

# 6장. 데이터베이스 언어 SQL

---

- ◆ SQL에서의 단순 질의
- ◆ 하나 이상의 릴레이션을 포함하는 질의
- ◆ 부질의(subquery)
- ◆ 중복(duplicates)
- ◆ 집단화(aggregation)
- ◆ 데이터베이스 변경
- ◆ SQL에서 릴레이션 스키마의 정의
- ◆ 뷰 정의
- ◆ NULL 값
- ◆ 조인 수식

# SQL

---

## ◆ SQL

- **Structured Query Language**의 줄임말
- 관계 데이터베이스에서 사용하는 질의어

## ◆ SQL의 버전들

- **ANSI SQL (SQL-89)** : 89년에 표준화됨
- **SQL2** : ANSI SQL을 92년에 갱신
- **SQL3 (SQL-99)**: SQL2에 **recursion, trigger, object concept** 등을 추가한 표준안
- 대부분의 상용 **DBMS**는 **SQL2**를 따른다.

# SQL에서의 단순 질의

---

## ◆ SQL 질의의 기본적인 형태

- 한 릴레이션에서 어떤 조건을 만족하는 튜플들을 검색
- 기본적인 세 개의 키워드 : **SELECT, FROM, WHERE**
  - » **FROM** 절: 질의의 대상이 되는 하나 이상의 릴레이션들
  - » **WHERE** 절: 튜플들이 질의 결과에 포함되기 위해 만족해야 하는 조건
  - » **SELECT** 절: 결과로 생성할 애트리뷰트들을 기술

[예] 1990년도에 Disney 스튜디오에서 제작된 모든 영화를 찾아라.

```
SELECT *  
FROM Movie  
WHERE studioName = 'Disney' AND year = 1990;
```

# SQL에서의 단순 질의 (계속)

---

## ◆ SQL에서의 프로젝션

- 릴레이션을 일부 애트리뷰트들 만으로 프로젝션

[예] 1990년도에 Disney 스튜디오에서 제작된 모든 영화들의 제목과 상영시간을 찾아라.

```
SELECT title, length
FROM Movie
WHERE studioName = 'Disney' AND year = 1990;
```

title	length
Pretty Woman	119

$\pi_{title, length} (\sigma_{studioName = 'Disney' \text{ AND } year = 1990} (Movie))$

# SQL에서의 단순 질의 (계속)

- **FROM** 절에 나타난 릴레이션의 애트리뷰트 이름과 다른 애트리뷰트 이름을 사용

```
SELECT title AS name, length AS duration
FROM Movie
WHERE studioName = 'Disney' AND year = 1990;
```



name	duration
Pretty Woman	119

☞ **SQL**은 키워드와 애트리뷰트의 이름에 대해서는 대소문자를 구별하지 않는다. 단지 인용부호 안에서만 대문자와 소문자를 구분한다.

# SQL에서의 단순 질의 (계속)

---

## ◆ 문자열 비교

- 튜플의 **title**에 저장된 값이 “**Star Wars**”인 경우 다음과 같은 문자열 비교는 적당하지 않다.

- » **title = 'star wars', title = 'Star wars', title = 'Star Wars'**

- » **title = 'Star Wars ', title = 'STAR WARS'**

- 대/소문자 변환 함수의 이용

- » **lower(title) = 'star wars'**

- » **upper(title) = 'STAR WARS'**

- 공백 제거 함수의 이용

- » **trim(title) = ' Star Wars ' - 'Star Wars'로 비교**

**[예] SELECT TRIM(' Star Wars ') FROM dual;**

TRIM

-----

Star Wars

# SQL에서의 단순 질의 (계속)

---

- **length**를 시간(hour) 단위로 표현한 결과

```
SELECT title AS name, length/60 AS duration
FROM Movie
WHERE studioName = 'Disney' AND year = 1990;
```

- **SELECT** 절의 항목으로 상수를 사용할 수도 있다.

```
SELECT title, length/60 AS duration, 'hrs.' AS inHours
FROM Movie
WHERE studioName = 'Disney' AND year = 1990;
```

title	duration	inHours
Pretty Woman	1.98334	hrs.

SELECT title, length/60 || ' hrs.' ...

# SQL에서의 단순 질의 (계속)

---

## ◆ SQL에서의 선택 연산

- **SQL의 WHERE 절은, 관계 대수의 선택 연산자와 그 이상의 것을 표현할 수 있다.**
  - 여섯 개의 일반적인 비교 연산자: =, <>, >, >=, <=, !=(Oracle)
  - 산술 연산자: +, -, \*, / 등
  - 문자열의 접합 연산자(concatenation operator) ||
  - **SQL에서 문자열은 주위에 단일 인용부호를 두어 나타낸다.**
  - 정수와 실수
  - 부울(boolean) 값들은 논리 연산자인 **AND, OR, NOT**으로 결합될 수 있다.
  - **DBMS가 제공하는 각종 함수들**



# SQL에서의 단순 질의 (계속)

---

**[예 1] 1970년 이후에 만들어진 모든 흑백 영화의 제목을 찾아라.**

```
SELECT title
```

```
FROM Movie
```

```
WHERE year > 1970 AND NOT inColor;
```

inColor는 부울 타입이다.

**[예 2] MGM 스튜디오에서 제작된 영화 중 1970년 이후의 것이거나 상영시간이 90분이 안 되는 영화의 제목을 찾아라.**

```
SELECT title
```

```
FROM Movie
```

```
WHERE (year > 1970 OR length < 90) AND studioName = 'MGM' ;
```

# SQL에서의 단순 질의 (계속)

---

## ◆ 문자열 비교

- 문자열 비교는 사전식(lexicographic) 순서 (예: 'at' < 'bar') 에 기반을 두고 있다.
- **s LIKE p** : 간단한 패턴 부합(match)을 기반으로 한 문자열 비교
  - » s는 문자열이며 p는 패턴이다.
  - » **NOT LIKE**도 가능
  - » 패턴: 특수 문자인 %와 \_을 선택적으로 가진 문자열
    - ◆ % : 0 이상의 길이를 가진 임의의 문자열
    - ◆ \_ : 임의의 한 문자

[예] 제목이 “Star” 로 시작하는 모든 영화를 찾아라.

```
SELECT title
FROM Movie                                /* 결과: {Star Wars, Star Trek} */
WHERE title LIKE 'Star _ _ _ _' ; /* 또는, 'Star%' */
```

# SQL에서의 단순 질의 (계속)

---

[예] 제목에 소유격( 's)을 갖는 모든 영화를 찾아라.

```
SELECT title
```

```
FROM Movie
```

```
WHERE title LIKE '% 's% ' ;
```

```
/* {Logan's Run, Alice's Restaurant}등의 결과가 나옴 */
```

연속되는 두개의 어포스트로피는 하나의 어포스트로피를 나타낸다.

## ◆ LIKE 수식에서 이스케이프(escape) 문자

- 키워드 **ESCAPE**와 사용하고자 하는 이스케이프 문자

[예] s LIKE 'x%%x%' ESCAPE 'x'

```
/* %로 시작해서 %로 끝나는 모든 문자열 */
```

# SQL에서의 단순 질의 (계속)

---

## ◆ 날짜와 시간의 비교

- 날짜와 시간은 일반적으로 특수한 데이터 타입으로 지원된다.
- 문자열로 표현한다.

- 날짜는 키워드 **DATE**로 표현된다.

**DATE** '2004-11-14'

- 시간은 키워드 **TIME**으로 표현된다.

**TIME** '15:00:02.5'

- 앞과 같은 비교 연산자를 이용하여 날짜와 시간을 비교할 수 있다.
  - **birthdate >= '1990-01-01' : 1990년 이후에 태어난...**

# NULL 값

---

- ◆ 의미 : 값이 무엇인지 정의할 수 없는 경우 사용될 수 있다
  - 알려지지 않은 값(**value unknown**)
    - » 값이 무엇인지 알 수 없는 경우
    - [예] 어떤 영화 스타의 알려지지 않은 생일
  - 적용 불가능한 값(**value inapplicable**)
    - » 적합한 값이 존재하지 않는 경우
    - [예] **MovieStar** 릴레이션에서 미혼인 영화 스타의 **spouse** 애트리뷰트 값
  - 보류된 값(**value withheld**)
    - » 값을 알 수 있는 자격이 없는 경우
    - [예] 공개되지 않은 전화번호
- 🌸 두 **NULL** 값은 서로 동일한 값이 아님을 주의하라.

# NULL 값 (계속)

---

## ◆ NULL에 대한 연산

❶  $x$  나  $+$ 와 같은 산술 연산자에 NULL을 사용 : 결과는 NULL

»  $x$ 의 값이 NULL이면,  $x + 3$  은 NULL이다.

❷  $=$  이나  $>$ 와 같은 비교 연산자에 NULL을 사용 : 결과는 UNKNOWN

»  $x$ 의 값이 NULL이면,  $x > 3$  은 UNKNOWN이다.

–  $x$ 의 값이 NULL인 경우  $x = \text{NULL}$  ?

» SQL 문법에 어긋남.

»  $x \text{ is NULL}$ ,  $x \text{ is NOT NULL}$ 을 사용

–  $x$ 의 값이 NULL인 경우  $x * 0$  ?,  $x - x$  ?

» 모두 결과는 NULL

# NULL 값 (계속)

## ◆ 진리값 *UNKNOWN*

- TRUE(1), UNKNOWN(1/2), FALSE(0)으로 이해
- $x \text{ AND } y : \min(x,y)$ ,  $x \text{ OR } y : \max(x,y)$ ,  $\text{NOT } x : 1-x$

X	y	x AND y	x OR y	NOT x
TRUE	UNKNOWN	UNKNOWN	TRUE	FALSE
FALSE	UNKNOWN	FALSE	UNKNOWN	TRUE
UNKNOWN	TRUE	UNKNOWN	TRUE	UNKNOWN
UNKNOWN	FALSE	FALSE	UNKNOWN	UNKNOWN
UNKNOWN	UNKNOWN	UNKNOWN	UNKNOWN	UNKNOWN

[예] **SELECT \* FROM Movie**

**WHERE length <= 120 OR length > 120; /\* length IS NOT NULL \*/**

➤ length가 NULL이 아닌 튜플을 찾아라 !!!

# SQL에서의 단순 질의 (계속)

---

## ◆ 결과의 정렬(ordering)

- 출력의 순서는 임의의 애트리뷰트의 값에 기반을 둘 수 있다.
- 정렬된 출력을 얻기 위해서는, 다음의 절을 추가한다:

`ORDER BY <애트리뷰트 리스트>`

- 순서는 디폴트로 오름차순이지만 키워드 `DESC`를 붙여 내림차순의 결과를 얻을 수 있다.

**[예] 상영시간이 짧은 순서대로, 그리고 상영시간이 같을 경우, 알파벳 순서의 영화 제목 순으로 영화들을 찾아라.**

```
SELECT *  
FROM Movie  
WHERE studioName = 'Disney' AND year = 1990  
ORDER BY length, title {ASC|DESC};  
/* ORDER BY 3, 1 로도 표현 가능 */
```



# 하나 이상의 릴레이션을 포함하는 질의

---

## ◆ SQL에서 카티션 프로덕트와 조인

- 하나의 질의에서 둘 이상의 릴레이션을 결합하는 간단한 방법은 **FROM** 절에 각 릴레이션을 나열하는 것이다.

[예] **Movie(title, year, length, inColor, studioName, producerC#),**  
**MovieExec(name, address, cert#, netWorth)**

**영화 Star Wars의 제작자 이름을 찾아라.**

```
SELECT name
FROM Movie, MovieExec
WHERE title = 'Star Wars' AND producerC# = cert#;
```

# 하나 이상의 릴레이션을 포함하는 질의 (계속)

---

## ◆ 애트리뷰트 이름의 모호성 제거(**disambiguity**)

- 여러 릴레이션들을 포함하는 질의에서는 둘 이상의 애트리뷰트가 같은 이름을 가질 수 있다.
- **R.A**의 기호를 사용한다. **R**은 릴레이션이고 **A**는 애트리뷰트이다.

[예] 주소가 같은 스타와 영화임원의 쌍을 찾아라.

```
SELECT MovieStar.name, MovieExec.name
FROM MovieStar, MovieExec
WHERE MovieStar.address = MovieExec.address;
```

- ☞ 모호성이 없는 경우에도 **R.A**와 같이 릴레이션 이름을 함께 사용할 수 있다.

# 하나 이상의 릴레이션을 포함하는 질의 (계속)

---

## ◆ 튜플 변수

- 한 릴레이션에서 서로 다른 튜플들을 지정하고자 할 때는, 릴레이션의 별명(alias)인 튜플 변수(tuple variable)를 사용한다.

[예] 같은 주소를 가진 스타들을 찾아라.

```
SELECT Star1.name, Star2.name
```

```
FROM MovieStar AS Star1, MovieStar AS Star2
```

```
WHERE Star1.address = Star2.address
```

```
AND Star1.name < Star2.name;
```

Oracle에서는  
사용 안함

- 만약 `Star1.name < Star2.name` 이 없으면, 동일한 스타 이름을 가진 모든 쌍이 질의 결과에 포함된다. 또한 같은 주소를 가진 스타들의 쌍을 단 한번만 생성하도록 해 준다.

# 하나 이상의 릴레이션을 포함하는 질의 (계속)

---

## ◆ 다중 릴레이션 질의의 해석

- 중첩 루프(**nested loop**): 여러 개의 튜플 변수들이 있다면, 각 튜플 변수에 대해 루프를 갖는 중첩 루프를 생각해 볼 수있다.
- 병렬 배정: 튜플 변수들에 임의의 순서 또는 병렬적으로 튜플들의 모든 가능한 배정을 고려한다.
- 관계 대수로 변환

**SELECT**    title, year, name  
**FROM**      Movie, MovieExec     $\rightarrow \pi_{\text{title, year, name}}(\sigma_{\text{producerC\# = cert\#}}(\text{Movie} \times \text{MovieExec}))$   
**WHERE**    producerC# = cert#;

# 하나 이상의 릴레이션을 포함하는 질의 (계속)

---

```
LET the tuple variables in the FROM clause range over
    relations  $R_1, R_2, \dots, R_n$ ;
FOR each tuple t1 in relation R1 DO
    FOR each tuple t2 in relation R2 DO
        . . .
    FOR each tuple tn in relation Rn DO
        IF the where clause is satisfied when the values
            from t1, t2, ... , tn are substituted for all
            attribute references
        THEN
            evaluate the attributes of the select clause
            according to t1, t2, ... , tn and produce the
            tuple of values that results.
```

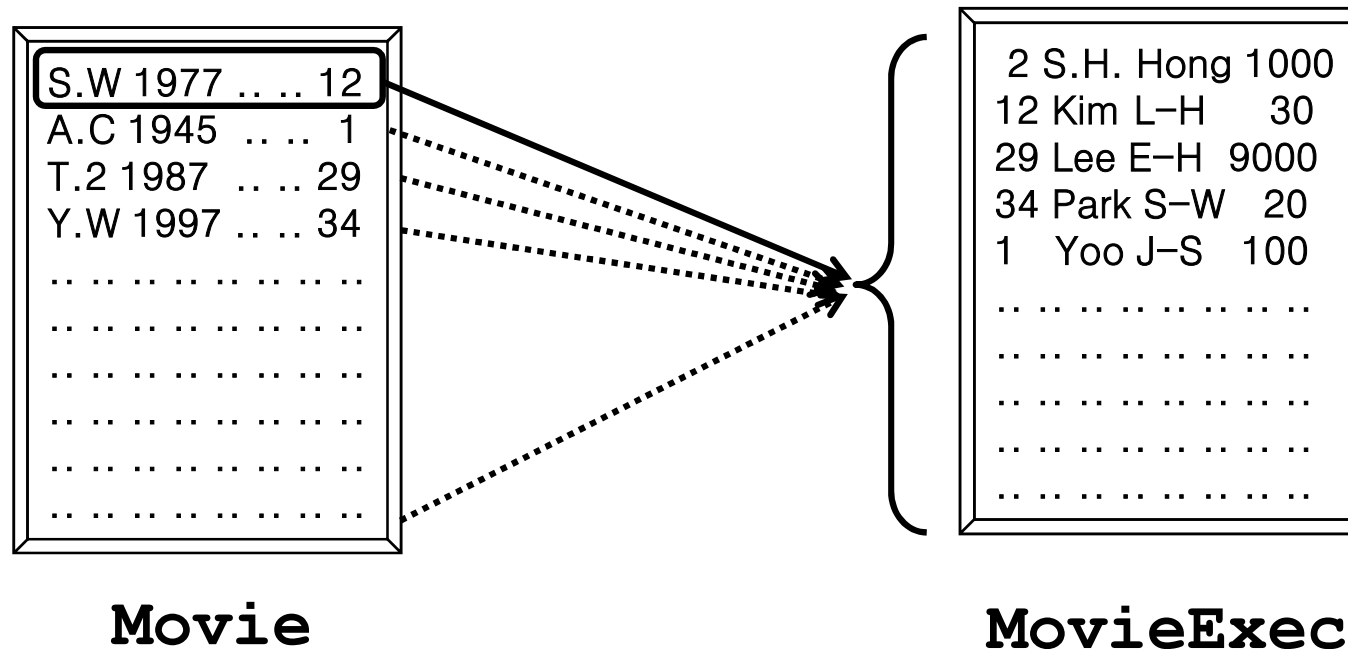
중첩 루프에 따른 알고리즘



# 하나 이상의 릴레이션을 포함하는 질의 (계속)

## ◆ 중첩 루프에 따른 **SQL**의 처리

```
SELECT title, year, name
FROM Movie, MovieExec
WHERE producerC# = cert#;
```



# 하나 이상의 릴레이션을 포함하는 질의 (계속)

---

## ◆ 관계 대수로의 변환

1. **FROM** 절의 튜플 변수들로 카티션 프로덕트를 구한다.
2. **WHERE** 절의 내용을 선택 조건으로 변환하여 카티션 프로덕트에 선택 연산자를 적용한다.
3. **SELECT** 절에 나타난 애트리뷰트들에 대해 프로젝션한다.

```
SELECT Star1.name, Star2.name
FROM MovieStar AS Star1, MovieStar AS Star2
WHERE Star1.address = Star2.address AND
      Star1.name < Star2.name;
```

$$\Rightarrow \pi_{A1, A6}(\sigma_{A2=A6 \text{ AND } A1 < A5}(\rho_{M(A1, A2, A3, A4)}(\text{MovieStar}) \times \rho_{N(A5, A6, A7, A8)}(\text{MovieStar})))$$

# SQL에서의 비직관적 결과

---

◆ 다음의 **SQL** 질의가  $R \cap (S \cup T)$ 와 동일한가 ?

**SELECT**     **R.A**

**FROM**       **R, S, T**

**WHERE**      **R.A = S.A OR R.A = T.A; /\* (R ∩ S) ∪ (R ∩ T) \*/**

- **T가 공집합인 경우 : R.A = T.A 항목은 항상 FALSE이므로 R.A = S.A 항목에 종속적인 결과가 나오는가 ?**
  - » 중첩루프/병렬배정 방식인 경우 : **T가 공집합이므로 알고리즘의 IF-문이 만족되는 경우는 없다**
  - » 관계 대수로 변환한 경우 :  **$R \times S \times T$ 에서 T가 공집합이므로 Cartesian Product 결과도 공집합이다.**





# SQL에서 집합 연산

---

## ◆ 질의들의 합집합, 교집합, 차집합

- **SQL**은 관계 대수의 합집합, 교집합, 차집합 연산을 제공한다.
- 사용되는 키워드는 **UNION, INTERSECT, EXCEPT(MINUS)** 이다.

[예1] 재산이 \$10,000,000보다 많고 영화 임원인 모든 여자 스타들의 이름과 주소를 찾아라.

```
(SELECT name, address
FROM MovieStar
WHERE gender = 'F' )
INTERSECT
(SELECT name, address
FROM MovieExec
WHERE netWorth > 10000000);
```

들의 이름과 주소는 ?

```
SELECT name, address
FROM MovieStar
MINUS
SELECT name, address
FROM MovieExec;
```

Oracle에서는 괄호 불필요

# 부질의 (subquery)

---

- 부질의는 릴레이션을 결과로 생성하는 수식이다.
- ◆ 스칼라 값을 생성하는 부질의
  - **select-from-where** 문에 의해 하나의 값만 생성될 때 그 문장은 하나의 상수처럼 사용될 수 있다.

[예] **Star Wars**의 제작자를 찾아라.

```
SELECT  name
FROM    MovieExec
WHERE   cert# = (SELECT producerC#
                  FROM Movie
                  WHERE title = 'Star Wars');
```

- **subquery** : 임의의 튜플들을 생성
- 한 개 이상의 값이 생성된다면?

# 부질의 (계속)

## ◆ 릴레이션이 비교 대상이 되는 조건

- **SQL**에는 부울 값을 결과로 생성하는 연산자들이 있다.

**R**은 릴레이션을 **s**는 스칼라 값을 나타낸다고 하자.

- **EXISTS R** : **R**에 튜플이 하나라도 존재하면(iff) 참  $\leftrightarrow$  **NOT EXISTS R**
- **s IN R** : **s**가 **R**에 있는 값 중 어느 하나와 일치하면(iff) 참  $\leftrightarrow$  **NOT IN R**
- **s > ALL R** : **s**가 단항 릴레이션 **R**의 모든 값보다 크면(iff) 참  $\leftrightarrow$  **s <= ANY R**
  - » “>” 연산자 대신 다른 비교 연산자가 사용될 수 있다.
  - » **s <> ALL R  $\equiv$  s NOT IN R**
- **s > ANY R** : **s**가 단항(unary) 릴레이션 **R**의 값 중 적어도 하나보다 크면(iff) 참  $\leftrightarrow$  **s <= ALL R**
  - » “>” 연산자 대신에 다른 비교 연산자도 사용될 수 있다.
  - » **s = ANY R  $\equiv$  s IN R**
  - » **NOT s > ANY R** : **s**는 **R**에 있는 모든 값 보다 작거나 같다. 즉, **R**에서 최소값
- **EXISTS**와 **IN**은 프레디케이트인 반면 **ALL**과 **ANY**(또는 **SOME**)은 정량자(quantifier)

# 부질의 (계속)

---

## ◆ 튜플이 비교 대상이 되는 조건

- **SQL**에서 하나의 튜플은 스칼라 값들의 리스트로 표현된다.
  - » **where (title, year) IN ( ('mighty ducks',1991), ('star wars', 1977))**
- 만약에 튜플 **t**가 릴레이션 **R**과 같은 수의 요소들을 갖는다면, **t**와 **R**을 비교할 수 있다.
  - » **t**와 **R**의 튜플은 같은 차수를 가져야 한다.
- 릴레이션 **R**의 원소와 튜플을 비교할 때에는, **R**의 애트리뷰트에 대한 표준 순서에 따라 요소들을 비교해야 한다.
- 부질의와 비교연산을 하는 경우 만일 부질의의 결과가 한 튜플 이상이라면 **ANY**또는 **ALL** 등의 정량자가 붙어야 한다.

## 부질의 (계속)

---

[예] Movie(title, year, length, inColor, studioName, producerC#)

StarsIn(movieTitle, movieYear, starName)

MovieExec(name, adress, cert#, netWorth)

- Harrison Ford가 출연한 영화의 제작자를 찾아라.

```
SELECT name
FROM   MovieExec
WHERE  cert# IN
      (SELECT producerC#
       FROM   Movie
       WHERE  (title, year) IN
             (SELECT movieTitle, movieYear
              FROM   StarsIn
              WHERE  starName = 'Harrison Ford')) ;
```

Harrison Ford가  
출연한 영화들

Harrison Ford가  
출연한 영화의 제작자들

# 부질의 (계속)

---

- 중첩된 질의는 하나의 **select-from-where** 문 형태로 변환될 수 있다.

```
SELECT name
```

```
FROM   MovieExec, Movie, StarsIn
```

```
WHERE  cert# = producerC# AND
```

```
       title = movieTitle AND
```

```
       year = movieYear AND
```

```
       starName = 'Harrison Ford' ;
```

결과에서 제작자의  
이름이 중복될 수 있다

- **FROM** : 주 질의나 부질의에서 사용되었던 릴레이션들을 기술한다.
- **WHERE** : **IN**은 등호로 대체된다.
- ☞ 중복의 발생에 있어 차이점이 있을 수 있다는 점에 주목하라.

[예] 다수의 영화 제작자인 '**George Lucas**'는 여러 번 나타날 수 있다.

# 부질의 (계속)

## ◆ 상호관련된 부질의(**correlated subquery**)

- 부질의의 외부에서 선언된 튜플 변수를 그 내부에서 사용하는 부질의

[예] `Movie(title, year, length, inColor, studioName, producerC#)`

### ■ **둘 이상의 영화에 사용된 영화 제목을 찾아라.**

```
SELECT title
FROM   Movie AS Old
WHERE  year < ANY
      (SELECT year
       FROM   Movie
       WHERE  title = Old.title);
```

애트리뷰트 이름에 대한  
영역 규칙(**scoping rules**)  
에 주목하라.

# 부질의 (계속)

---

## ◆ EXISTS의 사용

```
SELECT    name
FROM      MovieExec
WHERE     EXISTS
    (SELECT *
     FROM   Movie
     WHERE  cert# = producerC# AND
            (title, year) in (SELECT  movieTitle, movieYear
                              FROM      StarsIn
                              WHERE     starName = 'Harrison Ford')
    );
```



# 부질의 (계속)

---

## ◆ FROM절에 부질의 사용

```
SELECT    name
FROM      MovieExec, (SELECT producerC#
                      FROM Movie, StarsIn
                      WHERE title = MovieTitle AND
                             year = MovieYear AND
                             starName = 'Harrison Ford'
                      ) Prod
WHERE cert# = Prod.producerC#;
```

# 부질의 (계속)

---

## ◆ WITH 절의 사용

```
WITH prod AS (SELECT producerC#  
                FROM Movie, StarsIn  
                WHERE title = MovieTitle AND year = MovieYear AND  
                      starName = 'harrison ford')  
  
SELECT    name  
FROM      MovieExec, Prod  
WHERE     cert# = Prod.producerC#;
```

## 참고: SQL의 부울 수식에서 사용되는 프레디케이트

---

◆ 기본적인 비교 프레디케이트: =, <>(!=), <, >, <=, >=

◆ LIKE 프레디케이트

» `title LIKE '%love%'`

◆ IN 프레디케이트

» 튜플값 IN [부질의 | (값1, 값2, ..., 값n)]

◆ IS NULL, IS TRUE, IS FALSE, IS UNKNOWN 단항 프레디케이트

» `studioName IS NULL, cost = 100 IS TRUE`

◆ EXISTS와 UNIQUE 단항 프레디케이트

» `EXISTS subquery, UNIQUE subquery`

☞ ANY(또는 SOME) 와 ALL: 비교 프레디케이트와 같이 사용되는 정량자들 이다.

# 중복(Duplicates)

---

## ◆ 중복의 제거: 키워드 **DISTINCT**

- **SQL** 시스템은 보통 중복을 제거하지 않는다. (테이블은 **BAG** 형태)

» `SELECT DISTINCT name;`

## ◆ 합집합, 교집합, 차집합에서의 중복 : **ALL**

- 합집합, 교집합, 차집합 연산들은 기본적으로 중복을 제거한다.
- 중복 제거 방지: **ALL**

`(SELECT title, year FROM Movie)`

`UNION ALL`

`(SELECT movieTitle AS title, movieYear AS year FROM StarsIn);`

# 집 단 화

---

## ◆ 집 단 화(aggregation) 연산자

- **SQL**에는 릴레이션의 한 열에 대한 어떤 요약(summary)이나 집 단 화를 생성하는 다섯 개의 연산자가 있다.
  - **COUNT** : 값들의 개수
  - **SUM** : 한 열에 있는 값들의 합
  - **AVG** : 한 열에 있는 값들의 평균
  - **MIN** : 한 열에 있는 값 중 최소값
  - **MAX** : 한 열에 있는 값 중 최대값

## 집 단 화 (계 속)

---

[예 1] `SELECT COUNT (*)`  
`FROM MovieExec;`

\* 는 튜플 전체를 나타낸다.

❖ \* 는 집단화 연산자 중 **COUNT**에만 사용할 수 있는 용법이다.

[예 2] `SELECT COUNT (DISTINCT name)`  
`FROM MovieExec;`

**WHERE** 절이 없을 수도 있다

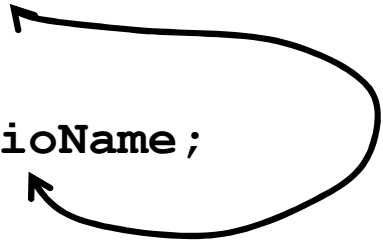
[예 3] `SELECT AVG (netWorth)`  
`FROM MovieExec;`

## 집 단 화 (계속)

### ◆ 그룹화(grouping): GROUP BY 절

- 하나 혹은 그 이상의 열의 값에 따라 그룹화(grouping)한 릴레이션의 튜플들을 생각해 보자.
- **GROUP BY** 다음에는 그룹화할 애트리뷰트들을 나열한다.

```
SELECT studioName, SUM(length)
FROM Movie
GROUP BY studioName;
```



studioName	SUM(length)
Disney	12345
Paramount	3432
... ..	... ..

- ☞ 집단화를 가진 **SELECT** 절에 집단화 되지않은 상태로 기술될 수 있는 애트리뷰트는 **GROUP BY**절에 나열된 애트리뷰트(즉, 그룹화 애트리뷰트)들 뿐이다.

## 집 단 화 (계 속)

### ◆ 그룹에 대한 조건 : HAVING 절

- 집단화 특성을 기반으로 한 그룹에 대한 조건

[예] 1930년 이전에 적어도 하나의 영화를 제작한 적이 있는 재산이 \$1,000,000이상인 제작자가 제작한 영화 상영시간의 합계를 구하라.

```
SELECT name, SUM(length)
FROM MovieExec, Movie
WHERE producerC# = cert# AND networth >= 1000000
GROUP BY name
HAVING MIN(year) < 1930;
```

각 제작자가 만든 영화의  
총 상영 시간을 구한다.

☞ **WHERE** : 조건을 만족하는 행(row)

☞ **HAVING** : 조건을 만족하는 그룹, 반드시 집단화 함수만 나타날 수 있다.



# 집 단 화 (계속)

---

## ◆ 해석 순서

- ① **FROM** 절을 기반으로 릴레이션들의 **Cartesian Product**를 계산한다.
- ② 튜플들이 **WHERE** 절을 기반으로 선택된다.
- ③ 이 튜플들은 그룹화 애트리뷰트들을 기반으로 그룹화 된다.
- ④ 그룹들은 **HAVING** 절을 기반으로 선택된다.

## ◆ SQL 질의에서 절들의 순서

- **SELECT → FROM → WHERE → GROUP BY → HAVING → ORDER BY**

# 데이터베이스 변경

---

## □ 데이터베이스 변경(modification)

– INSERT, DELETE, UPDATE

### ◆ 삽입

```
INSERT INTO R(A1, ... , An) VALUES (v1, ... , vn) ;
```

» 키워드 INSERT INTO,

» 릴레이션 이름 R, 괄호로 묶인 애트리뷰트들의 리스트

» 키워드 VALUES, 그리고 튜플 수식

```
INSERT INTO StarsIn(movieTitle, movieYear, starName)  
VALUES('The Maltese Falcon', 1942, 'Sydney Greenstreet') ;
```

# 데이터베이스 변경 (계속)

---

[예] `INSERT INTO StarsIn  
VALUES ('The Maltese Falcon', 1942, 'Sydney Greenstreet');`

[예] **Movie 릴레이션에는 언급되고 있지만, Studio 릴레이션에는 나타나지 않은 모든 영화 스튜디오를 Studio(name, address, presC#)에 추가하라.**

```
INSERT INTO studio(name)
SELECT DISTINCT studioName
FROM Movie
WHERE studioName NOT IN(SELECT name FROM Studio);
```

☞ 삽입된 **Studio** 튜플들의 **address**와 **presC#** 애트리뷰트에 대해서는 **NULL** 값이 사용된다.

# 데이터베이스 변경 (계속)

---

## ◆ 삭제

**DELETE FROM R WHERE <조건>;**

- » 키워드 **DELETE FROM**
- » 릴레이션 이름, 예를 들어 **R**,
- » 키워드 **WHERE**, 그리고 조건

[예 1] **DELETE FROM Movie; -- Movie table의 모든 tuple 삭제**

[예 2]

**DELETE FROM StarsIn**

**WHERE movieTitle = 'The Maltese Falcon' AND**

**movieYear = 1942 AND starName = 'Sydney Greenstreet' ;**

# 데이터베이스 변경 (계속)

---

## ◆ 갱신

**UPDATE R SET <새로운 값의 배정> WHERE <조건>;**

- » 키워드 **UPDATE**, 릴레이션의 이름, 예를 들어 **R**,
- » 키워드 **SET**, 식(**formula**)들의 리스트,
- » 키워드 **WHERE**, 그리고 조건

**[예] 영화 임원이 스튜디오 사장인 경우 ‘Pres’ 라는 직함을 이름 앞에 붙여라.**

```
UPDATE MovieExec
SET name = 'Pres. ' || name
WHERE cert# IN (SELECT presC# FROM Studio);
```

# SQL에서 릴레이션 스키마의 정의

---

- 데이터 정의 언어(**DDL**) 와 데이터 조작 언어(**DML**)
  - 데이터 정의 : 데이터베이스에 있는 정보의 구조를 표현
  - 데이터 조작 : 질의와 변경
- ◆ **SQL** 시스템에서 사용되는 주요 데이터 타입
  - **INT, INTEGER, SHORTINT(SMALLINT), NUMBER(p,s)**
  - **REAL, FLOAT, DOUBLE PRECISION, DECIMAL**
  - **CHAR(n), CHARACTER(n), VARCHAR(n), CHARACTER VARYING(n)**
    - » 고정 길이 또는 가변 길이의 문자열
  - **BIT(n), BIT VARYING(n)**
    - » 고정 길이 또는 가변 길이의 비트열
  - **DATE** 와 **TIME**

# SQL에서 릴레이션 스키마의 정의 (계속)

---

## ◆ 테이블 선언 : `CREATE TABLE table-name`

- **CREATE TABLE** 다음에 릴레이션의 이름, 애트리뷰트들과 그 애트리뷰트들의 타입들이 괄호로 묶인 리스트가 온다.

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

## ◆ 테이블 삭제: `DROP TABLE table-name`

```
DROP TABLE Movie;
```

# SQL에서 릴레이션 스키마의 정의 (계속)

---

## ◆ 릴레이션 스키마의 변경: `ALTER TABLE table-name`

### – `ALTER TABLE`

- » 키워드 `ADD`, 그 다음 애트리뷰트 이름과 그 데이터 타입이 나온다.
- » 키워드 `DROP`, 그 다음 애트리뷰트 이름이 나온다.

```
ALTER TABLE MovieStar ADD (phone CHAR(16), alias VARCHAR(20));
```

```
ALTER TABLE MovieStar DROP birthdate;
```



# SQL에서 릴레이션 스키마의 정의 (계속)

---

## ◆ 디폴트 값

- 구체적인 값이 주어지지 않았을 때 **NULL** 값이 사용된다.
- 일반적으로, 키워드 **DEFAULT** 와 특정 값을 기술할 수 있다.

```
gender CHAR(1) DEFAULT '?',
```

```
birthdate DATE DEFAULT DATE '0000-00-00',
```

```
ALTER TABLE MovieStar ADD (phone CHAR(16) DEFAULT 'unlisted');
```

# SQL에서 릴레이션 스키마의 정의 (계속)

---

## ◆ 도메인

- 어떤 데이터 타입을 나타내는 새로운 이름이다.

### ■ 도메인 정의 : CREATE DOMAIN

```
CREATE DOMAIN <이름> AS <타입 기술>;
```

```
CREATE DOMAIN MovieDomain AS VARCHAR(50) DEFAULT 'unknown' ;
```

- 도메인은 애틀리뷰트의 타입으로 사용될 수 있다.

```
title MovieDomain
```

### ■ 도메인에 지정된 디폴트 값을 변경 : ALTER DOMAIN

```
ALTER DOMAIN MovieDomain SET DEFAULT 'no such title' ;
```

### ■ 도메인을 삭제 : DROP DOMAIN

```
DROP DOMAIN MovieDomain;
```

# SQL에서 색인

---

## ◆ 색인

```
SELECT * FROM Movie
```

```
WHERE studioName = 'Disney' AND year = 1990;
```

- studioName과 year에 대한 index가 존재한다면 좀 더 빠르게 질의를 처리할 수 있다. : **Sequential V.S. Indexed Search**

## ◆ 색인의 생성: CREATE INDEX

```
CREATE INDEX <인덱스 이름> ON <테이블 이름(애트리뷰트 리스트)>
```

```
CREATE INDEX ExampleIndex ON Movie(studioName, year);
```

```
CREATE INDEX KeyIndex ON Movie(title, year);
```

### ■ 색인의 삭제: DROP INDEX

```
DROP INDEX YearIndex;
```

- ☞ 색인의 장단점: 질의 처리 속도가 빨라진다. 그러나 삽입, 삭제, 갱신 연산이 보다 복잡해지고 **DB** 공간이 더 필요하다.

# 뷰

---

- **CREATE TABLE** 문을 이용하여 정의된 릴레이션은 실제로 **DB** 내에 존재하는 반면 뷰(**view**)는 물리적으로 존재하지 않는다.

- ◆ 뷰의 선언

```
CREATE VIEW <뷰 이름> AS <뷰 정의>
```

- <뷰 정의>는 하나의 질의다.

```
CREATE VIEW ParamountMovie AS
    SELECT title, year
    FROM Movie
    WHERE studioName = 'Paramount';
```

- ☞ 기본(**base**) 테이블(기본 릴레이션) : 실제로 튜플이 저장된 테이블
- ☞ 뷰/가상(**virtual**) 테이블 : 실제로 튜플이 존재하지는 않고 기본 테이블 또는 다른 뷰를 기반으로 **SQL** 질의 형태로 정의(**defenition**)가 저장된 가상 테이블

# 뷰 (계속)

## ◆ 뷰에 대한 질의

- 뷰에 대한 질의가 주어지면 해당 튜플들을 기본 릴레이션으로 부터 가져온다.
- 뷰에 대한 질의를 기본 테이블에 대한 질의로 변환

```
SELECT title  
FROM ParamountMovie  
WHERE year = 1979;
```

뷰

```
SELECT title  
FROM Movie  
WHERE studioName = 'Paramount' AND year = 1979;
```

기본 테이블

질의 변경  
(query modification)

# 뷰 (계속)

---

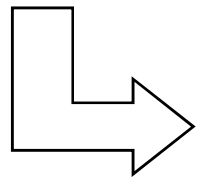
## ■ 여러 릴레이션들로 정의된 뷰

### [예1] 영화 제목과 그 영화의 제작자로 이루어진 뷰

```
CREATE VIEW MovieProd AS
  SELECT title, name
  FROM Movie, MovieExec
  WHERE producerC# = cert#;
```

### [예2] 기본 테이블과 뷰로 이루어진 질의

```
SELECT DISTINCT starName
FROM StarsIn, ParamountMovie
WHERE title = movieTitle AND year = movieYear;
```



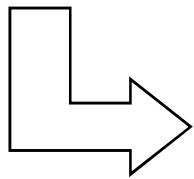
```
SELECT DISTINCT starName
FROM StarsIn, Movie
WHERE title = movieTitle AND year = movieYear
      AND studioName = 'Paramount';
```

## 뷰 (계속)

---

[예] ‘Gone With the Wind’ 라는 영화의 제작자는 ?

```
SELECT  name
FROM    MovieProd
WHERE   title = 'Gone With the Wind';
```



```
SELECT      name
FROM        Movie, MovieExec
WHERE       ProducerC# = cert# AND title = 'Gone With the Wind';
```

◆ 애틀리뷰트 이름의 변경

- 뷰에서 애틀리뷰트들의 이름을 새로 부여할 수 있다.

```
CERATE VIEW MovieProd(movieTitle, prodName) AS
SELECT title, name
FROM Movie, MovieExec
WHERE producerC# = cert#
```

# 뷰 (계속)

---

## ◆ 뷰의 변경

- 뷰는 갱신이 가능(**updatable**)할 수도 있고 가능하지 않을 수도 있다.
  - » 뷰에 대한 갱신은 기본 테이블에 대한 갱신으로 나타난다.
- **SQL2**에서는, 하나의 릴레이션 **R**로부터 애트리뷰트들을 선택하는 형태로 정의된 뷰에 대해서만 변경을 허용한다. 그리고 다음을 만족해야 한다.
  - » **SELECT** 절에는 충분한 애트리뷰트들이 있어야 한다.
    - [예1] 키 애트리뷰트들은 **NULL**이 아니어야 한다. **NOT NULL** 제약을 위반해서는 안된다. 뷰를 통해 삽입되는 튜플은 그 뷰를 통해 볼 수 있어야 한다.
    - [예2] **UNIQUE** 제약을 가지는 경우 이를 위반해서도 안된다.
  - » **WHERE** 절에 나타나는 어떤 부질의도 릴레이션 **R**을 포함하지 않아야 한다.



## 뷰 (계속)

---

[예] 뷰 **ParamountMovie**에 다음과 같은 튜플을 삽입한다고 하자.

```
INSERT INTO ParamountMovie
VALUES ('Star Trek', 1979)
```

**studioName** 애트리뷰트가 뷰의 애트리뷰트에 포함되어 있지 않으므로, 뷰에 삽입되는 튜플이 **Movie**에 반영될 때 **studioName** 값에 **NULL**이 들어간다. 이 튜플은 **ParamountMovie**의 조건을 만족하지 않는다. 즉, **ParamountMovie** 뷰에 나타나지 않는다. 따라서 뷰 **ParamountMovie**는 다음과 같이 수정되어야 한다:

```
CREATE VIEW ParamountMovie AS
SELECT studioName, title, year
FROM   Movie
WHERE  studioName = 'Paramount';
```

# 뷰 정의 (계속)

---

[예 1] 갱신가능 뷰로부터 튜플을 삭제

```
DELETE FROM ParamountMovie  
WHERE title LIKE '%Trek%';
```

[예 2] 갱신가능 뷰에 대한 갱신

```
UPDATE ParamountMovie  
SET year = 1979  
WHERE title = 'Star Trek the Movie';
```

■ 뷰의 삭제

```
DROP VIEW ParamountMovie;
```

## 참고: 뷰의 갱신 가능성

---

**Movie(title, year, length, inColor, studioName, producerC#)**

**MovieExec(name, address, cert#, netWorth)**

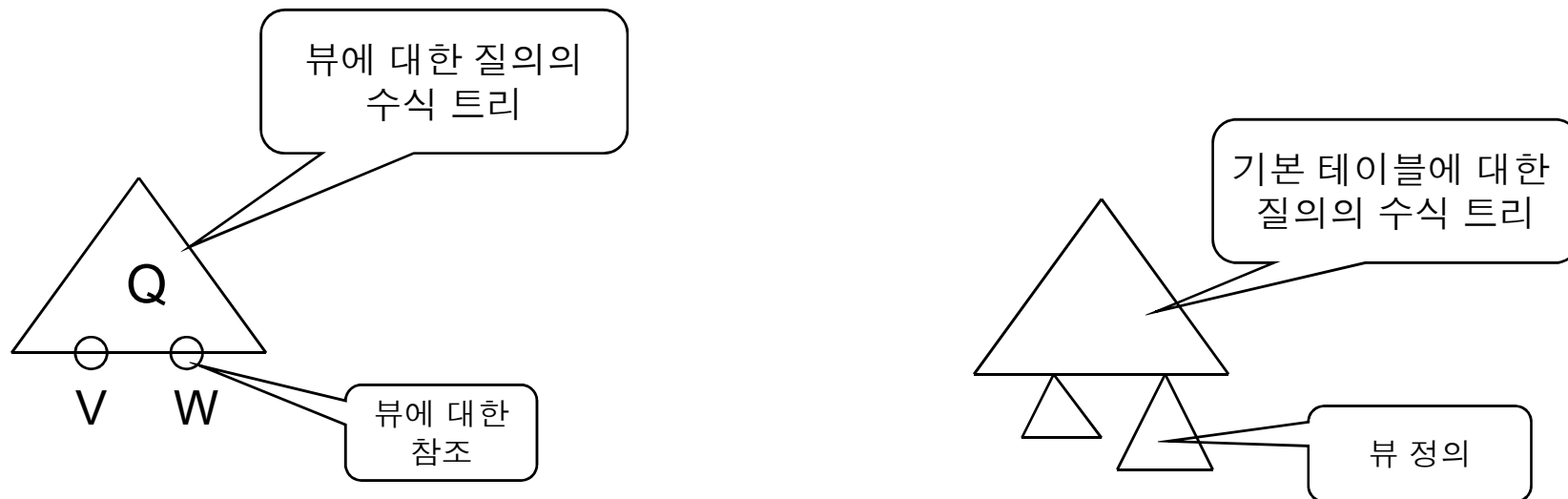
```
CREATE VIEW MovieProd AS
    SELECT title, name
    FROM Movie, MovieExec
    WHERE producerC# = cert#;
```

- 다음과 같은 튜플을 뷰 **MovieProd**에 삽입하려 한다고 하자:  
( 'Greatest Show on Earth' , 'cecil B. DeMille' )
  - **Movie**와 **MovieExec**의 키는 **NULL**이면 안 된다.
  - 조인이 이루어지는 애트리뷰트들은 **NULL**이면 안 된다.
    - » 두 **NULL** 값은 동일하지 않음에 주목하라.

# 뷰 (계속)

## ◆ 뷰를 포함하는 질의의 해석

- 기본적인 개념: 뷰에 대한 질의의 수식 트리를 기본 테이블에 대한 질의의 수식 트리로 변환한다.



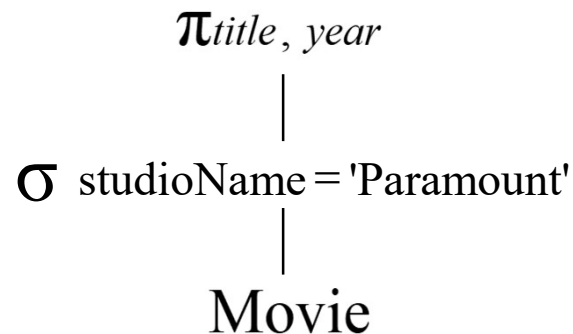
뷰에 대한 참조를 뷰의 정의로 대체  
(Q: 질의, V 와 W: 뷰)

## 뷰 (계속)

---

```
CREATE VIEW ParamountMovie AS
  SELECT title, year
  FROM Movie
  WHERE studioName = 'Paramount';
```

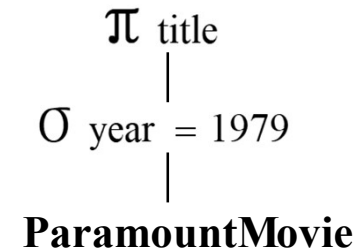
뷰 ParamountMovie에  
대한 질의



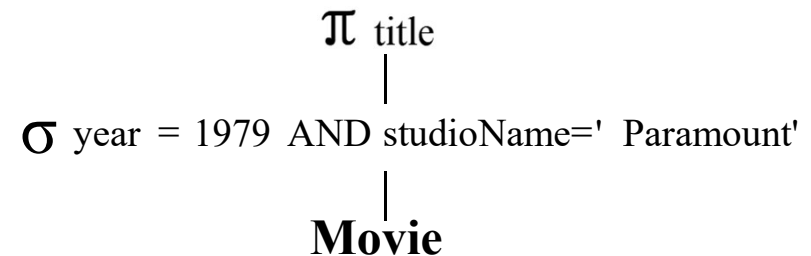
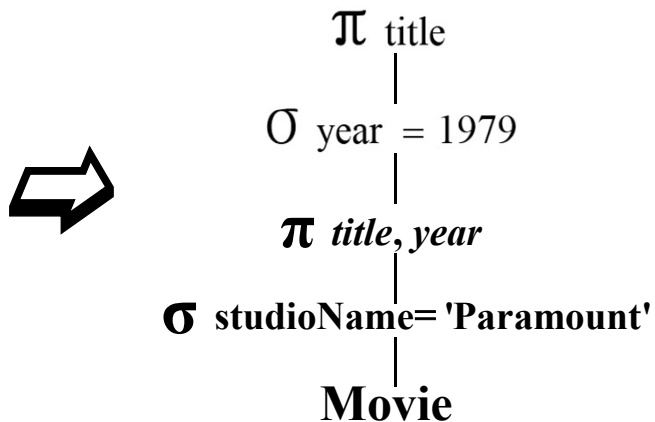
이 뷰를 정의하는 질의의  
수식 트리

## 뷰 (계속)

```
SELECT title
FROM ParamountMovie
WHERE year = 1979;
```



질의에 대한 수식 트리



단순화된 질의

기본 테이블을 사용한 질의의 수식 트리

# SQL2에서의 조인

---

## ◆ SQL2에서의 조인 수식

- CROSS JOIN: 카티션 프로덕트

`Movie CROSS JOIN StarsIn`

- JOIN ON: 세타 조인

`Movie JOIN StarsIn ON`

`title = movieTitle AND year = movieYear;`

- FROM 절의 조인 수식

`SELECT title, year, starName`

`FROM Movie JOIN StarsIn ON`

`title = movieTitle AND year = movieYear;`

- NATURAL JOIN: 자연 조인

`MovieStar NATURAL JOIN MovieExec`

# SQL2에서의 외부조인

---

## ◆ 외부조인(outerjoin)

- 적당한 애트리뷰트에 **NULL** 값을 넣어 허상 튜플(**dangling tuple**)을 결과에 추가한다.

» 허상 튜플 : 조인되지 못한 튜플

- **NATURAL [LEFT | RIGHT | FULL] OUTER JOIN**

**MovieStar NATURAL FULL OUTER JOIN MovieExec;**

» 스타지만 임원이 아닌, 혹은 임원이지만 스타가 아닌 사람들에 대한 정보도 함께 얻는다.

- **[LEFT | RIGHT | FULL] OUTER JOIN ON**

**Movie FULL OUTER JOIN StarsIN ON**

**title = movieTitle AND year = movieYear**