에는 검사가 이루어지지 않는다. 그러므로 만약 위의 제약이 총 상영시간에 대한 하한 값 이었다면(즉,  $\leq$ ) 이 제약은 위반될 수도 있다.

# 전역 제약 (계속)

## ◆ 제약들의 비교

제약의 형태	선언되는 장소	활성화되는 시점	성립의 보장 여부
애트리뷰트 기반 <b>CHECK</b>	애트리뷰트와 함께	릴레이션 삽입시 또는 애트리뷰트 갱신시	부질의를 사용할 경우 보장되지 않음
튜플 기반 <b>CHECK</b>	릴레이션 스키마의 요소	릴레이션 삽입시 또는 애트리뷰트 갱신시	부질의를 사용할 경우 보장되지 않음
무결성 단정	데이터베이스 스키마의 요소	참조되는 릴레이션 에 대한 변화시	항상 보장됨



# 제약의 변경

---

## ◆ 제약에 이름 부여

- 존재하는 제약을 변경하거나 삭제하기 위해서는 그 제약이 이름을 가지고 있어야 한다.

**CONSTRAINT <name> <constrains>**

- **name CHAR(30) CONSTRAINT NameIsKey PRIMARY KEY**
- **gender CHAR(1) CONSTRAINT NoAndro CHECK(gender IN ('F', 'M'))**
- **CREATE DOMAIN CertDomain INT**

**CONSTRAINT SixDigits CHECK(VALUE >= 100000)**

- **CONSTRAINT RightTitle**

**CHECK (gender = 'F' OR name NOT LIKE 'MS. %')**

# 제약의 변경 (계속)

## ◆ 테이블에 대한 제약의 변경: ALTER TABLE

- 애트리뷰트-기반 검사와 튜플-기반 검사에 모두 적용 가능

**ALTER TABLE <table name> DROP/ADD CONSTRAINT <constraint name>**

- ALTER TABLE MovieStar DROP CONSTRAINT NameIskey;
- ALTER TABLE MovieStar ENABLE CONSTRAINT NoAndro;
- ALTER TABLE MovieStar DISABLE CONSTRAINT RightTitle;
- ALTER TABLE MovieStar ADD ( CONSTRAINT NameIsKey  
PRIMARY KEY (name));
- ALTER TABLE MovieStar ADD ( CONSTRAINT NoAndro  
CHECK(gender IN ('F', 'M')));
- ALTER TABLE MovieStar ADD ( CONSTRAINT RightTitle  
CHECK(gender = 'F' OR name NOT LIKE 'Ms.%' ));

Oracle 에서 ...

이 제약들은  
튜플-기반이  
된다

# 제약의 변경 (계속)

---

## ◆ 도메인 제약의 변경 : **ALTER DOMAIN**

- 튜플-기반 검사를 삭제하거나 추가할 때와 같은 방법을 사용
  - **ALTER DOMAIN CertDomain DROP CONSTRAINT SixDigits;**
  - **ALTER DOMAIN CertDomain ADD CONSTRAINT SixDigits**  
**CHECK(VALUE >= 100000) ;**

## ◆ 무결성 단정의 변경: **DROP ASSERTION**

- **DROP ASSERTION** 문장으로 무결성 단정을 삭제  
**DROP ASSERTION RichPres ;**

# SQL3에서의 트리거(Trigger)

---

## ◆ 트리거와 제약

- 트리거는 사건-조건-조치 규칙(**Event-Condition-Action rule**) 또는 **ECA** 규칙이라고 불린다.

## ◆ 다른 제약들과의 차이점

- **DB** 프로그래머에 의해 명시된 어떤 사건(**event**)이 발생할 때에만 테스트 된다.
  - » 명시할 수 있는 사건 : 릴레이션에 대한 삽입, 삭제, 또는 갱신
- 트리거의 조건이 만족되면 관련된 조치(**action**)가 수행된다.
  - » 조치는 임의의 데이터베이스 연산
- 트리거는 조건의 만족여부에 관계없이 어떤 사건이 실행되도록 하거나 실행되지 않도록 할 수도 있다.
  - » **NOTE** : 제약에서는 그 조건이 만족되지 않으면 항상 그 사건이 금지된다.

# SQL3에서의 트리거 (계속)

---

[예] 스튜디오 임원의 재산을 감소시키는 연산을 원래의 상태로 복귀시킨다.

```
CREATE TRIGGER NetWorthTrigger
AFTER UPDATE OF netWorth ON MovieExec      /* 사건 */
REFERENCING
    OLD AS OldTuple,
    NEW AS NewTuple
WHEN (OldTuple.netWorth > NewTuple.netWorth) /* 조건 */
UPDATE MovieExec                             /* 조치 */
SET netWorth = OldTuple.netWorth
WHERE cert# = NewTuple.cert#
FOR EACH ROW                                /* 튜플-단위 트리거 */
```

# SQL3에서의 트리거 (계속)

---

## ◆ SQL3 트리거의 특징

- 조치는 사건의 전이나, 후 또는 그 사건 대신 실행될 수 있다.
- 조치는 사건에 의해 삽입이나 삭제 또는 갱신된 이전 튜플과 새 튜플 모두를 참조할 수 있다.
- 갱신 사건은 특정 열(들)을 명시할 수 있다.
- 트리거의 조건은 **WHEN** 절에서 명시된다.
- 조치가 실행되는 방식
  - » 튜플-단위(**tuple-level**) 트리거 : 변경된 각 튜플에 대해 한번씩 실행
  - » 문-단위(**statement-level**) 트리거 : 한 데이터베이스 연산에서 변경된 모든 튜플들에 대해 한 번만 실행

# SQL3에서의 트리거 (계속)

---

## ◆ BEFORE, AFTER, INSTEAD OF

- **BEFORE** : 사건 이전에 **WHEN** 조건을 검사
  - » **WHEN** 조건을 만족시 조치가 수행됨
  - » 그 후에, 사건(**INSERT**, **DELETE**, **UPDATE**)은 **WHEN** 조건의 결과와 상관없이 발생함
- **AFTER** : 사건 이후에 **WHEN** 조건을 검사
- **INSTEAD OF** : **WHEN** 조건이 만족하면 조치가 수행된다. 그러나 사건은 조건의 만족 여부에 상관없이 수행되지 않는다.

## ◆ “UPDATE OF” 에서의 “OF”는 INSERT나 DELETE에서는 허용되지 않는다.

## ◆ OLD AS, NEW AS

- 갱신 이전 튜플과 갱신 이후 튜플은 **OLD AS**나 **NEW AS**절을 이용하여 이름을 부여한다.
- 사건이 삽입(또는 삭제)일 경우, **OLD AS** (또는 **NEW AS**)는 허용되지 않는다.

# SQL3에서의 트리거 (계속)

---

## ◆ 튜플-단위 트리거, 문-단위 트리거

### – 튜플-단위 트리거 : **FOR EACH ROW**

» 변경된 각 튜플에 대해 한번씩 실행

### – 문-단위 트리거 (**FOR EACH ROW**를 생략할 경우)

» 사건을 생성하는 각 문장에 한번씩 만 실행

» **OLD AS** (또는 **NEW AS**) 대신에 **OLD\_TABLE AS** (또는 **NEW\_TABLE AS**)를 사용



# SQL3에서의 트리거 (계속)

---

[예] 영화 임원의 평균 재산이 \$500,000이하가 되는 것을 금지시킨다.

- ❏ MovieExec(name, address, cert#, netWorth)의 netWorth 열에 대한 삽입, 삭제, 또는 갱신 사건 각각에 대해 하나씩 트리거를 작성한다.
- ❏ INSERT, DELETE, UPDATE 각각에 대해 조치가 다르기 때문에 세 개의 트리거가 필요

```
CREATE TRIGGER AvgNetWorthTrigger
INSTEAD OF UPDATE OF netWorth ON MovieExec
REFERENCING
    OLD_TABLE AS OldStuff
    NEW_TABLE AS NewStuff
WHEN (500000 <=
    (SELECT AVG(netWorth)
     FROM ((MovieExec EXCEPT OldStuff) UNION NewStuff))
DELETE FROM MovieExec
WHERE (name, address, cert#, netWorth) IN OldStuff;

INSERT INTO MovieExec
    (SELECT * FROM NewStuff)
```

**INSTEAD OF : netWorth 열을**  
갱신하려는 연산들의 수행은  
데이터베이스에 반영되지 않는다

# SQL3에서의 무결성 단정

## ◆ SQL3에서의 무결성 단정 : SQL2 무결성 단정의 확장

- 프로그래머가 명시한 사건에 의해 트리거 된다.
- (테이블 전체가 아니라) 테이블의 각 튜플을 참조할 수도 있다.

[예] 재산이 최소 \$10,000,000이 되지 못하면 스튜디오 사장이 될 수 없다고 하자.

MovieExec(name, address, cert#, netWorth), Studio(name, address, presC#)

```
CREATE ASSERTION RichPres
AFTER
```

```
    INSERT ON Studio,
    UPDATE OF cert# ON MovieExec,
    UPDATE OF presC# ON Studio,
    UPDATE OF netWorth ON MovieExec,
    INSERT ON MovieExec
```

```
CHECK (NOT EXISTS
    (SELECT * FROM Studio, MovieExec
    WHERE presC# = cert# AND netWorth < 10000000))
```

사용자는 제약을 트리거 시킬 수  
있는 모든 사건들을 찾아내야 한다.

