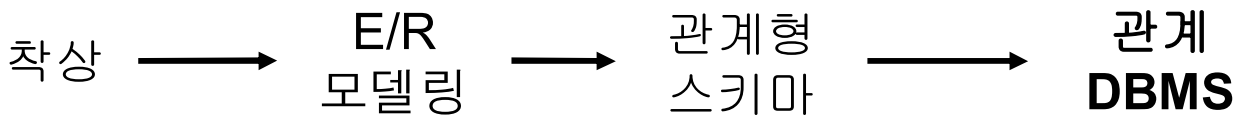
3장. 데이터 모델링과 MovieDB

- **E/R 모델의 요소**
- **E/R 다이어그램**
- **관계성의 표현**
- **설계 원칙**
- **제약의 모델링**
- **E/R 모델과 관계형 모델**
- **MovieDB Schema**

데이터베이스 모델링

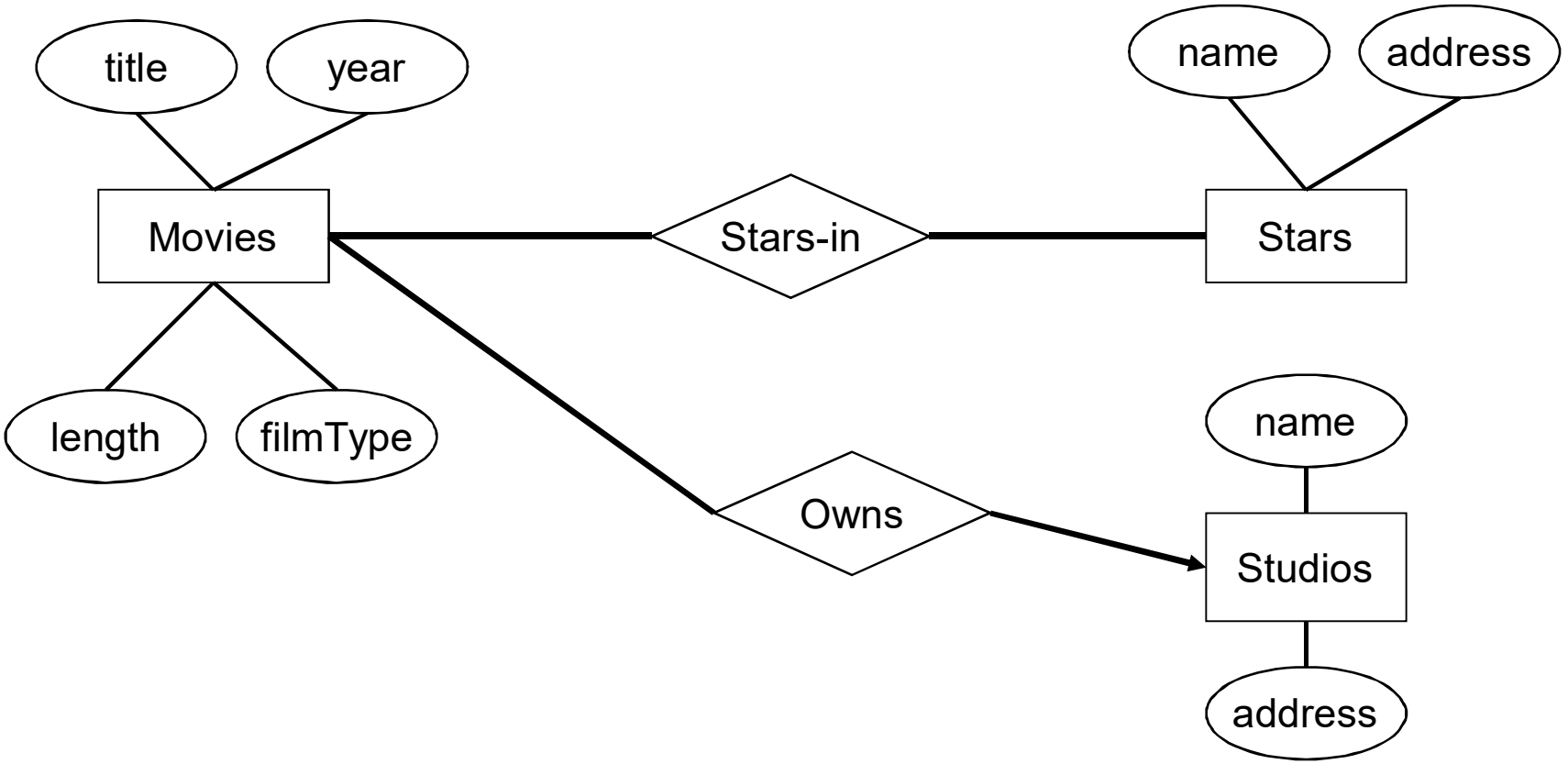
- **데이터 모델(data model)**
  - 데이터와 데이터들간의 관계를 기술하는 개념적 도구
  - 데이터베이스의 논리적 구조를 명시
- **데이터베이스 모델링과 구현 과정**



엔티티-관계성 다이어그램

- 엔티티-관계성(Entity-Relationship:E/R) 다이어그램
  - 데이터베이스 모델링을 그래프로 표현
- 엔티티-관계성 다이어그램의 세 가지 구성요소
  - 엔티티 집합(entity set) : 클래스, 엔티티는 객체에 해당
  - 애트리뷰트 : 엔티티의 특성을 기술하는 값
  - 관계성 : 둘 이상 엔티티 집합들간의 연결

엔티티-관계성 다이어그램 (계속)

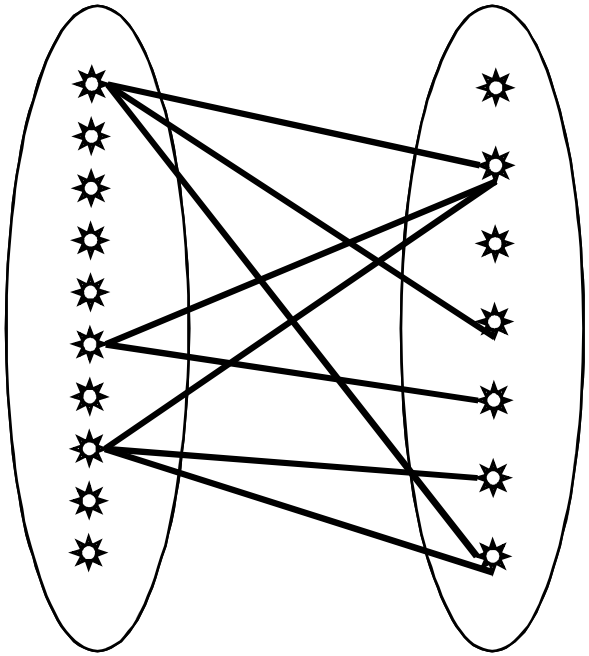


영화 데이터베이스를 위한 **E/R** 다이어그램

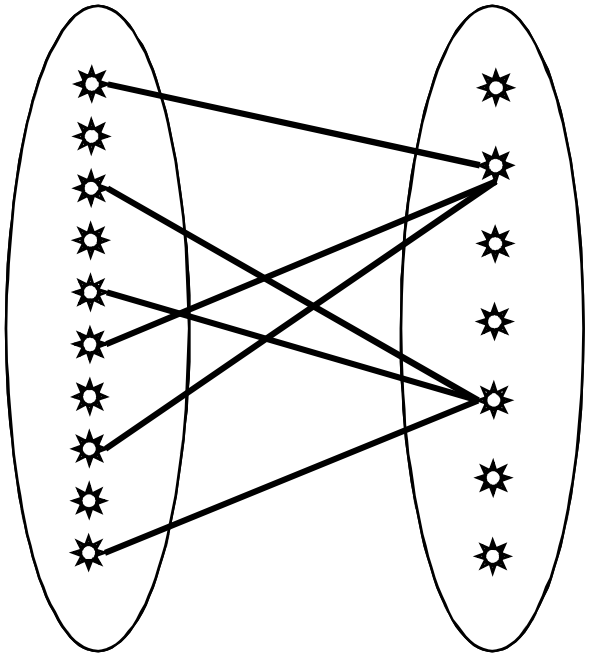
관계성의 다중연관성(multiplicity)

- 클래스 C로부터 클래스 D로의 다대다(many-many) 관계성은 C에 있는 하나의 객체가 D에 있는 객체들의 집합과 연관되어지는 것이며, 역관계성에서는 D에 있는 하나의 객체가 C에 있는 객체들의 집합과 연관되어지는 것이다.
- 클래스 C로부터 클래스 D로의 다대일(many-one) 관계성은 C에 있는 하나의 객체에 대해 D에 있는 객체는 하나만 연관될 수 있다. 그러나, 역관계성에서는 D에 있는 하나의 객체에 대해 C에 있는 객체들의 집합이 연관되어진다.
- 클래스 C로부터 클래스 D로의 일대일(one-one) 관계성은 하나의 C 객체는 하나의 D 객체와만 연관되어질 수 있고, 역관계성에서도 하나의 D 객체는 하나의 C 객체와만 연관되어질 수 있다.

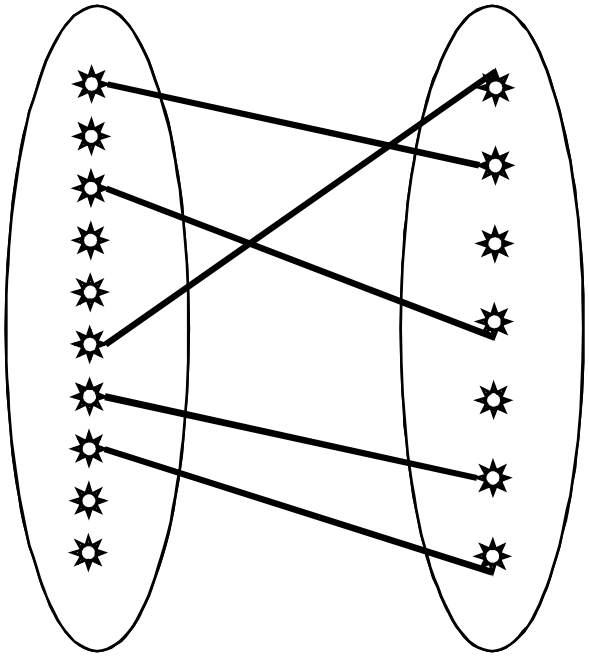
관계성의 다중연관성(계속)



Movie M:N Star



Movie M:1 Studio

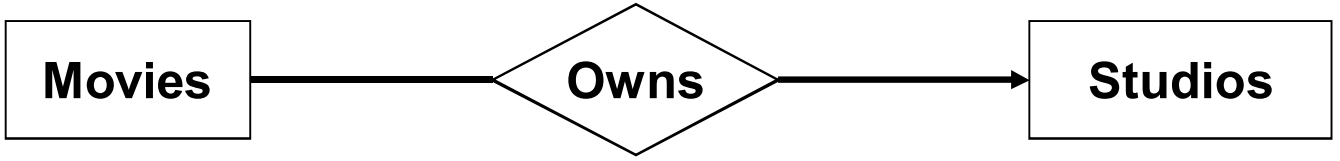


Studio 1:1 President

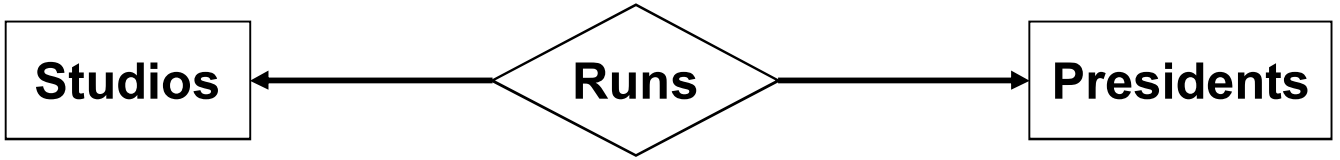
E/R 관계성의 다중연관성

◆ 다대다 관계성

◆ 다대일 관계성



◆ 일대일 관계성



설계 원칙

- 충실성(faithfulness)
  - 설계는 다루고자 하는 실제 상황을 충실하게 나타내야 한다.

  [예] Stars와 Movies간의 Stars-in 관계성은 M:N이 되어야 한다.
- 중복(redundancy) 회피
  - 하나의 사실을 나타내기 위해 하나의 정보만을 유지하는 것이 좋다.

  [예] Movies와 Studios간의 Owns 관계성외에 Movies에 studioName와 같은 애트리뷰트는 중복된다. – 공간낭비, 불일치 가능성
- 단순화(simplicity)
  - 필요 이상의 요소는 설계에 넣지 않는다.

설계 원칙 (계속)

◆ 올바른 요소의 선택

– 실세계의 개념을 나타내기 위한 애트리뷰트와 엔티티 집합사이의 선택

(예) **Studios** 엔티티 집합 대신, 스튜디오의 이름과 주소를 **Movies**의 애트리뷰트로 만들자.

» 주소 항목이 중복(**redundancy**)된다.

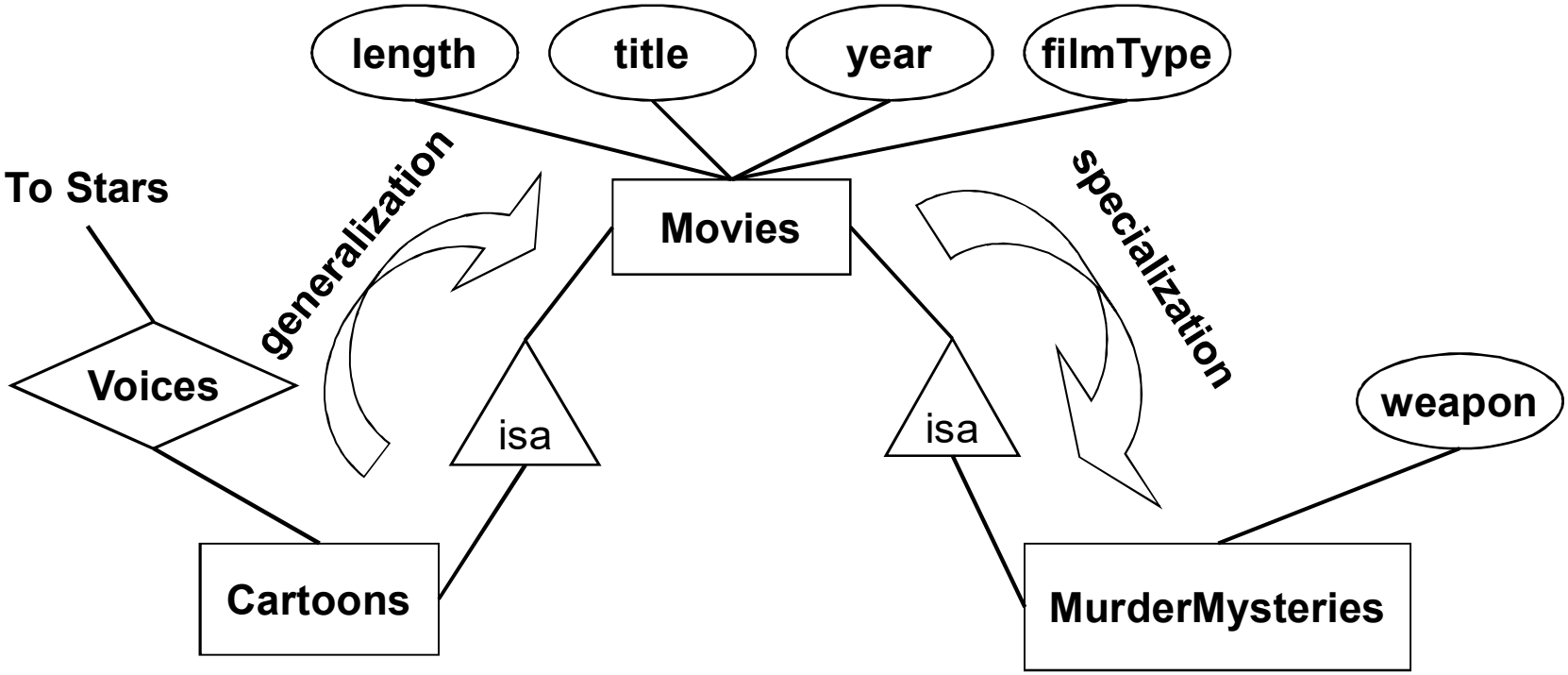
◆ 공간의 낭비

◆ **data inconsistency** 문제 : **Fox** 영화사가 **Pusan**으로 이사가면 **??**

» 어떤 스튜디오가 소유하는 영화가 하나도 없다면, 그 스튜디오의 주소는 저장할 수 없다. **(information loss)**

– 이름뿐만 아니라 좀 더 많은 정보를 필요로 한다면 엔티티 집합이나 클래스를 사용하는 것이 더 바람직하다. 그러나, 단지 이름만 필요하다면 애트리뷰트로 만드는 것이 더 좋다.

E/R 다이어그램에서의 서브클래스

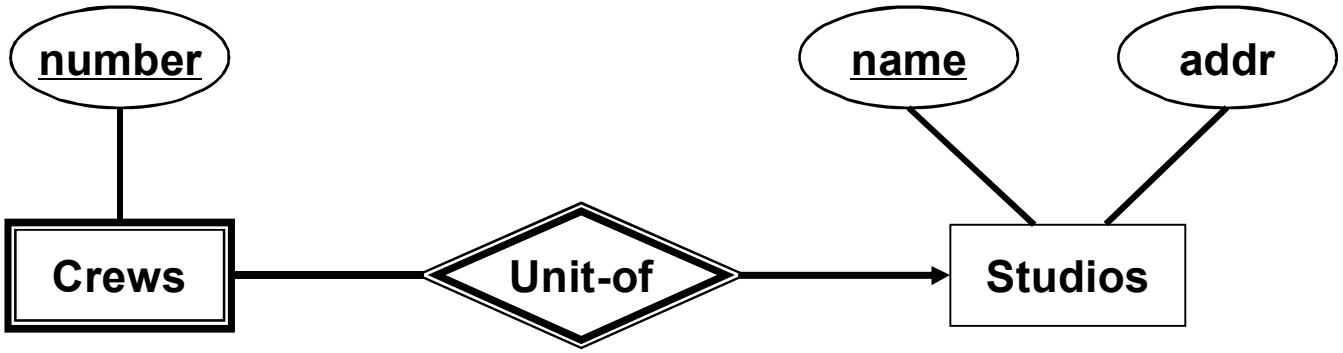


- Avante **is a** Hyun-Dai Car : isa-relationship

약 엔티티 집합

◆ 약 엔티티 집합(weak entity set)

- 키의 일부 또는 전부가 다른 어떤 엔티티 집합에 있는 애트리뷰트들로 만들어진 엔티티 집합
- 자신의 주키를 가지고는 **uniqueness**가 보장될 수 없어서 약 엔티티 집합이라고 한다.
- 오너**(owner)-**엔티티 없이는 식별될 수 없다.

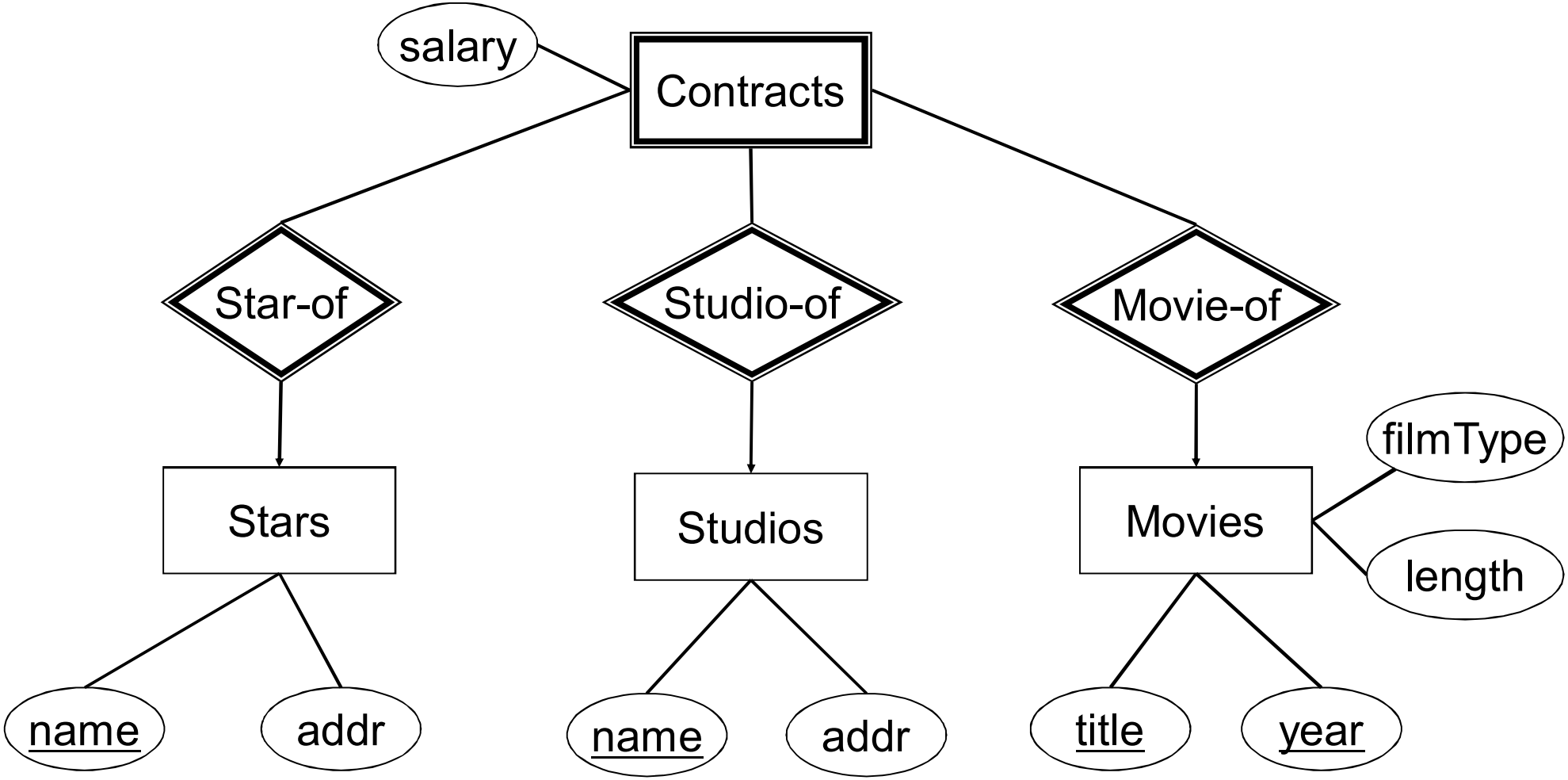


Owner Entity Set

number
1
2
1
2
2
3

Crews 엔티티 집합

약 엔티티 집합 (계속)


salary
Contracts
Star-of
Studio-of
Movie-of
Stars
Studios
Movies
filmType
length
name
addr
name
addr
title
year

약 엔티티 집합에 대한 요구사항

◆ 약 엔티티 집합 **W**의 키에 애트리뷰트를 제공하는 엔티티 집합 **S**는 어떤 관계성 **R**로 **W**에 반드시 연관되어 있어야 한다. 또한 다음과 같은 조건들이 반드시 지켜져야 한다.

❶ **R**은 **W**로부터 **S**로의 이진, 다대일 관계성이어야 한다.

❷ **W**의 키로 사용된 **S**의 애트리뷰트는 **S**의 키 애트리뷰트이어야 한다.

❸ **S**도 약 엔티티 집합이면 **(1)**과 **(2)**가 **S**에 같은 요령으로 적용된다.

☞ 다대일 관계성은 (다대일의 특별한 경우인) 일대일 관계성을 포함한다.

제약(constraint)을 모델링

- 실세계를 올바르게 모델링할 수 있도록 데이터베이스를 유지
  - 제약은 스키마의 한 부분 : **DB**의 데이터는 제약을 항상 만족해야 한다

❶ 키**(key)** : 클래스내의 객체나 엔티티 집합내의 엔티티를 유일하게 구별할 수 있는 하나의 애트리뷰트나 애트리뷰트들의 집합

❷ 단일 값 **(single-value)** 제약 : 특정한 역할을 하는 값이 유일해야 한다.

❸ 참조 무결성**(referential integrity)** 제약 : 어떤 다른 객체에 의해 참조되는 값이 데이터베이스에 실제로 존재해야 한다.

❹ 도메인**(domain)** 제약 : 애트리뷰트의 값이 특정 값의 집합에 속해야 한다.

❺ 일반**(general)** 제약 **:** 데이터베이스에서 지켜져야 할 임의의 무결성 단정**(assertion)**을 말한다.

키

◆ 정의

– 키를 하나 이상의 애트리뷰트로 구성되는 집합 **K**라고 하면, 클래스 (또는 엔티티 집합)의 두 객체 O_1과 O_2 는 키 **K**를 구성하는 애트리뷰트에 서로 같은 값을 가질 수 없다.

– 성질 : 유일성(**uniqueness**)과 최소성(**minimality**)

◆ 키의 종류

❶ 후보키(Candidate Key) : 여러 개의 가능한 키들

[예] 학생 클래스의 학번과 주민등록번호는 모두 후보키

❷ 주키(Primary Key) : 후보키들중 주로 사용할 키를 선언함

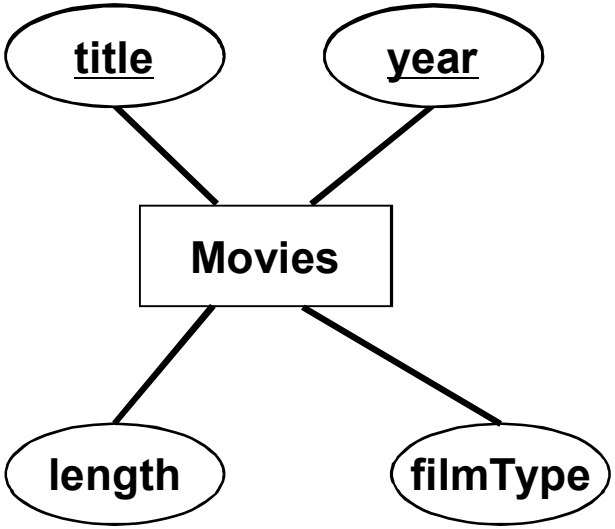
[예] 학생 클래스에 대해서 학번을 주키로 사용

❸ 수퍼키(Super Key) : 키의 성질중 최소성을 만족하지 않음

키 (계속)

◆ **E/R 모델에서 키의 표현**

– 키에 속하는 애트리뷰트에 밑줄을 긋는다.



– **E/R** 모델은 키가 두 개 이상 존재하는 것을 나타내는 표기법은 제공하지 않는다. 주석 등으로 처리하기도 한다.

단일 값 제약

- 단일 값을 가지는 애트리뷰트에 대한 두 가지 경우
  - ❶ 애트리뷰트의 값이 반드시 존재해야 하는 경우**(exactly one)**
    - » 키에 속하는 애트리뷰트
  - ❷ 애트리뷰트의 값이 있을 수도 있고 없을 수도 있는 경우**(at most one)**
    - » 실제 값 대신에 널**(null)** 값이 사용될 수도 있음
    - » 널 값은 애트리뷰트에 대한 유효한 정보가 없을 때 사용 **: Movie** 객체에서 **length** 값을 모르는 경우 **–1**이나 **NULL**을 저장
- 단일 값 제약 표현
  - **E/R** 모델에서는, 각 애트리뷰트는 다른 표시가 없는 한 단일 값을 갖는다고 간주되며, 일반적으로 값이 널이 될 수 있다고 가정한다.

참조 무결성 (1/2)

◆ 참조 무결성 제약

– 관계성에서 참조되어지는 객체나 엔티티는 반드시 존재해야 한다.

– 허상(dangling) 포인터(이미 삭제된 객체에 대한 포인터) 를 허용 안함

[참고] 쓰레기(garbage) 객체 : 존재하지만 참조할 포인터가 상실된 객체

◆ 프로그래밍 예제

```
int   *a, *b;
a = malloc(sizeof(int));
b = a;
free(a);
printf("%d", *b);
```

```
int  *a;
a = malloc(sizeof(int));
a = malloc(sizeof(int));
```

참조 무결성 (2/2)

◆ 참조 무결성 제약이 지켜지도록 하기 위한 방법

- 참조되어지는 객체의 삭제를 금지한다.
- 참조되어지는 객체가 삭제되면 이 객체를 참조하고 있던 모든 객체도 같이 삭제한다.
- 참조하는 객체가 새로 만들어 지면, 기존에 존재하는 객체를 그 관계성의 값으로 가져야 한다.
- 관계성의 값이 변경되었을 때, 새로운 값은 반드시 이미 존재하는 객체이어야 한다.

E/R 다이어그램에서의 참조 무결성

- 둥근 화살표는 다음을 나타내기 위해 사용된다.
  - 참조 무결성 제약
  - 관계성은 다대일이나 일대일이다.

  - 삭제되어서는 안되는 엔티티 집합으로 **one**의 관계를 가지는 경우 **referencial integrity**를 가진다.

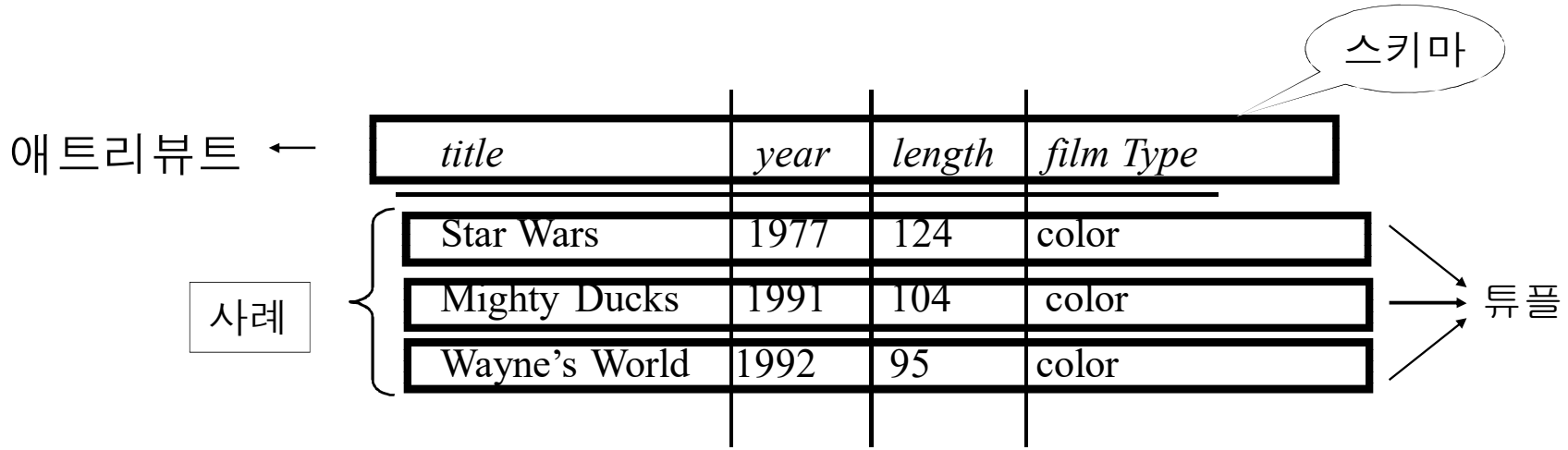
그 밖에 다른 제약

- 도메인(domain) 제약
  - 애트리뷰트의 값은 어떤 제한된 집합에 있는 것이어야 한다.
    - » **ODL**에서는 각 애트리뷰트의 타입을 지정해야 한다. 그리고 이 타입은 도메인 제약의 기본적인 형태이다. **(length**는 **positive integer** 형이다)
- 관계성의 차수(degree)에 대한 제약
  - 관계성에 참여하는 엔티티의 수를 제한한다.

- 화살표는 "≤ 1" 제약을 나타낸다.
- 참조 무결성은 "= 1" 제약을 나타낸다.
- **edge**에 숫자를 기입하여 표시

관계 모델의 기본

◆ 릴레이션 **Movie**



관계 모델의 기본 (계속)

- 릴레이션 (사례:instance) : 2차원 테이블
  - 요소들이 원자적 값을 가지는 튜플들의 집합
- 애트리뷰트 : 릴레이션의 각 열에 대한 이름
- 튜플**(tuple)** : 릴레이션의 행
- 스키마 : 릴레이션의 이름과 릴레이션의 애트리뷰트들의 집합

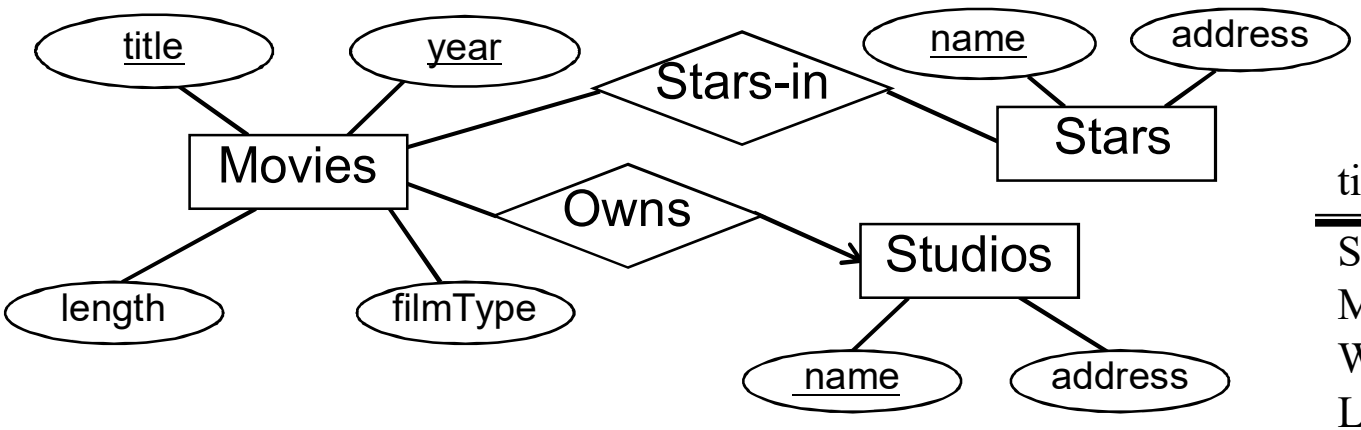
  [예] `Movie(title, year, length, filmType)`
- 도메인 : 릴레이션의 각 애트리뷰트에 연관된 타입

☞ 스키마와 사례**(instance)** : 스키마는 릴레이션에 대한 애트리뷰트들의 이름이며, 사례는 릴레이션에 대한 튜플들의 집합을 나타내는 것이다.

- 사례는 시간의 흐름에 따라서 그 내용이 변한다.

- 스키마는 거의 변하지 않는다.

E/R에서 관계 모델로의 변환



title	year	studioName
Star Wars	1977	Fox
Mighty Ducks	1991	Disney
Wayne's World	1992	Paramount
Little Mermaid	1993	Disney

관계성 `Owns` 에 대한 릴레이션

title	year	starName
Star Wars	1977	Carrie Fisher
Star Wars	1977	Mark Hamill
Star Wars	1977	Harrison Ford
Mighty Ducks	1991	Emilio Estevez
Wayne's World	1992	Dana Carvey
Wayne's World	1992	Mike Meyers

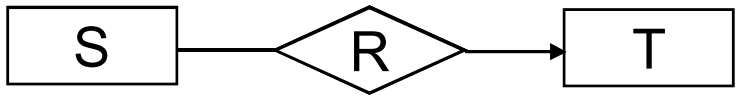
관계성 `Stars-in` 에 대한 릴레이션

name	address
Carrie Fisher	123 Maple St., Hollywood
Mark Hamill	456 Oak Rd., Brentwood
Harrison Ford	789 Palm Dr., Beverly Hills

엔티티 `Stars` 에 대한 릴레이션

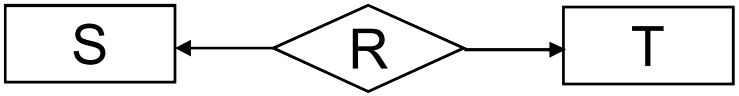
릴레이션들의 결합

- 어떤 관계성에 대해서는 테이블을 생성할 필요가 없다.
  - 다대일 관계성



- **R**에 대한 테이블을 생성하는 대신, **S**에 **R**의 모든 애트리뷰트들과 **T**의 키를 포함시켜도 된다.

  - 일대일 관계성

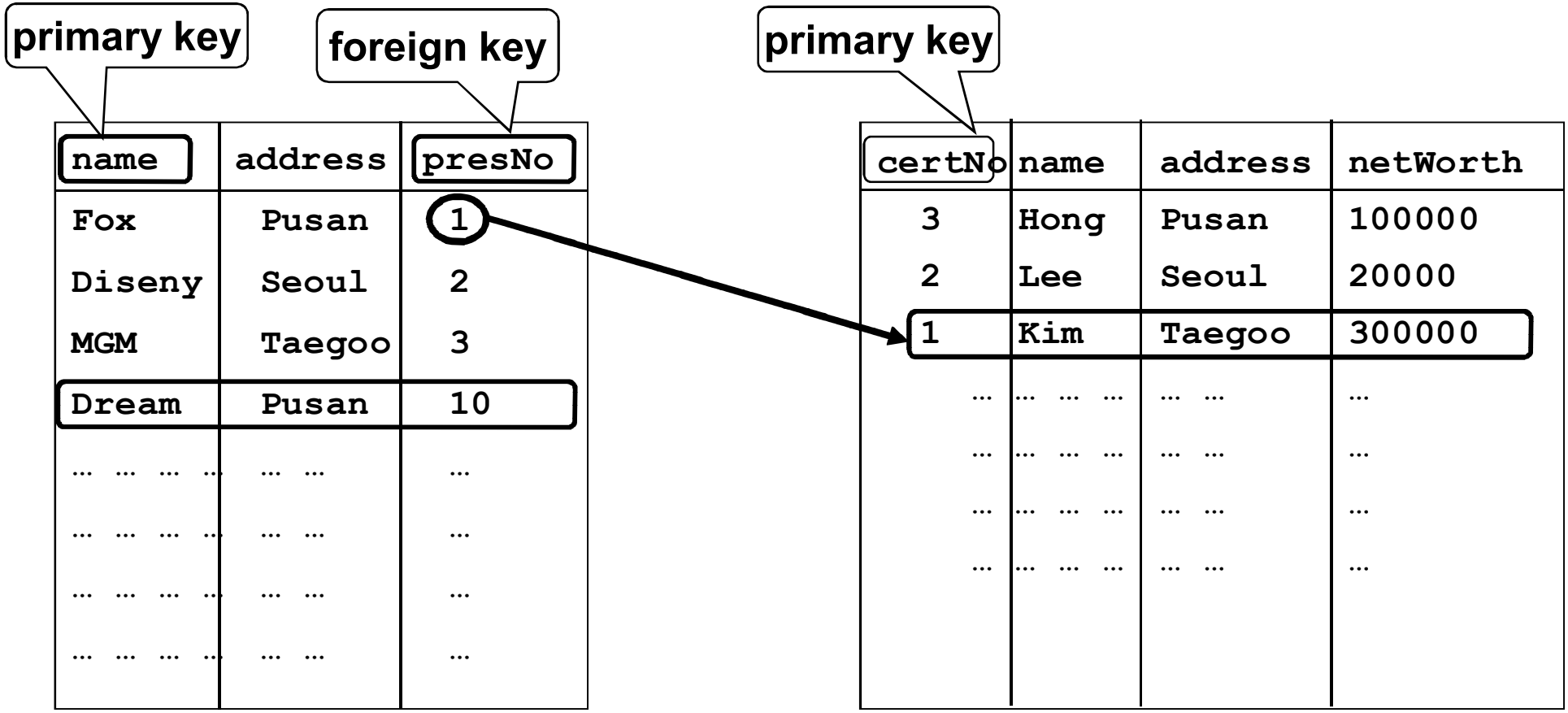


- **R**에 대한 테이블을 생성하는 대신, **S**에 **R**의 모든 애트리뷰트와 **T**의 키를 포함시켜도 된다. 그 역도 가능하다.
- **R**에 대한 테이블을 생성하는 대신, 모든 것을 하나의 테이블로 표현해도 된다.

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),
  presNo   INT REFERENCES
        MovieExec(certNo));
```

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

☞ 어떤 studio 튜플이 presNo 요소의 값으로 NULL을 갖더라도,NULL이 MovieExec 안의 certNo 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

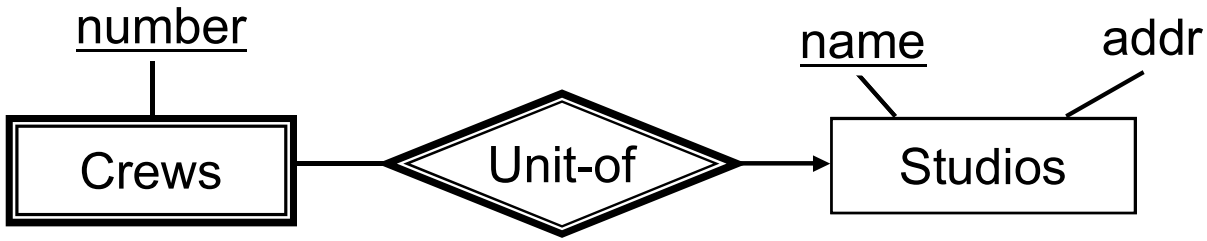
  [예2] MovieExec의 (1, 'Kim', 'Taegoo', 300000)을 삭제시 Studio의 어떤 튜플도 cert#로 1을 가지지 말아야 성공
- 연쇄(cascade) 정책
  - 연쇄 삭제(cascade deletion)
    - » 참조되는 튜플을 삭제하면 참조하는 튜플도 삭제
  - 연쇄 갱신(cascade update)
    - » 참조되는 튜플을 갱신하면 참조하는 튜플도 갱신

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 10g**에서는 **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
);
```

약 엔터티 집합을 관계 모델로 변환



Crews(number,studioName)
Studios(name,addr)

number	studioName
1	Fox
2	Fox
3	Fox
4	Fox
2	Disney
3	Disney
2	Paramount

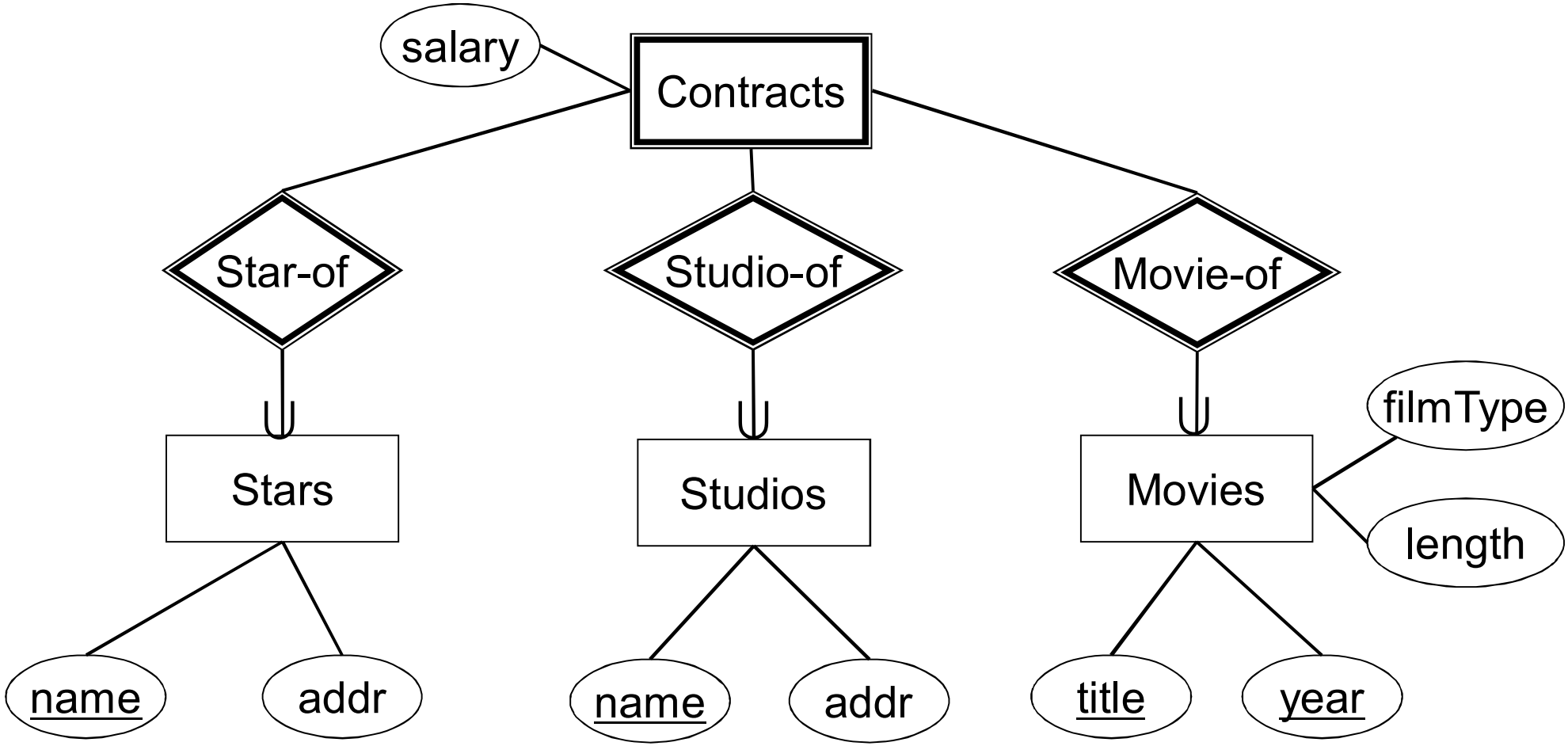
Unit-Of 릴레이션의 사례

- 만일 Unit-of를 릴레이션으로 만든다면 ?
  - Unit-of(number, studioName, studioName') : studioName과 studioName'는 동일하므로 하나로 표현해도 상관없으므로 ..

    [예] (3, Disney, Disney), (5, Fox, Fox) ...

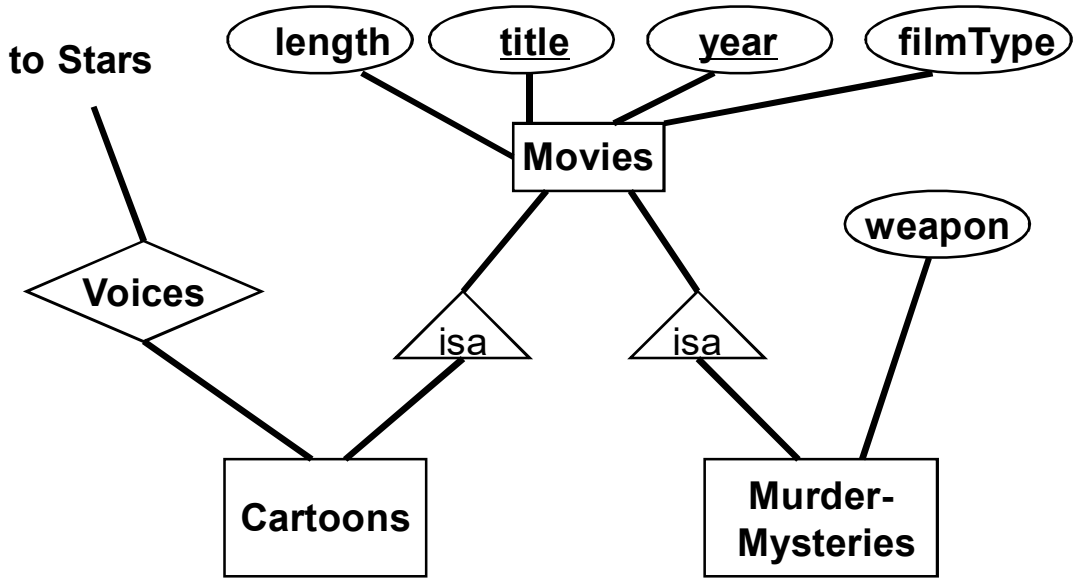
    » Unit-of(number,studioName)으로 변형됨 : Crews 릴레이션과 동일하다.

약 엔터티 집합을 관계 모델로 변환 (계속)




```
Contracts(starName,studioName,title,year,salary)
```

서브클래스를 릴레이션으로 표현



♦ 무성만화영화는 어떻게 표현되는가?
Voices 관계성에 starName이 없으므로 NULL로 표현할 수 있다.

```
Movies(title, year, length, filmType)
MurderMysteries(title, year, weapon)
Cartoons(title, year)
Voices(title, year, starName)
```

애트리뷰트 값에 대한 제약

- 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),
      presNo  INT REFERENCES MovieExec(certNo)
              CHECK (presC# >= 100000));

gender CHAR(1) CHECK (gender IN ('F', 'M'))
```

애트리뷰트 값에 대한 제약 (계속)

- **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.%')
);
```

MovieDB Schema 구조

- **Movie**
  - **TITLE : string, YEAR : integer, length : integer, inColor : string**
  - **studioName : string, producerNo : integer**
- **StarsIn**
  - **MOVIETITLE : string, MOVIEYEAR : integer, STARNAME : string**
- **MovieStar**
  - **NAME : string, address : string, gender : string, birthdate : date**
- **MovieExec**
  - **name : string, address : string,**
  - **CertNo : integer, netWorth : integer**
- **Studio**
  - **NAME : string, address : string, presNo : integer**