저자서문

이 책은 관계형 데이터베이스 관리 시스템을 활용하기 위하여 필요한 기본 지식에 대하여 설명한다. 먼저, 데이터베이스 관리 시스템의 핵심 엔진이라고 할 수 있는 저장시스템의 구조와 기능을 설명함으로써 데이터베이스 관리 시스템의 기본 기능을 이해한다. 그리고 관계형 데이터베이스 관리 시스템을 활용하기 위하여 데이터를 모델링하는 방법과 모델링된 데이터를 데이터베이스로 구축하고 데이터를 검색, 갱신하기 위한 질의어 활용 방법에 대하여 설명한다. 질의어에 대해서는 먼저 이론적인 기반이 되는 관계 대수에 대하여 간단하게 소개하고, 실제로 활용되는 SQL의 활용 방법에 대하여 MySQL을 이용하여 구체적으로 설명한다.

이 책은 다음과 같은 내용을 다룬다.

- ■ 데이터베이스 저장 구조의 이해
  - • 저장시스템이란?
  - • 데이터 저장 기능
  - • 데이터 관리 기능광선추적 기법 소개
- ■ 질의어 : 관계 대수
  - • 단항 관계 연산자
  - • 이항 관계 연산자
- ■ 관계형 데이터베이스 관리 시스템의 활용
  - • 데이터베이스 설계
  - • MySQL의 설치
  - • 릴레이션 생성
  - • SQL을 이용한 데이터베이스 관리 시스템의 활용

이 책이 나오기까지 수고해 주신 EGO expertgroup 고봉기사장님과 직원들께 감사의 말을 전한다.

한국산업기술대학교 게임공학과
장지웅 교수
jwchang@kpu.ac.kr

목 차

1. 데이터베이스 저장구조의 이해

1.1 저장 시스템이란?

본 장에서는 데이터베이스 관리 시스템의 핵심 엔진이라고 할 수 있는 저장 시스템에 대하여 설명한다.

데이터베이스 관리 시스템은 크게 질의어 형태로 입력되는 사용자의 요구사항을 분석하여 최선의 데이터 액세스 전략을 수립하는 질의 처리 및 최적화기 부분과 데이터를 저장 매체에 직접 저장하고 관리하면서 질의 처리 및 최적화기가 수립한 데이터 엑세스 전략에 따라 직접적으로 데이터를 액세스하는 기능을 수행하는 저장 시스템 부분으로 구성된다.

다시 말하면, 저장 시스템은 실제로 데이터를 조작하는 데이터베이스 관리 시스템의 핵심 모듈이라고 할 수 있다.

저장 시스템은 데이터를 디스크에 저장하고 검색하는 부분을 직접 관리함으로써 데이터베이스 관리 시스템의 성능에 직접적인 영향을 미친다.

저장 시스템이 수행하는 기능은 여러 가지가 있는데 이를 나열하면 다음과 같다.

1. 데이터 저장 기능
2. 데이터 관리 기능
3. 색인 기능

4. 동시성 제어 기능

5. 파손 회복 기능

6. 사용자 인터페이스 담당 기능

본 저서에서는 여러 가지 저장 시스템의 기능 중에서 직접적으로 데이터를 저장하고, 관리하는 기능에 대하여 좀 더 구체적으로 살펴보기로 한다.

1.2 데이터 저장 기능

데이터 저장 기능은 저장 시스템의 기능 중에서도 직접 저장 매체를 관리하는 가장 기본적인 기능이라고 할 수 있다.

특히, 저장 매체는 데이터를 직접 저장하는 물리적인 장치로서 저장 매체에 대한 접근은 컴퓨터의 여러 다른 부분에 비하여 입출력 비용이 매우 높기 때문에 데이터 액세스 성능에 가장 큰 영향을 미친다.

그러므로 데이터 저장 기능은 저장 매체의 특성과 운영체제의 저장 매체 관리 방법을 잘 이해하고, 그에 따라 저장 매체에 최적화된 형태로 설계되어야 데이터베이스 관리 시스템의 성능을 극대화할 수 있다.

일반적으로 널리 사용되는 저장 매체로는 하드 디스크를 들 수 있지만, 최근에는 기술의 급격한 발달로 메인 메모리가 저장 매체로 사용되는 경우도 있으며, 또한 플래시 메모리가 저장 매체로 사용되기도 한다.

하드 디스크는 액세스 속도가 매우 느리지만 가격이 저렴하고 크기에 제한이 매우 적다는 장점이 있어서 가장 널리 사용되는 저장 매체이다.

반면에 메인 메모리는 액세스 속도가 매우 빨라서 높은 성능을 보장할 수 있는 반면에 가격이 매우 비싸고, 운영체제에 의해서 크기에 제한을 많이 받는다.

더구나 휘발성이 있어서 전력 공급이 정지되는 경우에는 저장하고 있는 데이터가 모두 사라지는 치명적인 문제점을 가지고 있다.

플래시 메모리는 하드 디스크와 메인 메모리의 장점을 모두 가지고 있는 저장 매체이다.

플래시 메모리는 하드 디스크에 비해 속도가 매우 빠르면서도 메인 메모리와 같은 휘발성이 없다.

그러나 읽기 속도와 쓰기 속도의 차이가 매우 크고, 매체의 수명에 제한이 있다는 문제점이 있다.

이와 같이 저장 매체들은 각각 하드웨어적인 특성이 매우 다양하므로 응용 분야의 특성에 따라 적절한 저장 매체를 선택하여야 원하는 성능을 얻을 수 있으며, 관련 분야의 기술적 수준 또한 심각하게 고려하여야만 한다.

본 장에서는 먼저 기본적이고 가장 대중적인 저장 매체인 하드 디스크와 널리 사용되는 운영체제인 유닉스 시스템을 기준으로 하여 데이터 저장 기능을 설명하기로 한다.

일반적으로 운영체제에서는 파일이라는 개념을 이용하여 데이터를 저장하는데 이 때, 저장 관리되는 데이터는 동일한 포맷을 가지는 여러 개의 레코드이다.

다시 말하면, 파일은 동일한 포맷을 가지는 여러 개의 레코드가 순서대로 저장되어 있는 구조이다.

파일을 구성하는 레코드는 또한 여러 개의 필드로 이루어져 있다.

유닉스의 파일 시스템에서 레코드를 저장하기 위한 저장 단위를 데이

터 페이지라고 부른다.

즉, 파일은 물리적으로는 여러 개의 데이터 페이지로 구성되며, 레코드들은 데이터 페이지에 저장되는 것이다.

하나의 파일에 속한 레코드의 개수는 경우에 따라 다르지만 일반적으로는 매우 많아서 하나의 데이터 페이지에 모든 레코드를 저장할 수는 없다.

그러므로 하나의 파일은 여러 개의 데이터 페이지로 구성되는 것이며, 여러 개의 데이터 페이지를 관리하기 위한 자료 구조가 필요하다.

여러 개의 자료 구조를 관리하는 가장 일반적인 방법은 물리적으로 인접한 위치에 두는 것이지만 데이터 페이지의 경우에는 이러한 방법을 사용할 수 없다.

왜냐하면 파일의 크기는 데이터의 양에 따라서 매우 가변적이기 때문에 항상 물리적으로 연속된 공간을 확보하는 것이 매우 어렵고 비용이 많이 발생하는 일이기 때문이다.

두 번째로 생각할 수 있는 방법은 여러 자료 구조를 연결 리스트를 사용하여 관리하는 것이다.

이 방법의 경우에는 구조가 매우 간단한 장점이 있지만 연결 리스트의 중간 부분에 위치한 데이터 페이지를 액세스하기 위해서 리스트의 처음 부분부터 링크를 따라 가야 하기 때문에 매우 많은 디스크 액세스를 요구한다는 문제가 있다.

세 번째로 생각할 수 있는 방법은 파일을 구성하는 각 데이터 페이지들에 대한 물리적 주소를 관리하는 별도의 리스트를 만드는 것이다.

이 방법은 파일의 크기에 비하여 데이터 페이지에 대한 리스트의 크기가 훨씬 작으므로 작은 수의 디스크 액세스 만으로 원하는 데이터 페이지의 물리적 주소를 알아낼 수 있다는 장점이 있다.

그러나 파일의 크기가 커짐에 따라서 데이터 페이지의 물리적 주소를 기록한 리스트의 크기도 커지게 되므로 리스트의 크기가 가변적으로 조정될 수 있어야 한다.

유닉스 파일 시스템에서는 데이터 페이지를 관리하는 별도의 리스트를 사용하며, 이러한 리스트 구조를 inode라고 부른다.

그림 1은 유닉스 파일 시스템에서 사용하는 inode와 데이터 페이지의 관계를 도식화 한 것이다.

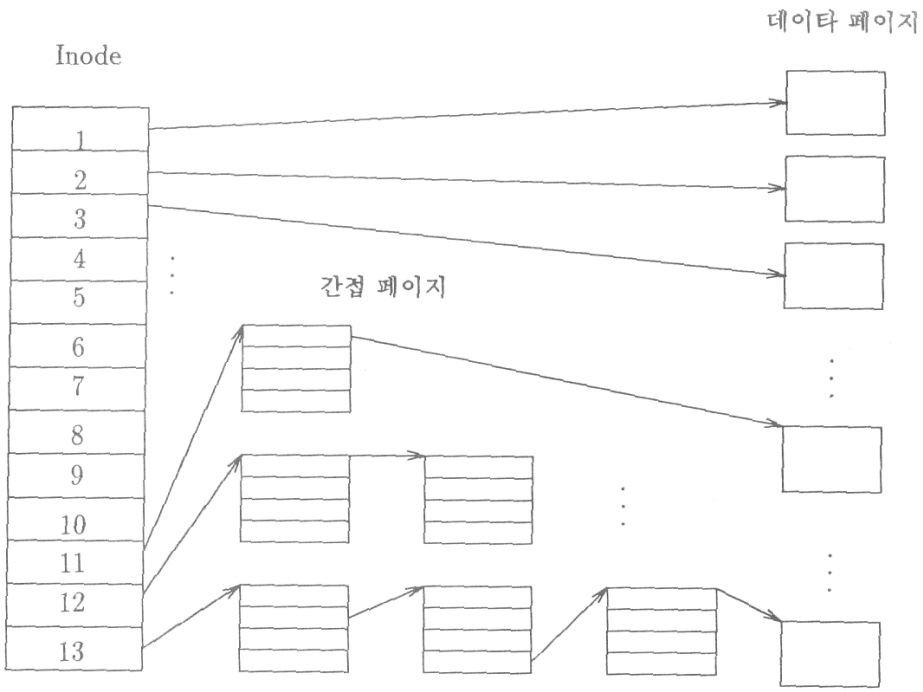

그림 1. 유닉스 파일 시스템에서 inode와 데이터 페이지의 구조.

파일을 구성하는 데이터 페이지는 물리적으로는 하드 디스크의 여러 부분에 흩어져 있지만 논리적으로는 하나의 파일에 속하여 순서적으로 관리된다.

사용자가 파일에 저장된 레코드를 액세스 하기 위해서는 데이터 페이

지에 대한 물리적인 주소가 필요하므로 유닉스 파일 시스템에서는 Inode를 사용하여 파일 내에서의 논리적인 주소를 하드 디스크 상의 물리적인 주소로 바꾸어 주는 역할을 수행한다.

그림 1.1에서 보는 바와 같이 각 inode에는 데이터 페이지에 대한 물리적인 주소를 나타내는 여러 개의 엔트리가 존재한다.

그림에서는 초기 10개의 엔트리에는 직접 데이터 페이지의 물리적 주소가 저장되지만, 그 이후의 엔트리에서는 데이터 페이지의 물리적 주소 리스트가 저장된 간접 페이지의 주소가 저장된다.

첫 번째 간접 페이지를 통하여 데이터를 액세스하기 위해서는 먼저 첫 번째 간접 페이지를 디스크로부터 읽어 들여, 해당되는 데이터페이지를 가리키는 엔트리를 찾고, 이를 이용하여 데이터 페이지를 읽어 들여야 한다.

또한 inode의 12번째 엔트리는 첫 번째 간접 페이지를 가리키는 엔트리들의 리스트가 저장된 두 번째 간접 페이지를 가리킨다.

따라서 두 번째 간접페이지를 통하여 데이터를 액세스하는 경우 데이터 페이지 액세스 외에도 간접페이지를 액세스하기 위한 두 번의 디스크 액세스가 추가로 발생된다.

마지막으로 inode의 13번째 엔트리도 유사한 방식으로 세 번째 간접 페이지를 가리키게 된다.

저장매체를 관리하기 위하여 제공되어야 하는 기능으로는 다음과 같은 것들이 있다.

- Initialize

  저장 매체를 관리하기 위한 주 기억장치내의 모든 자료 구조를 초기화 하는 기능.

- Finalize

  저장 매체를 관리하기 위한 주 기억장치내의 모든 자료 구조를 정리하는 기능.

- Format

  하나의 볼륨을 관리할 수 있는 구조로 포맷하는 기능.

- Mount

  하나의 볼륨을 사용할 수 있도록 등록하는 기능.

- Dismount

  Mount에 의하여 등록된 볼륨을 등록에서 제거하는 기능.

- Allocate

  자유 페이지 풀에서 페이지를 찾아 할당해 주는 기능.

- Free

  Allocate에 의하여 할당되어 있던 페이지를 자유 페이지 풀로 반환해주는 기능.

- Read

  페이지의 내용을 디스크로부터 읽는 기능.

- Write

  페이지의 내용을 디스크에 쓰는 기능.

1.3 데이터 관리 기능

저장 시스템에서 데이터는 레코드의 형태로 관리되며, 데이터 페이지에 저장된다.

레코드는 여러 개의 필드로 구성되는데 이 때 각각의 필드는 특정 데이터 값을 가지게 된다.

즉, 레코드는 서로 연관이 있는 데이터들의 모임이라고 할 수 있다.

그림 2는 모든 필드의 크기가 고정되어 있는 경우 일반적인 레코드의 구조를 도식화 한 것이다.

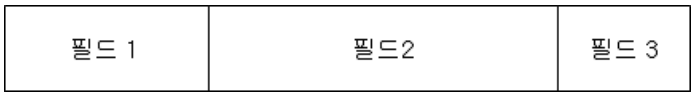

그림 2. 고정길이 레코드의 표현 방법.

레코드는 데이터 페이지에 저장되어 관리되는데, 이 때 데이터 페이지의 자투리 공간에 대한 활용 여부에 따라 신장조직과 비신장조직으로 나눌 수 있다.

비신장조직은 레코드의 전체 부분을 하나의 데이터 페이지에 저장하는 방법을 말한다.

그림 3은 비신장조직의 예를 도식화 한 것이다.

그림에서는 각 레코드의 길이가 다를 수도 있다고 가정하였다.

각각의 데이터 페이지에는 전체 레코드가 저장될 수 있는 경우에만 저장되었고, 남은 영역은 사용되지 않는 것을 볼 수 있다.

그림 3. 비신장조직의 레코드 저장 방법.

일반적으로 데이터 페이지는 고정 길이로 관리되기 때문에 레코드의 길이가 고정되어 있는 경우 하나의 데이터 페이지에 저장될 수 있는 레코드의 개수도 고정적으로 결정된다.

이렇게 하나의 데이터 페이지에 저장될 수 있는 레코드의 개수를 해당 데이터 페이지의 블로킹 팩터(bfr)라고 부른다.

데이터 페이지의 크기가 B바이트이고, 레코드의 크기가 R바이트인 경우 해당 데이터 페이지의 블로킹 팩터는 ⌊B/R⌋이 된다.

예를 들어, 데이터 페이지의 크기가 2048바이트이고, 레코드의 크기가 100바이트인 경우 데이터 페이지의 블로킹 팩터는 20이 되는 것이다.

일반적으로 데이터 페이지의 크기는 레코드의 크기로 나누어지는 값이 아니므로 데이터 페이지에는 레코드를 넣고 남는 공간이 존재하게 된다.

특히, 레코드의 크기가 큰 경우에는 남는 공간이 가볍게 여길 수 없을 정도로 커질 수도 있다.

극단적인 경우, 예를 들어 데이터 페이지의 크기가 2048바이트이고, 레코드의 크기가 1025바이트인 경우에는 블로킹 팩터가 1이 되고, 데이터 페이지 내의 남는 공간이 1023바이트가 된다.

다시 말하면, 전체 사용 공간 중 50%정도의 공간을 제대로 사용하지 못하고 낭비하는 결과를 초래할 수도 있는 것이다.

이러한 자투리 공간을 사용하기 위하여 레코드의 일부분만을 저장하고 남은 부분은 다른 데이터 페이지에 저장하는 방식을 신장조직이라고 한다.

그림 4는 신장조직의 예를 도식화 한 것이다.

그림에서 레코드4는 블록i와 블록i+ 1로 나뉘어 저장되어 있는 것을 확

인 할 수 있다.

이 경우 레코드4를 액세스하기 위해서는 먼저 블록 I를 액세스하고, 레코드4의 뒷부분에 있는 포인터 정보를 따라 블록i+1을 액세스 하여 레코드4의 나머지 부분을 액세스 하여야 한다.

그림 4. 신장조직의 레코드 저장 방법.

신장조직은 저장 공간이 낭비되는 것은 방지할 수 있지만, 레코드의 저장 방식이 2원화 되므로 레코드 관리에 오버헤드로 작용할 수 있다는 문제가 있다.

특히, 레코드의 크기가 작은 경우에는 절약할 수 있는 저장 공간은 작은 반면에 레코드 관리 오버헤드는 커지는 현상이 발생할 수 있다.

신장조직과 비신장조직 중 특별히 어떤 방식이 레코드 관리에 더 적합하다고 일반적으로 말하는 것은 무리가 있다.

레코드의 특성에 따라 적합한 방식을 선택하여야 할 것이다.

과거의 저장 시스템에서는 일반적으로 비신장조직을 주로 사용하였으나 최근에는 멀티미디어 데이터가 늘어남에 따라 데이터 레코드의 크기가 큰 응용 분야가 많아지고 있으므로 신장조직을 사용하는 경우도 늘어나고 있다.

심한 경우에는 레코드의 크기가 데이터 페이지 보다 더 큰 경우도 있으므로 이러한 경우에는 필연적으로 신장조직을 사용할 수밖에 없는 상황이 발생하게 된다.

이렇게 크기가 데이터 페이지보다 큰 레코드를 대용량 레코드라고 부른다.

이에 반하여 하나의 페이지에 저장될 수 있는 레코드를 소용량 레코드라고 정의한다.

이와 같이 레코드의 크기가 한 페이지를 초과하는지의 여부에 따라 레코드를 분류하는 이유는 한 페이지를 초과하게 되면 공간 할당(space allocation), 레코드 조작(object manipulation) 등이 복잡해지고, 동시성 제어, 회복 기능의 구현에도 다른 방법이 요구되기 때문이다.

레코드의 크기가 커져서 대용량 레코드가 되는 이유는 크게 두 가지로 나눌 수 있다.

첫째는 한 레코드의 각 필드들의 크기의 합이 한 페이지의 크기를 넘게 된 경우이며, 둘째는 레코드를 구성하는 필드들 중에 크기가 한 페이지 이상 되는 큰 필드가 있는 경우이다.

즉, 전자의 경우 하나의 레코드가 많은 필드로 구성된 경우이고, 후자의 경우는 앞에서 많이 언급된 영상, 음성, 문서 등의 멀티미디어 정보, X선 사진, 지도 등의 큰 데이터가 하나의 필드가 되는 경우이다.

관계형 데이터베이스에서는 튜플을 구성하는 필드들이 많아서 튜플이 대용량 레코드가 되는 경우는 드물다.

그러나 복잡한 응용 분야에서는 작은 필드들의 크기의 합으로 인하여 대용량 레코드가 되는 경우가 빈번하게 발생한다.

예를 들면 고급 데이터 모델을 지원하는 응용 분야에서는 집합, 긴 리스트(long list), 삽입가능 배열(insertable array) 등을 표현하기 위하여 대용량 레코드가 사용된다.

즉, 각각의 구성원소를 필드로 간주함으로써 필드가 많아 대용량 레코드가 되는 경우에 해당된다.

최근의 데이터베이스 응용에서는 대용량 레코드에 대한 요구 사항이 급격하게 증가하고 있으므로 저장 시스템에서도 대용량 레코드를 효율적으로 관리할 수 있는 기능이 개발되고 있다.

대용량 객체의 용도를 고려할 때, 대용량 객체를 지원하는 저장 시스템은 다음과 같은 기능을 제공할 수 있어야 한다.

첫째, 레코드 안의 임의의 위치에서 임의의 양의 바이트들에 대한 판독(read), 기록(write), 삽입(insert), 삭제(delete) 등의 연산이 제공되어야 한다.

연산의 대상이 레코드 전체인 연산을 레코드 단위 연산이라 정의하고, 연산의 대상이 레코드의 부분인 연산을 바이트 단위 연산이라 정의한다.

소용량 레코드의 경우에는 데이터가 모두 한 페이지 안에 저장되어 있으며 따라서 데이터를 모두 기억장치에 로드할 수 있기 때문에 레코드를 조작하는 연산들이 레코드 단위로 이루어져도 성능에 영향을 미치지 않는다.

그러나 대용량 레코드의 경우는 다음과 같은 이유로 인하여 바이트 단위의 연산의 제공이 필수적이다.

- 한 번에 레코드 전체를 생성, 검색, 갱신하는 레코드 단위 연산이 주기억장치의 물리적 제한으로 인하여 불가능한 경우가 있다. 예를 들어 최대 230크기의 레코드를 지원하는 저장시스템이 있다고할 때, 이러한 레코드가 한 순간에 주기억장치에 로드 된다는 것은 실제적으로 불가능하다. 그러므로 레코드를 부분적으로도 로드하여 연산할 수 있는 기능이 있어야 한다.
- 레코드의 부분만을 액세스하는 응용 분야들에 있어서 레코드 단위

의 연산은 성능 저하를 가져온다. 여러 개의 곡이 녹음되어 있는 CD를 저장하고 있는 레코드를 생각해보면, CD에 수록되어 있는 곡을 전부 듣는 경우도 있지만 원하는 곡만 골라서 듣고 싶은 경우도 많다. 이러한 경우에 레코드의 한 위치에서 시작하여 일정량의 바이트를 순차적으로 액세스하는 연산이 제공되어야 한다. 또한 영화 편집을 위해 사용되는 레코드에서는 프레임의 수정 또는 새로운 프레임의 삽입으로 인하여 데이터들을 레코드 중간에서 삽입하고 삭제하는 연산이 빈번하게 일어난다. 이와 같이 레코드에 가해지는 조작이 레코드의 부분에서만 행해지는데도 불구하고 레코드 전체를 메모리에 로드하는 것은 전체 시스템의 성능을 크게 떨어뜨린다. 그러므로 필요한 부분만 액세스 할 수 있도록 하는 기능이 있어야 한다.

둘째, 대용량 레코드의 임의의 위치를 효율적으로 찾아갈 수 있어야 한다.

위에서 열거한 이유로 인하여 바이트 단위 연산은 대용량 레코드를 지원하는 저장 시스템이 제공해 주어야 하는 꼭 필요한 기능이다.

그런데 바이트 단위 연산은 먼저 레코드 안에서 연산이 시작될 위치를 찾고, 다음에 찾은 위치에서 일정량의 바이트를 검색하거나 삽입, 삭제하게 되므로 먼저 레코드 안에서 연산이 시작하는 위치를 찾아가는 것이 효율적으로 수행되어야 한다.

셋째, 대용량 레코드의 끝에 임의의 양의 바이트들을 덧붙이는 첨가(append) 연산이 저장장치 이용률을 높일 수 있도록 효율적으로 제공되어야 한다.

위에서 언급했듯이 대용량 레코드의 경우에는 주기억장치의 물리적인 제한 조건 때문에 한 번의 연산으로 레코드 전체를 생성할 수 없는 경우가 있다.

따라서 레코드를 점차적으로 생성해가야 하는데, 삽입 연산을 그대로 적용하는 것은 저장장치 오버헤드 측면에서 문제가 있다.

그러므로 레코드의 끝에 데이터를 덧붙이는 첨가 연산은 삽입 연산과는 별도로 저장장치 이용률을 증가시킬 수 있게 효율적으로 제공되어야 한다.

넷째, 대용량 레코드를 순차적으로 검색할 때 빠른 검색이 가능해야 한다.

영화와 같은 영상 자료를 대용량 레코드로 저장한 경우 만일 검색시간이 너무 길다면, 사용자는 생동감 있는 영화가 아닌 느린 영상의 움직임을 보게 될 것이며 화면과 음향이 동기화되지 않는 이상 현상이 발생할 것이다.

따라서 같은 레코드를 구성하고 있는 페이지들은 가능한 한 인접한 위치에 할당함으로써 디스크 탐색 시간(seek time)을 줄여야 한다.

2. 질의어 : 관계 대수

본 장에서는 데이터베이스 관리 시스템을 사용하기 위한 질의어에 대하여 설명한다.

관계형 데이터 모델에 따라 저장된 데이터에 대한 질의 방법을 관계 언어라 한다. 대표적인 관계 언어에는 관계 대수와 관계 해석이 있으며, 관계 해석은 다시 투플 관계 해석과 도메인 관계해석으로 구분할 수 있다.

관계 언어는 액세스하고자 하는 데이터가 무엇이냐를 기술할 뿐 어떻게 그 데이터를 액세스하는지 방법을 기술하지 않는 선언적 언어이다.

SQL은 이러한 관계 언어를 이론적 기반으로 하여 만들어진 질의어이다.

그러므로 SQL에 대하여 설명하기에 앞서 관계 대수에 대하여 설명하기로 한다.

관계 대수란 관계형 데이터 모델의 기본 구조인 릴레이션에 대하여 여러 가지 연산자를 정의함으로써 릴레이션에 대한 가공 방법을 제공하는 대수식을 의미한다.

관계 대수에서 정의하는 연산자를 관계 연산자라고 하는데, 관계 연산자의 종류에는 단항 관계 연산자와 이항 관계 연산자가 있다.

단항 관계 연산자는 하나의 릴레이션에 대하여 정의된 관계 연산자이고, 이항 관계 연산자는 두 개의 릴레이션에 대하여 정의된 관계 연산자를 의미한다.

관계대수는 2가지 단항 관계 연산자(셀렉션, 프로젝션)와 3가지 집합 연산자(합집합, 차집합, 카테시안 프로덕트)로 이루어진 5가지 필수 연산자로 구성된다.

2.1 단항 관계 연산자

가) 프로젝션(Projection)

프로젝션 연산자는 릴레이션을 구성하는 여러 속성들 중에서 지정한 속성들만을 추출하는 연산자이다.

그러므로 릴레이션에 속산 투플들은 지정된 속성들만으로 이루어진 투플로 변형된다.

프로젝션 연산자의 표기법은 다음과 같다.

표기법 : Π A1, A2, ..., Am(R)

의미 : 릴레이션 R을 구성하는 속성들 중 일부만을 추출하는 것

다음과 같이 5개의 속성(film_id, title, rating, rental_rate, replacement_cost)으로 이루어진 FILM 릴레이션에서 title, rating, rental_rate 속성만을 프로젝션하는 관계 대수식을 작성해 보자.

FILM	film_id	title	rating	rental_rate	replacement_cost
	1	ACADEMY DINOSAUR	PG	0.99	20.99
	2	ACE GOLDFINGER	G	4.99	12.99
	3	ADAPTATION HOLES	NC-17	2.99	18.99
	4	AFFAIR PREJUDICE	G	2.99	26.99
	5	AFRICAN EGG	G	2.99	22.99

6	AGENT TRUMAN	PG	2.99	17.99
7	AIRPLANE SIERRA	PG-13	4.99	28.99
8	AIRPORT POLLOCK	R	4.99	15.99
9	ALABAMA DEVIL	PG-13	2.99	21.99
10	ALADDIN CALENDAR	NC-17	4.99	24.99

$$\Pi_{title,rating,\ rental\_rate}(FILM)$$

이 프로젝션 연산의 수행 결과는 그림 5와 같다.

즉, FILM 릴레이션의 5개 속성 중 관계 대수식에 기술된 3개의 속성만이 추출된다.

이 때, 투플을 구성하는 속성은 감소했지만, 릴레이션에 속한 투플의 개수는 총 10개로 그대로 유지된다.

RESULT	title	rating	rental_rate
	ACADEMY DINOSAUR	PG	0.99
	ACE GOLDFINGER	G	4.99
	ADAPTATION HOLES	NC-17	2.99
	AFFAIR PREJUDICE	G	2.99
	AFRICAN EGG	G	2.99
	AGENT TRUMAN	PG	2.99
	AIRPLANE SIERRA	PG-13	4.99
	AIRPORT POLLOCK	R	4.99
	ALABAMA DEVIL	PG-13	2.99

그림 5. 프로젝션 수행 결과.

나) 셀렉션(Selection)

셀렉션 연산자는 릴레이션에 속한 투플들 중에서 주어진 조건을 만족하는 투플들만을 선택하여 추출하는 연산자이다.

그러므로 릴레이션에 속산 투플들 중 조건을 만족하지 않는 투플들은 결과 릴레이션에 나타나지 않게 된다.

섹렉션 연산자의 표기법은 다음과 같다.

표기법 : σF(R)

의미 : 릴레이션 R에 속한 투플들 중 주어진 조건 F를 만족하는 투플들만을 추출하는 것

투플에 대한 조건 F는 투플의 특정 속성값에 대한 비교 조건을 부여하는 것이다.

이제, 셀렉션 연산자 사용을 연습하기 위해 위에서 살펴본 FILM 릴레이션에서 rental_rate 속성의 값이 2.99인 투플들만을 추출해보자.

이 질의를 위한 관계 대수식은 다음과 같다.

$$\sigma_{\text{rental\_rate = 2.99}}(\text{FILM})$$

그림 6은 위의 셀렉션 연산 수행 결과를 보인 것이다.

결과에서 총 10개의 투플 중에서 rental_rate 속성의 값이 2.99인 5개

RESULT	film_id	title	rating	rental_rate	replacement_cost
	3	ADAPTATION HOLES	NC-17	2.99	18.99
	4	AFFAIR PREJUDICE	G	2.99	26.99
	5	AFRICAN EGG	G	2.99	22.99
	6	AGENT TRUMAN	PG	2.99	17.99
	9	ALABAMA DEVIL	PG-13	2.99	21.99

그림 6. 셀렉션 수행 결과.

의 투플들만이 추출되었다.

여러 비교 조건을 연결해서 사용하는 경우에는 AND, OR, NOT 등의 논리 연산자를 사용한다.

예를 들어, FILM 릴레이션에서 rental_rate 속성의 값이 2보다 크고, 5보다 작은 모든 투플을 추출하는 관계 대수식은 다음과 같이 작성할 수 있다.

$$\sigma_{\text{rental\_rate} > 2 \text{ AND } \text{rental\_rate} < 5}(\text{FILM})$$

그림 7은 위의 질의 결과를 나타낸 것이다.

질의 결과를 살펴보면 rental_rate의 속성값이 2보다 크고, 5보다 작은 투플들만이 추출된 것을 확인할 수 있다.

RESULT	film_id	title	rating	rental_rate	replacement_cost
	2	ACE GOLDFINGER	G	4.99	12.99
	3	ADAPTATION HOLES	NC-17	2.99	18.99

4	AFFAIR PREJUDICE	G	2.99	26.99
5	AFRICAN EGG	G	2.99	22.99
6	AGENT TRUMAN	PG	2.99	17.99
7	AIRPLANE SIERRA	PG-13	4.99	28.99
8	AIRPORT POLLOCK	R	4.99	15.99
9	ALABAMA DEVIL	PG-13	2.99	21.99
10	ALADDIN CALENDAR	NC-17	4.99	24.99

그림 7. 셀렉션에서 여러 조건을 사용한 결과.

2.2 이항 관계 연산자

관계 대수의 필수 연산자 중 이항 관계 연산자는 합집합, 차집합, 카테시안 프로덕트가 있다.

이 중, 합집합과 차집합 연산자는 모든 릴레이션에 대하여 적용될 수 있는 것이 아니라 합집합 호환성이 있는 두 개의 릴레이션에 대해서만 적용할 수 있다.

차수가 각각 n과 m인 두 개의 릴레이션 R(A1, A2, A3, ..., An)과 S(B1, B2, B3, ..., Bm)가 있을 때에 n = m이고 상호 대응되는 모든 속성들의 도메인이 같으면 R과 S는 합집합 호환성이 있다고 한다.

예를들어 학생(학번, 이름, 전화번호, 주소, 성별)과 교수(사번, 이름, 전화번호, 주소, 성별)은 두 릴레이션 모두 속성의 개수가 5개이므로 차수가 동일하고, 서로 대응되는 속성들의 도메인이 도일하므로 합집합 호환성이 있다.

가) 합집합(Union)

합집합 연산은 주어진 두 릴레이션의 투플을 모두 포함하는 결과 릴레이션을 만드는 것이다.

이 때, 모든 속성값이 동일한 두 개의 투플이 생기는 경우에는 중복이 제거되어 하나의 투플만이 남게 된다.

합집합 연산의 표기법은 다음과 같다.

표기법 : R ∪ S

의미 : 릴레이션 R 또는 S에 속한 투플들의 집합

다음과 같은 FILM2 릴레이션을 생각해보자.

FILM2	film_id	title	rating	rental_rate	replacement_cost
	2	ACE GOLDFINGER	G	4.99	12.99
	6	AGENT TRUMAN	PG	2.99	17.99
	7	AIRPLANE SIERRA	PG-13	4.99	28.99
	8	AIRPORT POLLOCK	R	4.99	15.99
	13	ALI FOREVER	PG	4.99	21.99
	14	ALICE FANTASIA	NC-17	2.99	23.99
	15	ALIEN CENTER	NC-17	2.99	10.99
	16	ALLEY EVOLUTION	NC-17	2.99	23.99
	17	ALONE TRIP	R	0.99	14.99

18	ALTER VICTORY	PG-13	0.99	27.99
19	AMADEUS HOLY	PG	0.99	20.99
20	AMELIE HELLFIGHTERS	R	4.99	23.99

앞서 살펴본 FILM 릴레이션과 FILM2 릴레이션은 합집합 호환성이 있으므로 합집합 연산이 가능하다.

두 릴레이션을 합집합하는 관계 대수식은 다음과 같다.

FILM ∪ FILM2

그림 8은 두 릴레이션의 합집합 결과를 보인 것이다.

RESULT	film_id	title	rating	rental_rate	replacement_cost
	1	ACADEMY DINOSAUR	PG	0.99	20.99
	2	ACE GOLDFINGER	G	4.99	12.99
	3	ADAPTATION HOLES	NC-17	2.99	18.99
	4	AFFAIR PREJUDICE	G	2.99	26.99
	5	AFRICAN EGG	G	2.99	22.99
	6	AGENT TRUMAN	PG	2.99	17.99
	7	AIRPLANE SIERRA	PG-13	4.99	28.99
	8	AIRPORT POLLOCK	R	4.99	15.99
	9	ALABAMA DEVIL	PG-13	2.99	21.99
	10	ALADDIN CALENDAR	NC-17	4.99	24.99
	13	ALI FOREVER	PG	4.99	21.99

14	ALICE FANTASIA	NC-17	2.99	23.99
15	ALIEN CENTER	NC-17	2.99	10.99
16	ALLEY EVOLUTION	NC-17	2.99	23.99
17	ALONE TRIP	R	0.99	14.99
18	ALTER VICTORY	PG-13	0.99	27.99
19	AMADEUS HOLY	PG	0.99	20.99
20	AMELIE HELLFIGHTERS	R	4.99	23.99

그림 8. 합집합 연산의 결과.

결과인 RESULT 릴레이션에는 FILM 릴레이션에 속한 모든 투플들과 FILM2 릴레이션에 속한 모든 투플들이 포함되어 있는 것을 알 수 있다.

또한, film_id의 속성값이 2, 6, 7, 8인 네 개의 투플은 FILM과 FILM2 릴레이션에 모두 속해있기 때문에 중복이 제거되어 총 투플의 개수는 18개뿐이다.

나) 차집합(Set Difference)

차집합 연산의 표기법은 다음과 같다.

표기법 : R - S

의미 : 릴레이션 R 에는 속하지만 S에는 속하지 않는 투플들의 집합

차집합 연산은 주어진 두 릴레이션 중 선행 릴레이션에는 속하면서 후행 릴레이션에는 속하지 않는 투플을 포함하는 결과 릴레이션을 만드는 것이다.

앞서 살펴본 FILM 릴레이션과 FILM2 릴레이션은 합집합 호환성이 있으므로 차집합 연산이 가능하다.

두 릴레이션을 차집합하는 관계 대수식은 다음과 같다.

FILM - FILM2

그림 9는 두 릴레이션의 차집합 결과를 보인 것이다.

RESULT	film_id	title	rating	rental_rate	replacement_cost
	1	ACADEMY DINOSAUR	PG	0.99	20.99
	3	ADAPTATION HOLES	NC-17	2.99	18.99
	4	AFFAIR PREJUDICE	G	2.99	26.99
	5	AFRICAN EGG	G	2.99	22.99
	9	ALABAMA DEVIL	PG-13	2.99	21.99
	10	ALADDIN CALENDAR	NC-17	4.99	24.99

그림 9. FILM - FILM2 차집합 연산의 결과.

결과인 RESULT 릴레이션에는 FILM 릴레이션에 속한 투플들 중에서 FILM2 릴레이션에 속한 투플들을 제외한 나머지 투플들만이 포함되어 있는 것을 알 수 있다.

이번에는 반대로 다음과 같이 FILM2 릴레이션에서 FILM 릴레이션의

투플을 제외하는 차집합 연산을 생각해 보자.

FILM2 - FILM

그림 10은 두 릴레이션의 차집합 결과를 보인 것이다.

이 결과에도 FILM2 릴레이션에 속한 투플들 중에서 FILM 릴레이션에도 속해 있는 투플(film_id가 2, 6, 7, 8인 투플)들을 제외한 나머지 투플들만이 나타나는 것을 알 수 있다.

RESULT	film_id	title	rating	rental_rate	replacement_cost
	13	ALI FOREVER	PG	4.99	21.99
	14	ALICE FANTASIA	NC-17	2.99	23.99
	15	ALIEN CENTER	NC-17	2.99	10.99
	16	ALLEY EVOLUTION	NC-17	2.99	23.99
	17	ALONE TRIP	R	0.99	14.99
	18	ALTER VICTORY	PG-13	0.99	27.99
	19	AMADEUS HOLY	PG	0.99	20.99
	20	AMELIE HELLFIGHTERS	R	4.99	23.99

그림 10. FILM2 - FILM 차집합 연산의 결과.

다) 카테시안 프로덕트(Cartesian Product)

카테시안 프로덕트는 두 개의 릴레이션에 속한 투플들간의 모든 가능한 조합을 나타내는 연산자이다.

카테시안 프로덕트의 표기법은 다음과 같다.

표기법 : R X S

의미 : 릴레이션 R에 속한 각 투플들에 대하여 릴레이션 S에 속한 모든 투플들을 결합시킨 투플들의 집합

차집합 연산은 주어진 두 릴레이션 중 선행 릴레이션에는 속하면서 후행 릴레이션에는 속하지 않는 투플을 포함하는 결과 릴레이션을 만드는 것이다.

다음과 같은 ACTOR 릴레이션을 생각해보자.

ACTOR	actor_id	first_name	last_name
	1	PENELOPE	GUINESS
	2	NICK	WAHLBERG
	3	ED	CHASE
	4	JENNIFER	DAVIS
	5	JOHNNY	LOLLOBRIGIDA
	6	BETTE	NICHOLSON
	7	GRACE	MOSTEL
	8	MATTHEW	JOHANSSON
	9	JOE	SWANK
	10	CHRISTIAN	GABLE

이제 다음과 같이 FILM 릴레이션과 ACTOR 릴레이션에 대한 카테시안 프로덕트 연산을 생각해보자.

FILM X ACTOR

이 연산의 수행 결과는 그림 11과 같다.

그림에서 결과 릴레이션의 투플들은 FILM 릴레이션의 투플과 ACTOR 릴레이션의 투플들간의 모든 가능한 조합을 나타내고 있는 것을 알 수 있다.

RESULT	film_id	title	rating	rental_rate	replacement_cost	actor_id	first_name	last_name
	1	ACADEMY DINOSAUR	PG	0.99	20.99	1	PENELOPE	GUINESS
	1	ACADEMY DINOSAUR	PG	0.99	20.99	2	NICK	WAHLBERG
	1	ACADEMY DINOSAUR	PG	0.99	20.99	3	ED	CHASE
	1	ACADEMY DINOSAUR	PG	0.99	20.99	4	JENNIFER	DAVIS
	1	ACADEMY DINOSAUR	PG	0.99	20.99	5	JOHNNY	LOLLOBRIGIDA
	1	ACADEMY DINOSAUR	PG	0.99	20.99	6	BETTE	NICHOLSON
	1	ACADEMY DINOSAUR	PG	0.99	20.99	7	GRACE	MOSTEL
	1	ACADEMY DINOSAUR	PG	0.99	20.99	8	MATTHEW	JOHANSSON
	1	ACADEMY DINOSAUR	PG	0.99	20.99	9	JOE	SWANK
	1	ACADEMY DINOSAUR	PG	0.99	20.99	10	CHRISTIAN	GABLE
	2	ACE GOLDFINGER	G	4.99	12.99	1	PENELOPE	GUINESS
	2	ACE GOLDFINGER	G	4.99	12.99	2	NICK	WAHLBERG
	2	ACE GOLDFINGER	G	4.99	12.99	3	ED	CHASE
	2	ACE GOLDFINGER	G	4.99	12.99	4	JENNIFER	DAVIS
	2	ACE GOLDFINGER	G	4.99	12.99	5	JOHNNY	LOLLOBRIGIDA
	2	ACE GOLDFINGER	G	4.99	12.99	6	BETTE	NICHOLSON
	2	ACE GOLDFINGER	G	4.99	12.99	7	GRACE	MOSTEL
	2	ACE GOLDFINGER	G	4.99	12.99	8	MATTHEW	JOHANSSON
	2	ACE GOLDFINGER	G	4.99	12.99	9	JOE	SWANK
	2	ACE GOLDFINGER	G	4.99	12.99	10	CHRISTIAN	GABLE
	3	ADAPTATION HOLES	NC-17	2.99	18.99	1	PENELOPE	GUINESS
	3	ADAPTATION HOLES	NC-17	2.99	18.99	2	NICK	WAHLBERG
	3	ADAPTATION HOLES	NC-17	2.99	18.99	3	ED	CHASE
	3	ADAPTATION HOLES	NC-17	2.99	18.99	4	JENNIFER	DAVIS

3	ADAPTATION HOLES	NC-17	2.99	18.99	5	JOHNNY	LOLLOBRIGIDA
3	ADAPTATION HOLES	NC-17	2.99	18.99	6	BETTE	NICHOLSON
3	ADAPTATION HOLES	NC-17	2.99	18.99	7	GRACE	MOSTEL
3	ADAPTATION HOLES	NC-17	2.99	18.99	8	MATTHEW	JOHANSSON
3	ADAPTATION HOLES	NC-17	2.99	18.99	9	JOE	SWANK
3	ADAPTATION HOLES	NC-17	2.99	18.99	10	CHRISTIAN	GABLE
4	AFFAIR PREJUDICE	G	2.99	26.99	1	PENELOPE	GUINESS
4	AFFAIR PREJUDICE	G	2.99	26.99	2	NICK	WAHLBERG
4	AFFAIR PREJUDICE	G	2.99	26.99	3	ED	CHASE
4	AFFAIR PREJUDICE	G	2.99	26.99	4	JENNIFER	DAVIS
4	AFFAIR PREJUDICE	G	2.99	26.99	5	JOHNNY	LOLLOBRIGIDA
4	AFFAIR PREJUDICE	G	2.99	26.99	6	BETTE	NICHOLSON
4	AFFAIR PREJUDICE	G	2.99	26.99	7	GRACE	MOSTEL
4	AFFAIR PREJUDICE	G	2.99	26.99	8	MATTHEW	JOHANSSON
4	AFFAIR PREJUDICE	G	2.99	26.99	9	JOE	SWANK
4	AFFAIR PREJUDICE	G	2.99	26.99	10	CHRISTIAN	GABLE
5	AFRICAN EGG	G	2.99	22.99	1	PENELOPE	GUINESS
5	AFRICAN EGG	G	2.99	22.99	2	NICK	WAHLBERG
5	AFRICAN EGG	G	2.99	22.99	3	ED	CHASE
5	AFRICAN EGG	G	2.99	22.99	4	JENNIFER	DAVIS
5	AFRICAN EGG	G	2.99	22.99	5	JOHNNY	LOLLOBRIGIDA
5	AFRICAN EGG	G	2.99	22.99	6	BETTE	NICHOLSON
5	AFRICAN EGG	G	2.99	22.99	7	GRACE	MOSTEL
5	AFRICAN EGG	G	2.99	22.99	8	MATTHEW	JOHANSSON
5	AFRICAN EGG	G	2.99	22.99	9	JOE	SWANK
5	AFRICAN EGG	G	2.99	22.99	10	CHRISTIAN	GABLE

6	AGENT TRUMAN	PG	2.99	17.99	1	PENELOPE	GUINESS
6	AGENT TRUMAN	PG	2.99	17.99	2	NICK	WAHLBERG
6	AGENT TRUMAN	PG	2.99	17.99	3	ED	CHASE
6	AGENT TRUMAN	PG	2.99	17.99	4	JENNIFER	DAVIS
6	AGENT TRUMAN	PG	2.99	17.99	5	JOHNNY	LOLLOBRIGIDA
6	AGENT TRUMAN	PG	2.99	17.99	6	BETTE	NICHOLSON
6	AGENT TRUMAN	PG	2.99	17.99	7	GRACE	MOSTEL
6	AGENT TRUMAN	PG	2.99	17.99	8	MATTHEW	JOHANSSON
6	AGENT TRUMAN	PG	2.99	17.99	9	JOE	SWANK
6	AGENT TRUMAN	PG	2.99	17.99	10	CHRISTIAN	GABLE
7	AIRPLANE SIERRA	PG-13	4.99	28.99	1	PENELOPE	GUINESS
7	AIRPLANE SIERRA	PG-13	4.99	28.99	2	NICK	WAHLBERG
7	AIRPLANE SIERRA	PG-13	4.99	28.99	3	ED	CHASE
7	AIRPLANE SIERRA	PG-13	4.99	28.99	4	JENNIFER	DAVIS
7	AIRPLANE SIERRA	PG-13	4.99	28.99	5	JOHNNY	LOLLOBRIGIDA
7	AIRPLANE SIERRA	PG-13	4.99	28.99	6	BETTE	NICHOLSON
7	AIRPLANE SIERRA	PG-13	4.99	28.99	7	GRACE	MOSTEL
7	AIRPLANE SIERRA	PG-13	4.99	28.99	8	MATTHEW	JOHANSSON
7	AIRPLANE SIERRA	PG-13	4.99	28.99	9	JOE	SWANK
7	AIRPLANE SIERRA	PG-13	4.99	28.99	10	CHRISTIAN	GABLE
8	AIRPORT POLLOCK	R	4.99	15.99	1	PENELOPE	GUINESS
8	AIRPORT POLLOCK	R	4.99	15.99	2	NICK	WAHLBERG
8	AIRPORT POLLOCK	R	4.99	15.99	3	ED	CHASE
8	AIRPORT POLLOCK	R	4.99	15.99	4	JENNIFER	DAVIS
8	AIRPORT POLLOCK	R	4.99	15.99	5	JOHNNY	LOLLOBRIGIDA
8	AIRPORT POLLOCK	R	4.99	15.99	6	BETTE	NICHOLSON

8	AIRPORT POLLOCK	R	4.99	15.99	7	GRACE	MOSTEL
8	AIRPORT POLLOCK	R	4.99	15.99	8	MATTHEW	JOHANSSON
8	AIRPORT POLLOCK	R	4.99	15.99	9	JOE	SWANK
8	AIRPORT POLLOCK	R	4.99	15.99	10	CHRISTIAN	GABLE
9	ALABAMA DEVIL	PG-13	2.99	21.99	1	PENELOPE	GUINESS
9	ALABAMA DEVIL	PG-13	2.99	21.99	2	NICK	WAHLBERG
9	ALABAMA DEVIL	PG-13	2.99	21.99	3	ED	CHASE
9	ALABAMA DEVIL	PG-13	2.99	21.99	4	JENNIFER	DAVIS
9	ALABAMA DEVIL	PG-13	2.99	21.99	5	JOHNNY	LOLLOBRIGIDA
9	ALABAMA DEVIL	PG-13	2.99	21.99	6	BETTE	NICHOLSON
9	ALABAMA DEVIL	PG-13	2.99	21.99	7	GRACE	MOSTEL
9	ALABAMA DEVIL	PG-13	2.99	21.99	8	MATTHEW	JOHANSSON
9	ALABAMA DEVIL	PG-13	2.99	21.99	9	JOE	SWANK
9	ALABAMA DEVIL	PG-13	2.99	21.99	10	CHRISTIAN	GABLE
10	ALADDIN CALENDAR	NC-17	4.99	24.99	1	PENELOPE	GUINESS
10	ALADDIN CALENDAR	NC-17	4.99	24.99	2	NICK	WAHLBERG
10	ALADDIN CALENDAR	NC-17	4.99	24.99	3	ED	CHASE
10	ALADDIN CALENDAR	NC-17	4.99	24.99	4	JENNIFER	DAVIS
10	ALADDIN CALENDAR	NC-17	4.99	24.99	5	JOHNNY	LOLLOBRIGIDA
10	ALADDIN CALENDAR	NC-17	4.99	24.99	6	BETTE	NICHOLSON
10	ALADDIN CALENDAR	NC-17	4.99	24.99	7	GRACE	MOSTEL
10	ALADDIN CALENDAR	NC-17	4.99	24.99	8	MATTHEW	JOHANSSON
10	ALADDIN CALENDAR	NC-17	4.99	24.99	9	JOE	SWANK
10	ALADDIN CALENDAR	NC-17	4.99	24.99	10	CHRISTIAN	GABLE

그림 11. 카테시안 프로덕트 연산의 예.

3. 관계형 데이터베이스 관리 시스템의 활용

3.1 데이터베이스 설계

데이터베이스란 일반적으로 여러 사용자나 응용 프로그램에 의해 공유될 수 있도록 통합 관리되는 데이터의 집합을 의미한다.

데이터베이스를 만들기 위한 데이터 모델링 방법에는 여러 가지가 있다.

1960년대에는 데이터베이스 관리 시스템 보다는 파일 시스템을 주로 사용하였고 데이터 모델링의 개념도 매우 초보적인 수준이었다.

그 후 데이터베이스 관리시스템이 개발되면서 계층적 데이터 모델, 네트워크 모델 등이 사용되었다.

그 중에서도 특히 관계형 데이터 모델은 1970년 E.F. Codd가 제안한 모델로서 1990년대를 지나면서 객체 지향 개념이 도입되어 현재는 객체-관계형 모델로 발전하였으며 현재 가장 널리 사용되고 있다.

본 저서에서는 관계형 데이터 모델에 대하여 설명하기에 앞서, 보다 이해하기 쉽고 강한 표현력을 가지고 있는 ER 모델을 이용한 데이터 모델링 방법에 대하여 설명하기로 한다.

데이터를 모델링한다는 것은 데이터베이스화 하고자 하는 현실 세계의 여러 가지 데이터 요소들을 데이터베이스를 이용하여 빠르고 효율적으로 검색, 갱신할 수 있도록 구조화하는 작업이다.

매우 많은 종류의 데이터들을 효율적으로 검색하기 위해서는 응용 프로그램의 액세스 패턴에 따라 빠르게 데이터를 액세스 할 수 있도록

여러 가지 데이터 요소들을 서로 연관성이 큰 것들끼리 모아서 그룹화하는 것이 필요하다.

그러므로 데이터 모델링을 잘 한다는 것은 데이터의 연관성을 잘 파악하여 합리적으로 그룹화하는 것을 의미한다.

데이터 모델링을 용이하게 하기 위하여 널리 사용되는 모델링 방법이 ER(Entity Relationship) 모델이다.

ER 모델은 1976년에 P.P. Chen이 제안한 모델이며 현재는 EER(Enhanced Entity Relationship) 모델로 확장되어 사용되고 있다.

ER 모델이 널리 사용되는 이유는 다음과 같은 몇가지 장점을 가지고 있기 때문이다.

첫째, 높은 수준의 추상화를 제공한다.

둘째, 이해하기가 매우 쉽다.

셋째, 표현력이 크다.

ER 모델은 데이터를 모델링할 때에 엔티티, 애트리뷰트, 릴레이션쉽의 3가지 요소만을 사용하며, 그림을 이용하여 ER 다이어그램을 그림으로써 모델링을 수행한다.

또한, ER 모델은 몇가지 기계적인 단계를 거치면 관계형 데이터 모델로 변환할 수 있어 바로 데이터베이스 관리 시스템에서 사용할 수 있다는 장점도 있다.

이제 ER 모델을 이용한 데이터 모델링 방법에 대하여 살펴보기로 하자.

가) 엔티티

엔티티는 독립적으로 존재하면서 고유하게 식별이 가능한 객체를 의미한다.

예를 들어, 학교, 학생, 자동차 등이 엔티티가 될 수 있다.

경우에 따라서는 추상적인 개념이 엔티티가 될 수도 있다.

예를 들어, 강의 같은 개념은 형체가 있는 것은 아니지만 엔티티가 될 수 있다.

ER 다이어그램에서 엔티티는 직사각형 도형으로 표현된다.

엔티티는 그 특징에 따라 크게 강한 엔티티와 약한 엔티티로 구분할 수 있다.

강한 엔티티는 데이터베이스 내에서 독립적으로 의미를 가지는 엔티티를 의미한다.

일반적으로 대부분의 엔티티들은 강한 엔티티이며, ER 다이어그램에서는 실선 직사각형으로 표현된다.

반면에 약한 엔티티는 데이터베이스 내에서 독립적인 의미를 가지지 못하고, 다른 강한 엔티티에 의존적인 의미를 가지는 엔티티이다.

약한 엔티티는 ER 다이어그램에서 점선 직사각형으로 표현된다.

나) 속성

일반적으로 엔티티는 그 엔티티를 구성하고 있는 여러 가지 데이터로 표현될 수 있다.

예를 들어, 학생은 학번, 이름, 소속 학과, 전화 번호, 주소, 나이 등 여러 가지 데이터로 표현된다.

이렇게 엔티티를 구성하는 데이터들을 속성(attribute)이라고 부른다.

물론 엔티티를 구성하는 속성이 항상 확정적으로 결정되어 있는 것은 아니다.

실제로 엔티티를 구성하는 속성은 매우 많을 수 있지만 그 중에서 데이터베이스 설계자가 관심을 가지고 데이터베이스화 하고자 하는 데이터만을 선정하여 실제 데이터 모델에 삽입한다.

속성은 그 특징에 따라 크게 다음과 같은 6가지 속성으로 구분된다.

1. 단순 속성
2. 단일값 속성
3. 저장된 속성
4. 복합 속성
5. 다치 속성
6. 유도된 속성

이제 각각의 속성에 대하여 설명한다.

1) 단순 속성

단순 속성은 다른 속성으로 분할될 수 없는 가장 간단한 형태의 속성을 의미한다.

대부분의 속성들은 단순 속성에 해당하며 ER 다이어그램에서는 단순 속성을 실선 타원으로 표현한다.

예를 들어, 'film'이라는 엔티티에 대하여 film_id, title, description, release_year, rental_duration, rental_rate, length, replacement_cost, rating, special_features, last_update라는 속성이

있다고 하고, 모든 속성이 단순 속성이라고 하자.

film 엔티티를 ER 다이어그램으로 표현하면 그림 12와 같다.

그림에서 엔티티에 해당하는 film은 실선 직사각형으로, 나머지 단순 속성들은 실선 타원으로 표현되었으며, 속성들이 엔티티에 실선으로 연결되어 있는 것을 알 수 있다.

속성을 엔티티와 연결한 것은 해당 속성이 해당 엔티티를 구성하는 요소라는 것을 의미한다.

또한, 속성 중에서 이름에 밑줄이 있는 film_id 속성은 키 속성이라는 것을 의미한다.

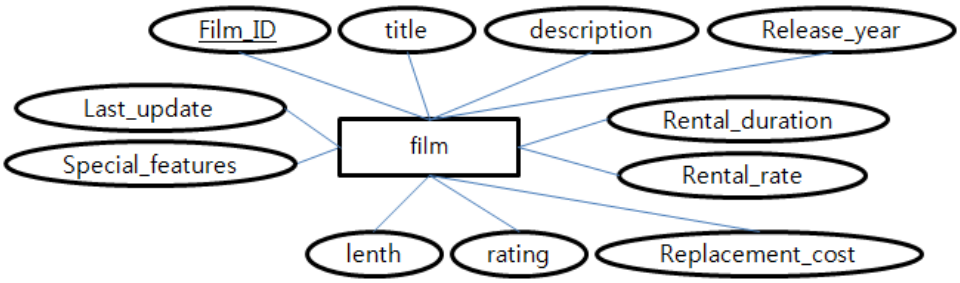

그림 12. 엔티티와 단순 속성에 대한 ER 다이어그램의 예.

2) 단일값 속성

단일값 속성은 각 엔티티마다 하나의 값만을 가지는 속성을 의미하며, 대부분의 속성들이 단일값 속성에 해당된다.

예를 들어, 위에서 살펴본 film 엔티티의 속성들은 모두 하나의 값만을 가지므로 단일값 속성에 해당된다.

그러나 어떤 속성이 단일값 속성인지 아닌지는 확정적으로 정해져 있는 것이 아니다.

위의 예에서 예를 들어 어떤 영화의 rental_rate가 경우에 따라 다르게 책정될 수 있는 경우에는 rental_rate 속성값이 여러개가 될 수 있으므로 단일값 속성이 아니게 된다.

일반적으로 사람 엔티티의 속성 중 하나가 되는 이름의 경우 이름이 두 개 이상인 사람의 정보를 표현하고자 하는 경우에는 이름 속성이 단일값 속성이 아니게 된다.

이러한 속성을 다치 속성이라고 부르며, 이에 대해서는 뒤에서 자세히 설명하기로 한다.

일반적인 ER 다이어그램에서 단일값 속성은 단순 속성과 동일하게 실선 타원으로 표현한다.

그러므로 그림 12는 단순 속성에 대한 예도 되지만, 단일값 속성에 대한 예도 된다고 할 수 있다.

3) 저장된 속성

저장된 속성은 다른 속성들과는 독립적인 의미를 가지는 속성으로서 실제로 데이터베이스에 속성의 값이 저장되는 속성이다.

일반적으로 대부분의 속성들이 저장된 속성에 해당되며, ER 다이어그램에서는 단순 속성이나 단일값 속성과 마찬가지로 실선 타원으로 표현된다.

4) 복합 속성

복합 속성은 두 개 이상의 속성이 모여서 통합적으로 대표되는 특정 의미를 가지는 속성이다.

예를 들어, film 엔티티에서 title, description, release_year의 세 가

지 속성을 묶어서 film_info라는 이름의 대표 속성을 만든다고 가정하면, film_info는 복합 속성이 된다.

복합 속성은 ER 다이어그램에서 다른 속성들과 연결된 형태로 표현된다.

그림 13은 복합 속성 film_info를 ER 다이어그램으로 표현한 것 이다.

그림에서 film_info는 복합 속성으로서 실선 타원으로 표현되어 film 엔티티에 연결되어 있으며, film_info를 구성하는 세가지 속성인 title, description, release_year는 실선 타원으로 표현되어 film_info와 연결되어 있는 것을 알 수 있다.

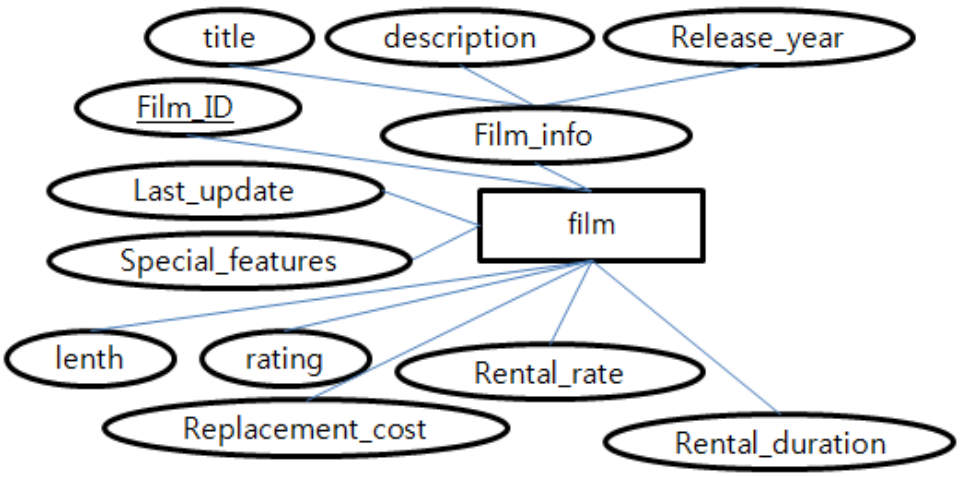

그림 13. 복합 속성에 대한 ER 다이어그램의 예.

film_info의 예에서 살펴본 바와 같이 어떤 속성이 단순 속성인지 복합 속성인지는 뚜렷하게 정해진 것이 아니다.

단순 속성과 복합 속성 중 의미가 더욱 명확하게 표현될 수 있는 방법에 따라 데이터베이스 설계자가 선택적으로 결정하는 문제라고 할 수

있다.

그러나 경우에 따라서는 복합 속성으로 모델링 하는 것과 단순 속성으로 모델링 하는 것에 단순히 의미의 명확성 뿐만 아니라 데이터베이스의 표현력 차이가 있을 수 있다.

‘주소’ 속성에 대하여 생각해 보자.

주소 데이터를 모델링 할 때에 주소 전체를 하나의 단순, 단일값 속성으로 나타낼 수도 있다.

또는 우리나라의 주소는 시/도, 구/군, 동, 번지수 등으로 이루어져 있으므로 주소 속성을 시, 구, 동, 번지수로 이루어진 복합 속성으로 모델링 할 수도 있다.

이 두 경우 실제 데이터베이스로 구성했을 때에는 데이터 검색 기능에 차이가 발생할 수 있다.

주소 전체를 하나의 속성으로 모델링 한 경우에는 주소 전체에 대한 문자열 검색만이 가능하다.

반면에 주소를 복합 속성으로 모델링한 경우에는 주소 정보가 세부 정보로 분할되어 관리되므로, 시 또는 구 정보를 기반으로 하는 검색 기능이 가능하다.

그러나 단순히 기능이 많다고 모델링이 잘 되었다고 판단할 수는 없다.

왜냐하면 응용 프로그램에서 주소 세부 정보를 활용한 검색 기능이 필요 없는 경우에는 굳이 복잡하게 모델링을 할 이유가 없기 때문이다.

이와 같이 데이터 모델링은 고정된 왕도가 있다기 보다는 응용의 데이터 액세스 형태에 알맞게 모델링 하는 것이 가장 좋은 데이터 모델링이라고 할 수 있을 것이다.

5) 다치 속성

다치 속성은 단일값 속성과는 여러 개의 값을 동시에 가질 수 있는 속성을 의미한다.

예를 들어 film 엔티티에서 rental_rate가 경우에 따라 여러 값을 가질 수 있도록 모델링을 한다면, 다치 속성이 된다.

다치 속성은 ER 다이어그램에서 두 줄 실선 타원으로 표현한다.

그림 14은 film 엔티티의 rental_rate가 다치 속성인 경우의 ER 다이어그램을 나타낸 것이다.

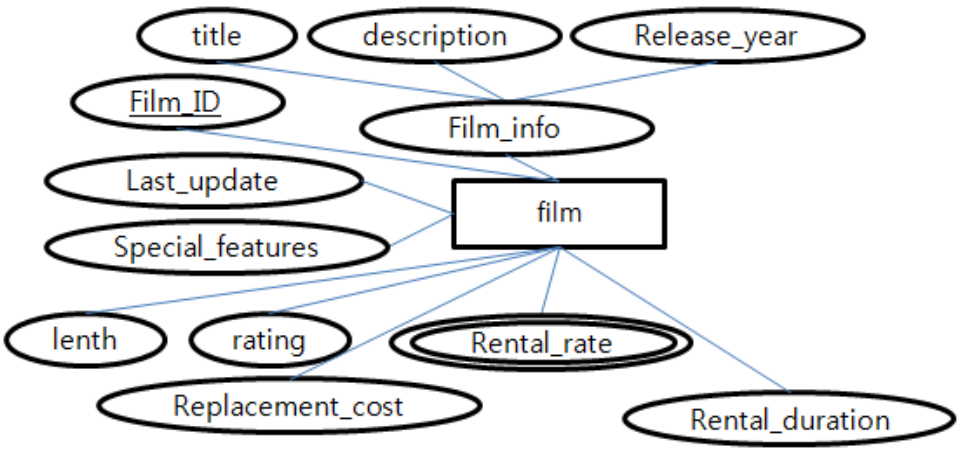

그림 14. 다치 속성에 대한 ER 다이어그램의 예.

6) 유도된 속성

유도된 속성은 저장된 속성과는 다른 속성의 값으로부터 유도될 수 있는 값을 가지는 속성이다.

일반적으로 유도된 속성은 관계형 데이터 모델에서는 잘 포함시키지 않는 경향이 있다.

왜냐하면 유도된 속성을 모델링하는 경우 데이터의 중복으로 인한 여러 가지 갱신 이상 현상이 발생할 수 있기 때문이다.

film 엔티티의 예에서 rental_duration 속성은 임의로 정해지는 것이 아니고 그 영화의 rating 속성 값에 따라서 자동으로 정해질 수 있다고 가정하면 rental_duration 속성이 유도된 속성으로 모델링 될 수 있다.

유도된 속성은 ER 다이어그램에서 점선 타원으로 표현된다.

그림 15은 유도된 속성을 ER 다이어그램으로 표현한 예를 보인 것이다.

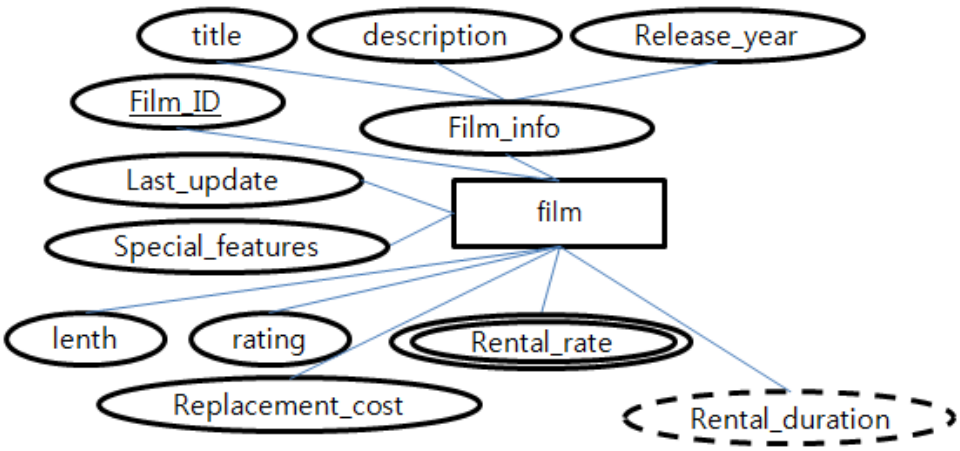

그림 15. 유도된 속성에 대한 ER 다이어그램의 예.

다) 릴레이션쉽

엔티티는 독립적으로 존재하는 객체이기는 하지만, 여러 엔티티들 사이에는 어떤 연관 관계가 있을 수 있다.

예를 들어, 영화 엔티티와 배우 엔티티 사이에는 배우가 영화에 출연

한다는 관계가 있을 것이다.

이러한 관계를 모델링하는 방법이 릴레이션쉽이다.

즉, 릴레이션쉽은 여러 엔티티들 사이의 연관 관계를 모델링한다.

ER 다이어그램에서 릴레이션쉽은 실선 마름모로 표현한다.

그림 16은 영화 엔티티와 배우 엔티티, 그리고 두 엔티티 사이의 출연 릴레이션쉽을 도식화하여 나타낸 것이다.

배우 엔티티는 actor_id 속성을 키로 하고, 성 및 이름 속성과 last_update 속성으로 구성되어 있다.

릴레이션쉽을 표현할 때 중요하게 고려해야 하는 것 중 해당 릴레이션쉽에 참여하는 엔티티들의 가능한 조합을 제한하는 제약조건이다.

이 제약 조건을 관계 제약조건이라고 한다.

대표적인 관계 제약조건으로는 카디날리티 제약조건과 참여 제약조건이 있다.

카디날리티 제약조건은 해당 릴레이션쉽에 참여하는 엔티티들이 어떤 대응 관계를 가지고 있는지를 나타낸 것이다.

카디날리티 제약조건의 종류에는 1:1 관계, 1:N 관계, M:N 관계가 있다.

1:1 관계는 릴레이션쉽에 참여하는 두 엔티티의 대응관계가 1:1 관계라는 것을 의미한다.

예를 들어, 남성 엔티티와 여성 엔티티 사이에 '부부'라는 릴레이션쉽을 생각할 수 있을 것이다.

우리나라는 일부일처제를 따르므로, 부부 릴레이션쉽의 카디날리티 제약조건은 1:1 관계가 될 것이다.

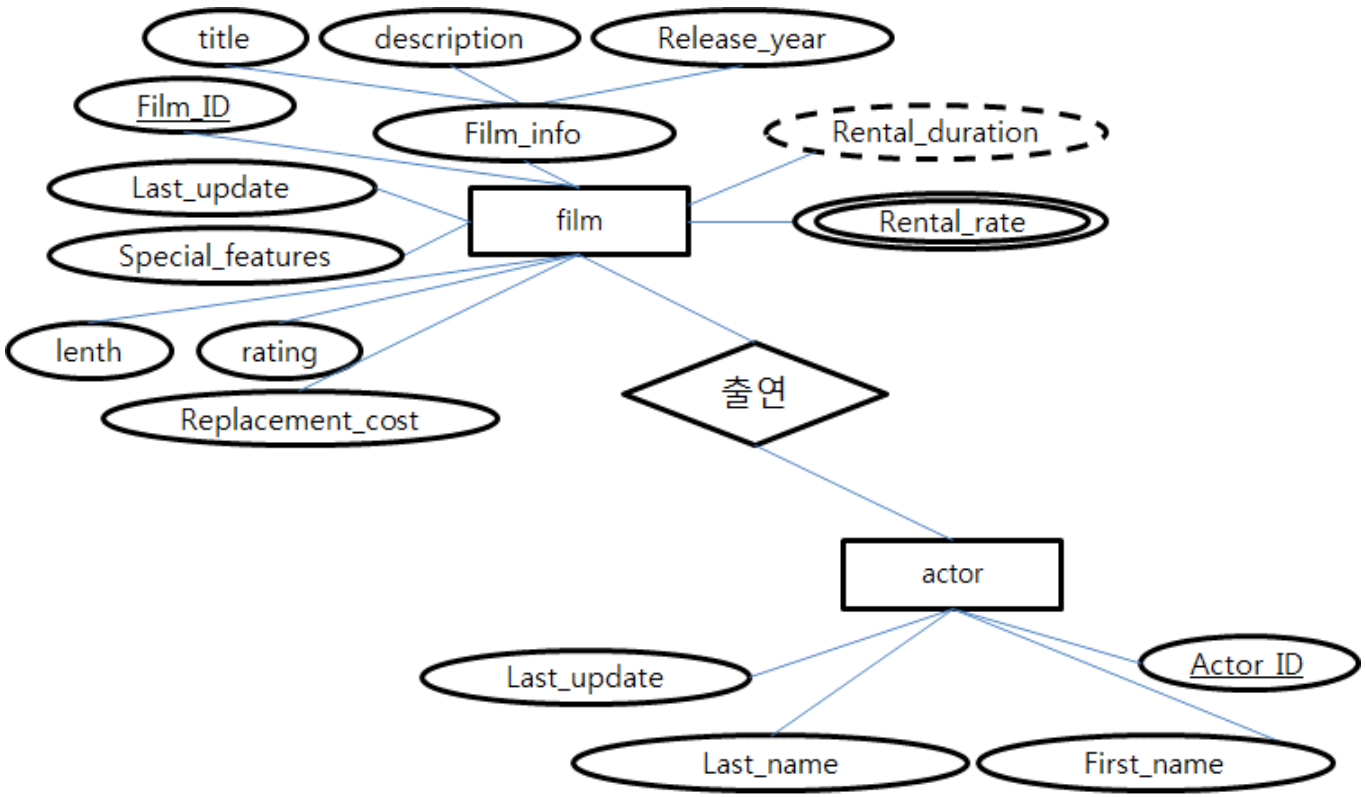

그림 16. 릴레이션쉽에 대한 ER 다이어그램의 예.

학생 엔티티와 학과 엔티티 사이에는 '소속'이라는 릴레이션쉽을 생각할 수 있다.

일반적으로 학생들은 하나의 학과에 소속되지만, 하나의 학과에는 여러명의 학생이 속해있으므로 소속 릴레이션쉽의 카디날리티 제약조건은 1:N 관계가 된다.

이제 위에서 예를 들었던 film 엔티티와 actor 엔티티 사이의 출연 릴레이션쉽을 생각해보자.

하나의 영화에는 여러 배우가 출연하고, 한 배우는 여러 영화에 출연할 수 있으므로 출연 릴레이션쉽의 카디날리티 제약조건은 N:M 관계가 된다.

이렇게 릴레이션쉽의 카디날리티 제약조건은 실제 상황에 맞게 설정된다.

카디날리티 제약조건이 결정되면 ER 다이어그램의 릴레이션쉽과 엔티티를 연결한 실선 위에 (최소값, 최대값)의 형태로 나타낼 수 있다.

최소값은 릴레이션쉽에 참여하는 엔티티의 최소 참여 개수를 의미하고, 최대값은 최대 참여 개수를 의미한다.

이 때, 최소 참여 개수가 0인 경우를 생각할 수 있다.

최소 참여 개수가 0이라는 의미는 해당 엔티티가 릴레이션쉽에 참여하지 않을 수도 있다는 것을 의미하고, 1 이상이면 반드시 참여해야 한다는 것을 의미한다.

이렇게, 엔티티가 릴레이션쉽에 반드시 참여해야만 하는지, 또는 선택적으로 참여하지 않을 수도 있는지에 대한 제한 조건을 참여 제약조건이라고 한다.

그림 17은 film 엔티티와 actor 엔티티, 그리고 그 사이의 출연 릴레이션쉽에 대한 카디날리티 및 참여 제약조건을 표현한 것이다.

그림에서 영화에 출연 배우가 한명도 없을 수는 없고, 1인 출연 영화는 가능하므로 film 엔티티의 출연 릴레이션쉽에 대한 최소 참여 개수는 1이라고 할 수 있다.

반면에, 배우는 아직 등록만 되어 있고 출연한 영화가 없을 수도 있으므로, actor 엔티티의 출연 릴레이션쉽에 대한 최소 참여 개수는 0이 될 것이다.

3.2 MySQL 설치

본 절에서는 MySQL을 윈도우즈 7 운영체제를 사용하는 개인용 컴퓨터에 설치하는 방법에 대하여 설명한다.

MySQL은 세계적으로 가장 널리 사용되는 데이터베이스 관리 시스템 회사인 오라클이 보유한 개방 소스 데이터베이스 관리 시스템이다.

MySQL은 관계형 데이터 모델을 지원하고 표준 SQL을 사용하며, 유닉스, 리눅스, 윈도우즈 등 다양한 운영체제하에서 동작할 수 있고, 동시성 제어, 파손 회복 등 기본적인 데이터베이스 관리 프로그램의 기능을 제공할 뿐만 아니라 다중 쓰레드를 지원하고 C, C++, JAVA, PHP, Python과 같은 다양한 언어를 위한 인터페이스를 제공하고 있다.

특히, 개방 소스 정책을 표방하고 있는 무료 프로그램이므로 개인 사용자나 소규모 사업자들에게 널리 사용되고 있다.

MySQL은 비상업적인 용도로 사용하는 경우 인터넷을 통하여 무료로 다운로드 받을 수 있다.

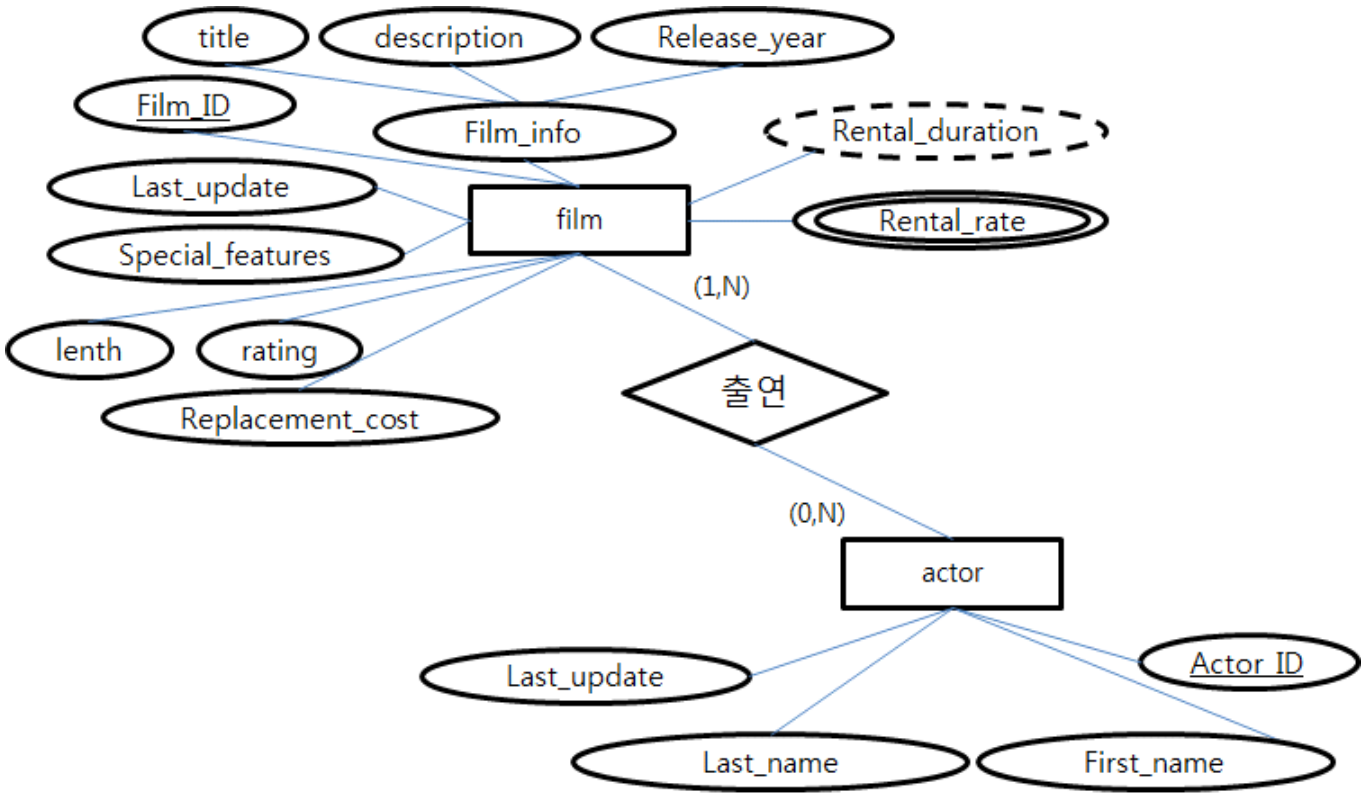

그림 17. 카디날리티 및 참여 제약조건을 표현한 ER 다이어그램의 예.

가) 무료 다운로드 받기

먼저 웹 브라우저의 주소창에 “http://dev.mysql.com/downloads/”를 입력하면 그림 18과 같은 웹 페이지를 만나게 된다.

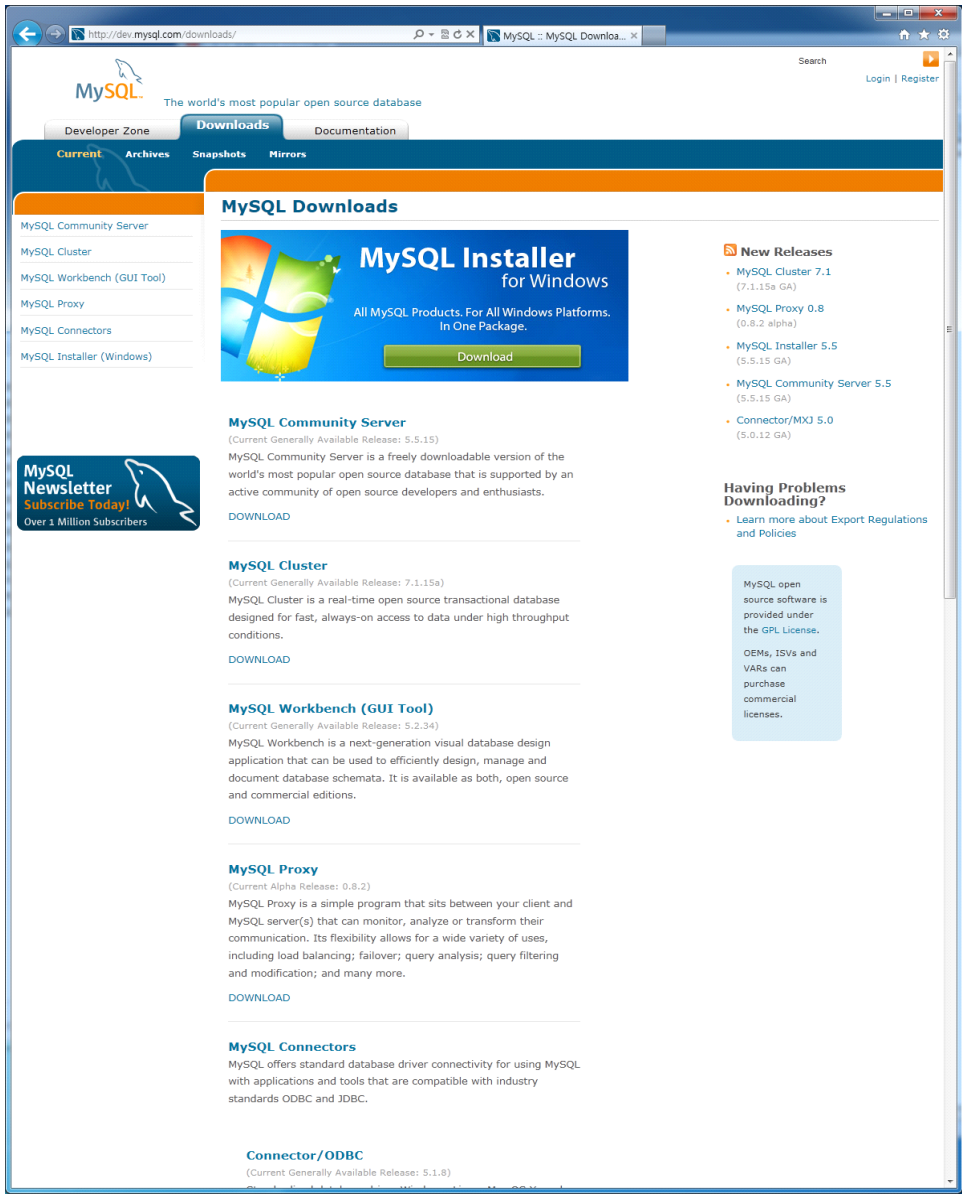

그림 18. MySQL 다운로드를 위한 초기 웹 페이지.

그림의 웹 페이지는 윈도우즈 운영체제를 위한 MySQL을 다운로드 받는 페이지이므로 윈도우즈가 아닌 다른 운영체제를 사용하는 경우에는 별도의 사이트에 접속해야만 한다.

이제 화면 상단 중앙의 초록색 "Download" 버튼을 클릭하면, 그림 19와 같은 웹 페이지가 나타나게 된다.

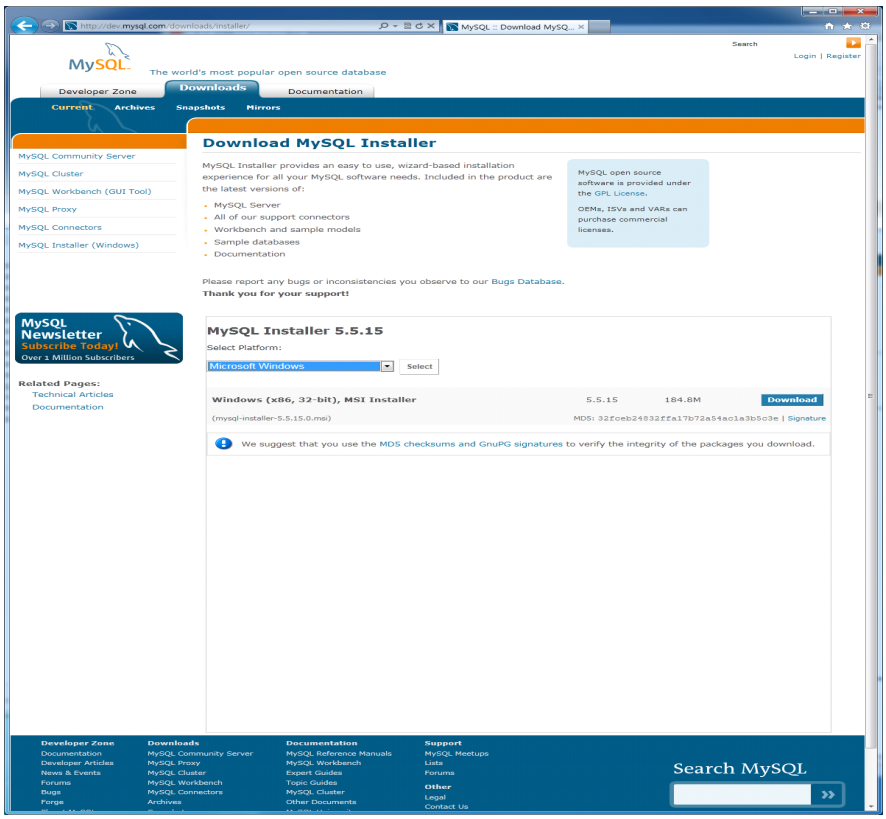

그림 19. MySQL Platform 선택 웹 페이지.

이 웹 페이지에서 platform을 window로 선택한 후 “Download" 버튼을 클릭하면 그림 20의 MySQL 계정 생성 웹 페이지가 나타나게 된다.

MySQL은 웹 계정을 생성하지 않고도 무료로 다운로드 받을 수 있으

므로 반드시 필요한 경우가 아니면 계정 입력 박스의 하단부에 있는 “No thanks, just take me to the downloads!" 메시지를 클릭하여 다음 단계로 진행할 수 있다.

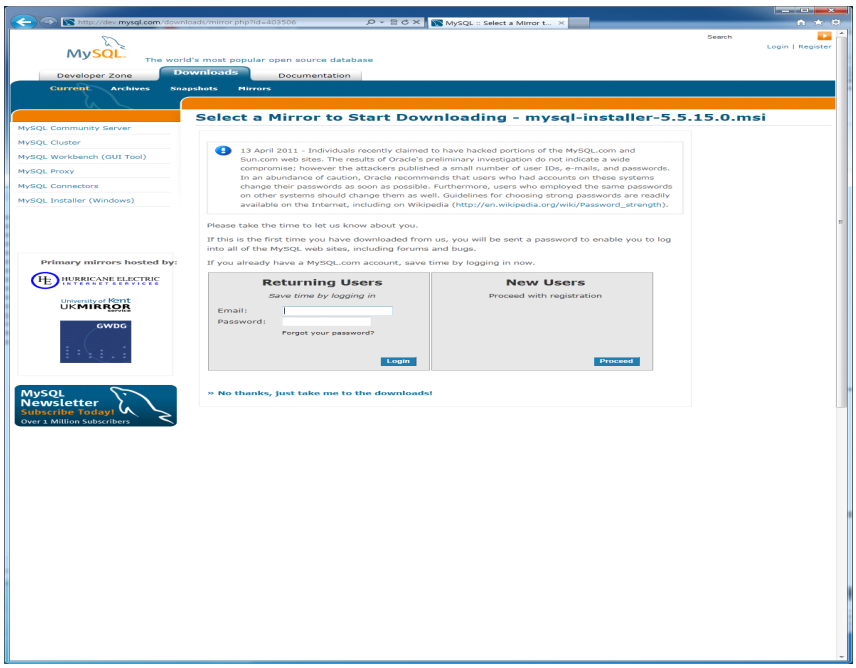

그림 20. MySQL 계정 생성 웹 페이지.

다음 단계에서 MySQL을 다운로드 받을 서버를 선택하여 mysql-installer-5.5.17.0.msi 파일을 다운로드 받는다.

이 파일은 MySQL을 시스템에 인스톨해주는 작업을 수행하는 파일로서 약 214MB의 크기이다.

다운로드가 끝나면 이 파일을 수행시켜 MySQL을 본격적으로 설치할 수 있다.

나) MySQL 설치하기

그림 21는 MySQL 설치의 첫 화면이다. 본 화면에서 첫 번째 항목인

Install MySQL Products를 선택하여 설치를 시작하도록 하자.

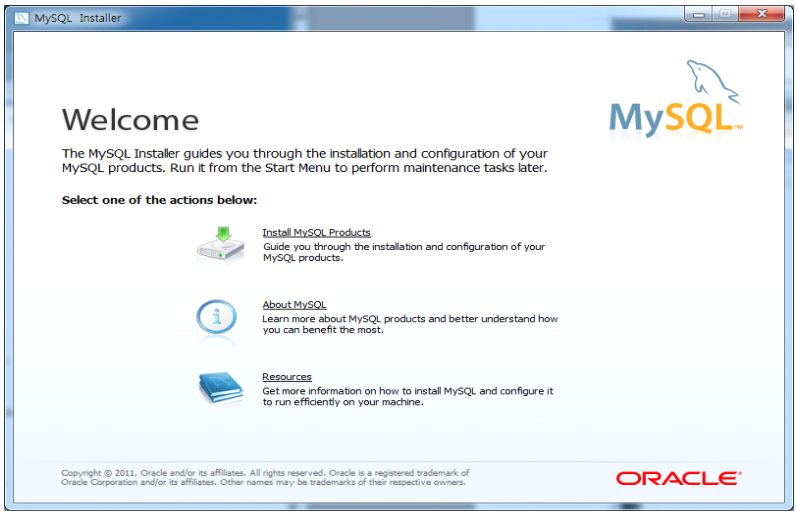

그림 21. MySQL 설치 첫 화면.

그림 22는 MySQL 설치 및 사용을 위한 여러 가지 사용자 라이센스 조건을 설명하고 있는 화면이다.

이 조건에 동의한다는 체크박스를 선택하고 Next 단추를 클릭한다.

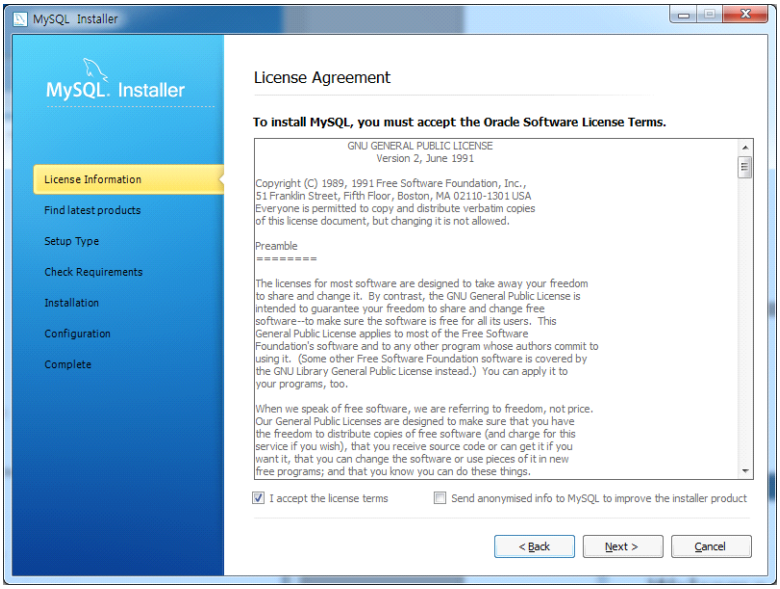

그림 22. 사용자 라이센스에 동의 화면.

그림 23은 설치하려는 MySQL이 최신 버전인지 인터넷을 통하여 확인하고, 최신 버전이 아닌 경우에는 패치를 수행하여 최신 버전을 설치하도록 하는 과정이다.

이 과정이 반드시 필요하지 않는 경우에는 수행하지 않을 수도 있으나 특별한 사유가 없다면 수행하는 것이 좋을 것이다.

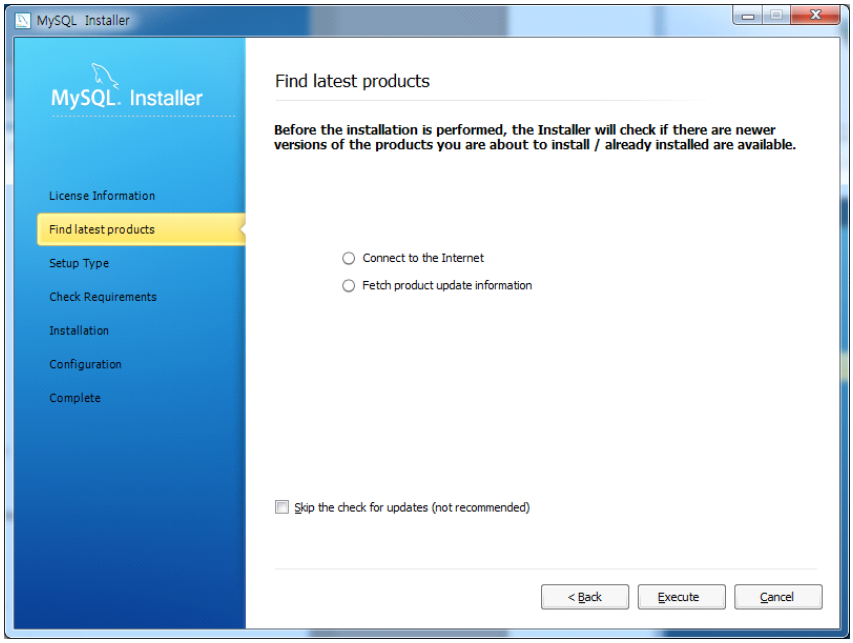

그림 23. 최신 버전 확인 및 패치 화면.

다음 단계는 Setup Type을 설정하는 과정으로서 그림 24에 나타나 있다.

사용자는 필요에 따라 서버만 설치하거나 클라이언트만 설치할 수도 있으며, 또한 사용자의 PC에 MySQL을 설치할 위치를 지정하는 것도 가능하다.

우리는 MySQL을 전체 다 설치할 수 있도록 Full을 선택하여 설치하기로 한다.

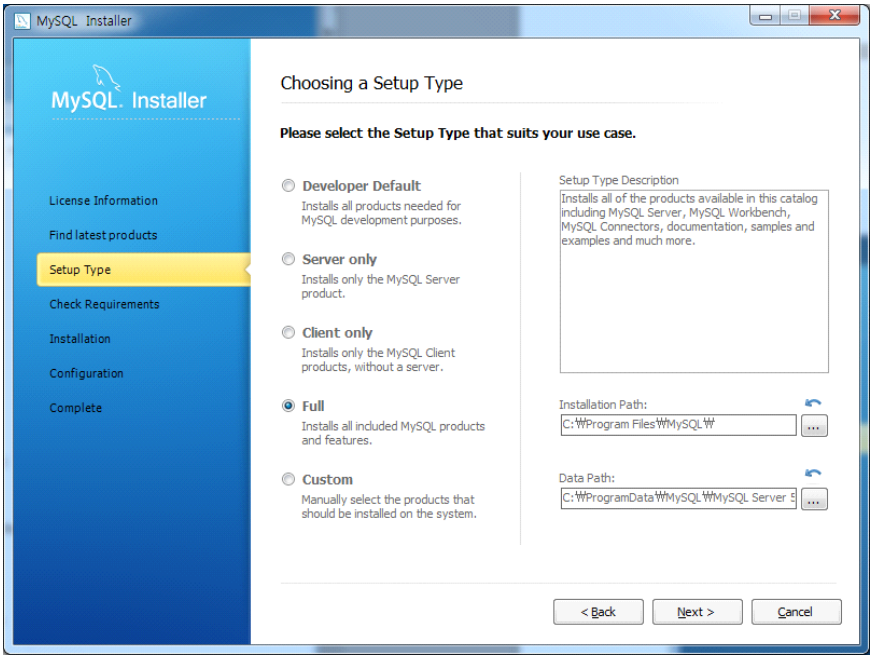

그림 24. Setup Type 설정.

다음 단계는 MySQL을 수행하기 위하여 PC에 설치되어 있어야만 하는 구성 요소를 확인하고 설치하는 단계로서 그림 25에 나타나 있다.

그림에서 보는 바와 같이 MySQL을 수행하기 위해서는 Microsoft .Net Framework4 Client Profile MySQL Workbench CE 5.2.35와 Microsoft Visual C++ 2010 32-bit runtime MySQL Workbench CE 5.2.35가 필요하다.

만일 사용자의 PC에 위의 모듈이 설치되어 있지 않은 경우에는 해당 모듈의 설치 과정이 진행된다.

이러한 모듈들은 모두 무료로 설치할 수 있다.

다음 단계는 드디어 그림 26에서 보는 바와 같이 MySQL을 설치하는 단계이다. 본 단계에서는 사용자가 선택한 설치 조건에 따라 MySQL의 여러 가지 구성 요소들이 설치된다.

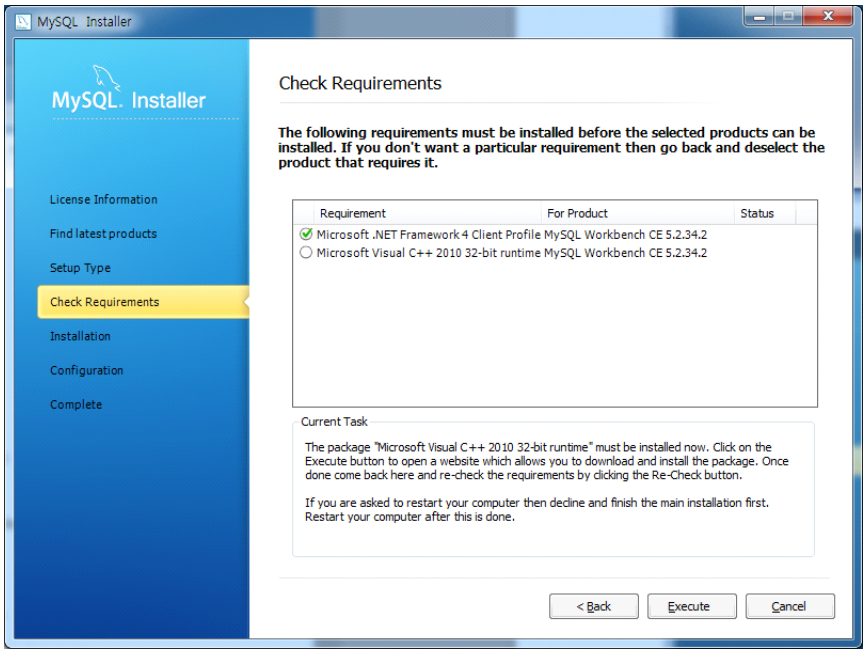

그림 25. 필수 설치 요구사항 확인.

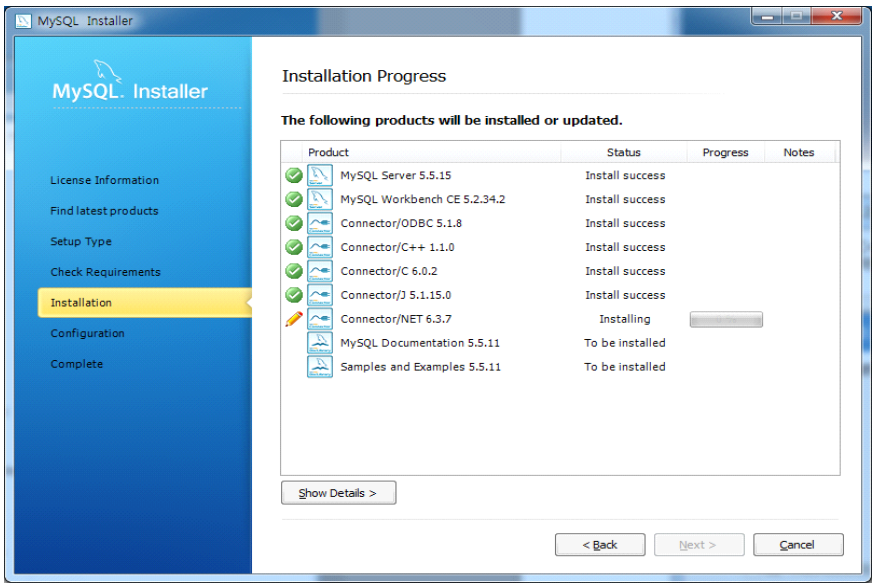

그림 26. MySQL 설치 단계.

MySQL의 설치 단계가 종료하면 각종 시스템 설정을 하는 과정이 수행된다.

그림 27, 28은 MySQL Server 5.5.15에 대한 설정 과정이다.

본 과정에서는 MySQL을 사용하는 목적에 따라 Developer Machine, Server Machine, Dedicated Machine 중 하나를 선택할 수 있다.

본 과정에서는 Developer Machine을 선택하기로 한다.

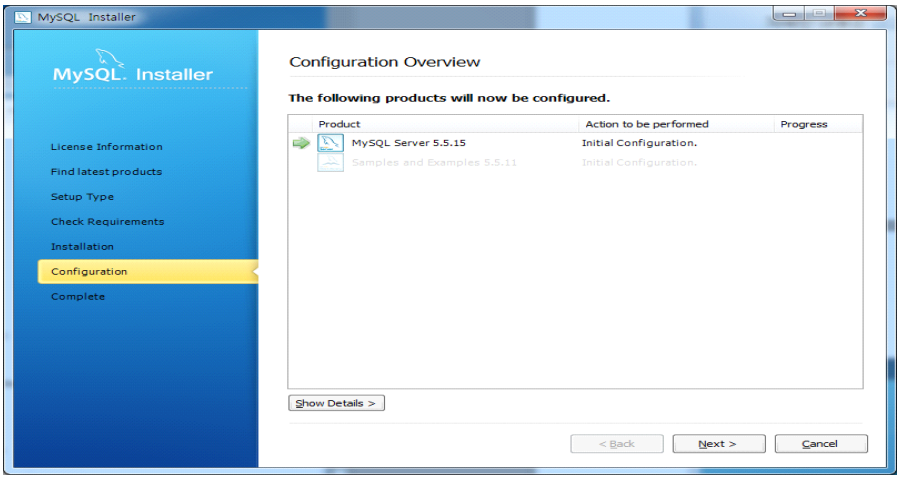

그림 27. MySQL 서버 설정 단계 I.

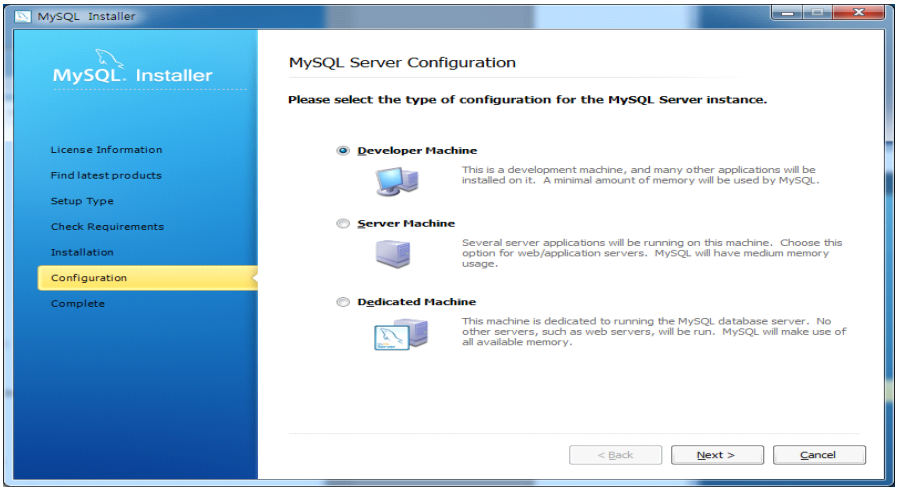

그림 28. MySQL 서버 설정 단계 II.

서버에 대한 설정 단계가 끝나면 그림 29와 같이 MySQL 설치 완료 화면이 나타난다.

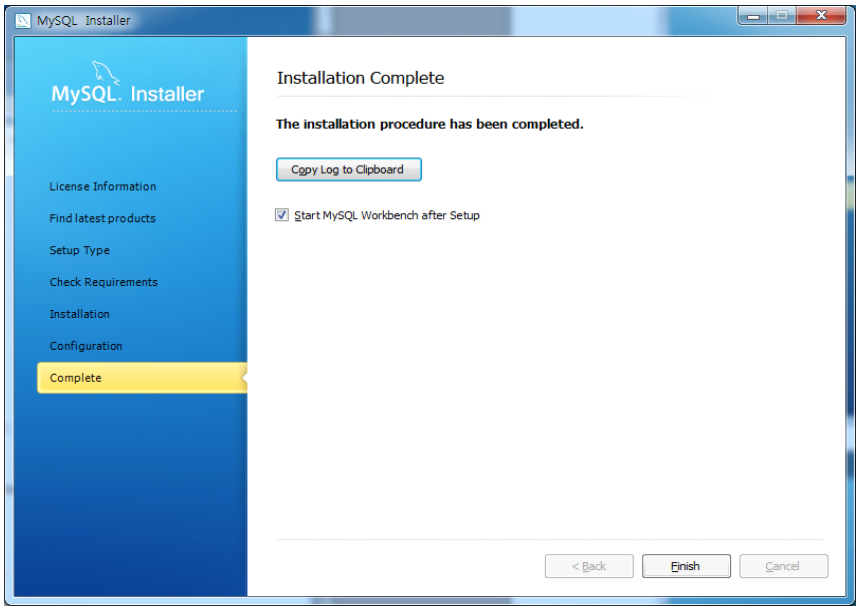

그림 29. MySQL 설치 완료 화면.

이제 MySQL 설치가 완료되었으므로 MySQL을 이용하여 데이터베이스를 구축하고 각종 질의어를 사용하여 데이터를 액세스해 보도록 하자.

3.3 릴레이션 생성

MySQL에서는 기본적으로 sakila 데이터베이스를 샘플로 제공하고 있다.

본 저서에서는 sakila 데이터베이스를 이용하여 여러 가지 질의어를

설명하기로 한다.

sakila 데이터베이스는 기본적으로 제공되는 데이터베이스이므로 별도로 릴레이션을 생성하고 데이터를 삽입할 필요가 없지만, 설명을 위해서 릴레이션을 생성하는 것부터 기술하기로 한다.

sakila 데이터베이스는 영화 DVD 대여점의 영업 활동을 대상으로 데이터베이스를 구축한 것으로서 다음과 같은 총 15개의 릴레이션으로 구성된다.

- actor(actor_id, first_name, last_name, last_update)

  : 영화 배우에 대한 정보가 기록된 릴레이션으로 200개의 투플이 입력되어 있다.

  다음은 actor 릴레이션 상태의 일부를 보인 것이다.

표. actor 릴레이션의 상태.

actor_id	first_name	last_name	last_update
1	PENELOPE	GUINESS	2006-02-15 4:34
2	NICK	WAHLBERG	2006-02-15 4:34
3	ED	CHASE	2006-02-15 4:34
4	JENNIFER	DAVIS	2006-02-15 4:34
5	JOHNNY	LOLLOBRIGIDA	2006-02-15 4:34
6	BETTE	NICHOLSON	2006-02-15 4:34
7	GRACE	MOSTEL	2006-02-15 4:34
8	MATTHEW	JOHANSSON	2006-02-15 4:34
9	JOE	SWANK	2006-02-15 4:34
10	CHRISTIAN	GABLE	2006-02-15 4:34
11	ZERO	CAGE	2006-02-15 4:34

12	KARL	BERRY	2006-02-15 4:34
13	UMA	WOOD	2006-02-15 4:34
14	VIVIEN	BERGEN	2006-02-15 4:34
15	CUBA	OLIVIER	2006-02-15 4:34
16	FRED	COSTNER	2006-02-15 4:34
17	HELEN	VOIGHT	2006-02-15 4:34
18	DAN	TORN	2006-02-15 4:34
19	BOB	FAWCETT	2006-02-15 4:34
…	…	…	…

- category(category_id, name, last_update)

  : 영화 장르에 대한 정보가 기록된 릴레이션

표. category 릴레이션.

category_id	name	last_update
1	Action	2006-02-15 4:46
2	Animation	2006-02-15 4:46
3	Children	2006-02-15 4:46
4	Classics	2006-02-15 4:46
5	Comedy	2006-02-15 4:46
6	Documentary	2006-02-15 4:46
7	Drama	2006-02-15 4:46
8	Family	2006-02-15 4:46

9	Foreign	2006-02-15 4:46
10	Games	2006-02-15 4:46
11	Horror	2006-02-15 4:46
12	Music	2006-02-15 4:46
13	New	2006-02-15 4:46
14	Sci-Fi	2006-02-15 4:46
15	Sports	2006-02-15 4:46
16	Travel	2006-02-15 4:46

- address(address_id, address, address2, district, city_id, postal_code, phone, last_update)

  : 주소에 대한 정보가 기록된 릴레이션

표. address 릴레이션의 상태.

address_id	address	address2	district	city_id	postal_code	phone	last_update
1	47 MySakila Drive	NULL	Alberta	300			2006-02-15 4:45
2	28 MySQL Boulevard	NULL	QLD	576			2006-02-15 4:45
3	23 Workhaven Lane	NULL	Alberta	300		14033335568	2006-02-15 4:45
4	1411 Lillydale Drive	NULL	QLD	576		6172235589	2006-02-15 4:45
5	1913 Hanoi Way		Nagasaki	463	35200	28303384290	2006-02-15 4:45
6	1121 Loja Avenue		California	449	17886	838635286649	2006-02-15 4:45
7	692 Joliet Street		Attika	38	83579	448477190408	2006-02-15 4:45
8	1566 Inegl Manor		Mandalay	349	53561	705814003527	2006-02-15 4:45
9	53 Idfu Parkway		Nantou	361	42399	10655648674	2006-02-15 4:45

10	1795 Santiago de Compostela Way		Texas	295	18743	860452626434	2006-02-15 4:45
11	900 Santiago de Compostela Parkway		Central Serbia	280	93896	716571220373	2006-02-15 4:45
12	478 Joliet Way		Hamilton	200	77948	657282285970	2006-02-15 4:45
13	613 Korolev Drive		Masqat	329	45844	380657522649	2006-02-15 4:45
14	1531 Sal Drive		Esfahan	162	53628	648856936185	2006-02-15 4:45
15	1542 Tarlac Parkway		Kanagawa	440	1027	635297277345	2006-02-15 4:45
16	808 Bhopal Manor		Haryana	582	10672	465887807014	2006-02-15 4:45
17	270 Amroha Parkway		Osmaniye	384	29610	695479687538	2006-02-15 4:45
18	770 Bydgoszcz Avenue		California	120	16266	517338314235	2006-02-15 4:45
19	419 Iligan Lane		Madhya Pradesh	76	72878	990911107354	2006-02-15 4:45
...	...	...	...	...	...	...	...

- city(city_id, city, country_id, last_update)

  : 도시에 대한 정보가 기록된 릴레이션

표. city 릴레이션의 상태.

city_id	city	country_id	last_update
1	A Corua (La Corua)	87	2006-02-15 4:45
2	Abha	82	2006-02-15 4:45
3	Abu Dhabi	101	2006-02-15 4:45
4	Acua	60	2006-02-15 4:45
5	Adana	97	2006-02-15 4:45
6	Addis Abeba	31	2006-02-15 4:45
7	Aden	107	2006-02-15 4:45
8	Adoni	44	2006-02-15 4:45
9	Ahmadnagar	44	2006-02-15 4:45
10	Akishima	50	2006-02-15 4:45
11	Akron	103	2006-02-15 4:45
12	al-Ayn	101	2006-02-15 4:45
13	al-Hawiya	82	2006-02-15 4:45
14	al-Manama	11	2006-02-15 4:45
15	al-Qadarif	89	2006-02-15 4:45
16	al-Qatif	82	2006-02-15 4:45
17	Alessandria	49	2006-02-15 4:45
18	Allappuzha (Alleppey)	44	2006-02-15 4:45
19	Allende	60	2006-02-15 4:45
...	...	...	...

- country(country_id, country, last_update)

  : 국가에 대한 정보가 기록된 릴레이션

표. country 릴레이션의 상태.

country_id	country	last_update
1	Afghanistan	2006-02-15 4:44
2	Algeria	2006-02-15 4:44
3	American Samoa	2006-02-15 4:44
4	Angola	2006-02-15 4:44
5	Anguilla	2006-02-15 4:44
6	Argentina	2006-02-15 4:44
7	Armenia	2006-02-15 4:44
8	Australia	2006-02-15 4:44
9	Austria	2006-02-15 4:44
10	Azerbaijan	2006-02-15 4:44
11	Bahrain	2006-02-15 4:44
12	Bangladesh	2006-02-15 4:44
13	Belarus	2006-02-15 4:44
14	Bolivia	2006-02-15 4:44
15	Brazil	2006-02-15 4:44
16	Brunei	2006-02-15 4:44
17	Bulgaria	2006-02-15 4:44
...	...	...

- film_actor(actor_id, film_id, last_update)

  : 영화 출연 배우에 대한 정보가 기록된 릴레이션

표. film_actor 릴레이션의 상태.

actor_id	film_id	last_update
1	1	2006-02-15 5:05
1	23	2006-02-15 5:05
1	25	2006-02-15 5:05
1	106	2006-02-15 5:05
1	140	2006-02-15 5:05
1	166	2006-02-15 5:05
1	277	2006-02-15 5:05
1	361	2006-02-15 5:05
1	438	2006-02-15 5:05
1	499	2006-02-15 5:05
1	506	2006-02-15 5:05
1	509	2006-02-15 5:05
1	605	2006-02-15 5:05
1	635	2006-02-15 5:05
1	749	2006-02-15 5:05
1	832	2006-02-15 5:05
1	939	2006-02-15 5:05
1	970	2006-02-15 5:05
1	980	2006-02-15 5:05
2	3	2006-02-15 5:05
...	...	...

- customer(customer_id, store_id, first_name, last_name, email, address_id, active, create_date, last_update)

  : 고객에 대한 정보가 기록된 릴레이션

표. customer 릴레이션의 상태.

customer_id	store_id	first_name	last_name	email	address_id	active	create_date	last_update
1	1	MARY	SMITH	MARY.SMITH@sakilacustomer.org	5	1	2006-02-14 22:04	2006-02-15 4:57
2	1	PATRICIA	JOHNSON	PATRICIA.JOHNSON@sakilacustomer.org	6	1	2006-02-14 22:04	2006-02-15 4:57
3	1	LINDA	WILLIAMS	LINDA.WILLIAMS@sakilacustomer.org	7	1	2006-02-14 22:04	2006-02-15 4:57
4	2	BARBARA	JONES	BARBARA.JONES@sakilacustomer.org	8	1	2006-02-14 22:04	2006-02-15 4:57
5	1	ELIZABETH	BROWN	ELIZABETH.BROWN@sakilacustomer.org	9	1	2006-02-14 22:04	2006-02-15 4:57
6	2	JENNIFER	DAVIS	JENNIFER.DAVIS@sakilacustomer.org	10	1	2006-02-14 22:04	2006-02-15 4:57
7	1	MARIA	MILLER	MARIA.MILLER@sakilacustomer.org	11	1	2006-02-14 22:04	2006-02-15 4:57
8	2	SUSAN	WILSON	SUSAN.WILSON@sakilacustomer.org	12	1	2006-02-14 22:04	2006-02-15 4:57
9	2	MARGARET	MOORE	MARGARET.MOORE@sakilacustomer.org	13	1	2006-02-14 22:04	2006-02-15 4:57

10	1	DOROTHY	TAYLOR	DOROTHY.TAYLOR@sakilacustomer.org	14	1	2006-02-14 22:04	2006-02-15 4:57
11	2	LISA	ANDERSON	LISA.ANDERSON@sakilacustomer.org	15	1	2006-02-14 22:04	2006-02-15 4:57
12	1	NANCY	THOMAS	NANCY.THOMAS@sakilacustomer.org	16	1	2006-02-14 22:04	2006-02-15 4:57
13	2	KAREN	JACKSON	KAREN.JACKSON@sakilacustomer.org	17	1	2006-02-14 22:04	2006-02-15 4:57
14	2	BETTY	WHITE	BETTY.WHITE@sakilacustomer.org	18	1	2006-02-14 22:04	2006-02-15 4:57
15	1	HELEN	HARRIS	HELEN.HARRIS@sakilacustomer.org	19	1	2006-02-14 22:04	2006-02-15 4:57
16	2	SANDRA	MARTIN	SANDRA.MARTIN@sakilacustomer.org	20	0	2006-02-14 22:04	2006-02-15 4:57
17	1	DONNA	THOMPSON	DONNA.THOMPSON@sakilacustomer.org	21	1	2006-02-14 22:04	2006-02-15 4:57
18	2	CAROL	GARCIA	CAROL.GARCIA@sakilacustomer.org	22	1	2006-02-14 22:04	2006-02-15 4:57
...	...	...	...	...	...	...	...	...

- film_category(film_id, category_id, last_update)

  : 각 영화의 장르에 대한 정보가 기록된 릴레이션

표. film_category 릴레이션의 상태.

film_id	category_id	last_update
1	6	2006-02-15 5:07
2	11	2006-02-15 5:07
3	6	2006-02-15 5:07
4	11	2006-02-15 5:07
5	8	2006-02-15 5:07
6	9	2006-02-15 5:07
7	5	2006-02-15 5:07
8	11	2006-02-15 5:07
9	11	2006-02-15 5:07
10	15	2006-02-15 5:07
11	9	2006-02-15 5:07
12	12	2006-02-15 5:07
13	11	2006-02-15 5:07
14	4	2006-02-15 5:07
15	9	2006-02-15 5:07
16	9	2006-02-15 5:07
17	12	2006-02-15 5:07
...	...	...

- film(film_id, title, description, release_year, language_id, original_language, rental_duration, rental_rate, length, replacement_cost, rating, special_features, last_update)

  : 영화에 대한 정보가 기록된 릴레이션

`표. film 릴레이션의 상태.

film_id	title	description	release_year	language_id	original_language_id	rental_duration	rental_rate	length	replacement_cost	rating	special_features	last_update
1	ACADEMY DINOSAUR	A Epic Drama of a Feminist And a Mad Scientist who must Battle a Teacher in The Canadian Rockies	2006	1	NULL	6	0.99	86	20.99	PG	Deleted Scenes,Behind the Scenes	2006-02-15 5:03
2	ACE GOLDFINGER	A Astounding Epistle of a Database Administrator And a Explorer who must Find a Car in Ancient China	2006	1	NULL	3	4.99	48	12.99	G	Trailers,Deleted Scenes	2006-02-15 5:03
3	ADAPTATION HOLES	A Astounding Reflection of a Lumberjack And a Car who must Sink a Lumberjack in A Baloon Factory	2006	1	NULL	7	2.99	50	18.99	NC-17	Trailers,Deleted Scenes	2006-02-15 5:03
4	AFFAIR PREJUDICE	A Fanciful Documentary of a Frisbee And a Lumberjack who must Chase a Monkey in A Shark Tank	2006	1	NULL	5	2.99	117	26.99	G	Commentaries, Behind the Scenes	2006-02-15 5:03
5	AFRICAN EGG	A Fast-Paced Documentary of a Pastry Chef And a Dentist who must Pursue a Forensic Psychologist in The Gulf of Mexico	2006	1	NULL	6	2.99	130	22.99	G	Deleted Scenes	2006-02-15 5:03
6	AGENT TRUMAN	A Intrepid Panorama of a Robot And a Boy who must Escape a Sumo Wrestler in Ancient China	2006	1	NULL	3	2.99	169	17.99	PG	Deleted Scenes	2006-02-15 5:03

7	AIRPLANE SIERRA	A Touching Saga of a Hunter And a Butler who must Discover a Butler in A Jet Boat	2006	1	NULL	6	4.99	62	28.99	PG-13	Trailers,Deleted Scenes	2006-02-15 5:03
8	AIRPORT POLLOCK	A Epic Tale of a Moose And a Girl who must Confront a Monkey in Ancient India	2006	1	NULL	6	4.99	54	15.99	R	Trailers	2006-02-15 5:03
9	ALABAMA DEVIL	A Thoughtful Panorama of a Database Administrator And a Mad Scientist who must Outgun a Mad Scientist in A Jet Boat	2006	1	NULL	3	2.99	114	21.99	PG-13	Trailers,Deleted Scenes	2006-02-15 5:03
10	ALADDIN CALENDAR	A Action-Packed Tale of a Man And a Lumberjack who must Reach a Feminist in Ancient China	2006	1	NULL	6	4.99	63	24.99	NC-17	Trailers,Deleted Scenes	2006-02-15 5:03
11	ALAMO VIDEOTAPE	A Boring Epistle of a Butler And a Cat who must Fight a Pastry Chef in A MySQL Convention	2006	1	NULL	6	0.99	126	16.99	G	Commentaries, Behind the Scenes	2006-02-15 5:03
12	ALASKA PHANTOM	A Fanciful Saga of a Hunter And a Pastry Chef who must Vanquish a Boy in Australia	2006	1	NULL	6	0.99	136	22.99	PG	Commentaries, Deleted Scenes	2006-02-15 5:03
13	ALI FOREVER	A Action-Packed Drama of a Dentist And a Crocodile who must Battle a Feminist in The Canadian Rockies	2006	1	NULL	4	4.99	150	21.99	PG	Deleted Scenes,Behind the Scenes	2006-02-15 5:03
...	...	...	...	...	...	...	...	...	...	...	...	...

- inventory(inventory_id, film_id, store_id, last_update)

  : 대여점에 구비된 영화에 대한 정보가 기록된 릴레이션

표. inventory 릴레이션의 상태.

inventory_id	film_id	store_id	last_update
1	1	1	2006-02-15 5:09
2	1	1	2006-02-15 5:09
3	1	1	2006-02-15 5:09
4	1	1	2006-02-15 5:09
5	1	2	2006-02-15 5:09
6	1	2	2006-02-15 5:09
7	1	2	2006-02-15 5:09
8	1	2	2006-02-15 5:09
9	2	2	2006-02-15 5:09
10	2	2	2006-02-15 5:09
11	2	2	2006-02-15 5:09
12	3	2	2006-02-15 5:09
13	3	2	2006-02-15 5:09
14	3	2	2006-02-15 5:09
15	3	2	2006-02-15 5:09
16	4	1	2006-02-15 5:09
17	4	1	2006-02-15 5:09
...	...	...	...

- language(language_id, name, lst_update)

  : 언어에 대한 정보가 기록된 릴레이션

표. language 릴레이션의 상태.

language_id	name	last_update
1	English	2006-02-15 5:02
2	Italian	2006-02-15 5:02
3	Japanese	2006-02-15 5:02
4	Mandarin	2006-02-15 5:02
5	French	2006-02-15 5:02
6	German	2006-02-15 5:02

- store(store_id, manager_staff_id, address_id, last_update)

  : 영화 DVD 대여점 정보가 기록된 릴레이션

표. store 릴레이션의 상태.

store_id	manager_staff_id	address_id	last_update
1	1	1	2006-02-15 4:57
2	2	2	2006-02-15 4:57

- staff(staff_id, first_name, last_name, address_id, picture, email, store_id, active, username, password, last_update)

  : 대여점 스텝에 대한 정보가 기록된 릴레이션

표. staff 릴레이션의 상태.

staff_id	first_name	last_name	address_id	picture	email	store_id	active	username	password	last_update
1	Mike	Hillyer	3		Mike.Hillyer@sakilastaff.com	1	1	Mike	8cb2237d0679ca88db6464eac60da96345513964	2006-02-15 4:57
2	Jon	Stephens	4	NULL	Jon.Stephens@sakilastaff.com	2	1	Jon	8cb2237d0679ca88db6464eac60da96345513964	2006-02-15 4:57

- payment(payment_id, customer_id, staff_id, rental_id, amount, payment_date, last_update)

  : 고객의 과금 정보가 기록된 릴레이션

표. payment 릴레이션의 상태.

payment_id	customer_id	staff_id	rental_id	amount	payment_date	last_update
1	1	1	76	2.99	2005-05-25 11:30	2006-02-15 22:12
2	1	1	573	0.99	2005-05-28 10:35	2006-02-15 22:12
3	1	1	1185	5.99	2005-06-15 0:54	2006-02-15 22:12
4	1	2	1422	0.99	2005-06-15 18:02	2006-02-15 22:12
5	1	2	1476	9.99	2005-06-15 21:08	2006-02-15 22:12
6	1	1	1725	4.99	2005-06-16 15:18	2006-02-15 22:12
7	1	1	2308	4.99	2005-06-18 8:41	2006-02-15 22:12
8	1	2	2363	0.99	2005-06-18 13:33	2006-02-15 22:12
9	1	1	3284	3.99	2005-06-21 6:24	2006-02-15 22:12
10	1	2	4526	5.99	2005-07-08 3:17	2006-02-15 22:12
11	1	1	4611	5.99	2005-07-08 7:33	2006-02-15 22:12
12	1	1	5244	4.99	2005-07-09 13:24	2006-02-15 22:12
13	1	1	5326	4.99	2005-07-09 16:38	2006-02-15 22:12

14	1	1	6163	7.99	2005-07-11 10:13	2006-02-15 22:12
15	1	2	7273	2.99	2005-07-27 11:31	2006-02-15 22:12
16	1	1	7841	4.99	2005-07-28 9:04	2006-02-15 22:12
17	1	2	8033	4.99	2005-07-28 16:18	2006-02-15 22:12
18	1	1	8074	0.99	2005-07-28 17:33	2006-02-15 22:12
19	1	2	8116	0.99	2005-07-28 19:20	2006-02-15 22:12
20	1	2	8326	2.99	2005-07-29 3:58	2006-02-15 22:12
21	1	2	9571	2.99	2005-07-31 2:42	2006-02-15 22:12
22	1	2	10437	4.99	2005-08-01 8:51	2006-02-15 22:12
23	1	2	11299	3.99	2005-08-02 15:36	2006-02-15 22:12
24	1	1	11367	0.99	2005-08-02 18:01	2006-02-15 22:12
25	1	2	11824	4.99	2005-08-17 12:37	2006-02-15 22:12
…	…	…	…	…	…	…

- rental(rental_id, rental_date, inventory_id, customer_id, return_date, staff_id, last_update)

  : 고객의 대여 정보가 기록된 릴레이션

표. rental 릴레이션의 상태.

rental_id	rental_date	inventory_id	customer_id	return_date	staff_id	last_update
1	2005-05-24 22:53	367	130	2005-05-26 22:04	1	2006-02-15 21:30
2	2005-05-24 22:54	1525	459	2005-05-28 19:40	1	2006-02-15 21:30
3	2005-05-24 23:03	1711	408	2005-06-01 22:12	1	2006-02-15 21:30
4	2005-05-24 23:04	2452	333	2005-06-03 1:43	2	2006-02-15 21:30
5	2005-05-24 23:05	2079	222	2005-06-02 4:33	1	2006-02-15 21:30
6	2005-05-24 23:08	2792	549	2005-05-27 1:32	1	2006-02-15 21:30
7	2005-05-24 23:11	3995	269	2005-05-29 20:34	2	2006-02-15 21:30
8	2005-05-24 23:31	2346	239	2005-05-27 23:33	2	2006-02-15 21:30
9	2005-05-25 0:00	2580	126	2005-05-28 0:22	1	2006-02-15 21:30
10	2005-05-25 0:02	1824	399	2005-05-31 22:44	2	2006-02-15 21:30

11	2005-05-25 0:09	4443	142	2005-06-02 20:56	2	2006-02-15 21:30
12	2005-05-25 0:19	1584	261	2005-05-30 5:44	2	2006-02-15 21:30
13	2005-05-25 0:22	2294	334	2005-05-30 4:28	1	2006-02-15 21:30
14	2005-05-25 0:31	2701	446	2005-05-26 2:56	1	2006-02-15 21:30
15	2005-05-25 0:39	3049	319	2005-06-03 3:30	1	2006-02-15 21:30
16	2005-05-25 0:43	389	316	2005-05-26 4:42	2	2006-02-15 21:30
17	2005-05-25 1:06	830	575	2005-05-27 0:43	1	2006-02-15 21:30
18	2005-05-25 1:10	3376	19	2005-05-31 6:35	2	2006-02-15 21:30
19	2005-05-25 1:17	1941	456	2005-05-31 6:00	1	2006-02-15 21:30
20	2005-05-25 1:48	3517	185	2005-05-27 2:20	2	2006-02-15 21:30
...	...	...	...	...	...	...

MySQL에서는 이미 sakila 데이터베이스를 샘플로 제공하고 있으므로 본 저서에서는 대표적으로 actor 릴레이션의 생성 방법만 설명하기로 한다.

먼저 MySQL Workbench 5.2 CE를 관리자 권한으로 실행한다.

그림 30은 MySQL Workbench를 실행했을 때의 초기 화면이다.

MySQL Workbench에서는 SQL Development, Data Modeling, Server Administration의 세가지 기능을 수행할 수 있다.

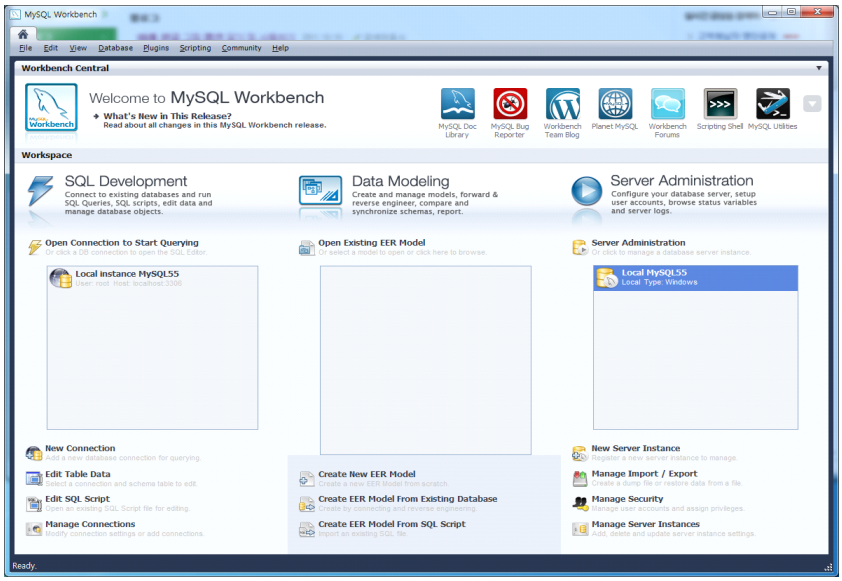

그림 30. MySQL Workbench 실행 초기 화면.

우리는 데이터베이스를 생성하여 각종 질의를 수행하고 결과를 확인하는 작업을 수행할 것이므로 SQL Development의 기능을 수행하기로 한다.

먼저 새로운 DB Connection을 생성해야 하므로 그림에서 New Connection을 클릭한다.

그림 31은 새로운 DB Connection을 생성하는 팝업 창이다.

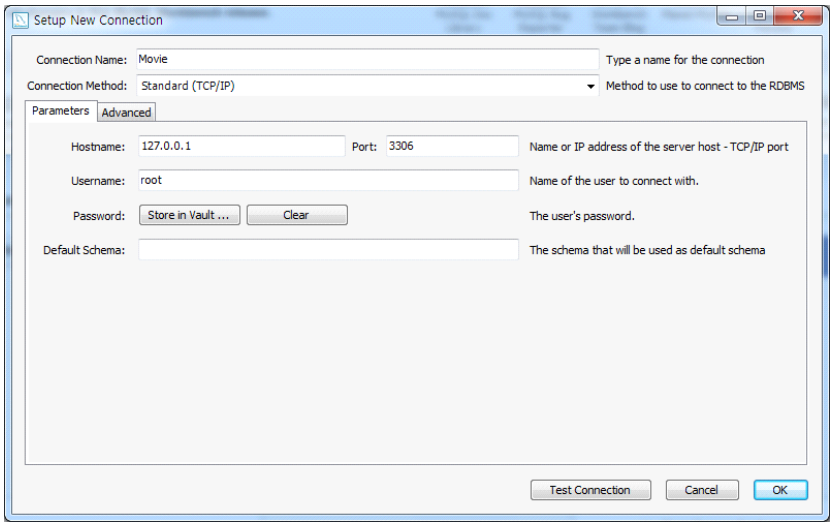

그림 31. 새로운 DB Connection 생성하기.

팝업 창에서 Connection Name을 설정하고(본 예에서는 Moviefkrh 설정하였다), 사용자 패스워드를 설정한다.

그리고 OK 버튼을 클릭하면 새로운 DB Connection이 생성되고, MySQL Workbench 초기 화면에 생성된 DB Connection의 이름이 나타나게 된다.

이 이름을 더블 클릭하고, 패스워드를 입력하면 그림 32과 같은 SQL Editor 화면이 나타난다.

그림에서 이미 sakila schema가 생성되어 있으며, 릴레이션들도 존재하는 것을 알 수 있다.

우리는 앞으로 이 sakila 데이터베이스를 예제로 활용할 것이다.

그러나 그에 앞서 Schema 생성 방법과 릴레이션 생성 방법을 설명하기로 한다.

릴레이션을 생성하기에 앞서 먼저 Schema를 생성해야 한다.

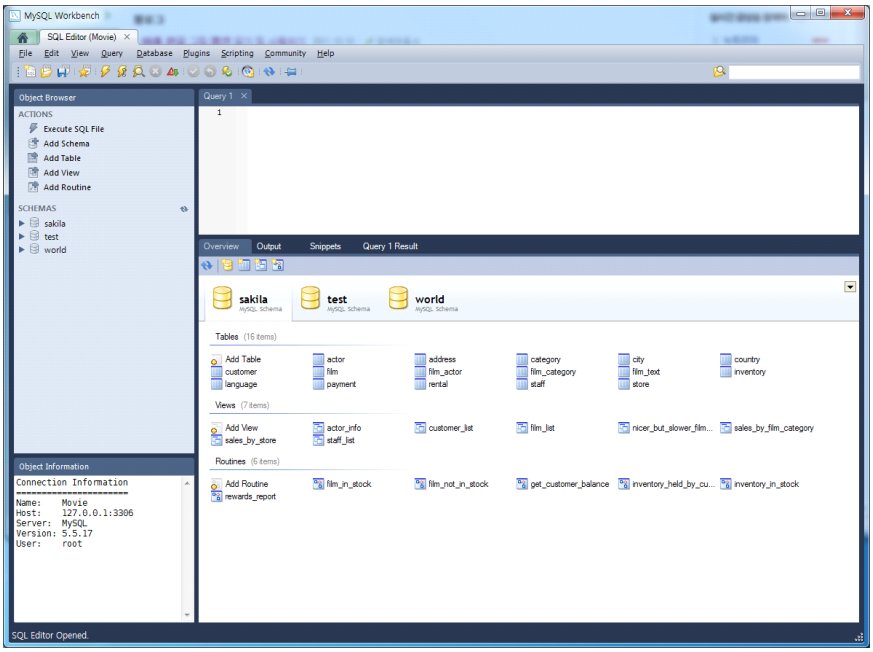

그림 32. SQL Editor 화면.

이를 위해 다음과 같은 SQL 명령어를 수행한다.

```
CREATE SCHEMA 'Movie';
```

그림 33은 이 SQL 명령어를 실행하는 화면이다. 본 화면에서 Apply 버튼을 클릭하면 새로운 Schema가 생성된다.

이제 생성된 Schema에 릴레이션들을 만들어보자.

생성해야 할 릴레이션이 15개이므로 15개의 Create Table을 작성 및 수행해야 하지만 본 장에서는 예제로 actor 릴레이션을 생성하는 SQL 질의어만 작성해 보기로 한다.

Add Table 버튼을 클릭한 후 팝업창에서 릴레이션 이름인 actor를 입력한다.

이제 Columns에 대한 정보를 입력할 차례이다.

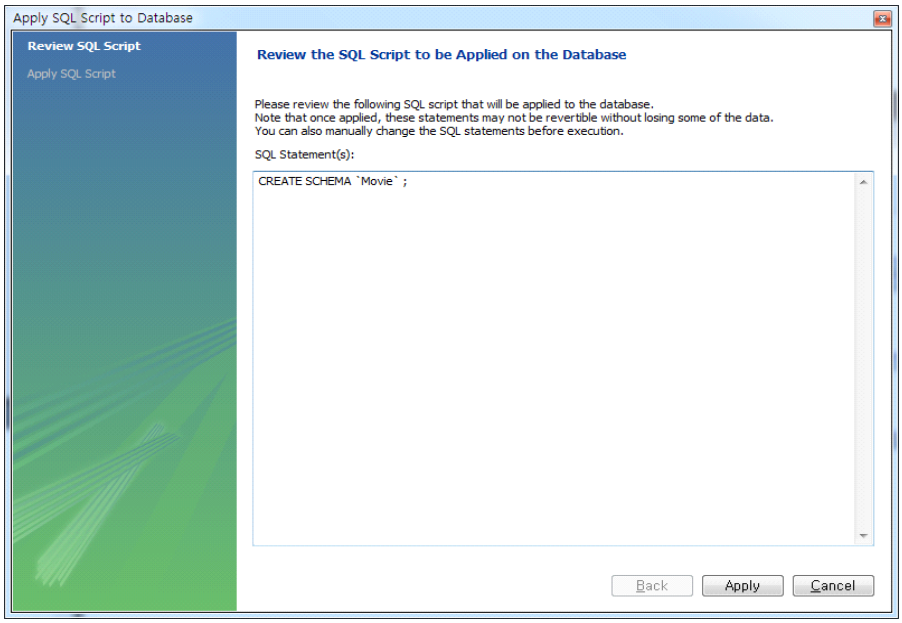

그림 33. Create Schema 실행 화면.

actor 릴레이션은 4개의 속성(actor_id, first_name, last_name, last_update)으로 구성된다.

이중 actor_id가 기본 키라는 정보와 함께 모든 속성의 데이터 타입을 입력한 후 Apply 버튼을 클릭하면 다음과 같은 SQL 질의어가 생성된다.

```
CREATE  TABLE `movie`.`actor` (
  `actor_id` DECIMAL NOT NULL ,
  `first_name` VARCHAR(45) NULL ,
  `last_name` VARCHAR(45) NULL ,
  `last_update` DATETIME NULL ,
  PRIMARY KEY (`actor_id`) );
```

생성된 질의어에 NOT NULL 이라는 키워드가 있다.

일반적으로 속성에는 특정한 값이 기록된다.

예를 들어 actor 릴레이션에 다음과 같은 값을 가지는 투플이 있다고 가정하자.

1	PENELOPE	GUINESS	2006-02-15 4:34

이 투플은 first_name 속성에 "PENELOPE"라는 값을 가지고 있다.

이는 actor_id가 1번인 사람의 이름이 “PENELOPE"라는 것을 의미한다.

그러나 경우에 따라서는 어떤 투플의 속성 값으로 NULL이 기록되는 경우가 있다.

예를 들어 다음과 같은 값을 가지는 투플이 있다고 가정하자.

500	NULL	GUINESS	2006-02-15 4:34

이 투플은 first_name 속성에 “NULL"이 기록되어 있다.

그러나 그것이 actor_id가 500번인 사람의 이름이 “NULL"이라는 것을 의미하는 것은 아니다.

왜냐하면 NULL은 일반적으로 인식되는 특정한 값이 아니라 해당 속성의 값이 없거나 아직 알지 못한다는 것을 의미하는 특수 문자이기 때문이다.

그러므로 NULL에 대해서는 일반적인 값과는 다른 별도의 처리 방식이 필요하다.

NULL이 기록된 속성에 대하여 비교 연산을 수행하는 경우 어떠한 비

교 연산도 그 결과는 false가 된다.

일반적인 속성의 경우 그 값이 없거나 알려져 있지 않을 수 있으므로 NULL을 기록하는 것이 허용되는 경우가 많다.

그러나 특별히 기본 키인 속성에 대해서는 NULL이 허용되지 않는다.

왜냐하면 기본 키인 속성은 해당 릴레이션에 속하는 투플들 중에서 특정한 투플을 식별하는 식별자 역할을 해야하기 때문이다.

actor 릴레이션에서 actor_id는 기본 키에 해당하는 속성이므로, 특별히 NOT NULL로 설정하여 NULL 값이 기록되지 않도록 강제할 수 있다.

그림 34는 입력한 정보에 따라 생성된 SQL 질의어를 보여주고 있다.

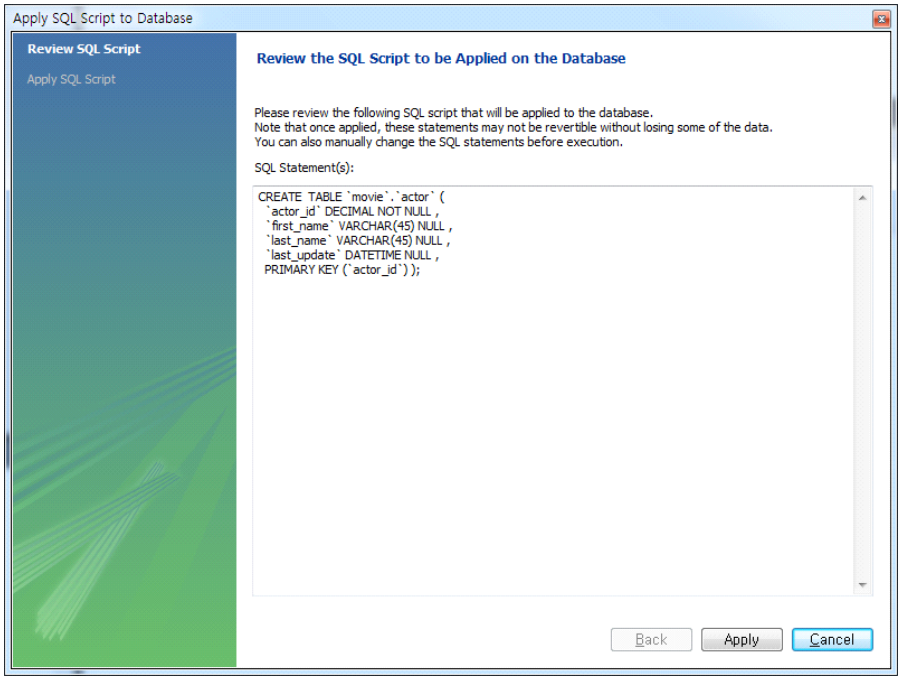

그림 34. actor 릴레이션 생성 화면.

Apply 버튼을 클릭하면 Movie schema 내에 actor 릴레이션이 생성된다.

이와 같은 방식으로 나머지 14개의 릴레이션을 모두 생성할 수 있다.

이제 다음부터는 sakila schema를 사용하여 SQL 질의어를 설명하기로 한다.

3.4 SQL을 이용한 데이터베이스 관리 시스템의 활용

가) 기본적인 SELECT 질의문

SELECT 문은 SQL 질의의 기본이 되는 구문으로서 데이터를 조회하는데 사용된다.

SELECT 문의 기본 형식은 다음과 같다.

```
SELECT <attribute list>
FROM  <table list>
WHERE <condition>
```

<attribute list> : 질의 결과에 나타나는 애트리뷰트 이름목록
<table list> : 질의의 대상이 되는 릴레이션 목록
<condition> : 질의 결과에 포함될 투플들을 표시하는조건식

SELECT 문의 기본 형식은 관계 대수의 셀렉션, 프로젝션, 카테시안 프로덕트를 모두 결합하여 사용할 수 있도록 설계된 것이다.

SELECT 절의 <attribute list>는 관계 대수 프로젝션 연산자에서 나오는 속성 리스트와 동일한 의미를 가진다.

즉, 결과 릴레이션에서 표현해야 할 속성들의 리스트를 의미하는 것이다.

FROM 절의 <table list>는 관계 대수에서 나타나는 릴레이션들에 해당한다.

WHERE 절의 <condition>은 관계 대수 셀렉션 연산자에서 나오는 조건문과 동일한 의미를 가진다.

즉, 주어진 릴레이션에서 추출할 투플이 만족해야 하는 조건을 기술한 것이다.

이제 가장 기본적인 SELECT 문을 사용해 보기로 하자.

1) 릴레이션의 조회

본 절에서 가장 간단한 형태의 SELECT 문을 사용해 보기로 한다.

가장 간단한 형태의 SELECT 문은 질의문으로서 한 릴레이션의 내용 전체를 조회하는 구문으로서 다음과 같이 SELECT 절과 FROM 절만으로 구성된다.

아래의 구문에서 '*'는 모든 속성을 나타낼 것을 의미한다.

질의 : category 릴레이션의 모든 쿠폴을 검색하라.

```
SELECT *
FROM CATEGORY;
```

질의 결과

```
> select *

from CATEGORY

+ ---------------- + --------- + ---------------- +

| category_id      | name      | last_update      |

+ ---------------- + --------- + ---------------- +

| 1                | Action    | 2006-02-15 04:46:27 |

| 2                | Animation | 2006-02-15 04:46:27 |

| 3                | Children  | 2006-02-15 04:46:27 |

| 4                | Classics  | 2006-02-15 04:46:27 |

| 5                | Comedy    | 2006-02-15 04:46:27 |

| 6                | Documentary | 2006-02-15 04:46:27 |

| 7                | Drama     | 2006-02-15 04:46:27 |

| 8                | Family    | 2006-02-15 04:46:27 |

| 9                | Foreign   | 2006-02-15 04:46:27 |

| 10               | Games     | 2006-02-15 04:46:27 |

| 11               | Horror    | 2006-02-15 04:46:27 |

| 12               | Music     | 2006-02-15 04:46:27 |

| 13               | New       | 2006-02-15 04:46:27 |

| 14               | Sci-Fi    | 2006-02-15 04:46:27 |

| 15               | Sports    | 2006-02-15 04:46:27 |

| 16               | Travel    | 2006-02-15 04:46:27 |

+ ---------------- + --------- + ---------------- +

16 rows
```

2) 릴레이션의 특정 속성 조회

본 절에서는 투플 전체를 조회하는 것이 아니라 일부 속성만을 선택하여 조회하는 SELECT 구문을 사용해 보자.

이 때에는 SELECT 절에 원하는 속성의 리스트를 나열하면 된다.

실행 결과는 매우 많은 투플들이 검색되므로 지면 관계상 상위 20여 개의 검색 결과만을 나타내기로 한다.

질의 : film 릴레이션에 속하는 모든 투플에 대하여 film_id, title, release_year를 검색하라.

```
SELECT film_id, title, release_year
FROM FILM;
```

질의 결과

```
> select film_id, title, release_year
from film
+ ------------ + ---------- + ----------------- +
| film_id      | title      | release_year      |
+ ------------ + ---------- + ----------------- +
| 1            | ACADEMY DINOSAUR | 2006             |
| 2            | ACE GOLDFINGER | 2006             |
| 3            | ADAPTATION HOLES | 2006             |
| 4            | AFFAIR PREJUDICE | 2006             |
| 5            | AFRICAN EGG | 2006             |
| 6            | AGENT TRUMAN | 2006             |
| 7            | AIRPLANE SIERRA | 2006             |
```

```
| 8           | AIRPORT POLLOCK | 2006             |
| 9           | ALABAMA DEVIL | 2006               |
| 10          | ALADDIN CALENDAR | 2006              |
| 11          | ALAMO VIDEOTAPE | 2006              |
| 12          | ALASKA PHANTOM | 2006               |
| 13          | ALI FOREVER | 2006                |
| 14          | ALICE FANTASIA | 2006              |
| 15          | ALIEN CENTER | 2006               |
| 16          | ALLEY EVOLUTION | 2006              |
| 17          | ALONE TRIP | 2006                |
| 18          | ALTER VICTORY | 2006               |
| 19          | AMADEUS HOLY | 2006               |
| 20          | AMELIE HELLFIGHTERS | 2006     |
| 21          | AMERICAN CIRCUS | 2006              |
| 22          | AMISTAD MIDSUMMER | 2006              |
...
```

이와 같이 어떤 릴레이션의 일부 속성만을 추출하는 경우 결과에 출력되는 속성들의 나열 순서는 원 릴레이션에서 나타나는 속성의 순서와 무관하고, 오로지 SELECT 문에 기술된 속성 순서에 따라 나열된다.

위와 동일한 의미의 질의이지만 추출하는 속성의 순서를 변경시켜 보도록 하자.

결과 릴레이션의 속성 나열 순서가 위의 결과와는 달리 title, film_id, release_year의 순서대로 나열되어 있는 것을 확인할 수 있다.

질의 : film 릴레이션에 속하는 모든 투플에 대하여 title, film_id, release_year를 검색하라.

```
SELECT title, film_id, release_year
FROM FILM;
```

질의 결과

```
> SELECT title, film_id, release_year
FROM FILM
+ ---------- + ------------ + ----------------- +
| title      | film_id      | release_year      |
+ ---------- + ------------ + ----------------- +
| ACADEMY DINOSAUR | 1            | 2006              |
| ACE GOLDFINGER | 2            | 2006              |
| ADAPTATION HOLES | 3            | 2006              |
| AFFAIR PREJUDICE | 4            | 2006              |
| AFRICAN EGG | 5            | 2006              |
| AGENT TRUMAN | 6            | 2006              |
| AIRPLANE SIERRA | 7            | 2006              |
| AIRPORT POLLOCK | 8            | 2006              |
| ALABAMA DEVIL | 9            | 2006              |
| ALADDIN CALENDAR | 10           | 2006              |
| ALAMO VIDEOTAPE | 11           | 2006              |
| ALASKA PHANTOM | 12           | 2006              |
| ALI FOREVER | 13           | 2006              |
| ALICE FANTASIA | 14           | 2006              |
| ALIEN CENTER | 15           | 2006              |
```

```
| ALLEY EVOLUTION | 16          | 2006             |
| ALONE TRIP | 17           | 2006             |
| ALTER VICTORY | 18          | 2006             |
| AMADEUS HOLY | 19           | 2006             |
| AMELIE HELLFIGHTERS | 20     | 2006             |
| AMERICAN CIRCUS | 21        | 2006             |
| AMISTAD MIDSUMMER | 22      | 2006             |
 ...
```

3) 중복되지 않은 속성 값의 조회

일반적인 SQL 구문을 실행하면 결과에 중복된 값이 삭제되지 않은 채로 검색된다.

다음은 중복된 값이 검색되는 예이다.

질의 : film 릴레이션에서 대여료를 모두 검색하라.

```
SELECT rental_rate
FROM FILM;
```

질의 결과

```
> select rental_rate
from film
+ ---------------- +
| rental_rate      |
+ ---------------- +
```

```
| 0.99          |
| 4.99          |
| 2.99          |
| 2.99          |
| 2.99          |
| 2.99          |
| 4.99          |
| 4.99          |
| 2.99          |
| 4.99          |
| 0.99          |
| 0.99          |
| 4.99          |
| 0.99          |
| 2.99          |
| 2.99          |
| 0.99          |
| 0.99          |
| 0.99          |
| 4.99          |
| 4.99          |
| 2.99          |
| 0.99          |
| 2.99          |
| 2.99          |
| 0.99          |
...
```

결과에서 살펴볼 수 있듯이 대여료 0.99, 2.99, 4.99 등의 값이 중복해서 검색되는 것을 볼 수 있다.

경우에 따라서 사용자는 중복이 제거된 상태의 결과를 검색하고 싶은 경우가 있을 것이다.

이 경우에는 Disinct 라는 키워드를 사용한다.

질의 : film 릴레이션에서 대여료 종류를 모두 검색하라.

```
SELECT Distinct rental_rate
FROM FILM;
```

질의 결과

```
> select Distinct rental_rate
from film
+ ---------------- +
| rental_rate      |
+ ---------------- +
| 0.99             |
| 4.99             |
| 2.99             |
+ ---------------- +
3 rows
```

질의 결과에서 보는 바와 같이 중복되었던 값들이 모두 제거되고 0.99, 4.99, 2.99 세 개의 값만이 검색되는 것을 알 수 있다.

4) 조건을 만족하는 투플의 조회

본 절에서는 주어진 조건을 만족하는 투플들만을 검색하는 질의문에 대하여 설명한다.

주어진 조건을 만족하는 투플을 검색하는 연산은 관계 대수의 셀렉션 연산에 해당한다.

SELECT 문에서는 WHERE 절에 조건을 기술하여 이 구문을 처리한다.

WHERE 절에 기술하는 조건은 특정 속성의 값을 다른 속성값 또는 상수값과 비교하는 조건으로 논리 비교 연산자 <,>,=,<=,>=,<>를 사용할 수 있으며, 조건이 여러개인 경우에는 AND 또는 OR 키워드를 사용하여 여러 조건을 나열할 수 있다.

다음과 같은 조건을 만족하는 투플을 검색해 보자.

질의 : actor 릴레이션에서 성이 “ALLEN”인 배우의 정보를 검색하라.

```
select *
from actor
where last_name = "allen";
```

질의 결과

```
> select *
from actor
where last_name = "allen"
+ ------------ + --------------- + -------------- + ---------------- +
```

```
| actor_id      | first_name      | last_name      | last_update      |
+ ------------- + --------------- + -------------- + ---------------- +
| 118           | CUBA            | ALLEN          | 2006-02-15 04:34:33 |
| 145           | KIM             | ALLEN          | 2006-02-15 04:34:33 |
| 194           | MERYL           | ALLEN          | 2006-02-15 04:34:33 |
+ ------------- + --------------- + -------------- + ---------------- +
3 rows
```

질의 결과에서 성이 ALLEN인 3인의 배우가 검색되는 것을 알 수 있다.

두 개 이상의 조건을 연결하여 부여할 수도 있다.

다음과 같은 질의를 생각해 보자.

이 질의에서는 두 개의 조건을 부울 연산자 AND를 이용해 연결하여 사용하였다.

질의 : actor 릴레이션에서 성이 “ALLEN"이고, 이름이 ”KIM" 인 배우의 정보를 검색하라.

```
select *
from actor
where last_name = "allen"
        and first_name = "Kim";
```

질의 결과

```
> select *
```

```
from actor
where last_name = "allen" and first_name = "Kim"
+ ------------ + --------------- + -------------- + ---------------- +
| actor_id     | first_name      | last_name      | last_update      |
+ ------------ + --------------- + -------------- + ---------------- +
| 145          | KIM             | ALLEN          | 2006-02-15 04:34:33 |
+ ------------ + --------------- + -------------- + ---------------- +
1 rows
```

이 경우 WHERE절에 있는 두 개의 조건문 순서를 바꾸어도 검색 결과에는 영향을 미치지 않는다.

다음의 SQL 질의문을 수행해 보자.

이 SQL 질의문은 위의 질의문에서 WHERE절의 두 조건문 기술 순서를 뒤바꾼 것이다.

이 경우에도 결과는 변하지 않는 것을 확인할 수 있다.

```
select *
from actor
where first_name = "Kim"
      and last_name = "allen";
```

질의 결과

```
> select *
from actor
where first_name = "Kim"
      and last_name = "allen"
```

```
+ ------------ + -------------- + ------------- + ---------------- +
| actor_id     | first_name     | last_name     | last_update      |
+ ------------ + -------------- + ------------- + ---------------- +
| 145          | KIM            | ALLEN         | 2006-02-15 04:34:33 |
+ ------------ + -------------- + ------------- + ---------------- +
1 rows
```

이제 여러 가지 논리비교 연산자와 부울 연산자를 연결하여 사용해 보자.

질의 : actor 릴레이션에서 성이 “ALLEN"이고, 이름이 ”KIM" 이 아닌 배우의 정보를 검색하라.

```
select *
from actor
where last_name = "allen"
      and first_name <> "Kim";
```

질의 결과

```
> select *
from actor
where last_name = "allen" and first_name <> "Kim"
+ ------------ + -------------- + ------------- + ---------------- +
| actor_id     | first_name     | last_name     | last_update      |
+ ------------ + -------------- + ------------- + ---------------- +
| 118          | CUBA           | ALLEN         | 2006-02-15 04:34:33 |
| 194          | MERYL          | ALLEN         | 2006-02-15 04:34:33 |
+ ------------ + -------------- + ------------- + ---------------- +
2 rows
```

이 질의는 다음과 같이 부울 연산자 NOT을 이용하여 표현할 수도 있다.

질의 : actor 릴레이션에서 성이 “ALLEN"고,”KIM"이 아닌 배우의 정보를 검색하라.

```
select *
from actor
where last_name = "allen"
        and not first_name = "Kim";
```

질의 결과

```
> select *
from actor
where last_name = "allen" and not first_name = "Kim"
+ ------------ + --------------- + -------------- + ----------------- +
| actor_id     | first_name      | last_name      | last_update       |
+ ------------ + --------------- + -------------- + ----------------- +
| 118          | CUBA            | ALLEN          | 2006-02-15 04:34:33 |
| 194          | MERYL           | ALLEN          | 2006-02-15 04:34:33 |
+ ------------ + --------------- + -------------- + ----------------- +
2 rows
```

이제 부울 연산자 OR를 사용해 보자.

다음 질의의 결과에서 성이 allen인 3인의 배우와 성이 wood인 2인의 배우가 검색된 것을 볼 수 있다.

질의 : actor 릴레이션에서 성이 “ALLEN"이거나 ”WOOD"인 배우의 정보를 검색하라.

```
select *
from actor
where last_name = "allen"
      OR last_name = "wood";
```

질의 결과

```
> select *
from actor
where last_name = "allen" OR last_name = "WOOD"
+ ------------ + --------------- + -------------- + ---------------- +
| actor_id     | first_name      | last_name      | last_update      |
+ ------------ + --------------- + -------------- + ---------------- +
| 118          | CUBA            | ALLEN          | 2006-02-15 04:34:33 |
| 145          | KIM             | ALLEN          | 2006-02-15 04:34:33 |
| 194          | MERYL           | ALLEN          | 2006-02-15 04:34:33 |
| 13           | UMA             | WOOD           | 2006-02-15 04:34:33 |
| 156          | FAY             | WOOD           | 2006-02-15 04:34:33 |
+ ------------ + --------------- + -------------- + ---------------- +
5 rows
```

5) 셀렉션과 프로젝션의 조합

본 절에서는 주어진 조건을 만족하는 투플을 검색하는 셀렉션 연산과 원하는 속성만을 검색하는 프로젝션 연산을 조합하는 방법에 대하여

살펴보자.

다음과 같은 질의를 생각해 보자.

질의 : actor 릴레이션에서 성이 “ALLEN"이거나 ”WOOD"인 배우의 actor_id, last_name을 검색하라.

```
select actor_id, last_name
from actor
where last_name = "allen"
        OR last_name = "wood";
```

질의 결과

```
> select actor_id, last_name
from actor
where last_name = "allen"
       OR last_name = "wood"
+ ------------ + -------------- +
| actor_id     | last_name      |
+ ------------ + -------------- +
| 118          | ALLEN          |
| 145          | ALLEN          |
| 194          | ALLEN          |
| 13           | WOOD           |
| 156          | WOOD           |
+ ------------ + -------------- +
5 rows
```

연습을 위하여 한 가지 질의를 더 살펴보자.

질의 : address 릴레이션에서 California 구역에 속한 address_id와 address, 우편번호를 검색하라.

```
select address_id, address, postal_code
from address
where district = "California";
```

질의 결과

```
> select address_id, address, postal_code
from address
where district = "California"
+ -------------- + ------------ + ---------------- +
| address_id     | address      | postal_code      |
+ -------------- + ------------ + ---------------- +
| 6              | 1121 Loja Avenue | 17886            |
| 18             | 770 Bydgoszcz Avenue | 16266            |
| 55             | 1135 Izumisano Parkway | 48150            |
| 116            | 793 Cam Ranh Avenue | 87057            |
| 186            | 533 al-Ayn Boulevard | 8862             |
| 218            | 226 Brest Manor | 2299             |
| 274            | 920 Kumbakonam Loop | 75090            |
| 425            | 1866 al-Qatif Avenue | 89420            |
| 599            | 1895 Zhezqazghan Drive | 36693            |
+ -------------- + ------------ + ---------------- +
9 rows
```

6) 문자열 비교 연산의 사용

SQL에서는 문자열에 대한 비교 연산을 지원하고 있다.

이 때에는 like 키워드와 '%'연산자, 그리고 '_'연산자를 사용한다.

다음의 질의를 살펴보자.

질의 : address 릴레이션에서 Ca로 시작되는 구역에 속한 address_id와 address, 우편번호, 구역이름을 검색하라.

```
select address_id, address, postal_code, district
from address
where district like "Ca%";
```

질의 결과

```
> select address_id, address, postal_code, district
from address
where district like "Ca%"
+ --------------- + ------------ + ----------------- + ------------- +
| address_id      | address      | postal_code       | district      |
+ --------------- + ------------ + ----------------- + ------------- +
| 6               | 1121 Loja Avenue | 17886             | California    |
| 18              | 770 Bydgoszcz Avenue | 16266             | California    |
| 55              | 1135 Izumisano Parkway | 48150             | California    |
| 69              | 698 Otsu Street | 71110             | Cayenne       |
| 98              | 152 Kitwe Parkway | 53182             | Caraga        |
| 116             | 793 Cam Ranh Avenue | 87057             | California    |
| 186             | 533 al-Ayn Boulevard | 8862              | California    |
```

```
| 192          | 1166 Changhwa Street | 58852          | Caraga       |
| 218          | 226 Brest Manor | 2299           | California   |
| 266          | 862 Xintai Lane | 30065          | Cagayan Valley |
| 274          | 920 Kumbakonam Loop | 75090          | California   |
| 309          | 827 Yuncheng Drive | 79047          | Callao       |
| 425          | 1866 al-Qatif Avenue | 89420          | California   |
| 465          | 734 Tanshui Avenue | 70664          | Caquet       |
| 545          | 163 Augusta-Richmond County Loop | 33030    | Carabobo     |
| 584          | 1819 Alessandria Loop | 53829          | Campeche     |
| 599          | 1895 Zhezqazghan Drive | 36693          | California   |
+ ------------- + ------------ + ---------------- + ------------ +
17 rows
```

질의 결과를 보면 구역 이름이 모두 “Ca"로 시작되는 것을 알 수 있다.

마찬가지로 ‘%’연산자를 문자열의 앞부분에 사용할 수도 있다.

이 경우에는 문자열의 앞 부분에 임의의 문자열이 있고, 뒷 부분은 주어진 문자열로 끝나는 속성값을 가진 투플을 검색하라는 의미가 된다.

다음과 같은 질의를 살펴보자.

질의 : address 릴레이션에서 nia로 끝나는 구역에 속한 address_id와 address, 우편번호, 구역이름을 검색하라.

```
select address_id, address, postal_code, district
from address
where district like "%nia";
```

질의 결과

```
> select address_id, address, postal_code, district
from address
where district like "%nia"
+ --------------- + ------------ + ----------------- + ------------- +
| address_id      | address      | postal_code       | district      |
+ --------------- + ------------ + ----------------- + ------------- +
| 6               | 1121 Loja Avenue | 17886            | California    |
| 18              | 770 Bydgoszcz Avenue | 16266            | California    |
| 55              | 1135 Izumisano Parkway | 48150          | California    |
| 116             | 793 Cam Ranh Avenue | 87057             | California    |
| 186             | 533 al-Ayn Boulevard | 8862             | California    |
| 218             | 226 Brest Manor | 2299              | California    |
| 274             | 920 Kumbakonam Loop | 75090            | California    |
| 296             | 1936 Cuman Avenue | 61195              | Virginia      |
| 348             | 785 Vaduz Street | 36170             | Baja California |
| 425             | 1866 al-Qatif Avenue | 89420            | California    |
| 599             | 1895 Zhezqazghan Drive | 36693          | California    |
+ --------------- + ------------ + ----------------- + ------------- +
11 rows
```

'%'연산자를 문자열의 앞부분과 뒷부분에 모두 사용할 수도 있다.

이 경우에는 문자열 중에 주어진 문자 (또는 문자열)이 포함된 모든 투플을 검색하는 질의가 된다.

다음과 같은 질의를 살펴보자.

질의 : address 릴레이션에서 구역이름에 'th'가 포함되는 구역에 속한 address_id와 address, 우편번호, 구역이름을 검색하라.

```
select address_id, address, postal_code, district
from address
where district like "%th%";
```

질의 결과

```
> select address_id, address, postal_code, district
from address
where district like "%th%"
+ --------------- + ------------ + ---------------- + ------------- +
| address_id      | address      | postal_code      | district      |
+ --------------- + ------------ + ---------------- + ------------- +
| 23              | 1417 Lancaster Avenue | 72192            | Northern Cape |
| 24              | 1688 Okara Way | 21954            | Nothwest Border Prov |
| 31              | 217 Botshabelo Place | 49521            | Southern Mindanao |
| 46              | 1632 Bislig Avenue | 61117            | Nonthaburi    |
| 64              | 81 Hodeida Way | 55561            | Rajasthan     |
| 113             | 682 Junan Way | 30418            | North West    |
| 162             | 369 Papeete Way | 66639            | North Carolina |
| 163             | 1440 Compton Place | 81037            | North Austria |
| 174             | 1877 Ezhou Lane | 63337            | Rajasthan     |
| 213             | 43 Dadu Avenue | 4855             | Rajasthan     |
| 228             | 60 Poos de Caldas Street | 82338            | Rajasthan     |
| 240             | 1479 Rustenburg Boulevard | 18727            | Southern Tagalog |
```

```
| 354            | 953 Hodeida Street | 18841            | Southern Tagalog |
| 376            | 1061 Ede Avenue | 57810            | Southern Tagalog |
| 378            | 1191 Tandil Drive | 6362            | Southern Tagalog |
| 393            | 717 Changzhou Lane | 21615            | Southern Tagalog |
| 411            | 264 Bhimavaram Manor | 54749            | St Thomas      |
| 412            | 1639 Saarbrcken Drive | 9827            | North West     |
| 413            | 692 Amroha Drive | 35575            | Northern       |
| 439          | 1351 Aparecida de Goinia Parkway | 41775     | Northern Mindanao |
| 466            | 118 Jaffna Loop | 10447            | Northern Mindanao |
| 479            | 1854 Okara Boulevard | 42123            | Drenthe        |
| 528            | 1176 Southend-on-Sea Manor | 81651        | Southern Tagalog |
+ -------------- + ------------ + ---------------- + ------------- +
23 rows
```

위와 같이 '%'연산자를 문자열의 일부분에 적절히 사용하면 다양한 문자열 관련 질의를 할 수 있다.

다음과 같은 질의를 살펴보자.

질의 : address 릴레이션에서 'th'가 포함되면서 'an'으로 끝나는 구역이름을 가진 구역에 속한 address_id와 address, 우편번호, 구역이름을 검색하라.

```
select address_id, address, postal_code, district
from address
where district like "%th%an";
```

질의 결과

```
> select address_id, address, postal_code, district
from address
where district like "%th%an"
+ --------------- + ------------ + ---------------- + ------------- +
| address_id      | address      | postal_code      | district      |
+ --------------- + ------------ + ---------------- + ------------- +
| 64              | 81 Hodeida Way | 55561            | Rajasthan     |
| 174             | 1877 Ezhou Lane | 63337            | Rajasthan     |
| 213             | 43 Dadu Avenue | 4855             | Rajasthan     |
| 228             | 60 Poos de Caldas Street | 82338            | Rajasthan     |
+ --------------- + ------------ + ---------------- + ------------- +
4 rows
```

문자열과 관련된 또다른 연산자로는 '_' 연산자가 있다.

이 연산자는 하나의 문자에 대해서 와일드 카드의 역할을 수행하는 연산자이다.

다음과 같은 질의를 살펴보자.

질의 : address 릴레이션에서 첫 번째 문자와는 관계없이 두 번째부터 'ali'가 들어가고 그 뒷 부분은 어떤 문자열이 와도 관계없는 이름의 속한 address_id와 address, 우편번호, 구역이름을 검색하라.

```
select address_id, address, postal_code, district
from address
where district like "_ali%";
```

질의 결과

```
> select address_id, address, postal_code, district
from address
where district like "_ali%"
+ -------------- + ------------ + ---------------- + ------------- +
| address_id     | address      | postal_code      | district      |
+ -------------- + ------------ + ---------------- + ------------- +
| 6              | 1121 Loja Avenue | 17886            | California    |
| 18             | 770 Bydgoszcz Avenue | 16266            | California    |
| 55             | 1135 Izumisano Parkway | 48150            | California    |
| 56             | 939 Probolinggo Loop | 4166             | Galicia       |
| 116            | 793 Cam Ranh Avenue | 87057            | California    |
| 186            | 533 al-Ayn Boulevard | 8862             | California    |
| 218            | 226 Brest Manor | 2299             | California    |
| 274            | 920 Kumbakonam Loop | 75090            | California    |
| 306            | 1883 Maikop Lane | 68469            | Kaliningrad   |
| 369            | 817 Laredo Avenue | 77449            | Jalisco       |
| 371            | 1912 Emeishan Drive | 33050            | Balikesir     |
| 425            | 1866 al-Qatif Avenue | 89420            | California    |
| 470            | 1088 Ibirit Place | 88502            | Jalisco       |
| 474            | 1469 Plock Lane | 95835            | Galicia       |
| 511            | 1152 al-Qatif Lane | 44816            | Kalimantan Barat |
```

```
| 555            | 1764 Jalib al-Shuyukh Parkway | 77642          | Galicia      |
| 599            | 1895 Zhezqazghan Drive | 36693            | California   |
+ -------------- + ------------ + ---------------- + ------------- +
17 rows
```

‘_’ 연산자를 연속으로 사용하여 문자 임의의 문자 개수를 제한하는 질의문을 작성해 보자.

질의 : address 릴레이션에서 첫 번째, 두번째 문자와는 관계없이 세 번째부터 'li'가 들어가고 그 뒷 부분은 어떤 문자열이 와도 관계없는 이름의 구역에 속한 address_id와 address, 우편번호, 구역이름을 검색하라.

```
select address_id, address, postal_code, district
from address
where district like "__li%";
```

질의 결과

```
> select address_id, address, postal_code, district
from address
where district like "__li%"
+ -------------- + ------------ + ---------------- + ------------- +
| address_id     | address      | postal_code      | district      |
+ -------------- + ------------ + ---------------- + ------------- +
```

6	1121 Loja Avenue	17886	California
18	770 Bydgoszcz Avenue	16266	California
55	1135 Izumisano Parkway	48150	California
56	939 Probolinggo Loop	4166	Galicia
116	793 Cam Ranh Avenue	87057	California
143	1029 Dzerzinsk Manor	57519	Ynlin
147	374 Bat Yam Boulevard	97700	Kilis
186	533 al-Ayn Boulevard	8862	California
215	1333 Haldia Street	82161	Jilin
218	226 Brest Manor	2299	California
274	920 Kumbakonam Loop	75090	California
280	1980 Kamjanets-Podilskyi Street	89502	Illinois
306	1883 Maikop Lane	68469	Kaliningrad
335	587 Benguela Manor	91590	Illinois
369	817 Laredo Avenue	77449	Jalisco
371	1912 Emeishan Drive	33050	Balikesir
425	1866 al-Qatif Avenue	89420	California
458	138 Caracas Boulevard	16790	Zulia
462	1485 Bratislava Place	83183	Illinois
470	1088 Ibirit Place	88502	Jalisco
474	1469 Plock Lane	95835	Galicia
509	786 Matsue Way	37469	Illinois
511	1152 al-Qatif Lane	44816	Kalimantan Barat
544	183 Haiphong Street	69953	Jilin

```
| 555            | 1764 Jalib al-Shuyukh Parkway | 77642          | Galicia       |
| 599            | 1895 Zhezqazghan Drive | 36693            | California    |
+ -------------- + ------------ + ---------------- + ------------- +
26 rows
```

'_' 연산자를 문자열의 중간 부분에 사용하는 질의문을 작성해 보자.

질의 : address 릴레이션에서 A로 시작하고 임의의 3개의 문자가 나온 후 r이 나오는 이름의 구역에 속한 address_id와 address, 우편번호, 구역이름을 검색하라.

```
select address_id, address, postal_code, district
from address
where district like "A___r%";
```

질의 결과

```
> select address_id, address, postal_code, district
from address
where district like "A___r%"
+ -------------- + ------------ + ---------------- + ------------- +
| address_id     | address      | postal_code      | district      |
+ -------------- + ------------ + ---------------- + ------------- +
| 1              | 47 MySakila Drive |                 | Alberta       |
| 3              | 23 Workhaven Lane |                 | Alberta       |
| 372            | 230 Urawa Drive | 2738            | Andhra Pradesh |
```

```
| 399            | 331 Bydgoszcz Parkway | 966              | Asturia       |
| 513            | 758 Korolev Parkway | 75474            | Andhra Pradesh |
| 535            | 635 Brest Manor | 40899            | Andhra Pradesh |
+ -------------- + ------------ + ---------------- + ------------- +
6 rows
```

'%' 연산자와 '_' 연산자의 차이 중 하나는 '%' 연산자는 빈 문자열도 허용하는 반면에 '_' 연산자는 반드시 하나의 문자만을 허용한다는 것이다.

예를 들어 다음과 같은 두 질의문을 생각해보자.

질의 1 : select address_id, address, postal_code, district
from address
where district like "%lin%";

질의 2 : select address_id, address, postal_code, district
from address
where district like "%lin_";

질의 1과 질의 2는 district 속성값이 되는 문자열의 조건에 '%' 또는 '_' 연산자를 사용한 것을 제외하면 모두 동일한 질의문이다.

이제 이 두 질의문을 수행한 결과를 살펴보자.

질의 1 :

```
select address_id, address, postal_code, district
```

```
from address
where district like "%lin%";
```

질의 결과

```
> select address_id, address, postal_code, district
from address
where district like "%lin%"
+ -------------- + ------------ + ---------------- + ------------- +
| address_id     | address      | postal_code      | district      |
+ -------------- + ------------ + ---------------- + ------------- +
| 143            | 1029 Dzerzinsk Manor | 57519            | Ynlin         |
| 162            | 369 Papeete Way | 66639            | North Carolina |
| 215            | 1333 Haldia Street | 82161            | Jilin         |
| 280            | 1980 Kamjanets-Podilskyi Street | 89502            | Illinois
 |
| 306            | 1883 Maikop Lane | 68469            | Kaliningrad   |
| 335            | 587 Benguela Manor | 91590            | Illinois      |
| 462            | 1485 Bratislava Place | 83183            | Illinois      |
| 509            | 786 Matsue Way | 37469            | Illinois      |
| 544            | 183 Haiphong Street | 69953            | Jilin         |
+ -------------- + ------------ + ---------------- + ------------- +
9 rows
```

질의 2 :

```
select address_id, address, postal_code, district
from address
where district like "%lin_";
```

질의 결과

```
> select address_id, address, postal_code, district

from address

where district like "%lin_"

+ --------------- + ------------ + ---------------- + ------------- +

| address_id      | address      | postal_code      | district      |

+ --------------- + ------------ + ---------------- + ------------- +

| 162             | 369 Papeete Way | 66639            | North Carolina |

+ --------------- + ------------ + ---------------- + ------------- +

1 rows
```

질의 1의 결과에서 address_id가 143, 215, 544 인 세 개의 투플을 살펴보자.

세 투플의 구역이름은 각각 ‘Ynlin’, ‘Jilin’, ‘Jilin’ 이다.

즉, 구역이름이 'lin'으로 끝나는 투플이다.

질의 1의 결과에는 이 세 투플이 포함되어 있으나, 질의 2의 결과에는 포함되지 않은 것을 확인할 수 있다.

‘%’ 연산자는 ‘lin’ 이후에 문자열이 따라 오지 않더라도 조건을 만족한 것으로 인식되는 반면에 ‘_’ 연산자는 반드시 하나의 문자가 따라 와야만 조건이 만족된다.

그러므로 두 연산자를 사용할 때 빈 문자에 대한 의미를 주의하여 사용하여야 한다.

이와 같은 문자열 연산은 특히 키워드 검색 기능을 제공하는 응용 프로그램을 작성할 때에 큰 도움이 된다.

7) 산술 연산자의 사용

관계 대수에는 산술 연산자가 없으나 SQL에서는 산술 연산자를 지원하고 있다.

다음의 질의를 살펴보자.

이 질의의 수행 결과를 살펴보면 마지막에 replacement_cost * 1.1이라는 이름의 속성이 추가되어 있는 것을 알 수 있다.

질의 : film 릴레이션에서 A로 시작하는 제목을 가진 영화 중에서 replacement cost가 $17 이상인 영화에 대해서 film_id, title, replacement_cost, 그리고 replacement cost를 10% 증가시킨 값을 검색하라.

```
select film_id, title, replacement_cost,
        replacement_cost *1.1
from film
where replacement_cost > 17.0
        AND title like "A%";
```

질의 결과

```
> select film_id, title, replacement_cost, replacement_cost *1.1
from film
where replacement_cost > 17.0 AND title like "A%"
+ ------------ + ---------- + --------------------- + -------------------------- +
| film_id      | title      | replacement_cost      | replacement_cost *1.1      |
+ ------------ + ---------- + --------------------- + -------------------------- +
| 1          | ACADEMY DINOSAUR | 20.99                | 23.089                     |
| 3          | ADAPTATION HOLES | 18.99                | 20.889                     |
```

4	AFFAIR PREJUDICE	26.99	29.689
5	AFRICAN EGG	22.99	25.289
6	AGENT TRUMAN	17.99	19.789
7	AIRPLANE SIERRA	28.99	31.889
9	ALABAMA DEVIL	21.99	24.189
10	ALADDIN CALENDAR	24.99	27.489
12	ALASKA PHANTOM	22.99	25.289
13	ALI FOREVER	21.99	24.189
14	ALICE FANTASIA	23.99	26.389
16	ALLEY EVOLUTION	23.99	26.389
18	ALTER VICTORY	27.99	30.789
19	AMADEUS HOLY	20.99	23.089
20	AMELIE HELLFIGHTERS	23.99	26.389
21	AMERICAN CIRCUS	17.99	19.789
24	ANALYZE HOOSIERS	19.99	21.989
30	ANYTHING SAVANNAH	27.99	30.789
34	ARABIA DOGMA	29.99	32.989
35	ARACHNOPHOBIA ROLLERCOASTER	24.99	27.489
37	ARIZONA BANG	28.99	31.889
38	ARK RIDGEMONT	25.99	28.589
40	ARMY FLINTSTONES	22.99	25.289
41	ARSENIC INDEPENDENCE	17.99	19.789
44	ATTACKS HATE	21.99	24.189

25 rows

연습을 위해 몇 가지 산술 연산자 사용 사례를 더 살펴보기로 하자.

산술 연산자의 우선 순위는 일반적인 사칙연산자 및 괄호의 우선 순위와 동일하다.

그리고 논리 및 비교 연산자의 우선 순위는 다음과 같다.

연산자	우선 순위
비교 연산자	1
NOT, AND	2
OR	3

다음 질의를 살펴보자.

다음의 질의는 여러개의 논리 연산자가 혼합되어 사용되고 있다. 질의의 결과를 살펴보면 OR 연산자의 우선 순위가 가장 낮게 적용되었고, AND와 NOT 연산자는 우선 순위가 같기 때문에 질의문에 나타난 순서에 따라 적용된 것을 알 수 있다.

질의 : film 릴레이션에서 A로 시작하는 제목을 가진 영화 중에서 replacement cost가 $17 이상이거나 B로 시작하지만 Ba 또는 Be 또는 Bo 또는 Bi로 시작하지는 않는 모든 영화에 대해서 film_id, title, replacement_cost를 검색하라.

```
select film_id, title, replacement_cost
from film
where replacement_cost > 17.0
        AND title like "A%"
        OR NOT title like "Ba%"
```

AND NOT title like "Bo%"

AND NOT title like "Be%"

AND NOT title like "Bi%"

AND title like "B%" ;

질의 결과

```
> select film_id, title, replacement_cost
from film
where replacement_cost > 17.0
       AND title like "A%"
       OR NOT title like "Ba%"
       AND NOT title like "Bo%"
       AND NOT title like "Be%"
       AND NOT title like "Bi%"
       AND title like "B%"
+ ------------ + ---------- + --------------------- +
| film_id      | title      | replacement_cost      |
+ ------------ + ---------- + --------------------- +
| 1            | ACADEMY DINOSAUR | 20.99                |
| 3            | ADAPTATION HOLES | 18.99                |
| 4            | AFFAIR PREJUDICE | 26.99                |
| 5            | AFRICAN EGG | 22.99                |
| 6            | AGENT TRUMAN | 17.99                |
| 7            | AIRPLANE SIERRA | 28.99                |
| 9            | ALABAMA DEVIL | 21.99                |
```

10	ALADDIN CALENDAR	24.99
12	ALASKA PHANTOM	22.99
13	ALI FOREVER	21.99
14	ALICE FANTASIA	23.99
16	ALLEY EVOLUTION	23.99
18	ALTER VICTORY	27.99
19	AMADEUS HOLY	20.99
20	AMELIE HELLFIGHTERS	23.99
21	AMERICAN CIRCUS	17.99
24	ANALYZE HOOSIERS	19.99
30	ANYTHING SAVANNAH	27.99
34	ARABIA DOGMA	29.99
35	ARACHNOPHOBIA ROLLERCOASTER	24.99
37	ARIZONA BANG	28.99
38	ARK RIDGEMONT	25.99
40	ARMY FLINTSTONES	22.99
41	ARSENIC INDEPENDENCE	17.99
44	ATTACKS HATE	21.99
78	BLACKOUT PRIVATE	12.99
79	BLADE POLISH	10.99
80	BLANKET BEVERLY	21.99
81	BLINDNESS GUN	29.99
82	BLOOD ARGONAUTS	13.99
83	BLUES INSTINCT	18.99

```
| 93          | BRANNIGAN SUNRISE | 27.99                |
| 94          | BRAVEHEART HUMAN | 14.99                 |
| 95          | BREAKFAST GOLDFINGER | 18.99                 |
| 96          | BREAKING HOME | 21.99                 |
| 97          | BRIDE INTRIGUE | 24.99                 |
| 98          | BRIGHT ENCOUNTERS | 12.99                 |
| 99          | BRINGING HYSTERICAL | 14.99                 |
| 100         | BROOKLYN DESERT | 21.99                 |
| 101         | BROTHERHOOD BLANKET | 26.99                 |
| 102         | BUBBLE GROSSE | 20.99                 |
| 103         | BUCKET BROTHERHOOD | 27.99                 |
| 104         | BUGSY SONG | 17.99                 |
| 105         | BULL SHAWSHANK | 21.99                 |
| 106         | BULWORTH COMMANDMENTS | 14.99                 |
| 107         | BUNCH MINDS | 13.99                 |
| 108         | BUTCH PANTHER | 19.99                 |
| 109         | BUTTERFLY CHOCOLAT | 17.99                 |
+ ----------- + ---------- + --------------------- +
48 rows
```

8) NULL 에 대한 비교 조건문

앞에서 NULL은 속성의 값이라기 보다는 해당 속성의 값이 없거나 알지 못한다는 의미를 나타내는 특수 기호라는 설명을 한 바 있다.

이번에는 속성에 NULL이 기록되어 있는 투플에 대한 질의문을 살펴보기로 하자.

질의 : address 릴레이션에서 address_id가 10보다 작은 투플의 address_id, address, address2를 검색하라.

```
select address_id, address, address2
from address
where address_id < 10;
```

질의 결과

```
> select address_id, address, address2
from address
where address_id < 10
+ --------------- + ------------ + ------------- +
| address_id      | address      | address2      |
+ --------------- + ------------ + ------------- +
| 1               | 47 MySakila Drive |  NULL            |
| 2               | 28 MySQL Boulevard |  NULL             |
| 3               | 23 Workhaven Lane |  NULL            |
| 4               | 1411 Lillydale Drive |  NULL              |
| 5               | 1913 Hanoi Way |               |
| 6               | 1121 Loja Avenue |              |
| 7               | 692 Joliet Street |              |
| 8               | 1566 Inegl Manor |              |
| 9               | 53 Idfu Parkway |              |
+ --------------- + ------------ + ------------- +
9 rows
```

이 질의의 결과를 살펴보면 address_id가 1번부터 4번까지는

address2에 NULL이 기록되어 있는 것을 알 수 있다.

이제 address2에 NULL이 기록되지 않은 투플만을 검색하도록 질의를 해보자.

질의 : address 릴레이션에서 address_id가 10보다 작고 address2에 NULL이 기록되어 있지 않은 투플의 address_id, address, address2를 검색하라.

```
select address_id, address, address2
from address
where address_id < 10
      AND address2 is NOT NULL;
```

질의 결과

```
> select address_id, address, address2
from address
where address_id < 10
     AND address2 is NOT NULL
+ -------------- + ------------ + ------------- +
| address_id     | address      | address2      |
+ -------------- + ------------ + ------------- +
| 5              | 1913 Hanoi Way |              |
| 6              | 1121 Loja Avenue |             |
| 7              | 692 Joliet Street |            |
| 8              | 1566 Inegl Manor |             |
| 9              | 53 Idfu Parkway |              |
+ -------------- + ------------ + ------------- +
5 rows
```

이번에는 address2에 NULL 값이 기록되어 있던 4개의 투플이 사라진 것을 확인할 수 있다.

이 때 SQL 질의문에서 is NOT NULL이라는 키워드를 사용한 것에 주목해야 한다.

NULL을 일반적인 상수값과 같은 방식으로 처리한다면 is NOT NULL 대신에 <> NULL과 같은 표현을 사용할 수 있을 것이다.

MySQL에서 이와 같은 질의문을 사용하고 그 결과를 확인해 보기로 하자.

```
select address_id, address, address2
from address
where address_id < 10
      AND address2 <> NULL;
```

질의 결과

```
> select address_id, address, address2
from address
where address_id < 10
     AND address2 <> NULL
+ --------------- + ------------ + ------------- +
| address_id      | address      | address2      |
+ --------------- + ------------ + ------------- +
+ --------------- + ------------ + ------------- +
0 rows
```

질의 결과 투플이 전혀 검색되지 않은 것을 알 수 있다.

이는 <> NULL 이라는 표현이 부정확한 것이기 때문에 MySQL의 질의 처리기가 수행을 하지 못한 것이다.

마찬가지로 address2에 대해서 NULL 값을 가지는 투플만을 검색하는 경우에도 = NULL이라는 표현을 사용해서는 안되고 is NULL이라는 키워드를 사용해야 한다.

다음의 질의를 살펴보자.

```
select address_id, address, address2
from address
where address_id < 10
      AND address2 is NULL;
```

질의 결과

```
> select address_id, address, address2
from address
where address_id < 10
     AND address2 is NULL
+ --------------- + ------------ + ------------- +
| address_id      | address      | address2      |
+ --------------- + ------------ + ------------- +
| 1               | 47 MySakila Drive | NULL       |
| 2               | 28 MySQL Boulevard | NULL          |
| 3               | 23 Workhaven Lane | NULL         |
| 4               | 1411 Lillydale Drive | NULL          |
+ --------------- + ------------ + ------------- +
4 rows
```

9) 속성의 이름 변경

SQL 질의문을 작성하다보면 경우에 따라 속성의 이름을 변경할 필요성을 느낄 수 있다.

대부분은 속성의 이름이 너무 길어서 편의상 간단하게 줄이기 위해서 사용하지만 때에 질의문 해석 상의 혼란을 피하기 위하여 반드시 이름을 변경해야 하는 경우도 있다.

본 절에서는 속성의 이름을 변경하는 방법에 대하여 살펴보자.

먼저 다음과 같은 일반적인 질의를 생각해 보자.

질의 : category 릴레이션에서 category_id와 name을 모두 출력하라.

```
select category_id, name
from category;
```

질의 결과

```
> select category_id, name
from category
+ ---------------- + --------- +
| category_id      | name      |
+ ---------------- + --------- +
| 1                | Action    |
| 2                | Animation |
| 3                | Children  |
```

```
| 4               | Classics  |
| 5               | Comedy    |
| 6               | Documentary |
| 7               | Drama     |
| 8               | Family    |
| 9               | Foreign   |
| 10              | Games     |
| 11              | Horror    |
| 12              | Music     |
| 13              | New       |
| 14              | Sci-Fi    |
| 15              | Sports    |
| 16              | Travel    |
+ --------------- + --------- +
16 rows
```

질의 결과를 살펴보면 결과 릴레이션의 속성 이름이 그대로 category_id와 name이라고 되어있는 것을 알 수 있다.

이제 결과 릴레이션의 name 속성 이름을 category_name으로 바꿔보기로 하자.

이를 위해서는 AS 키워드를 사용한다.

SELECT 절에서 프로젝션 할 속성을 나열할 때 AS 키워드와 함께 변경 후 이름을 기술하면 결과 릴레이션의 속성 이름이 변경되는 것을 확인 할 수 있다.

질의 : category 릴레이션에서 category_id와 name을 모두 출력하되 name의 속성 이름을 category_name으로 바꾸어 출력하라.

```
select category_id, name as category_name
from category;
```

질의 결과

```
> select category_id, name as category_name
from category
+ ---------------- + ------------------ +
| category_id      | category_name      |
+ ---------------- + ------------------ +
| 1                | Action             |
| 2                | Animation          |
| 3                | Children           |
| 4                | Classics           |
| 5                | Comedy             |
| 6                | Documentary        |
| 7                | Drama              |
| 8                | Family             |
| 9                | Foreign            |
| 10               | Games              |
| 11               | Horror             |
| 12               | Music              |
| 13               | New                |
```

```
| 14              | Sci-Fi           |
| 15              | Sports           |
| 16              | Travel           |
+ --------------- + ---------------- +
16 rows
```

AS 키워드를 사용하지 않고도 결과 릴레이션의 속성 이름을 변경할 수 있다.

위와 동일한 기능을 하는 SQL 질의 문을 다음과 같이 작성할 수 있다.

```
select category_id, name "category_name"
from category;
```

마찬가지 방식으로 속성 이름을 다음과 같이 한글로 표시할 수도 있다.

```
select category_id, name "카탈로그 이름"
from category;
```

MySQL에서 수행했을 때 예상한 대로 결과가 출력되는지는 각자 실행해 보기로 하자.

10) 날짜 속성에 대한 연산

속성에는 여러 가지 데이터 타입이 존재하는 데 그 중 하나가 날짜 타입이다.

날짜 데이터에 대해서도 일반 데이터와 같이 비교 연산을 수행할 수 있다.

날짜 데이터를 상수값으로 표현 할 때에는 작은 따옴표(‘) 안에 연월일을 ’/‘ 또는 ’-‘으로 구분하여 사용한다.

다음과 같은 질의를 생각해 보자.

질의 : payment 릴레이션에서 2006년 1월 1일 이후에 $7 이상의 금액을 지불한 고객의 ID와 rental_id, 그리고 지불한 금액을 검색하라.

```
select customer_id, rental_id, amount, payment_date
from payment
where amount >= 7.0
      and payment_date >= '2006/01/01';
```

질의 결과

```
> select customer_id, rental_id, amount, payment_date
from payment
where amount >= 7.0
      and payment_date >= '2006/01/01'
+ --------------- + -------------- + ----------- + ----------------- +
| customer_id     | rental_id      | amount      | payment_date      |
+ --------------- + -------------- + ----------- + ----------------- +
```

```
| 53          | 11657       | 7.98      | 2006-02-14 15:16:03 |
| 60          | 12489       | 9.98      | 2006-02-14 15:16:03 |
| 75          | 13534       | 8.97      | 2006-02-14 15:16:03 |
| 155         | 11496       | 7.98      | 2006-02-14 15:16:03 |
| 163         | 11754       | 7.98      | 2006-02-14 15:16:03 |
| 267         | 12066       | 7.98      | 2006-02-14 15:16:03 |
| 354         | 12759       | 7.98      | 2006-02-14 15:16:03 |
+ ------------- + ------------ + ---------- + ----------------- +
7 rows
```

질의 결과를 살펴보면 payment date가 2006년 이후인 것을 확인 할 수 있다.

MySQL에서는 날짜 데이터에 대한 여러 가지 함수를 제공하고 있다.

이제 몇 가지 날짜 관련 함수의 사용방법에 대해 알아보기로 하자.

(1) 요일 검색

MySQL에서는 날짜를 입력하면 해당 요일을 출력해 주는 함수 dayofweek()를 제공한다.

dayofweek()는 일요일인 경우 1, 월요일인 경우 2와 같은 방식으로 토요일의 경우 7을 반환한다.

다음의 질의를 생각해보자.

질의 : payment 릴레이션에서 고객 ID가 10인 고객의 customer_id, payment_date, 그리고 payment_date의 요일을 검색하라.

```
select customer_id, payment_date,
       dayofweek(payment_date)
from payment
where customer_id = 10;
```

질의 결과

```
> select distinct customer_id, payment_date, dayofweek(payment_date)
from payment
where customer_id = 10
+ ---------------- + ----------------- + ---------------------------- +
| customer_id      | payment_date      | dayofweek(payment_date)      |
+ ---------------- + ----------------- + ---------------------------- +
| 10               | 2005-05-31 19:36:30 | 3                            |
| 10               | 2005-06-16 20:21:53 | 5                            |
| 10               | 2005-06-17 11:11:14 | 6                            |
| 10               | 2005-06-18 03:26:23 | 7                            |
| 10               | 2005-06-19 20:01:59 | 1                            |
| 10               | 2005-06-20 00:00:55 | 2                            |
| 10               | 2005-07-06 14:13:45 | 4                            |
| 10               | 2005-07-07 03:06:40 | 5                            |
| 10               | 2005-07-07 14:14:13 | 5                            |
| 10               | 2005-07-09 03:12:52 | 7                            |
```

```
| 10              | 2005-07-09 04:53:18 | 7                          |
| 10              | 2005-07-09 21:58:57 | 7                          |
| 10              | 2005-07-10 20:41:09 | 1                          |
| 10              | 2005-07-28 05:21:42 | 5                          |
| 10              | 2005-07-28 15:10:55 | 5                          |
| 10              | 2005-07-28 22:34:12 | 5                          |
| 10              | 2005-07-31 15:27:07 | 1                          |
| 10              | 2005-08-01 17:09:59 | 2                          |
| 10              | 2005-08-02 14:55:00 | 3                          |
| 10              | 2005-08-02 19:13:39 | 3                          |
| 10              | 2005-08-17 20:11:35 | 4                          |
| 10              | 2005-08-18 09:19:12 | 5                          |
| 10              | 2005-08-19 19:23:30 | 6                          |
| 10              | 2005-08-20 16:43:28 | 7                          |
| 10              | 2005-08-22 21:59:29 | 2                          |
+ --------------- + ---------------- + -------------------------- +
25 rows
```

(2) 날짜 데이터의 일자 구하기

MySQL에서 연/월/일로 구성된 날짜 데이터에서 해당 월 중 일자만을 얻는 함수로 dayofmonth()를 제공하고 있다.

다음의 질의를 생각해보자.

질의 : payment 릴레이션에서 고객 ID가 10인 고객의 payment_id, payment_date, 그리고 payment_date의 월 중 일자를 검색하라.

```
select payment_id, payment_date,
       dayofmonth(payment_date)
from payment
where customer_id = 10;
```

질의 결과

```
> select payment_id, payment_date, dayofmonth(payment_date)
from payment
where customer_id = 10
+ --------------- + ----------------- + ------------------------------ +
| payment_id      | payment_date      | dayofmonth(payment_date)       |
+ --------------- + ----------------- + ------------------------------ +
| 254             | 2005-05-31 19:36:30 | 31                           |
| 255             | 2005-06-16 20:21:53 | 16                           |
| 256             | 2005-06-17 11:11:14 | 17                           |
| 257             | 2005-06-18 03:26:23 | 18                           |
| 258             | 2005-06-19 20:01:59 | 19                           |
| 259             | 2005-06-20 00:00:55 | 20                           |
| 260             | 2005-07-06 14:13:45 | 6                            |
| 261             | 2005-07-07 03:06:40 | 7                            |
| 262             | 2005-07-07 14:14:13 | 7                            |
| 263             | 2005-07-09 03:12:52 | 9                            |
| 264             | 2005-07-09 04:53:18 | 9                            |
```

```
| 265            | 2005-07-09 21:58:57 | 9                          |
| 266            | 2005-07-10 20:41:09 | 10                         |
| 267            | 2005-07-28 05:21:42 | 28                         |
| 268            | 2005-07-28 15:10:55 | 28                         |
| 269            | 2005-07-28 22:34:12 | 28                         |
| 270            | 2005-07-31 15:27:07 | 31                         |
| 271            | 2005-08-01 17:09:59 | 1                          |
| 272            | 2005-08-02 14:55:00 | 2                          |
| 273            | 2005-08-02 19:13:39 | 2                          |
| 274            | 2005-08-17 20:11:35 | 17                         |
| 275            | 2005-08-18 09:19:12 | 18                         |
| 276            | 2005-08-19 19:23:30 | 19                         |
| 277            | 2005-08-20 16:43:28 | 20                         |
| 278            | 2005-08-22 21:59:29 | 22                         |
+ -------------- + ------------------ + ---------------------------- +
25 rows
```

비슷한 함수로 주어진 날짜 데이터의 연중 일자를 구하는 함수 dayofyear()도 제공한다.

다음의 연산을 생각해보자.

질의 : payment 릴레이션에서 고객 ID가 10인 고객의 payment_id, payment_date, 그리고 payment_date의 연 중 일자를 검색하라.

```
select payment_id, payment_date,
       dayofyear(payment_date)
from payment
where customer_id = 10;
```

질의 결과

```
> select payment_id, payment_date, dayofyear(payment_date)
from payment
where customer_id = 10
+ --------------- + ----------------- + ---------------------------- +
| payment_id      | payment_date      | dayofyear(payment_date)      |
+ --------------- + ----------------- + ---------------------------- +
| 254             | 2005-05-31 19:36:30 | 151                          |
| 255             | 2005-06-16 20:21:53 | 167                          |
| 256             | 2005-06-17 11:11:14 | 168                          |
| 257             | 2005-06-18 03:26:23 | 169                          |
| 258             | 2005-06-19 20:01:59 | 170                          |
| 259             | 2005-06-20 00:00:55 | 171                          |
| 260             | 2005-07-06 14:13:45 | 187                          |
| 261             | 2005-07-07 03:06:40 | 188                          |
| 262             | 2005-07-07 14:14:13 | 188                          |
| 263             | 2005-07-09 03:12:52 | 190                          |
| 264             | 2005-07-09 04:53:18 | 190                          |
| 265             | 2005-07-09 21:58:57 | 190                          |
| 266             | 2005-07-10 20:41:09 | 191                          |
| 267             | 2005-07-28 05:21:42 | 209                          |
| 268             | 2005-07-28 15:10:55 | 209                          |
```

```
| 269            | 2005-07-28 22:34:12 | 209                          |
| 270            | 2005-07-31 15:27:07 | 212                          |
| 271            | 2005-08-01 17:09:59 | 213                          |
| 272            | 2005-08-02 14:55:00 | 214                          |
| 273            | 2005-08-02 19:13:39 | 214                          |
| 274            | 2005-08-17 20:11:35 | 229                          |
| 275            | 2005-08-18 09:19:12 | 230                          |
| 276            | 2005-08-19 19:23:30 | 231                          |
| 277            | 2005-08-20 16:43:28 | 232                          |
| 278            | 2005-08-22 21:59:29 | 234                          |
+ -------------- + ------------------ + ---------------------------- +
25 rows
```

유사한 방식으로 month(), year(), hour() 등의 함수도 사용할 수 있다.

각각의 함수는 사용 사례는 지면 관계상 생략하기로 한다.

(3) 날짜 데이터에 대한 산술 연산

MySQL에서 날짜 데이터에 더하기, 빼기 등의 산술 연산을 가능하도록 하는 기능을 제공하고 있다.

이 기능을 하는 함수는 DATE_ADD()와 DATE_SUB()이다.

이 두 함수의 사용 형식은 다음과 같다.

DATE_ADD(날짜, INTERVAL 숫자 type)

DATE_SUB(날짜, INTERVAL 숫자 type)

이 때, type으로는 다음과 같은 것들이 있다.

SECOND(초), MINUTE(분), HOUR(시), DAY(일), MONTH(월), YEAR(년) 등

이제 이 함수를 사용하는 다음의 질의를 생각해보자.

질의 : payment 릴레이션에서 고객 ID가 10인 고객의 payment_id, payment_date, 그리고 payment_date에 1개월을 더한 날짜 값을 검색하라.

```
select payment_id, payment_date,
       date_add(payment_date, interval 1 month)
       as add_1_month
from payment
where customer_id = 10;
```

질의 결과

```
> select payment_id, payment_date, date_add(payment_date, interval 1 month) as
add_1_month
from payment
where customer_id = 10
+ --------------- + ----------------- + ---------------- +
| payment_id      | payment_date      | add_1_month      |
+ --------------- + ----------------- + ---------------- +
| 254             | 2005-05-31 19:36:30 | 2005-06-30 19:36:30 |
| 255             | 2005-06-16 20:21:53 | 2005-07-16 20:21:53 |
| 256             | 2005-06-17 11:11:14 | 2005-07-17 11:11:14 |
```

```
| 257             | 2005-06-18 03:26:23 | 2005-07-18 03:26:23 |
| 258             | 2005-06-19 20:01:59 | 2005-07-19 20:01:59 |
| 259             | 2005-06-20 00:00:55 | 2005-07-20 00:00:55 |
| 260             | 2005-07-06 14:13:45 | 2005-08-06 14:13:45 |
| 261             | 2005-07-07 03:06:40 | 2005-08-07 03:06:40 |
| 262             | 2005-07-07 14:14:13 | 2005-08-07 14:14:13 |
| 263             | 2005-07-09 03:12:52 | 2005-08-09 03:12:52 |
| 264             | 2005-07-09 04:53:18 | 2005-08-09 04:53:18 |
| 265             | 2005-07-09 21:58:57 | 2005-08-09 21:58:57 |
| 266             | 2005-07-10 20:41:09 | 2005-08-10 20:41:09 |
| 267             | 2005-07-28 05:21:42 | 2005-08-28 05:21:42 |
| 268             | 2005-07-28 15:10:55 | 2005-08-28 15:10:55 |
| 269             | 2005-07-28 22:34:12 | 2005-08-28 22:34:12 |
| 270             | 2005-07-31 15:27:07 | 2005-08-31 15:27:07 |
| 271             | 2005-08-01 17:09:59 | 2005-09-01 17:09:59 |
| 272             | 2005-08-02 14:55:00 | 2005-09-02 14:55:00 |
| 273             | 2005-08-02 19:13:39 | 2005-09-02 19:13:39 |
| 274             | 2005-08-17 20:11:35 | 2005-09-17 20:11:35 |
| 275             | 2005-08-18 09:19:12 | 2005-09-18 09:19:12 |
| 276             | 2005-08-19 19:23:30 | 2005-09-19 19:23:30 |
| 277             | 2005-08-20 16:43:28 | 2005-09-20 16:43:28 |
| 278             | 2005-08-22 21:59:29 | 2005-09-22 21:59:29 |
+ --------------- + ------------------ + ---------------- +
25 rows
```

11) 범위 연산자의 사용

범위 연산자는 특정 속성의 값이 지정한 범위 내에 있을 때에 True를 반환하는 연산자를 의미한다.

SQL에서 범위 연산을 하는 방법은 크게 두가지가 있다.

첫 번째 방법은 비교 연산자와 논리 연산자를 결합하여 사용하는 방법이다.

다음과 같은 질의를 생각해 보자.

질의 : payment 릴레이션에서 고객 ID가 10보다 크고 20보다 작은 사람들 중에 $6 이상 $7 이하의 금액을 지불한 고객의 ID와 rental_id, 그리고 지불한 금액을 검색하라.

```
select customer_id, rental_id, amount
from payment
where amount >= 6.0 and amount <= 7.0
      and customer_id  > 10 and customer_id < 20;
```

질의 결과

```
> select customer_id, rental_id, amount
from payment
where amount >= 6.0 and amount <= 7.0
      and customer_id  > 10 and customer_id < 20
+ ---------------- + -------------- + ----------- +
| customer_id      | rental_id      | amount      |
+ ---------------- + -------------- + ----------- +
```

```
| 11              | 987           | 6.99       |
| 11              | 1470          | 6.99       |
| 11              | 6146          | 6.99       |
| 11              | 9292          | 6.99       |
| 11              | 11166         | 6.99       |
| 14              | 8035          | 6.99       |
| 14              | 10526         | 6.99       |
| 14              | 15015         | 6.99       |
| 15              | 8615          | 6.99       |
| 16              | 1934          | 6.99       |
| 16              | 13480         | 6.99       |
| 19              | 179           | 6.99       |
+ --------------- + ------------- + ---------- +
12 rows
```

동일한 질의를 다음과 같이 Between 키워드를 사용하여 작성할 수도 있다.

```
select customer_id, rental_id, amount
from payment
where amount between 6.0 and 7.0
      and customer_id  > 10 and customer_id < 20;
```

질의 결과

```
> select customer_id, rental_id, amount
```

```
from payment
where amount between 6.0 and 7.0
      and customer_id  > 10 and customer_id < 20
+ --------------- + -------------- + ----------- +
| customer_id     | rental_id      | amount      |
+ --------------- + -------------- + ----------- +
| 11              | 987            | 6.99        |
| 11              | 1470           | 6.99        |
| 11              | 6146           | 6.99        |
| 11              | 9292           | 6.99        |
| 11              | 11166          | 6.99        |
| 14              | 8035           | 6.99        |
| 14              | 10526          | 6.99        |
| 14              | 15015          | 6.99        |
| 15              | 8615           | 6.99        |
| 16              | 1934           | 6.99        |
| 16              | 13480          | 6.99        |
| 19              | 179            | 6.99        |
+ --------------- + -------------- + ----------- +
12 rows
```

질의 결과를 살펴 보면 두 개의 질의문이 동일한 결과를 검색하는 것을 알 수 있다.

BETWEEN A AND B 연산은 특정 속성의 값이 A 이상이고 B 이하인 경우에 True를 반환한다.

그러므로 본 질의에서 customer_id에 대하여 between 연산을 사용하면 잘 못된 결과를 얻게 된다.

다음의 질의 결과를 살펴보자.

질의 결과에 customer_id가 10인 고객과 20인 고객도 포함되어 있는 것을 알 수 있다.

```
select customer_id, rental_id, amount
from payment
where amount between 6.0 and 7.0
      and customer_id  between 10 and  20;
```

질의 결과

```
> select customer_id, rental_id, amount
from payment
where amount between 6.0 and 7.0
      and customer_id  between 10 and  20
+ ---------------- + -------------- + ----------- +
| customer_id      | rental_id      | amount      |
+ ---------------- + -------------- + ----------- +
| 10               | 8001           | 6.99        |
| 11               | 987            | 6.99        |
| 11               | 1470           | 6.99        |
| 11               | 6146           | 6.99        |
| 11               | 9292           | 6.99        |
| 11               | 11166          | 6.99        |
| 14               | 8035           | 6.99        |
```

```
| 14              | 10526          | 6.99        |
| 14              | 15015          | 6.99        |
| 15              | 8615           | 6.99        |
| 16              | 1934           | 6.99        |
| 16              | 13480          | 6.99        |
| 19              | 179            | 6.99        |
| 20              | 497            | 6.99        |
| 20              | 6267           | 6.99        |
| 20              | 15057          | 6.99        |
+ --------------- + -------------- + ----------- +
16 rows
```

Between 연산자와 NOT 연산자를 조합하면, 특정 값 미만이거나 특정 값을 초과하는 투플들도 검색할 수 있다.

다음과 같은 질의를 살펴보자.

질의 : payment 릴레이션에서 고객 ID가 10인 고객이 $3 미만 또는 $7 초과의 금액을 지불한 고객의 ID와 rental_id, 그리고 지불한 금액을 검색하라.

```
select customer_id, rental_id, amount
from payment
where amount NOT between 3.0 and 7.0
      and customer_id =10;
```

질의 결과

```
> select customer_id, rental_id, amount
from payment
where amount NOT between 3.0 and 7.0
      and customer_id =10
+ ---------------- + -------------- + ----------- +
| customer_id      | rental_id      | amount      |
+ ---------------- + -------------- + ----------- +
| 10               | 2814           | 0.99        |
| 10               | 2865           | 0.99        |
| 10               | 4255           | 1.99        |
| 10               | 5038           | 7.99        |
| 10               | 5068           | 2.99        |
| 10               | 5444           | 0.99        |
| 10               | 5905           | 2.99        |
| 10               | 7738           | 2.99        |
| 10               | 10671          | 8.99        |
| 10               | 11289          | 2.99        |
| 10               | 11405          | 0.99        |
| 10               | 12031          | 2.99        |
| 10               | 12400          | 2.99        |
| 10               | 13917          | 2.99        |
+ ---------------- + -------------- + ----------- +
14 rows
```

문자열 타입의 속성에 대해서도 범위 연산이 가능하다.

문자열 타입의 속성에 대해서는 알파벳 순서에 따라 순서가 부여되어 범위 연산을 수행하게 된다.

다음과 같은 질의를 생각해 보자.

질의 : 고객 중에서 이름이 'an'과 'at' 사이에 있는 고객의 customer_id, first_name, last_name을 검색하라.

```
select customer_id, first_name, last_name
from customer
where first_name between 'an' and 'at';
```

질의 결과

```
> select customer_id, first_name, last_name
from customer
where first_name between 'an' and 'at'
+ ---------------- + --------------- + -------------- +
| customer_id      | first_name      | last_name      |
+ ---------------- + --------------- + -------------- +
| 29               | ANGELA          | HERNANDEZ      |
| 33               | ANNA            | HILL           |
| 48               | ANN             | EVANS          |
| 63               | ASHLEY          | RICHARDSON     |
| 81               | ANDREA          | HENDERSON      |
| 85               | ANNE            | POWELL         |
| 97               | ANNIE           | RUSSELL        |
| 136              | ANITA           | MORALES        |
| 142              | APRIL           | BURNS          |
```

```
| 175            | ANNETTE        | OLSON          |
| 181            | ANA            | BRADLEY        |
| 225            | ARLENE         | HARVEY         |
| 320            | ANTHONY        | SCHWAB         |
| 333            | ANDREW         | PURDY          |
| 346            | ARTHUR         | SIMPKINS       |
| 398            | ANTONIO        | MEEK           |
| 503            | ANGEL          | BARCLAY        |
| 515            | ANDRE          | RAPP           |
| 522            | ARNOLD         | HAVENS         |
| 556            | ARMANDO        | GRUBER         |
| 582            | ANDY           | VANHORN        |
+ ---------------- + --------------- + -------------- +
21 rows
```

질의 결과에서 이름이 'AN'으로 시작하는 고객부터 'AS'로 시작하는 고객들이 모두 검색되는 것을 확인할 수 있다.

이름이 'AT'로 시작하는 고객들은 'AT' 보다는 큰 값의 이름을 가진 것으로 간주되므로 검색되지 않는다.

12) IN 연산자의 사용

IN 연산자는 특정 속성의 값이 주어진 몇 개의 값 중 하나인지 여부를 검사할 때에 사용하는 연산자이다.

예를 들어 다음과 같은 질의문을 생각해 보자.

질의 : payment 릴레이션에서 2006년 1월 1일 이후에 지불 금액이 $1.98 또는 $7.98 또는 $9.98인 고객의 고객 ID와 지불 금액, 그리고 지불 날짜를 검색하라.

```
select customer_id, amount, payment_date
from payment
where (payment_date > '2006/01/01')
      AND (amount = 1.98
           OR amount = 7.98
           OR amount = 9.98);
```

질의 결과

```
> select customer_id, amount, payment_date
from payment
where (payment_date > '2006/01/01')
      AND (amount = 1.98
           OR amount = 7.98
           OR amount = 9.98)
+ ---------------- + ----------- + ----------------- +
| customer_id      | amount      | payment_date      |
+ ---------------- + ----------- + ----------------- +
| 53               | 7.98        | 2006-02-14 15:16:03 |
| 60               | 9.98        | 2006-02-14 15:16:03 |
| 107              | 1.98        | 2006-02-14 15:16:03 |
| 155              | 7.98        | 2006-02-14 15:16:03 |
| 163              | 7.98        | 2006-02-14 15:16:03 |
| 267              | 7.98        | 2006-02-14 15:16:03 |
```

```
| 354             | 7.98        | 2006-02-14 15:16:03 |
+ --------------- + ----------- + ------------------ +
7 rows
```

이 질의문은 여러 개의 OR 연산자를 나열해야 하는 번거로움이 있다. 이러한 경우 다음과 같이 IN 연산자를 사용하여 손쉽게 질의문을 작성할 수 있다.

질의 결과 동일한 결과 값을 얻는 것을 확인 할 수 있다.

```
select customer_id, amount, payment_date
from payment
where (payment_date > '2006/01/01')
      AND amount IN (1.98, 7.98, 9.98);
```

질의 결과

```
> select customer_id, amount, payment_date
from payment
where (payment_date > '2006/01/01')
      AND amount IN (1.98, 7.98, 9.98)
+ --------------- + ----------- + ------------------ +
| customer_id     | amount      | payment_date       |
+ --------------- + ----------- + ------------------ +
| 53              | 7.98        | 2006-02-14 15:16:03 |
| 60              | 9.98        | 2006-02-14 15:16:03 |
```

```
| 107             | 1.98        | 2006-02-14 15:16:03 |
| 155             | 7.98        | 2006-02-14 15:16:03 |
| 163             | 7.98        | 2006-02-14 15:16:03 |
| 267             | 7.98        | 2006-02-14 15:16:03 |
| 354             | 7.98        | 2006-02-14 15:16:03 |
+ ---------------- + ------------ + ------------------ +
7 rows
```

IN 연산자의 경우에도 NOT 연산자와 결합하여 사용할 수 있다.

다음의 예를 살펴보자.

질의 : payment 릴레이션에서 2006년 1월 1일 이후에 지불 금액이 $0.00, $0.99, $1.98, $2.99, $4.99, $7.98, $9.98이 아닌 고객의 고객 ID와 지불 금액, 그리고 지불 날짜를 검색하라.

```
select customer_id, amount, payment_date
from payment
where (payment_date > '2006/01/01')
      AND amount NOT IN
      (0.00, 0.99, 1.98, 2.99, 4.99, 7.98, 9.98);
```

질의 결과

```
> select customer_id, amount, payment_date
from payment
where (payment_date > '2006/01/01')
```

```
      AND amount NOT IN
      (0.00, 0.99, 1.98, 2.99, 4.99, 7.98, 9.98)
+ ---------------- + ----------- + ----------------- +
| customer_id      | amount      | payment_date      |
+ ---------------- + ----------- + ----------------- +
| 15               | 3.98        | 2006-02-14 15:16:03 |
| 42               | 5.98        | 2006-02-14 15:16:03 |
| 43               | 3.98        | 2006-02-14 15:16:03 |
| 75               | 8.97        | 2006-02-14 15:16:03 |
| 175              | 3.98        | 2006-02-14 15:16:03 |
| 208              | 5.98        | 2006-02-14 15:16:03 |
| 216              | 5.98        | 2006-02-14 15:16:03 |
| 228              | 3.98        | 2006-02-14 15:16:03 |
| 269              | 3.98        | 2006-02-14 15:16:03 |
| 284              | 5.98        | 2006-02-14 15:16:03 |
| 361              | 3.98        | 2006-02-14 15:16:03 |
| 448              | 3.98        | 2006-02-14 15:16:03 |
| 457              | 3.98        | 2006-02-14 15:16:03 |
| 516              | 5.98        | 2006-02-14 15:16:03 |
| 560              | 5.98        | 2006-02-14 15:16:03 |
| 576              | 5.98        | 2006-02-14 15:16:03 |
+ ---------------- + ----------- + ----------------- +
16 rows
```

나) 조인을 사용한 SQL 질의문

관계형 데이터 모델에서는 데이터가 여러 릴레이션이 분산되어 있는 경우가 많다.

왜냐하면 데이터 중복으로 인한 일관성 저하 및 이상 현상 발생을 줄이기 위하여 정규화를 수행하는데 이 때 릴레이션이 여러개로 분할되기 때문이다.

그러므로 필요한 경우 조인 연산을 사용하여 분할된 릴레이션을 하나의 릴레이션으로 병합하는 작업을 수행하여야 한다.

본 절에서는 SQL 질의문에서 조인을 사용하여 검색을 수행하는 방법에 대하여 설명한다.

1) 카테시안 프로덕트

카테시안 프로덕트는 두 릴레이션에 속한 투플들 간의 모든 가능한 조합을 표현한 것이다.

SQL에서는 From 절에 카테시안 프로덕트를 수행할 두 릴레이션을 기술하는 것만으로 카테시안 프로덕트가 수행된다.

다음의 질의를 생각해 보자.

본 질의 결과는 검색되는 투플이 총 65400개로 너무 많으므로 지면 관계 상 일부만 표시하기로 한다.

질의 : city 릴레이션과 country 릴레이션의 카테시안 프로덕트를 구하라.

select *

from city, country;

질의 결과

```
> select *
from city, country
+ ----------- + --------- + -------------- + ---------------- + -------------- + ------------ + ---------------- +
| city_id     | city      | country_id     | last_update      | country_id     | country      | last_update      |
+ ----------- + --------- + -------------- + ---------------- + -------------- + ------------ + ---------------- +
| 1           | A Corua (La Corua) | 87          | 2006-02-15 04:45:25 | 1            | Afghanistan  | 2006-02-15 04:44:00 |
| 1           | A Corua (La Corua) | 87          | 2006-02-15 04:45:25 | 2            | Algeria      | 2006-02-15 04:44:00 |
| 1           | A Corua (La Corua) | 87          | 2006-02-15 04:45:25 | 3            | American Samoa | 2006-02-15 04:44:00 |
| 1           | A Corua (La Corua) | 87          | 2006-02-15 04:45:25 | 4            | Angola       | 2006-02-15 04:44:00 |
| 1           | A Corua (La Corua) | 87          | 2006-02-15 04:45:25 | 5            | Anguilla     | 2006-02-15 04:44:00 |
| 1           | A Corua (La Corua) | 87          | 2006-02-15 04:45:25 | 6            | Argentina    | 2006-02-15 04:44:00 |
| 1           | A Corua (La Corua) | 87          | 2006-02-15 04:45:25 | 7            | Armenia      | 2006-02-15 04:44:00 |
| 1           | A Corua (La Corua) | 87          | 2006-02-15 04:45:25 | 8            | Australia    | 2006-02-15 04:44:00 |
```

```
| 1          | A Corua (La Corua) | 87          | 2006-02-15 04:45:25 | 9          | Austria      | 2006-02-15 04:44:00 |
| 1          | A Corua (La Corua) | 87          | 2006-02-15 04:45:25 | 10         | Azerbaijan   | 2006-02-15 04:44:00 |
| 1          | A Corua (La Corua) | 87          | 2006-02-15 04:45:25 | 11         | Bahrain      | 2006-02-15 04:44:00 |
| 1          | A Corua (La Corua) | 87          | 2006-02-15 04:45:25 | 12         | Bangladesh   | 2006-02-15 04:44:00 |
| 1          | A Corua (La Corua) | 87          | 2006-02-15 04:45:25 | 13         | Belarus      | 2006-02-15 04:44:00 |
| 1          | A Corua (La Corua) | 87          | 2006-02-15 04:45:25 | 14         | Bolivia      | 2006-02-15 04:44:00 |
| 1          | A Corua (La Corua) | 87          | 2006-02-15 04:45:25 | 15         | Brazil       | 2006-02-15 04:44:00 |
| 1          | A Corua (La Corua) | 87          | 2006-02-15 04:45:25 | 16         | Brunei       | 2006-02-15 04:44:00 |
| 1          | A Corua (La Corua) | 87          | 2006-02-15 04:45:25 | 17         | Bulgaria     | 2006-02-15 04:44:00 |
| 1          | A Corua (La Corua) | 87          | 2006-02-15 04:45:25 | 18         | Cambodia     | 2006-02-15 04:44:00 |
| 1          | A Corua (La Corua) | 87          | 2006-02-15 04:45:25 | 19         | Cameroon     | 2006-02-15 04:44:00 |
| 1          | A Corua (La Corua) | 87          | 2006-02-15 04:45:25 | 20         | Canada       | 2006-02-15 04:44:00 |
| 1          | A Corua (La Corua) | 87          | 2006-02-15 04:45:25 | 21         | Chad         | 2006-02-15 04:44:00 |
| 1          | A Corua (La Corua) | 87          | 2006-02-15 04:45:25 | 22         | Chile        | 2006-02-15 04:44:00 |
| 1          | A Corua (La Corua) | 87          | 2006-02-15 04:45:25 | 23         | China        | 2006-02-15 04:44:00
....
```

2) 동등 조인

동등 조인은 가장 일반적인 조인 방법으로서 조인에 참여하는 두 릴레이션의 특정 속성 값이 동일한 투플을 연결하여 검색하는 방법이다.

이를 위해서는 조인에 참여하는 두 릴레이션에 동일한 의미를 가지는 속성이 존재하여야 한다.

동등 조인은 대개의 경우 외래키로 설정되어 참조 관계를 가지는 두 릴레이션을 하나의 릴레이션으로 병합하여 원하는 데이터를 검색할 때에 주로 사용된다.

또한 정규화로 인하여 분할된 두 릴레이션을 하나의 릴레이션으로 복원할 수도 있다.

다음과 같은 질의를 생각해 보자.

질의 : city 릴레이션과 country 릴레이션을 동등 조인하라.

이 질의는 앞서 수행했던 city 릴레이션과 country 릴레이션의 카테시안 프로덕트에서 양쪽 country_id 값이 동일한 투플만을 추출한 것이 된다.

이렇게 하면 각 도시의 정보와 그 도시가 속한 국가의 정보가 하나의 투플에 표시된다.

본 예제에서는 질의 결과 투플이 600개로서 너무 많으므로 지면 관계상 일부만 표현하기로 한다.

```
select *
from city, country
where city.country_id = country.country_id;
```

질의 결과

```
> select *
from city, country
where city.country_id = country.country_id
+ ---------- + --------- + ------------ + ---------------- + ------------ + ---------- + ---------------- +
| city_id    | city      | country_id   | last_update      | country_id   | country    | last_update      |
+ ---------- + --------- + ------------ + ---------------- + ------------ + ---------- + ---------------- +
| 251        | Kabul     | 1            | 2006-02-15 04:45:25 | 1            | Afghanistan  | 2006-02-15 04:44:00 |
| 59         | Batna     | 2            | 2006-02-15 04:45:25 | 2            | Algeria      | 2006-02-15 04:44:00 |
| 63         | Bchar     | 2            | 2006-02-15 04:45:25 | 2            | Algeria      | 2006-02-15 04:44:00 |
| 483        | Skikda    | 2            | 2006-02-15 04:45:25 | 2            | Algeria      | 2006-02-15 04:44:00 |
| 516        | Tafuna    | 3            | 2006-02-15 04:45:25 | 3            | American Samoa | 2006-02-15 04:44:00 |
| 67         | Benguela  | 4            | 2006-02-15 04:45:25 | 4            | Angola       | 2006-02-15 04:44:00 |
```

```
| 360         | Namibe    | 4           | 2006-02-15 04:45:25 | 4           | Angola      | 2006-02-15 04:44:00 |
| 493         | South Hill | 5          | 2006-02-15 04:45:25 | 5           | Anguilla    | 2006-02-15 04:44:00 |
| 20          | Almirante Brown | 6     | 2006-02-15 04:45:25 | 6           | Argentina   | 2006-02-15 04:44:00 |
| 43          | Avellaneda | 6          | 2006-02-15 04:45:25 | 6           | Argentina   | 2006-02-15 04:44:00 |
| 45          | Baha Blanca | 6         | 2006-02-15 04:45:25 | 6           | Argentina   | 2006-02-15 04:44:00 |
| 128         | Crdoba    | 6           | 2006-02-15 04:45:25 | 6           | Argentina   | 2006-02-15 04:44:00 |
| 161         | Escobar   | 6           | 2006-02-15 04:45:25 | 6           | Argentina   | 2006-02-15 04:44:00 |
| 165         | Ezeiza    | 6           | 2006-02-15 04:45:25 | 6           | Argentina   | 2006-02-15 04:44:00 |
| 289         | La Plata  | 6           | 2006-02-15 04:45:25 | 6           | Argentina   | 2006-02-15 04:44:00 |
| 334         | Merlo     | 6           | 2006-02-15 04:45:25 | 6           | Argentina   | 2006-02-15 04:44:00 |
| 424         | Quilmes   | 6           | 2006-02-15 04:45:25 | 6           | Argentina   | 2006-02-15 04:44:00 |
| 454         | San Miguel de Tucumn | 6 | 2006-02-15 04:45:25 | 6           | Argentina   | 2006-02-15 04:44:00 |
| 457         | Santa F   | 6           | 2006-02-15 04:45:25 | 6           | Argentina   | 2006-02-15 04:44:00 |
| 524         | Tandil    | 6           | 2006-02-15 04:45:25 | 6           | Argentina   | 2006-02-15 04:44:00 |
....
```

동등 조인을 수행한 이후 원하는 속성만을 프로젝션 하는 것도 가능하다.

이때는 하나의 릴레이션에서 프로젝션을 수행하듯이 SELECT 절에 프로젝션 할 속성의 이름을 나열하면 된다.

단, 동일한 이름의 속성이 두 릴레이션에 모두 존재하는 경우에는 혼란을 피하기 위하여 속성 이름 앞에 릴레이션 이름과 '.'을 기술한다.

다음의 질의를 살펴보자.

이 경우에도 결과 릴레이션의 투플 개수가 600개이므로 지면 관계상 일부만 소개하기로 한다.

질의 : 도시 이름과 도시가 속한 국가의 이름, 그리고 country 릴레이션의 last_update를 검색하라.

```
select city, country, country.last_update
from city, country
where city.country_id = country.country_id;
```

질의 결과

```
> select city, country, country.last_update
from city, country
where city.country_id = country.country_id
+ --------- + ------------ + ---------------- +
| city      | country      | last_update      |
+ --------- + ------------ + ---------------- +
| Kabul     | Afghanistan  | 2006-02-15 04:44:00 |
```

```
| Batna     | Algeria      | 2006-02-15 04:44:00 |
| Bchar     | Algeria      | 2006-02-15 04:44:00 |
| Skikda    | Algeria      | 2006-02-15 04:44:00 |
| Tafuna    | American Samoa | 2006-02-15 04:44:00 |
| Benguela  | Angola       | 2006-02-15 04:44:00 |
| Namibe    | Angola       | 2006-02-15 04:44:00 |
| South Hill | Anguilla     | 2006-02-15 04:44:00 |
| Almirante Brown | Argentina    | 2006-02-15 04:44:00 |
| Avellaneda | Argentina    | 2006-02-15 04:44:00 |
| Baha Blanca | Argentina    | 2006-02-15 04:44:00 |
| Crdoba    | Argentina    | 2006-02-15 04:44:00 |
| Escobar   | Argentina    | 2006-02-15 04:44:00 |
| Ezeiza    | Argentina    | 2006-02-15 04:44:00 |
| La Plata  | Argentina    | 2006-02-15 04:44:00 |
| Merlo     | Argentina    | 2006-02-15 04:44:00 |
| Quilmes   | Argentina    | 2006-02-15 04:44:00 |
| San Miguel de Tucumn | Argentina    | 2006-02-15 04:44:00 |
| Santa F   | Argentina    | 2006-02-15 04:44:00 |
| Tandil    | Argentina    | 2006-02-15 04:44:00 |
| Vicente Lpez | Argentina    | 2006-02-15 04:44:00 |
| Yerevan   | Armenia      | 2006-02-15 04:44:00 |
| Woodridge | Australia    | 2006-02-15 04:44:00 |
...
```

마찬가지로 동등 조인을 수행한 이후 조건을 만족하는 투플만을 셀렉션 하는 것도 가능하다.

이때는 하나의 릴레이션에서 셀렉션을 수행하듯이 WHERE 절에 필요한 조건을 조인 조건과 AND로 연결하여 나열하면 된다.

다음의 질의를 생각해 보자.

질의 : Argentina에 속한 도시 이름과 국가의 이름을 검색하라.

```
select city, country
from city, country
where city.country_id = country.country_id
        AND country = 'Argentina';
```

질의 결과

```
> select city, country
from city, country
where city.country_id = country.country_id
      AND country = 'Argentina'
+ --------- + ------------ +
| city      | country      |
+ --------- + ------------ +
| Almirante Brown | Argentina    |
| Avellaneda | Argentina    |
| Baha Blanca | Argentina    |
| Crdoba    | Argentina    |
| Escobar   | Argentina    |
| Ezeiza    | Argentina    |
| La Plata  | Argentina    |
| Merlo     | Argentina    |
| Quilmes   | Argentina    |
```

```
| San Miguel de Tucumn | Argentina    |
| Santa F   | Argentina    |
| Tandil    | Argentina    |
| Vicente Lpez | Argentina    |
+ --------- + ---------- +
13 rows
```

실수로 조인 조건을 기술하지 않는 경우에 대하여 생각해보자.

다음과 같은 SQL 질의문을 생각해보자.

```
select city, country
from city, country, language
where city.country_id = country.country_id
        AND country = 'Argentina';
```

이 질의문의 위의 SQL 질의문에서 FROM 절에 language 릴레이션을 추가한 것이다.

이 때, language 릴레이션은 city 또는 country 릴레이션과 조인 연산을 수행하는 조건이 WHERE 절에 기술되어 있지 않다.

그러므로 city 릴레이션과 country 릴레이션을 조인한 결과 릴레이션과 language 릴레이션을 카테시안 프로덕트한 것이 결과가 된다.

그 결과는 다음과 같다.

결과적으로 위의 질의 결과로 나타난 13개 투플과 language 릴레이션에 속한 6개 투플의 카테시안 프로덕트로 총 78개 투플이 검색된다.

질의 결과

```
> select city, country

from city, country, language

where city.country_id = country.country_id

       AND country = 'Argentina'

+ --------- + ------------ +

| city      | country      |

+ --------- + ------------ +

| Almirante Brown | Argentina    |

| Avellaneda | Argentina    |

| Baha Blanca | Argentina    |

| Crdoba    | Argentina    |

| Escobar   | Argentina    |

| Ezeiza    | Argentina    |

| La Plata  | Argentina    |

| Merlo     | Argentina    |

| Quilmes   | Argentina    |

| San Miguel de Tucumn | Argentina    |

| Santa F   | Argentina    |

| Tandil    | Argentina    |

| Vicente Lpez | Argentina    |

| Almirante Brown | Argentina    |

| Avellaneda | Argentina    |

| Baha Blanca | Argentina    |

... 중간 생략 ...

| Crdoba    | Argentina    |
```

```
| Escobar   | Argentina    |
| Ezeiza    | Argentina    |
| La Plata  | Argentina    |
| Merlo     | Argentina    |
| Quilmes   | Argentina    |
| San Miguel de Tucumn | Argentina    |
| Santa F   | Argentina    |
| Tandil    | Argentina    |
| Vicente Lpez | Argentina    |
+ --------- + ------------ +
78 rows
```

경우에 따라서는 3개 이상의 릴레이션을 동등 조인할 필요가 있는 경우가 있다.

ER 모델에서 N:M 릴레이션쉽의 경우 관계형 데이터 모델로 변환하는 과정에서 별도의 릴레이션이 생성된다.

이 때, 원하는 정보를 얻기 위해서는 릴레이션쉽에 참여하는 두 엔티티가 변환되어 생성된 두 릴레이션과, 릴레이션쉽이 변환되어 생성된 릴레이션까지 총 3개의 릴레이션이 조인되어야 할 필요가 있는 것이다.

다음과 같은 질의를 생각해 보자.

이 질의는 actor 릴레이션, film 릴레이션, film_actor 릴레이션을 조인하여 결과를 추출하는 질의문이다.

질의 : 배우 ZERO CAGE가 출연한 모든 영화를 검색하여, 배우 이름과 영화 제목을 출력하라.

```
select actor.first_name, actor.last_name, title
from actor, film, film_actor
where actor.actor_id = film_actor.actor_id
      and film.film_id = film_actor.film_id
      and actor.first_name = 'ZERO'
      and actor.last_name = 'CAGE';
```

질의 결과

```
> select actor.first_name, actor.last_name, title
from actor, film, film_actor
where actor.actor_id = film_actor.actor_id
      and film.film_id = film_actor.film_id
      and actor.first_name = 'ZERO'
      and actor.last_name = 'CAGE'
+ --------------- + -------------- + ---------- +
| first_name      | last_name      | title      |
+ --------------- + -------------- + ---------- +
| ZERO            | CAGE           | CANYON STOCK |
| ZERO            | CAGE           | DANCES NONE |
| ZERO            | CAGE           | ENCINO ELF |
| ZERO            | CAGE           | ENDING CROWDS |
| ZERO            | CAGE           | GANDHI KWAI |
| ZERO            | CAGE           | GODFATHER DIARY |
| ZERO            | CAGE           | HANDICAP BOONDOCK |
```

```
| ZERO           | CAGE           | HONEY TIES |
| ZERO           | CAGE           | HORN WORKING |
| ZERO           | CAGE           | IMAGE PRINCESS |
| ZERO           | CAGE           | JERSEY SASSY |
| ZERO           | CAGE           | LOSER HUSTLER |
| ZERO           | CAGE           | MEET CHOCOLATE |
| ZERO           | CAGE           | MOD SECRETARY |
| ZERO           | CAGE           | MOONWALKER FOOL |
| ZERO           | CAGE           | OLEANDER CLUE |
| ZERO           | CAGE           | RACER EGG  |
| ZERO           | CAGE           | STORY SIDE |
| ZERO           | CAGE           | STRANGERS GRAFFITI |
| ZERO           | CAGE           | THIN SAGEBRUSH |
| ZERO           | CAGE           | TOOTSIE PILOT |
| ZERO           | CAGE           | UPTOWN YOUNG |
| ZERO           | CAGE           | VELVET TERMINATOR |
| ZERO           | CAGE           | WEST LION  |
| ZERO           | CAGE           | WORKER TARZAN |
+ -------------- + -------------- + ---------- +
25 rows
```

이 질의에서 film_actor 릴레이션은 ER 모델에서 film과 actor의 릴레이션쉽으로 모델링되었던 것이 관계형 데이터 모델로 변환되면서 릴레이션이 된 것이다.

그러므로 film 릴레이션과 actor 릴레이션을 연결하는 다리 역할을 하는 것을 알 수 있다.

조인 연산은 검색하려는 데이터가 여러 릴레이션으로 나뉘어 저장되어 있는 경우에 주로 사용된다.

여러 릴레이션을 조합하여 데이터를 검색하게 되기 때문에 질의 내용을 보고 조인 연산이 필요한지 여부를 쉽게 파악하지 못하는 경우가 있으므로 많은 연습이 필요하다.

이를 위하여 조인이 사용되는 질의의 예를 몇가지 더 살펴보고 연습해 보기로 하자.

질의 : 가족영화 중 S로 시작하는 영화의 제목과 장르를 출력하라.

```
select title, category.name
from film, category, film_category
where film.film_id = film_category.film_id
      and category.category_id = film_category.category_id
      and category.name = 'Family'
      and film.title like 'S%';
```

질의 결과

```
> select title, category.name
from film, category, film_category
where film.film_id = film_category.film_id
      and category.category_id = film_category.category_id
      and category.name = 'Family'
      and film.title like 'S%'
+ ---------- + --------- +
| title      | name      |
```

```
+ ---------- + --------- +
| SECRETS PARADISE | Family    |
| SENSIBILITY REAR | Family    |
| SIEGE MADRE | Family    |
| SLUMS DUCK | Family    |
| SOUP WISDOM | Family    |
| SPARTACUS CHEAPER | Family    |
| SPINAL ROCKY | Family    |
| SPLASH GUMP | Family    |
| SUNSET RACER | Family    |
| SUPER WYOMING | Family    |
+ ---------- + --------- +
10 rows
```

질의 : 배우 'Tom Miranda'가 출연한 영화 가족 영화를 찾아 영화 제목, 영화 장르를 출력하라.

```
select title, category.name
from film, category, film_category, actor, film_actor
where film.film_id = film_category.film_id
    and category.category_id = film_category.category_id
    and film.film_id = film_actor.film_id
    and actor.actor_id = film_actor.actor_id
    and category.name = 'Family'
    and actor.first_name = 'Tom'
    and actor.last_name = 'Miranda';
```

질의 결과

```
> select title, category.name
from film, category, film_category, actor, film_actor
where film.film_id = film_category.film_id
     and category.category_id = film_category.category_id
     and film.film_id = film_actor.film_id
     and actor.actor_id = film_actor.actor_id
     and category.name = 'Family'
     and actor.first_name = 'Tom'
     and actor.last_name = 'Miranda'
+ ---------- + --------- +
| title      | name      |
+ ---------- + --------- +
| CHASING FIGHT | Family    |
| FEUD FROGMEN | Family    |
| SUPER WYOMING | Family    |
+ ---------- + --------- +
3 rows
```

질의 : 배우 'Tom Miranda'가 출연한 영화 가족 영화의 이름과 그 영화에 함께 출연했던 배우의 이름을 검색하라.

```
select title, category.name, a2.first_name, a2.last_name
from film, category, film_category, actor as a1,
      actor as a2, film_actor as fa1, film_actor as fa2
where film.film_id = film_category.film_id
    and category.category_id = film_category.category_id
```

```
and film.film_id = fa1.film_id
and a1.actor_id = fa1.actor_id
and category.name = 'Family'
and a1.first_name = 'Tom'
and a1.last_name = 'Miranda'
and film.film_id = fa2.film_id
and a2.actor_id = fa2.actor_id
and a2.actor_id <> a1.actor_id;
```

질의 결과

```
> select title, category.name, a2.first_name, a2.last_name
from film, category, film_category, actor as a1, actor as a2, film_actor as
fa1, film_actor as fa2
where film.film_id = film_category.film_id
     and category.category_id = film_category.category_id
     and film.film_id = fa1.film_id
     and a1.actor_id = fa1.actor_id
     and category.name = 'Family'
     and a1.first_name = 'Tom'
     and a1.last_name = 'Miranda'
     and film.film_id = fa2.film_id
     and a2.actor_id = fa2.actor_id
     and a2.actor_id <> a1.actor_id
+ ---------- + --------- + --------------- + -------------- +
| title      | name      | first_name      | last_name      |
+ ---------- + --------- + --------------- + -------------- +
| CHASING FIGHT | Family    | JADA            | RYDER          |
```

```
| CHASING FIGHT | Family    | FAY             | WINSLET        |
| CHASING FIGHT | Family    | RUSSELL         | TEMPLE         |
| CHASING FIGHT | Family    | MENA            | HOPPER         |
| FEUD FROGMEN | Family     | TIM             | HACKMAN        |
| FEUD FROGMEN | Family     | PENELOPE        | CRONYN         |
| FEUD FROGMEN | Family     | MARY            | KEITEL         |
| SUPER WYOMING | Family    | NICK            | DEGENERES      |
| SUPER WYOMING | Family    | HUMPHREY        | GARLAND        |
+ ---------- + --------- + --------------- + -------------- +
9 rows
```

질의 : 'Jean Bell'이라는 고객이 빌렸던 모든 영화의 제목을 검색하라.

```
select first_name, last_name, title
from rental, inventory, customer, film
where rental.inventory_id = inventory.inventory_id
      and rental.customer_id = customer.customer_id
      and inventory.film_id = film.film_id
      and first_name = 'Jean'
      and last_name = 'Bell';
```

질의 결과

```
> select first_name, last_name, title
from rental, inventory, customer, film
where rental.inventory_id = inventory.inventory_id
      and rental.customer_id = customer.customer_id
      and inventory.film_id = film.film_id
```

```
        and first_name = 'Jean'
        and last_name = 'Bell'
+--------------+-------------+----------+
| first_name   | last_name   | title    |
+--------------+-------------+----------+
| JEAN         | BELL        | DOOM DANCING |
| JEAN         | BELL        | EASY GLADIATOR |
| JEAN         | BELL        | LANGUAGE COWBOY |
| JEAN         | BELL        | ROCKY WAR  |
| JEAN         | BELL        | GREASE YOUTH |
| JEAN         | BELL        | DOOM DANCING |
| JEAN         | BELL        | HUMAN GRAFFITI |
| JEAN         | BELL        | MIDSUMMER GROUNDHOG |
| JEAN         | BELL        | HORROR REIGN |
| JEAN         | BELL        | INNOCENT USUAL |
| JEAN         | BELL        | WIFE TURN  |
| JEAN         | BELL        | KWAI HOMEWARD |
| JEAN         | BELL        | DOLLS RAGE |
| JEAN         | BELL        | BRIDE INTRIGUE |
| JEAN         | BELL        | SWEDEN SHINING |
| JEAN         | BELL        | COMMAND DARLING |
| JEAN         | BELL        | ILLUSION AMELIE |
| JEAN         | BELL        | HARPER DYING |
| JEAN         | BELL        | FEVER EMPIRE |
| JEAN         | BELL        | LUST LOCK  |
| JEAN         | BELL        | CHOCOLAT HARRY |
```

```
| JEAN           | BELL           | SHANGHAI TYCOON |
| JEAN           | BELL           | DURHAM PANKY |
| JEAN           | BELL           | HYSTERICAL GRAIL |
| JEAN           | BELL           | OPEN AFRICAN |
| JEAN           | BELL           | FIDELITY DEVIL |
| JEAN           | BELL           | RIDGEMONT SUBMARINE |
+ -------------- + -------------- + ---------- +
27 rows
```

3) 셀프 조인

셀프 조인은 조인에 참여하는 두 릴레이션이 사실은 동일한 릴레이션인 경우를 의미한다.

이 경우 조인 속성이 어느 쪽 릴레이션의 속성인지에 대한 혼란이 있으므로 명확하게 구별할 수 있도록 릴레이션의 이름을 일시적으로 변경하여 사용한다.

다음과 같은 질의문을 생각해보자.

질의 : 영화 'Dirty Ace'와 같은 등급의 영화 중에서 replacement cost가 $11.99인 영화를 찾아 제목과 등급, replacement cost를 출력하라.

```
select f2.title, f2.rating, f2.replacement_cost
from film as f1, film as f2
where f1.rating = f2.rating
      and f1.title = 'Dirty Ace'
      and f2.replacement_cost = 11.99;
```

질의 결과

```
> select f2.title, f2.rating, f2.replacement_cost
from film as f1, film as f2
where f1.rating = f2.rating
      and f1.title = 'Dirty Ace'
      and f2.replacement_cost = 11.99
+ ---------- + ---------- + -------------------- +
| title      | rating     | replacement_cost     |
+ ---------- + ---------- + -------------------- +
| ANTITRUST TOMATOES | NC-17      | 11.99              |
| ELIZABETH SHANE | NC-17      | 11.99              |
| IMPOSSIBLE PREJUDICE | NC-17      | 11.99              |
| JUNGLE CLOSER | NC-17      | 11.99              |
| MEMENTO ZOOLANDER | NC-17      | 11.99              |
| MODEL FISH | NC-17      | 11.99              |
| PIZZA JUMANJI | NC-17      | 11.99              |
| SONS INTERVIEW | NC-17      | 11.99              |
| UNTOUCHABLES SUNRISE | NC-17      | 11.99              |
| VAMPIRE WHALE | NC-17      | 11.99              |
| WESTWARD SEABISCUIT | NC-17      | 11.99              |
+ ---------- + ---------- + -------------------- +
11 rows
```

셀프 조인을 수행할 때에 조인 속성을 두 개 이상으로 설정하는 것도 가능하다.

다음의 질의를 생각해보자.

질의 : 영화 'Dirty Ace'와 등급과 replacement_cost가 동일한 제목과 등급, replacement cost를 출력하라.

```
select f2.title, f2.rating, f2.replacement_cost
from film as f1, film as f2
where f1.rating = f2.rating
      and f1.title = 'Dirty Ace'
      and f1.replacement_cost = f2.replacement_cost;
```

질의 결과

```
> select f2.title, f2.rating, f2.replacement_cost
from film as f1, film as f2
where f1.rating = f2.rating
      and f1.title = 'Dirty Ace'
      and f1.replacement_cost = f2.replacement_cost
+ ---------- + ----------- + --------------------- +
| title      | rating      | replacement_cost      |
+ ---------- + ----------- + --------------------- +
| ARABIA DOGMA | NC-17       | 29.99                 |
| DIRTY ACE  | NC-17       | 29.99                 |
| EARTH VISION | NC-17       | 29.99                 |
| IDOLS SNATCHERS | NC-17       | 29.99                 |
| JERICHO MULAN | NC-17       | 29.99                 |
| PATIENT SISTER | NC-17       | 29.99                 |
| PRINCESS GIANT | NC-17       | 29.99                 |
| RANDOM GO  | NC-17       | 29.99                 |
+ ---------- + ----------- + --------------------- +
8 rows
```

4) 세타 조인

세타 조인은 가장 일반적인 형태의 조인을 의미한다.

동등 조인이 조인 조건으로 조인 속성들 간에 = 연산자만을 사용할 수 있는데 반하여 세타 조인은 <,> 등 다양한 비교 연산자를 사용할 수 있다.

이제 다양한 형태의 조인을 사용하는 SQL 질의문을 작성해보자.

질의 : G 등급이면서 영화 'Story Side' 보다 replacement cost가 더 높은 영화의 제목과 replacement cost를 검색하라.

```
select f2.title, f2.rating, f2.replacement_cost
from film as f1, film as f2
where f1.replacement_cost < f2.replacement_cost
    and f2.rating = 'G'
    and f1.title = 'Story Side';
```

질의 결과

```
> select f2.title, f2.rating, f2.replacement_cost
from film as f1, film as f2
where f1.replacement_cost < f2.replacement_cost
    and f2.rating = 'G'
    and f1.title = 'Story Side'
+ ---------- + ----------- + --------------------- +
| title      | rating      | replacement_cost      |
+ ---------- + ----------- + --------------------- +
| BALLROOM MOCKINGBIRD | G           | 29.99                |
| BEAUTY GREASE | G           | 28.99                |
```

```
| BONNIE HOLOCAUST | G           | 29.99                |
| CRUELTY UNFORGIVEN | G           | 29.99                |
| DESPERATE TRAINSPOTTING | G           | 29.99                |
| DOCTOR GRAIL | G           | 29.99                |
| EXTRAORDINARY CONQUERER | G           | 29.99                |
| FANTASIA PARK | G           | 29.99                |
| FARGO GANDHI | G           | 28.99                |
| FLATLINERS KILLER | G           | 29.99                |
| GOLDFINGER SENSIBILITY | G           | 29.99                |
| HILLS NEIGHBORS | G           | 29.99                |
| JAPANESE RUN | G           | 29.99                |
| LAWLESS VISION | G           | 29.99                |
| LOVER TRUMAN | G           | 29.99                |
| LUST LOCK  | G           | 28.99                |
| ROCK INSTINCT | G           | 28.99                |
| SAGEBRUSH CLUELESS | G           | 28.99                |
| SASSY PACKER | G           | 29.99                |
| SATURDAY LAMBS | G           | 28.99                |
| SIDE ARK   | G           | 28.99                |
| SMILE EARRING | G           | 29.99                |
| TRACY CIDER | G           | 29.99                |
| WATERSHIP FRONTIER | G           | 28.99                |
| WEST LION  | G           | 29.99                |
+ ---------- + ----------- + -------------------- +
25 rows
```

질의 : 영화 'Story Side'과 동일한 등급이면서 replacement cost는 더 높은 영화의 제목과 replacement cost를 검색하라.

```
select f2.title, f2.rating, f2.replacement_cost
from film as f1, film as f2
where f1.replacement_cost < f2.replacement_cost
    and f2.rating = f1.rating
    and f1.title = 'Story Side';
```

질의 결과

```
> select f2.title, f2.rating, f2.replacement_cost
from film as f1, film as f2
where f1.replacement_cost < f2.replacement_cost
    and f2.rating = f1.rating
    and f1.title = 'Story Side'
+ ---------- + ----------- + --------------------- +
| title      | rating      | replacement_cost      |
+ ---------- + ----------- + --------------------- +
| CHARIOTS CONSPIRACY | R           | 29.99                |
| CUPBOARD SINNERS | R           | 29.99                |
| DOUBLE WRATH | R           | 28.99                |
| FEUD FROGMEN | R           | 29.99                |
| FUGITIVE MAGUIRE | R           | 28.99                |
| GILMORE BOILED | R           | 29.99                |
| HANDICAP BOONDOCK | R           | 28.99                |
| HEAD STRANGER | R           | 28.99                |
| HONEY TIES | R           | 29.99                |
| ICE CROSSING | R           | 28.99                |
```

```
| JEEPERS WEDDING | R          | 29.99                |
| LOATHING LEGALLY | R          | 29.99                |
| QUEST MUSSOLINI | R          | 29.99                |
| SALUTE APOLLO | R          | 29.99                |
| SLACKER LIAISONS | R          | 29.99                |
| TIES HUNGER | R          | 28.99                |
| ZOOLANDER FICTION | R          | 28.99                |
+ ---------- + ---------- + -------------------- +
17 rows
```

질의 : 영화 'Wife Turn' 보다 상영 시간이 긴 영화의 이름과 상영 시간을 검색하라.

```
select f2.title, f2.length
from film as f1, film as f2
where f1.length < f2.length
      and f1.title = 'WIFE TURN';
```

질의 결과

```
> select f2.title, f2.length
from film as f1, film as f2
where f1.length < f2.length
      and f1.title = 'WIFE TURN'
+ ---------- + ---------- +
| title      | length     |
+ ---------- + ---------- +
| CHICAGO NORTH | 185        |
```

```
| CONSPIRACY SPIRIT | 184         |
| CONTROL ANTHEM | 185          |
| CRYSTAL BREAKING | 184        |
| DARN FORRESTER | 185          |
| GANGS PRIDE | 185          |
| HOME PITY  | 185          |
| KING EVOLUTION | 184          |
| MOONWALKER FOOL | 184         |
| MUSCLE BRIGHT | 185          |
| POND SEATTLE | 185          |
| SMOOCHY CONTROL | 184         |
| SOLDIERS EVOLUTION | 185         |
| SONS INTERVIEW | 184          |
| SORORITY QUEEN | 184          |
| SWEET BROTHERHOOD | 185         |
| THEORY MERMAID | 184          |
| WORST BANGER | 185          |
+ ---------- + ----------- +
18 rows
```

5) INNER 조인

Inner 조인은 ANSI에서 정의한 표준 SQL 조인 질의문의 하나로서 동등 조인을 좀 더 간편하게 사용할 수 있도록 정의한 것이다.

대부분의 상용 데이터베이스 관리 시스템이 ANSI 표준 SQL을 지원하고 있으므로 Inner 조인에 대해서 숙지하고 있다면 매우 유용하게 사

용할 수 있을 것이다.

Inner 조인은 조인 조건을 WHERE절에 기술하지 않고 FROM절에 기술한다는 특징이 있다.

SQL 질의문을 작성할 때에 WHERE절에 조인 조건문과 셀렉션 조건문이 함께 나타나기 때문에 질의문이 복잡해지면 혼란이 초래될 수 있다.

Innser 조인 구문은 조인 조건문을 FROM절에 기술함으로써 이러한 혼란을 방지할 수 있다.

Inner 조인의 구문 형식은 다음과 같다.

From t1 INNER JOIN t2 ON [조인 조건]

이제 Inner 조인을 사용한 SQL 질의문을 연습해 보자.

앞서 동등 조인을 설명할 때 살펴보았던 질의문을 Inner 조인을 이용하여 다시 한번 작성해 보기로 하자.

질의 : Argentina에 속한 도시 이름과 국가의 이름을 검색하라.

```
select city, country
from city inner join country
      on city.country_id = country.country_id
where country = 'Argentina';
```

질의 결과

```
> select city, country
```

```
from city inner join country on city.country_id = country.country_id
where country = 'Argentina'
+ --------- + ------------ +
| city      | country      |
+ --------- + ------------ +
| Almirante Brown | Argentina    |
| Avellaneda | Argentina    |
| Baha Blanca | Argentina    |
| Crdoba    | Argentina    |
| Escobar   | Argentina    |
| Ezeiza    | Argentina    |
| La Plata  | Argentina    |
| Merlo     | Argentina    |
| Quilmes   | Argentina    |
| San Miguel de Tucumn | Argentina    |
| Santa F   | Argentina    |
| Tandil    | Argentina    |
| Vicente Lpez | Argentina    |
+ --------- + ------------ +
13 rows
```

질의 결과를 살펴보면 앞서 작성했던 동등 조인 구문을 실행했을 때와 동일한 결과를 얻는 것을 알 수 있다.

Inner 조인을 사용할 때에 조인에 참여하는 두 릴레이션의 조인 속성이 동일한 이름을 가지고, 동등 조인을 하려한다면, 굳이 ON 구문을 사용하여 조인 조건을 모두 기술할 필요가 없다.

USING 구문을 이용하면 간단하게 속성 이름이 같은 동등 조인을 표현할 수 있기 때문이다.

위의 예에서 작성한 질의문은 USING을 이용하여 다음과 같이 작성할 수 있다.

```
select city, country
from city inner join country USING country_id
where country = 'Argentina';
```

이 경우에도 질의 결과는 동일한 결과를 얻을 수 있다.

6) 자연 조인

동일한 이름의 속성을 이용하여 두 릴레이션을 동등 조인하는 것을 자연 조인이라고 한다.

자연 조인은 동등 조인의 특별한 형태이기는 하지만 실제 상황에서 매우 자주 사용된다.

ANSI 표준 SQL에서는 자연 조인을 쉽게 사용할 수 있도록 하기 위하여 NATURAL JOIN 구문을 제공하고 있다.

NATURAL JOIN 구문의 형식은 다음과 같다.

```
FROM t1 NATURAL JOIN t2
```

이 구문의 의미는 릴레이션 t1과 t2에 있는 동일한 이름의 속성을 조인 속성으로 하여 두 릴레이션을 동등 조인하라는 것이다.

자연 조인을 사용한 SQL 질의문을 살펴보기로 하자.

이 질의문은 앞서 동등 조인을 다룰 때에 살펴보았던 질의문이지만 NATURAL JOIN 구문을 이용하여 다시 작성해 보기로 하자.

질의 : Argentina에 속한 도시 이름과 국가의 이름을 검색하라.

```
select city, country
from city natural join country
where country = 'Argentina';
```

다) 집합 연산자를 사용한 SQL 질의문

관계형 데이터 모델의 근간을 이루는 릴레이션은 동일한 스키마를 가지는 투플들의 집합으로 정의된다.

그러므로 합집합 호환성이 있는 두 개의 릴레이션에 대해서는 여러 가지 집합 연산이 가능하다.

앞에서 살펴본 SELECT 문의 결과는 릴레이션이다.

그러므로 두 SELECT 문의 결과 릴레이션이 합집합 호환성이 있으면 집합 연산을 수행할 수 있다.

본 절에서는 관계 대수의 연산자 중 집합 연산자에 해당하는 합집합, 교집합, 차집합 연산을 SQL로 표현하는 방법에 대하여 설명한다.

1) 합집합

합집합은 합집합 호환성이 있는 두 릴레이션을 합하여 하나의 릴레이션으로 만드는 연산이다.

이 때 두 릴레이션은 합집합 호환성이 있으므로 스키마는 그대로 유지되고, 두 릴레이션의 투플은 하나의 결과 릴레이션에 함께 속하게 된다.

이는 집합 연산이므로 중복되는 투플이 있는 경우에는 필요에 따라 중복이 제거될 수 있다.

SQL 질의문에서 합집합 연산자에는 UNION과 UNION ALL이 있다.

UNION은 합집합의 결과 중복되는 투플을 제거하고, UNION ALL은 중복을 제거하지 않는다.

다음과 같은 질의를 살펴보기로 하자.

질의 : replacement cost가 $29 이상이고, 대여료는 $4 이상이거나 배우 'Sandra Kilmer'가 출연한 스포츠영화의 ID와 제목을 검색하라.

이 질의는 다음과 같은 두 개의 질의로 나누어 수행한 후 그 결과를 합집합하면 처리될 수 있다.

질의 1: replacement cost가 $29 이상이고, 대여료는 $4 이상인 영화의 ID와 제목을 검색하라.

질의 2: 배우 'Sandra Kilmer'가 출연한 스포츠영화의 ID와 제목을 검색하라.

두 개의 질의에 대한 SQL 질의문과 그 수행 결과는 다음과 같다.

질의 1.

```
select film_id, title
from film
where replacement_cost >= 29
      and rental_rate >= 4;
```

질의 결과

```
> select film_id, title
from film
where replacement_cost >= 29
      and rental_rate >= 4
+ ------------ + ---------- +
| film_id      | title      |
+ ------------ + ---------- +
| 81           | BLINDNESS GUN |
| 224          | DESPERATE TRAINSPOTTING |
| 487          | JINGLE SAGEBRUSH |
| 510          | LAWLESS VISION |
| 691          | POSEIDON FOREVER |
| 731          | RIGHT CRANES |
| 803          | SLACKER LIAISONS |
| 944          | VIRGIN DAISY |
| 969          | WEST LION  |
| 994          | WYOMING STORM |
+ ------------ + ---------- +
10 rows
```

질의 2.

```
select film.film_id, title
from film, film_actor, actor, film_category, category
where film.film_id = film_actor.film_id
      and actor.actor_id = film_actor.actor_id
      and actor.first_name = 'Sandra'
      and actor.last_name = 'Kilmer'
      and film.film_id = film_category.film_id
      and film_category.category_id = category.category_id
      and category.name = 'Sports';
```

질의 결과

```
> select film.film_id, title
from film, film_actor, actor, film_category, category
where film.film_id = film_actor.film_id
      and actor.actor_id = film_actor.actor_id
      and actor.first_name = 'Sandra'
      and actor.last_name = 'Kilmer'
      and film.film_id = film_category.film_id
      and film_category.category_id = category.category_id
      and category.name = 'Sports'
+ ------------ + ---------- +
| film_id      | title      |
+ ------------ + ---------- +
| 42           | ARTIST COLDBLOODED |
| 254          | DRIVER ANNIE |
| 691          | POSEIDON FOREVER |
| 782          | SHAKESPEARE SADDLE |
```

```
+ ------------ + ---------- +

4 rows
```

최종 질의문

```
(select film_id, title
from film
where replacement_cost >= 29
      and rental_rate >= 4)
UNION
(select film.film_id, title
from film, film_actor, actor, film_category, category
where film.film_id = film_actor.film_id
      and actor.actor_id = film_actor.actor_id
      and actor.first_name = 'Sandra'
      and actor.last_name = 'Kilmer'
      and film.film_id = film_category.film_id
      and film_category.category_id = category.category_id
      and category.name = 'Sports')
```

질의 결과

```
> (select film_id, title
from film
where replacement_cost >= 29
      and rental_rate >= 4)
UNION
(select film.film_id, title
from film, film_actor, actor, film_category, category
```

```
where film.film_id = film_actor.film_id

      and actor.actor_id = film_actor.actor_id

      and actor.first_name = 'Sandra'

      and actor.last_name = 'Kilmer'

      and film.film_id = film_category.film_id

      and film_category.category_id = category.category_id

      and category.name = 'Sports')

+ ------------ + ---------- +

| film_id      | title      |

+ ------------ + ---------- +

| 81           | BLINDNESS GUN |

| 224          | DESPERATE TRAINSPOTTING |

| 487          | JINGLE SAGEBRUSH |

| 510          | LAWLESS VISION |

| 691          | POSEIDON FOREVER |

| 731          | RIGHT CRANES |

| 803          | SLACKER LIAISONS |

| 944          | VIRGIN DAISY |

| 969          | WEST LION  |

| 994          | WYOMING STORM |

| 42           | ARTIST COLDBLOODED |

| 254          | DRIVER ANNIE |

| 782          | SHAKESPEARE SADDLE |

+ ------------ + ---------- +

13 rows
```

질의 결과에서 질의 1의 결과에 나타나는 투플 개수는 10개이고, 질의 2의 결과에 나타나는 투플 개수는 4개인데, 합집합을 수행한 결과 투플 개수가 13개가 되는 것을 확인할 수 있다.

이는 film_id가 691인 영화가 양쪽 결과에 모두 포함되어 중복된 투플이 삭제되었기 때문이다.

이제 동일한 질의에 대하여 중복을 삭제하지 않고 수행한 결과를 확인해 보자.

이를 위하여 UNION 대신에 UNION ALL을 사용한다.

```
(select film_id, title
from film
where replacement_cost >= 29
      and rental_rate >= 4)
UNION ALL
(select film.film_id, title
from film, film_actor, actor, film_category, category
where film.film_id = film_actor.film_id
      and actor.actor_id = film_actor.actor_id
      and actor.first_name = 'Sandra'
      and actor.last_name = 'Kilmer'
      and film.film_id = film_category.film_id
      and film_category.category_id = category.category_id
      and category.name = 'Sports')
```

질의 결과

```
> (select film_id, title
from film
where replacement_cost >= 29
```

```
      and rental_rate >= 4)
UNION ALL
(select film.film_id, title
from film, film_actor, actor, film_category, category
where film.film_id = film_actor.film_id
      and actor.actor_id = film_actor.actor_id
      and actor.first_name = 'Sandra'
      and actor.last_name = 'Kilmer'
      and film.film_id = film_category.film_id
      and film_category.category_id = category.category_id
      and category.name = 'Sports')
+ ------------ + ---------- +
| film_id      | title      |
+ ------------ + ---------- +
| 81           | BLINDNESS GUN |
| 224          | DESPERATE TRAINSPOTTING |
| 487          | JINGLE SAGEBRUSH |
| 510          | LAWLESS VISION |
| 691          | POSEIDON FOREVER |
| 731          | RIGHT CRANES |
| 803          | SLACKER LIAISONS |
| 944          | VIRGIN DAISY |
| 969          | WEST LION  |
| 994          | WYOMING STORM |
| 42           | ARTIST COLDBLOODED |
| 254          | DRIVER ANNIE |
```

```
| 691          | POSEIDON FOREVER |
| 782          | SHAKESPEARE SADDLE |
+ ------------ + ---------- +
14 rows
```

질의를 수행한 결과 film_id가 691인 영화 Poseidon forever가 중복해서 나타나는 것을 확인할 수 있다.

만일 합집합 호환성이 없는 두 릴레이션에 대하여 합집합 연산을 수행하면 어떤 현상이 일어날지 확인해 보자.

합집합 호환성이 없도록 하기 위하여 위의 질의 1에서 desciption 속성도 프로젝션하도록 질의를 변경해 보자.

즉, 다음과 같이 질의문을 변경하여 수행해 보자.

```
(select film_id, title, description
from film
where replacement_cost >= 29
      and rental_rate >= 4)
UNION
(select film.film_id, title
from film, film_actor, actor, film_category, category
where film.film_id = film_actor.film_id
   and actor.actor_id = film_actor.actor_id
   and actor.first_name = 'Sandra'
   and actor.last_name = 'Kilmer'
   and film.film_id = film_category.film_id
```

```
and film_category.category_id = category.category_id
and category.name = 'Sports')
```

질의문을 바꾸어 수행하면 다음과 같은 에러 메시지가 출력되는 것을 확인할 수 있다.

Error Code: 1222. The used SELECT statements have a different number of columns

이 메시지는 합집합을 수행하려는 두 릴레이션의 속성 개수가 같지 않아 합집합 연산을 수행할 수 없다는 것을 의미한다.

그렇다면 속성의 개수는 동일하지만 대응되는 속성의 데이터 타입이 일치하지 않는 경우는 어떤 현상이 발생하는지 확인해 보자.

이를 위하여 질의 1의 질의문에서 프로젝션하는 속성을 film_id가 아니라 날짜 타입인 last_update로 바꾸어 수행해보자.

```
(select last_update, title
from film
where replacement_cost >= 29
      and rental_rate >= 4)
UNION
(select film.film_id, title
from film, film_actor, actor, film_category, category
where film.film_id = film_actor.film_id
   and actor.actor_id = film_actor.actor_id
   and actor.first_name = 'Sandra'
```

```
and actor.last_name = 'Kilmer'
and film.film_id = film_category.film_id
and film_category.category_id = category.category_id
and category.name = 'Sports')
```

질의 결과

```
> (select last_update, title
from film
where replacement_cost >= 29
      and rental_rate >= 4)
UNION
(select film.film_id, title
from film, film_actor, actor, film_category, category
where film.film_id = film_actor.film_id
   and actor.actor_id = film_actor.actor_id
   and actor.first_name = 'Sandra'
   and actor.last_name = 'Kilmer'
   and film.film_id = film_category.film_id
   and film_category.category_id = category.category_id
   and category.name = 'Sports')
+ ---------------- + ---------- +
| last_update      | title      |
+ ---------------- + ---------- +
| 2006-02-15 05:03:42 | BLINDNESS GUN |
| 2006-02-15 05:03:42 | DESPERATE TRAINSPOTTING |
| 2006-02-15 05:03:42 | JINGLE SAGEBRUSH |
| 2006-02-15 05:03:42 | LAWLESS VISION |
```

```
| 2006-02-15 05:03:42 | POSEIDON FOREVER |
| 2006-02-15 05:03:42 | RIGHT CRANES |
| 2006-02-15 05:03:42 | SLACKER LIAISONS |
| 2006-02-15 05:03:42 | VIRGIN DAISY |
| 2006-02-15 05:03:42 | WEST LION  |
| 2006-02-15 05:03:42 | WYOMING STORM |
| 42              | ARTIST COLDBLOODED |
| 254             | DRIVER ANNIE |
| 691             | POSEIDON FOREVER |
| 782             | SHAKESPEARE SADDLE |
+ --------------- + ---------- +
14 rows
```

질의 결과 문제없이 합집합이 수행되는 것을 확인할 수 있다.

즉, MySQL에서는 합집합 호환성을 체크할 때에 대응되는 속성의 데이터 타입까지는 체크하지 않는 것을 알 수 있다.

그러나 이것은 MySQL만의 특징으로서 모든 데이터베이스 관리 시스템이 합집합 호환성을 체크할 때에 데이터 타입을 고려하지 않는 것은 아니다.

2) 차집합

두 릴레이션 R의 S에 대한 차집합은 릴레이션 R에는 포함되지만 릴레이션 S에는 포함되지 않는 투플들을 추출하는 연산이다.

차집합 연산도 합집합 호환성이 존재하는 릴레이션에 대해서만 유효하

다.

차집합 연산을 수행하는 SQL 구문은 MINUS 이고, 사용 방식은 UNION과 같다.

다음과 같은 질의를 생각해 보자.

질의 : 액션 영화를 대여한 고객들 중에서 코메디 영화나 가족 영화는 한번도 대여하지 않았던 고객의 ID를 검색하라.

이 질의는 다음과 같은 두 개의 질의로 나누어 처리한 후 차집합 연산을 수행하면 처리할 수 있다.

질의 1: 액션 영화를 대여한 고객들의 ID를 검색하라.
질의 2: 코메디 영화나 가족영화를 대여한 고객 ID를 검색하라.

그러므로 최종 질의문을 SQL로 작성하면 다음과 같다.

```
(select distinct customer.first_name, customer.last_name
from customer, rental, film_category,
        category, inventory
where customer.customer_id = rental.customer_id
    and rental.inventory_id = inventory.inventory_id
    and inventory.film_id = film_category.film_id
    and film_category.category_id = category.category_id
    and category.name = 'Action')
MINUS
```

```
(select distinct customer.customer_id
from customer, rental, film_category,
        category, inventory
where customer.customer_id = rental.customer_id
   and rental.inventory_id = inventory.inventory_id
   and inventory.film_id = film_category.film_id
   and film_category.category_id = category.category_id
   and (category.name = 'Comedy'
         OR category.name = 'Family'))
```

그러나 우리가 사용하고 있는 MySQL에서는 차집합 연산자인 MINUS를 지원하지 않는다.

그러므로 다른 방식으로 이 연산을 수행하여야 한다.

이 연산의 수행 방법에 대해서는 “마) 중첩 질의문”절에서 다루기로 한다.

라) 부가절을 사용한 SQL 질의문

우리는 지금까지 일반적인 관계 질의어로서의 SQL 질의문에 대하여 소개하였다.

이러한 SQL 질의문은 관계 대수를 이론적인 배경으로 하여 SQL이 관계적으로 완전한 언어임을 보이는 것이다.

그러나 실용적인 측면에서는 아직 필요한 여러 가지 기능이 남아 있다.

본 절에서는 SQL을 보다 실용적으로 만들어주는 부가 구문에 대하여 설명하기로 한다.

1) ORDER BY

릴레이션은 투플의 집합으로 정의되므로 릴레이션에 속한 투플에는 특별한 순서가 존재하지 않는다.

마찬가지로 SQL의 수행 결과로 출력되는 투플에도 특정한 순서가 없이 임의로 출력될 수 있다.

그러나 실제 상황에서는 투플들을 정렬된 순서로 출력하는 것이 매우 유용한 기능이 될 수 있다.

Order By는 결과 릴레이션의 출력 순서를 특정 속성 값의 크기에 따라 정렬하는 명령어이다.

다음과 같은 질의문을 생각해 보자.

질의 : 영화 'Dirty Ace'와 같은 등급의 영화 중에서 제목이 알파벳 'C' 또는 'S'로 시작하고, 'replacement cost가 $11.99 이상 $17.99 이하인 영화를 찾아 제목과 등급, replacement cost를 출력하되 replacement cost의 크기에 따라 오름차순으로 출력하라.

```
select f2.title, f2.rating, f2.replacement_cost
from film as f1, film as f2
where f1.rating = f2.rating
      and f1.title = 'Dirty Ace'
      and f2.replacement_cost between 11.99 and 17.99
      and (f2.title like 'C%'
            or f2.title like 'S%')
order by f2.replacement_cost;
```

질의 결과

```
> select f2.title, f2.rating, f2.replacement_cost
from film as f1, film as f2
where f1.rating = f2.rating
      and f1.title = 'Dirty Ace'
      and f2.replacement_cost between 11.99 and 17.99
      and (f2.title like 'C%'
           or f2.title like 'S%')
order by f2.replacement_cost
+ ---------- + ----------- + --------------------- +
| title      | rating      | replacement_cost      |
+ ---------- + ----------- + --------------------- +
| SONS INTERVIEW | NC-17       | 11.99                 |
| CONEHEADS SMOOCHY | NC-17       | 12.99                 |
| CRANES RESERVOIR | NC-17       | 12.99                 |
| SIERRA DIVIDE | NC-17       | 12.99                 |
| CONFIDENTIAL INTERVIEW | NC-17       | 13.99                 |
| SCALAWAG DUCK | NC-17       | 13.99                 |
| STATE WASTELAND | NC-17       | 13.99                 |
| CHAMBER ITALIAN | NC-17       | 14.99                 |
| CANDLES GRAPES | NC-17       | 15.99                 |
| STEEL SANTA | NC-17       | 15.99                 |
| CARIBBEAN LIBERTY | NC-17       | 16.99                 |
| CHOCOLAT HARRY | NC-17       | 16.99                 |
| CADDYSHACK JEDI | NC-17       | 17.99                 |
| CRAFT OUTFIELD | NC-17       | 17.99                 |
```

```
| SOMETHING DUCK | NC-17       | 17.99                |
| SORORITY QUEEN | NC-17       | 17.99                |
+ ---------- + ----------- + -------------------- +
16 rows
```

질의 결과는 order by절에 나타낸 속성의 값에 따라 오름차순으로 정렬된다.

명시적으로 오름차순임을 나타내고 싶을 때는 ASC 키워드를 속성의 뒷부분에 붙인다.

만일 특별히 내림차순으로 정렬하고자 하는 경우에는 속성의 뒷부분에 DESC 키워드를 붙인다.

질의 : 영화 'Dirty Ace'와 같은 등급의 영화 중에서 제목이 알파벳 'C' 또는 'S'로 시작하고, 'replacement cost가 $11.99 이상 $17.99 이하인 영화를 찾아 제목과 등급, replacement cost를 출력하되 replacement cost의 크기에 따라 내림차순으로 출력하라.

```
select f2.title, f2.rating, f2.replacement_cost
from film as f1, film as f2
where f1.rating = f2.rating
      and f1.title = 'Dirty Ace'
      and f2.replacement_cost between 11.99 and 17.99
      and (f2.title like 'C%'
            or f2.title like 'S%')
order by f2.replacement_cost DESC;
```

질의 결과

```
> select f2.title, f2.rating, f2.replacement_cost
from film as f1, film as f2
where f1.rating = f2.rating
      and f1.title = 'Dirty Ace'
      and f2.replacement_cost between 11.99 and 17.99
      and (f2.title like 'C%'
           or f2.title like 'S%')
order by f2.replacement_cost DESC
+ ---------- + ----------- + --------------------- +
| title      | rating      | replacement_cost      |
+ ---------- + ----------- + --------------------- +
| CADDYSHACK JEDI | NC-17       | 17.99                 |
| CRAFT OUTFIELD | NC-17       | 17.99                 |
| SOMETHING DUCK | NC-17       | 17.99                 |
| SORORITY QUEEN | NC-17       | 17.99                 |
| CARIBBEAN LIBERTY | NC-17       | 16.99                 |
| CHOCOLAT HARRY | NC-17       | 16.99                 |
| CANDLES GRAPES | NC-17       | 15.99                 |
| STEEL SANTA | NC-17       | 15.99                 |
| CHAMBER ITALIAN | NC-17       | 14.99                 |
| CONFIDENTIAL INTERVIEW | NC-17       | 13.99                 |
| SCALAWAG DUCK | NC-17       | 13.99                 |
| STATE WASTELAND | NC-17       | 13.99                 |
| CONEHEADS SMOOCHY | NC-17       | 12.99                 |
| CRANES RESERVOIR | NC-17       | 12.99                 |
```

```
| SIERRA DIVIDE | NC-17       | 12.99                |
| SONS INTERVIEW | NC-17      | 11.99                |
+ ---------- + ----------- + -------------------- +
16 rows
```

경우에 따라서는 투플의 정렬 순서를 결정하는 속성을 두가지 이상으로 설정할 수도 있다.

이 경우 전체적으로는 첫 번째 속성값의 크기에 따라 정렬하지만, 첫 번째 속성값이 동일한 경우 두 번째 속성값에 따라 정렬된 순서를 유지하게 된다.

다음의 질의를 살펴보자.

질의 : 영화 'Dirty Ace'와 같은 등급의 영화 중에서 제목이 알파벳 'C' 또는 'S'로 시작하고, 'replacement cost가 $11.99 이상 $17.99 이하인 영화를 찾아 제목과 등급, replacement cost를 출력하되 replacement cost의 크기에 따라 오름차순으로 출력하고, replacement cost가 동일한 경우에는 영화 제목의 알파벳 순에 따라 내림차순으로 정렬하여 출력하라.

```
select f2.title, f2.rating, f2.replacement_cost
from film as f1, film as f2
where f1.rating = f2.rating
      and f1.title = 'Dirty Ace'
      and f2.replacement_cost between 11.99 and 17.99
      and (f2.title like 'C%'
            or f2.title like 'S%')
order by f2.replacement_cost ASC, f2.title DESC;
```

질의 결과

```
> select f2.title, f2.rating, f2.replacement_cost
from film as f1, film as f2
where f1.rating = f2.rating
      and f1.title = 'Dirty Ace'
      and f2.replacement_cost between 11.99 and 17.99
      and (f2.title like 'C%'
           or f2.title like 'S%')
order by f2.replacement_cost ASC, f2.title DESC
+ ---------- + ----------- + --------------------- +
| title      | rating      | replacement_cost      |
+ ---------- + ----------- + --------------------- +
| SONS INTERVIEW | NC-17       | 11.99                 |
| SIERRA DIVIDE | NC-17       | 12.99                 |
| CRANES RESERVOIR | NC-17       | 12.99                 |
| CONEHEADS SMOOCHY | NC-17       | 12.99                 |
| STATE WASTELAND | NC-17       | 13.99                 |
| SCALAWAG DUCK | NC-17       | 13.99                 |
| CONFIDENTIAL INTERVIEW | NC-17       | 13.99                 |
| CHAMBER ITALIAN | NC-17       | 14.99                 |
| STEEL SANTA | NC-17       | 15.99                 |
| CANDLES GRAPES | NC-17       | 15.99                 |
| CHOCOLAT HARRY | NC-17       | 16.99                 |
| CARIBBEAN LIBERTY | NC-17       | 16.99                 |
| SORORITY QUEEN | NC-17       | 17.99                 |
```

```
| SOMETHING DUCK | NC-17       | 17.99                |
| CRAFT OUTFIELD | NC-17       | 17.99                |
| CADDYSHACK JEDI | NC-17       | 17.99                |
+ ---------- + ----------- + -------------------- +
16 rows
```

질의 결과를 살펴보면, 투플들이 전체적으로는 replacement cost 속성값에 따라 오름차순으로 정렬되어 있지만 그 값이 동일한 경우에는 제목에 따라 내림차순으로 정렬되어 있는 것을 확인할 수 있다.

2) 집계함수

집계함수란 여러개의 데이터에 적용되어 하나의 값을 반환하는 함수를 의미한다.

대표적인 집계함수로는 데이터의 개수를 세는 count, 데이터의 총 합을 구하는 sum, 데이터의 평균값을 구하는 average, 최소값을 구하는 minimum, 최대값을 구하는 maximum 등이 있다.

본 절에서는 SQL 질의문에서 각각의 집계함수를 사용하는 방법에 대하여 설명한다.

(1) COUNT 함수

Count() 함수는 투플의 개수를 구하는 함수이다.

다음은 count()함수를 사용하여 작성한 SQL 질의문이다.

질의 : 영화 'Dirty Ace'와 같은 등급의 영화 개수와 해당 등급을 출력하라.

```
select f2.rating, count(*)
from film as f1, film as f2
where f1.rating = f2.rating
      and f1.title = 'Dirty Ace'
```

질의 결과

```
> select f2.rating, count(*)
from film as f1, film as f2
where f1.rating = f2.rating
      and f1.title = 'Dirty Ace'
+ ----------- + ------------- +
| rating      | count(*)      |
+ ----------- + ------------- +
| NC-17       | 210           |
+ ----------- + ------------- +
1 rows
```

count() 함수 앞에 distinct 키워드를 삽입하면 중복된 값을 제거한 후 count() 함수를 적용하게 된다.

다음의 두 질의를 비교해 보자.

질의 1은 film 릴레이션에서 rating 속성을 프로젝션하여 개수를 센 것이고, 질의 2는 프로젝션 후 중복을 제거하고 개수를 센 것이다.

결과적으로 질의 1은 총 투플의 개수가 출력된 셈이고, 질의 2는

rating 속성에 나타나는 속성값의 종류의 개수가 출력된 것이라고 할 수 있다.

질의 2의 결과로부터 5가지 종류의 영화 등급이 있다는 것을 알 수 있다.

질의 1. **select count(rating)** **from film ;**	**질의 2.** **select count(distinct rating)** **from film ;**
질의 결과 > select count(rating) from film + ---------- + | count(rating) | + ---------- + | 1000 | + ---------- + 1 rows	**질의 결과** > select count(distinct rating) from film + ---------- + | count(distinct rating) | + ---------- + | 5 | + ---------- + 1 rows

이러한 사실은 다음 질의를 통하여 확인할 수 있다.

질의 결과 영화 등급에는 PG, G, NC-17, PG-13, R의 5가지 등급이 있음을 확인할 수 있다.

```
select distinct rating
from film ;
```

질의 결과

```
> select distinct rating
from film
+ ---------- +
| rating     |
+ ---------- +
| PG         |
| G          |
| NC-17      |
| PG-13      |
| R          |
+ ---------- +
5 rows
```

(2) SUM 함수

SUM() 함수는 주어진 속성값의 총합을 구하는 함수이다.

다음의 질의문을 살펴보자.

질의 : 고객 'Lisa Anderson'이 영화를 빌려가고 대여금을 지불한 총 횟수와 총 금액을 검색하라.

```
select first_name, last_name, count(amount), sum(amount)
from customer, payment
```

```
where customer.customer_id = payment.customer_id
      and first_name = 'LISA'
      and last_name = 'ANDERSON';
```

질의 결과

```
> select first_name, last_name, count(amount), sum(amount)
from customer, payment
where customer.customer_id = payment.customer_id
      and first_name = 'LISA'
      and last_name = 'ANDERSON'
+ -------------- + ------------- + ----------------- + --------------- +
| first_name     | last_name     | count(amount)     | sum(amount)     |
+ -------------- + ------------- + ----------------- + --------------- +
| LISA           | ANDERSON      | 24                | 106.76          |
+ -------------- + ------------- + ----------------- + --------------- +
1 rows
```

질의 결과 Lisa Anderson은 24회에 걸쳐 대여금을 지불했으며, 총 $106.76을 지불한 것을 알 수 있다.

(3) AVERAGE 함수

MySQL에서 속성값의 평균을 구하는 함수는 AVG()이다.

다음의 질의문을 살펴보자.

질의 : 고객 'Lisa Anderson'이 영화를 빌려가고 대여금을 지불한 총 금액과 1회당 지불한 평균 금액을 검색하라.

```
select first_name, last_name, sum(amount), avg(amount)
from customer, payment
where customer.customer_id = payment.customer_id
      and first_name = 'LISA'
      and last_name = 'ANDERSON';
```

질의 결과

```
> select first_name, last_name, sum(amount), avg(amount)
from customer, payment
where customer.customer_id = payment.customer_id
      and first_name = 'LISA'
      and last_name = 'ANDERSON'
+ -------------- + -------------- + ---------------- + ---------------- +
| first_name     | last_name      | sum(amount)      | avg(amount)      |
+ -------------- + -------------- + ---------------- + ---------------- +
| LISA           | ANDERSON       | 106.76           | 4.448333         |
+ -------------- + -------------- + ---------------- + ---------------- +
1 rows
```

질의 결과로부터 Lisa Anderson은 총 $106.76의 대여금을 지불했으며, 이는 평균적으로 1회에 $4.448383을 지불한 셈인 것을 확인할 수 있다.

(4) MIN, MAX 함수

MySQL에서 속성값의 최소값과 최대값을 구하는 함수는 각각 MIN()과 MAX()이다.

다음의 질의문을 살펴보자.

질의 : 고객 'Lisa Anderson'이 영화를 빌려가고 지불한 대여금 중 최소값과 최대값을 검색하라.

```
select first_name, last_name, min(amount), max(amount)
from customer, payment
where customer.customer_id = payment.customer_id
      and first_name = 'LISA'
      and last_name = 'ANDERSON';
```

질의 결과

```
> select first_name, last_name, min(amount), max(amount)
from customer, payment
where customer.customer_id = payment.customer_id
      and first_name = 'LISA'
      and last_name = 'ANDERSON'
+ --------------- + -------------- + ---------------- + ---------------- +
| first_name      | last_name      | min(amount)      | max(amount)      |
+ --------------- + -------------- + ---------------- + ---------------- +
| LISA            | ANDERSON       | 0.99             | 9.99             |
+ --------------- + -------------- + ---------------- + ---------------- +
1 rows
```

질의 결과로부터 Lisa Anderson은 최소 $0.99의 대여금을 지불했으며, 최대로는 $9.99를 지불했던 것을 확인할 수 있다.

3) GROUP BY

우리는 앞 절에서 여러 가지 집계 함수를 사용하여 보았다.

이러한 집계 함수는 주어진 릴레이션에서 주어진 조건을 만족하는 모든 투플에 대하여 적용되는 함수이다.

그러나 경우에 따라서는 특정 속성 값이 같은 것들끼리 묶어서 개별적으로 집계 함수를 적용시켜야 하는 경우도 있다.

이러한 경우에 사용하는 것이 GROUP BY 구문이다.

다음의 질의문을 생각해 보자.

질의 : 등급별 영화 개수를 출력하라 .

```
select category.name, count(film_id)
from  category, film_category
where category.category_id = film_category.category_id
group by film_category.category_id;
```

질의 결과

```
> select category.name, count(film_id)
from  category, film_category
where category.category_id = film_category.category_id
group by film_category.category_id
```

```
+ --------- + ------------------- +
| name      | count(film_id)      |
+ --------- + ------------------- +
| Action    | 64                  |
| Animation | 66                  |
| Children  | 60                  |
| Classics  | 57                  |
| Comedy    | 58                  |
| Documentary | 68                  |
| Drama     | 62                  |
| Family    | 69                  |
| Foreign   | 73                  |
| Games     | 61                  |
| Horror    | 56                  |
| Music     | 51                  |
| New       | 63                  |
| Sci-Fi    | 61                  |
| Sports    | 74                  |
| Travel    | 57                  |
+ --------- + ------------------- +
16 rows
```

연습을 위해서 한 가지 질의를 더 살펴보기로 하자.

질의 : 고객번호 101번부터 110까지의 고객들에 대하여 각각 대여금의 지불 횟수, 총 지불금, 지불금의 최대값을 검색하라.

```
select customer.customer_id, count(amount),
        sum(amount), max(amount)
from customer, payment
where customer.customer_id = payment.customer_id
      and customer.customer_id between 101 and 110
group by customer.customer_id;
```

질의 결과

```
> select customer.customer_id, count(amount), sum(amount), max(amount)
 from customer, payment
 where customer.customer_id = payment.customer_id
       and customer.customer_id between 101 and 110
 group by customer.customer_id
+ --------------- + ------------------ + ---------------- + ---------------- +
| customer_id     | count(amount)      | sum(amount)      | max(amount)      |
+ --------------- + ------------------ + ---------------- + ---------------- +
| 101             | 24                 | 96.76            | 9.99             |
| 102             | 33                 | 137.67           | 9.99             |
| 103             | 31                 | 146.69           | 9.99             |
| 104             | 24                 | 92.76            | 10.99            |
| 105             | 26                 | 120.74           | 9.99             |
| 106             | 23                 | 100.77           | 8.99             |
```

```
| 107            | 30               | 126.70          | 7.99           |
| 108            | 30               | 132.70          | 8.99           |
| 109            | 26               | 107.74          | 7.99           |
| 110            | 14               | 59.86           | 8.99           |
+ ---------------- + ------------------ + ---------------- + ---------------- +
10 rows
```

질의 결과를 살펴보면 고객번호 101번부터 110번까지 10명의 고객에 대하여 각각의 대여금 지불 횟수와 총 지불금 및 1회에 최대로 지불한 금액이 검색되는 것을 알 수 있다.

GROUP BY 구문과 앞에서 설명한 ORDER BY 구문을 결합하여 사용하는 것도 가능하다.

위의 질의 결과를 총 지불금이 많은 고객부터 출력되도록 하는 경우에는 다음과 같이 SQL 질의문을 작성한다.

질의 : 고객번호 101번부터 110까지의 고객들에 대하여 각각 대여금의 지불 횟수, 총 지불금, 지불금의 최대값을 검색하되 총 지불금이 큰 값부터 내림차순으로 출력하라.

```
select customer.customer_id, count(amount),
       sum(amount), max(amount)
from customer, payment
where customer.customer_id = payment.customer_id
      and customer.customer_id between 101 and 110
group by customer.customer_id
order by sum(amount) DESC;
```

질의 결과

```
> select customer.customer_id, count(amount), sum(amount), max(amount)
from customer, payment
where customer.customer_id = payment.customer_id
      and customer.customer_id between 101 and 110
group by customer.customer_id
order by sum(amount) DESC
+ --------------- + ------------------ + ---------------- + ---------------- +
| customer_id     | count(amount)      | sum(amount)      | max(amount)      |
+ --------------- + ------------------ + ---------------- + ---------------- +
| 103             | 31                 | 146.69           | 9.99             |
| 102             | 33                 | 137.67           | 9.99             |
| 108             | 30                 | 132.70           | 8.99             |
| 107             | 30                 | 126.70           | 7.99             |
| 105             | 26                 | 120.74           | 9.99             |
| 109             | 26                 | 107.74           | 7.99             |
| 106             | 23                 | 100.77           | 8.99             |
| 101             | 24                 | 96.76            | 9.99             |
| 104             | 24                 | 92.76            | 10.99            |
| 110             | 14                 | 59.86            | 8.99             |
+ --------------- + ------------------ + ---------------- + ---------------- +
10 rows
```

4) HAVING

GROUP BY는 특정 속성 값을 기준으로 집계 함수를 적용할 수 있다는 점에서 매우 유용한 구문이다.

그러나 경우에 따라서는 집계 함수를 적용하려는 그룹에 대하여 좀 더 다양한 형태의 제한을 가하는 것이 필요한 경우도 있다.

다음과 같은 질의를 생각해보자.

질의 : 총 지불금이 $170이상인 고객들에 대하여 각각 대여금의 지불 횟수, 총 지불금, 지불금의 최대값을 검색하라.

이 질의를 처리하기 위하여 다음과 같이 SQL 질의문을 작성할 수 있다.

```
select customer.customer_id, count(amount),
        sum(amount), max(amount)
from customer, payment
where customer.customer_id = payment.customer_id
      and sum(amount) >= 170
group by customer.customer_id;
```

그러나 이 질의문을 수행하면 다음과 같은 에러 메시지를 얻게 된다.

Error Code: 1111. Invalid use of group function

이러한 에러 메시지가 나타나는 까닭은 집계 함수를 구하는 그룹에 대한 조건이 잘 못 기술되었기 때문이다.

특별히, 집계 함수를 적용할 그룹에 대하여 제한을 가하고자 할 때에는 HAVING 구문을 사용해야 한다.

즉, 위의 질의에 대한 올바른 SQL 질의문은 다음과 같다.

```
select customer.customer_id, count(amount),
        sum(amount), max(amount)
from customer, payment
where customer.customer_id = payment.customer_id
group by customer.customer_id
having sum(amount) >= 170;
```

질의 결과

```
> select customer.customer_id, count(amount), sum(amount), max(amount)
from customer, payment
where customer.customer_id = payment.customer_id
group by customer.customer_id
having sum(amount) >= 170
+ ---------------- + ------------------ + ---------------- + ---------------- +
| customer_id      | count(amount)      | sum(amount)      | max(amount)      |
+ ---------------- + ------------------ + ---------------- + ---------------- +
| 137              | 39                 | 194.61           | 9.99             |
| 144              | 42                 | 195.58           | 9.99             |
| 148              | 46                 | 216.54           | 10.99            |
| 176              | 37                 | 173.63           | 8.99             |
| 178              | 39                 | 194.61           | 10.99            |
| 181              | 34                 | 174.66           | 9.99             |
| 236              | 42                 | 175.58           | 8.99             |
| 259              | 33                 | 170.67           | 10.99            |
```

```
| 459            | 38               | 186.62          | 10.99          |
| 468            | 39               | 175.61          | 10.99          |
| 469            | 40               | 177.60          | 10.99          |
| 526            | 45               | 221.55          | 10.99          |
+ ---------------- + ------------------ + ---------------- + ---------------- +
12 rows
```

연습을 위하여 다음과 같은 질의도 생각해 보자.

질의 : 영화 출연 횟수가 35회 이상인 배우의 이름과 출연 횟수를 검색하되 출연 횟수가 높은 배우부터 내림차순으로 정렬하고, 출연 횟수가 동일한 경우에는 이름의 오름차순으로 정렬하여 출력하라.

```
select first_name, last_name, count(film_id)
from actor, film_actor
where actor.actor_id = film_actor.actor_id
group by actor.actor_id
having count(film_id) >= 35
order by count(film_id) desc, first_name;
```

질의 결과

```
> select first_name, last_name, count(film_id)
from actor, film_actor
where actor.actor_id = film_actor.actor_id
group by actor.actor_id
having count(film_id) >= 35
```

```
order by count(film_id) desc, first_name

+ --------------- + -------------- + ------------------- +

| first_name      | last_name      | count(film_id)      |

+ --------------- + -------------- + ------------------- +

| GINA            | DEGENERES      | 42                  |

| WALTER          | TORN           | 41                  |

| MARY            | KEITEL         | 40                  |

| MATTHEW         | CARREY         | 39                  |

| SANDRA          | KILMER         | 37                  |

| SCARLETT        | DAMON          | 36                  |

| ANGELA          | WITHERSPOON    | 35                  |

| GROUCHO         | DUNST          | 35                  |

| HENRY           | BERRY          | 35                  |

| UMA             | WOOD           | 35                  |

| VAL             | BOLGER         | 35                  |

| VIVIEN          | BASINGER       | 35                  |

+ --------------- + -------------- + ------------------- +

12 rows
```

마) 중첩 질의문

중첩 질의는 SELECT 문의 결과로 반환되는 결과 릴레이션을 또 다른 SELECT 문의 WHERE절에 사용하여 조건의 일부로 활용하는 질의문 형태를 의미한다.

이 때, SELECT문의 WHERE절에 사용되어 조건의 일부로 활용되는 SELECT문을 부질의(sub query)라고 한다.

중첩 질의문의 일반적인 형식은 다음과 같다.

SELECT <속성 리스트>

FROM <릴레이션 리스트>

WHERE <조건 리스트> AND

속성 <비교연산자> (SELECT <속성 리스트>

FROM <릴레이션 리스트>

WHERE <조건 리스트>)

다음과 같은 질의문을 생각해 보자.

질의 : Argentina에 속한 도시 이름을 검색하라.

이 질의는 다음의 두가지 질의로 나누어질 수 있다.

질의 1: Argentina의 country_id를 구하라.

질의 2: city 릴레이션에서 질의 1에서 구한 country_id와 동일한 country_id 값을 가지는 도시의 이름을 검색하라.

질의 1과 2에 대한 SQL 질의문은 각각 다음과 같다.

질의 1의 SQL 질의문

```
select country_id
from country
where country = 'Argentina'
```

질의 2의 SQL 질의문

```
select city
from city
where country_id = <질의 1의 결과>
```

이제 이 두가지 SQL 질의문을 이용하여 중첩 질의문을 만들어보기로 하자.

```
select city
from city
where country_id = (select country_id
                    from country
                    where country = 'Argentina');
```

질의 결과

```
> select city
from city
where country_id = (select country_id
                    from country
                    where country = 'Argentina')
+ --------- +
| city      |
+ --------- +
| Almirante Brown |
| Avellaneda |
| Baha Blanca |
| Crdoba    |
| Escobar   |
| Ezeiza    |
```

```
| La Plata  |
| Merlo     |
| Quilmes   |
| San Miguel de Tucumn |
| Santa F   |
| Tandil    |
| Vicente Lpez |
+ --------- +
13 rows
```

이 질의는 이미 앞에서 동등 조인에 대한 설명을 할 때에 살펴본 바 있다.

즉, 이 질의는 동등 조인을 사용하려 처리할 수도 있고, 지금과 같이 중첩 질의문을 사용하여 처리할 수도 있다.

본 질의에서는 부질의의 결과가 하나의 속성에 하나의 투플만을 가지는 릴레이션이었다.

다시 말하면, 부질의의 결과가 하나의 속성값이다.

그렇기 때문에 단순하게 “속성 =”라는 표현을 사용하는 것이 가능하였다.

만일 부질의의 결과가 여러 투플인 경우에도 문제없이 수행될 수 있는지 살펴보자.

다음의 질의를 생각해 보자.

질의 : Argentina 또는 Thailand에 속한 도시 이름을 검색하라.

이 질의도 앞에서와 유사한 방법으로 다음의 두가지 질의로 나누어질 수 있다.

질의 1: Argentina와 Thailand의 country_id를 구하라.
질의 2: city 릴레이션에서 질의 1에서 구한 country_id와 동일한 country_id 값을 가지는 도시의 이름을 검색하라.

이제 이 두가지 SQL 질의문을 이용하여 다음과 같은 중첩 질의문을 만들 수 있다.

```
select city
from city
where country_id = (select country_id
                    from country
                    where country = 'Argentina'
                          or country = 'Thailand');
```

그러나 이 질의를 수행하면 다음과 같은 에러 메시지가 출력된다.

Error Code: 1242. Subquery returns more than 1 row

이 메시지는 부질의의 결과가 두 개 이상의 투플을 반환한다는 의미이다.
즉, 부질의의 결과가 두 개 이상의 투플을 반환하는 경우에는 지금과는 다른 특별한 비교 연산이 필요하다.

여러 개의 투플이 반환되는 부질의를 사용하는 경우에는 비교 연산자와 함께 IN, ANY, ALL, EXIST 등의 연산자를 함께 사용해야 한다.

이제 각각의 연산자에 대하여 설명한다.

1) IN 연산자

IN 연산자는 이미 앞에서 한번 설명한 바 있다.

IN 연산자는 해당 속성 값이 IN 연산자를 따라오는 집합의 어떤 한 원소와 동일한 값을 갖는지 여부를 가리는 연산자이다.

다음과 같은 질의를 생각해보자.

질의 : Argentina 또는 Thailand에 속한 도시 이름을 검색하라.

이 질의는 앞에서 언급하였던 질의로서 다음의 두가지 질의로 나눌 수 있다.

질의 1: Argentina와 Thailand의 country_id를 구하라.
질의 2: city 릴레이션에서 질의 1에서 구한 country_id와 동일한 country_id 값을 가지는 도시의 이름을 검색하라.

그리고 이 두가지 질의문을 활용하고 IN 연산자를 이용하여 다음과 같은 중첩 질의문을 만들 수 있다.

```
select city
from city
where country_id IN (select country_id
```

```
                    from country
                    where country = 'Argentina'
                          or country = 'Thailand');
```

질의 결과

```
> select city
from city
where country_id IN (select country_id
                     from country
                     where country = 'Argentina'
                           or country = 'Thailand')
+ --------- +
| city      |
+ --------- +
| Almirante Brown |
| Avellaneda |
| Baha Blanca |
| Crdoba    |
| Escobar   |
| Ezeiza    |
| La Plata  |
| Merlo     |
| Nakhon Sawan |
| Pak Kret  |
| Quilmes   |
| San Miguel de Tucumn |
| Santa F   |
```

```
| Songkhla  |
| Tandil    |
| Vicente Lpez |
+ --------- +
16 rows
```

질의 결과를 살펴보면 우리가 원한 바와 같이 Argentina와 Thailand의 도시가 모두 검색되는 것을 알 수 있다.

우리는 앞에서 차집합 연산자인 MINUS에 대하여 설명한 바 있다.

그러나 MySQL에서는 MINUS 연산자를 허용하지 않는다.

이 경우 IN 연산자를 사용한 중첩 질의문으로 차집합 연산을 수행할 수 있다.

다음 질의에 대하여 생각해 보자.

질의 : 액션 영화를 대여한 고객들 중에서 코메디 영화나 가족 영화는 한번도 대여하지 않았던 고객의 ID를 검색하라.

이 질의는 다음의 두 질의를 수행한 후 차집합 연산을 수행하면 처리할 수 있다.

질의 1: 액션 영화를 대여한 고객들의 ID를 검색하라.

질의 2: 코메디 영화나 가족영화를 대여한 고객 ID를 검색하라.

질의문 1

```
select distinct customer.first_name, customer.last_name
```

```
from customer, rental, film_category, category, inventory
where customer.customer_id = rental.customer_id
   and rental.inventory_id = inventory.inventory_id
   and inventory.film_id = film_category.film_id
   and film_category.category_id = category.category_id
   and category.name = 'Action';
```

질의 결과

```
> select distinct customer.first_name, customer.last_name
from customer, rental, film_category, category, inventory
where customer.customer_id = rental.customer_id
   and rental.inventory_id = inventory.inventory_id
   and inventory.film_id = film_category.film_id
   and film_category.category_id = category.category_id
   and category.name = 'Action'
+ --------------- + -------------- +
| first_name      | last_name      |
+ --------------- + -------------- +
| JANE            | BENNETT        |
| DEBRA           | NELSON         |
중간 생략
| COURTNEY        | DAY            |
| TINA            | SIMMONS        |
| JACQUELINE      | LONG           |
| DANNY           | ISOM           |
| VIVIAN          | RUIZ           |
```

```
| REGINA          | BERRY          |
+ --------------- + -------------- +
510 rows
```

질의문 2

```
select distinct customer.first_name, customer.last_name
from customer, rental, film_category, category, inventory
where customer.customer_id = rental.customer_id
   and rental.inventory_id = inventory.inventory_id
   and inventory.film_id = film_category.film_id
   and film_category.category_id = category.category_id
   and (category.name = 'Comedy'
        OR category.name = 'Family');
```

질의 결과

```
> select distinct customer.first_name, customer.last_name
from customer, rental, film_category, category, inventory
where customer.customer_id = rental.customer_id
   and rental.inventory_id = inventory.inventory_id
   and inventory.film_id = film_category.film_id
   and film_category.category_id = category.category_id
   and (category.name = 'Comedy' OR category.name = 'Family')
+ --------------- + -------------- +
| first_name      | last_name      |
+ --------------- + -------------- +
| FRED            | WHEAT          |
```

```
| DEAN            | SAUER          |
| CRAIG           | MORRELL        |
| PRISCILLA       | LOWE           |
중간 생략
| JAY             | ROBB           |
| WENDY           | HARRISON       |
| MILDRED         | BAILEY         |
| DANNY           | ISOM           |
+ --------------- + -------------- +
581 rows
```

이제 질의문 1과 질의문 2에 대하여 차집합 대신 다음과 같은 질의문을 생각해 보자.

최종 질의문

```
select distinct customer.first_name, customer.last_name
from customer, rental, film_category, category, inventory
where customer.customer_id = rental.customer_id
   and rental.inventory_id = inventory.inventory_id
   and inventory.film_id = film_category.film_id
   and film_category.category_id = category.category_id
   and category.name = 'Action'
   and (customer.customer_id) NOT IN
   (select distinct customer.customer_id
    from customer, rental, film_category, category, inventory
    where customer.customer_id = rental.customer_id
       and rental.inventory_id = inventory.inventory_id
```

```
  and inventory.film_id = film_category.film_id
  and film_category.category_id = category.category_id
  and (category.name = 'Comedy'
       OR category.name = 'Family'))
```

질의 결과

```
> select distinct customer.first_name, customer.last_name
from customer, rental, film_category, category, inventory
where customer.customer_id = rental.customer_id
   and rental.inventory_id = inventory.inventory_id
   and inventory.film_id = film_category.film_id
   and film_category.category_id = category.category_id
   and category.name = 'Action'
   and (customer.customer_id) NOT IN
   (select distinct customer.customer_id
from customer, rental, film_category, category, inventory
where customer.customer_id = rental.customer_id
   and rental.inventory_id = inventory.inventory_id
   and inventory.film_id = film_category.film_id
   and film_category.category_id = category.category_id
   and (category.name = 'Comedy' OR category.name = 'Family'))
+ --------------- + -------------- +
| first_name      | last_name      |
+ --------------- + -------------- +
| DALE            | RATCLIFF       |
| MATTHEW         | MAHAN          |
| SHEILA          | WELLS          |
```

```
| JIMMY          | SCHRADER       |
| CHAD           | CARBONE        |
| SCOTT          | SHELLEY        |
| MARION         | OCAMPO         |
| ROBERTA        | HARPER         |
| TARA           | RYAN           |
| PETER          | MENARD         |
| LOUISE         | JENKINS        |
| SUSAN          | WILSON         |
| ANITA          | MORALES        |
| DOLORES        | WAGNER         |
| BERNARD        | COLBY          |
| SERGIO         | STANFIELD      |
| LEONARD        | SCHOFIELD      |
| TERRY          | GRISSOM        |
+ -------------- + -------------- +
18 rows
```

질의 결과는 차집합 연산을 수행한 것과 동일하다는 것을 확인할 수 있다.

2) ANY 연산자

ANY 연산자는 속성값이 ANY 연산자 뒤에 따라오는 집합의 원소 중 비교 조건이 성립하는 원소가 하나라도 있으면 true를 그렇지 않으면 false를 반환하는 연산자이다.

다음과 같은 질의를 생각해보자.

질의 : 고객 'April Burns'가 지불한 대여료 중에서 가장 큰 금액과 가장 작은 금액을 제외한 나머지 금액들을 검색하라.

이 질의는 차집합 연산을 이용해도 처리할 수 있지만 다음과 같이 ANY 연산자를 사용하는 중첩 질의문으로 처리할 수도 있다.

```
select distinct amount
from payment, customer
where payment.customer_id = customer.customer_id
      and customer.first_name = 'April'
      and customer.last_name = 'Burns'
      and amount < ANY (select distinct amount
                        from payment, customer
                        where payment.customer_id = customer.customer_id
                              and customer.first_name = 'April'
                              and customer.last_name = 'Burns')
      and amount > ANY (select distinct amount
                        from payment, customer
                        where payment.customer_id = customer.customer_id
                              and customer.first_name = 'April'
                              and customer.last_name = 'Burns');
```

질의 결과

```
> select distinct amount
from payment, customer
where payment.customer_id = customer.customer_id
```

```
        and customer.first_name = 'April'
        and customer.last_name = 'Burns'
        and amount < ANY (select distinct amount
                          from payment, customer
                          where payment.customer_id = customer.customer_id
                                and customer.first_name = 'April'
                                and customer.last_name = 'Burns')
        and amount > ANY (select distinct amount
                          from payment, customer
                          where payment.customer_id = customer.customer_id
                                and customer.first_name = 'April'
                                and customer.last_name = 'Burns')
+ ----------- +
| amount      |
+ ----------- +
| 8.99        |
| 1.99        |
| 2.99        |
| 4.99        |
| 7.99        |
| 5.99        |
+ ----------- +
6 rows
```

질의 결과 $1.99부터 $8.99까지 총 6 종류의 지불 금액이 검색되었다.

질의 결과의 정확성을 확인하기 위하여 다음의 질의를 수행해 보자.

질의 : 고객 'April Burns'가 지불한 대여료 금액들을 검색하라.

```
select distinct amount
from payment, customer
where payment.customer_id = customer.customer_id
      and customer.first_name = 'April'
      and customer.last_name = 'Burns'
order by amount;
```

질의 결과

```
> select distinct amount
from payment, customer
where payment.customer_id = customer.customer_id
      and customer.first_name = 'April'
      and customer.last_name = 'Burns'
order by amount
+ ---------- +
| amount     |
+ ---------- +
| 0.99       |
| 1.99       |
| 2.99       |
| 4.99       |
| 5.99       |
| 7.99       |
```

```
| 8.99        |
| 9.99        |
+ ----------- +
8 rows
```

질의 결과 최저 지불금은 $0.99, 최대 지불금은 $9.99이었던 것을 알 수 있다.

3) ALL 연산자

ALL 연산자는 속성값이 ALL 연산자 뒤에 따라오는 집합의 모든 원소와 비교하여 항상 비교 조건이 성립하는 경우에만 true를 그렇지 않으면 false를 반환하는 연산자이다.

다음과 같은 질의를 생각해보자.

질의 : 고객 'Lori Wood'가 지불한 대여료 중에서 가장 큰 금액보다 더 많은 대여료를 지불한 경험이 있는 고객의 이름을 검색하라.

이 질의는 집계 함수 MAX()를 이용하여 처리할 수도 있지만 다음과 같이 ALL 연산자를 사용할 수도 있다.

```
select first_name, last_name
from payment, customer
where payment.customer_id = customer.customer_id
      and amount > ALL (select amount
```

```
                    from payment, customer
                    where payment.customer_id = customer.customer_id
                          and customer.first_name = 'Lori'
                          and customer.last_name = 'Wood');
```

질의 결과

```
> select first_name, last_name
from payment, customer
where payment.customer_id = customer.customer_id
      and amount > ALL (select amount
                        from payment, customer
                        where payment.customer_id = customer.customer_id
                              and customer.first_name = 'Lori'
                              and customer.last_name = 'Wood')
+ --------------- + -------------- +
| first_name      | last_name      |
+ --------------- + -------------- +
| KAREN           | JACKSON        |
| VICTORIA        | GIBSON         |
| VANESSA         | SIMS           |
| ALMA            | AUSTIN         |
| ROSEMARY        | SCHMIDT        |
| TANYA           | GILBERT        |
| RICHARD         | MCCRARY        |
| NICHOLAS        | BARFIELD       |
| KENT            | ARSENAULT      |
```

```
| TERRANCE        | ROUSH           |
+ --------------- + --------------- +
10 rows
```

엄밀한 의미에서 중첩 질의문은 WHERE 절에만 SELECT 문이 올 수 있는 것이 아니라, FROM 절에도 SELECT 문이 올 수 있다.

즉, 이미 만들어져 있는 릴레이션에 대해서만 질의를 하는 것이 아니라 검색 결과로 반환되는 릴레이션에 대해서도 여러 가지 관계 연산을 할 수 있다는 의미이다.

다음과 같은 질의문을 생각해보자.

이 질의는 배우들의 출연 횟수에 대한 비교 연산이 필요한데 출연 횟수를 구하기 위해서는 집계 함수를 사용해야 하므로 집계 함수 값에 대한 비교 연산이 필요하다.

이러한 문제를 다음과 같이 FROM 절에 중첩 질의문을 사용하면 쉽게 해결할 수 있다.

질의 : 가장 많은 영화에 출연한 배우의 이름과 출연 횟수를 검색하라.

```
Select first_name, last_name, count
from actor,
     (select film_actor.actor_id, count(film_actor.film_id) as count
      from film_actor
      group by film_actor.actor_id) AS C
where actor.actor_id = C.actor_id
```

```
and C.count >= ALL (select count(film_actor.film_id) as count
                    from film_actor
                    group by film_actor.actor_id)
```

질의 결과

```
> select first_name, last_name, count
from actor,
     (select film_actor.actor_id, count(film_actor.film_id) as count
      from film_actor
      group by film_actor.actor_id) AS C
where actor.actor_id = C.actor_id
      and C.count >= ALL (select count(film_actor.film_id) as count
                          from film_actor
                          group by film_actor.actor_id)
+ --------------- + -------------- + ---------- +
| first_name      | last_name      | count      |
+ --------------- + -------------- + ---------- +
| GINA            | DEGENERES      | 42         |
+ --------------- + -------------- + ---------- +
1 rows
```

바) 삽입을 위한 SQL 질의문

본 절에서는 릴레이션에 새로운 투플을 삽입하는 SQL 질의문에 대하여 설명한다.

삽입을 위한 SQL 질의문의 기본 형식은 다음과 같다.

INSERT INTO 릴레이션 이름(속성 이름 리스트)
VALUES (속성 값 리스트)

INSERT문을 실행하면 릴레이션에 투플을 하나씩 입력할 수 있다.

이 때, VALLUES를 따라오는 속성 값들은 릴레이션 이름을 따라오는 속성 이름 리스트의 순서에 따라 순서대로 입력된다.

만일 모든 속성 값들이 원래 릴레이션의 속성 순서와 동일한 순서로 나열된 경우에는 굳이 속성 이름 리스트를 기술하지 않고 단지 릴레이션 이름을 기술하는 것만으로도 충분하다.

다음 질의어를 살펴보자.

질의 : 다음 값을 가지는 신규 고객 정보를 입력보자.

customer_id : 600
store_id : 2
first_name : James
last_name : Dean
email : jamesdean@sakilacustomer.org
address_id : 604
active : 1
create_date : 현재시간
last_upate : 현재시간

이 질의를 SQL 질의문으로 작성하면 다음과 같다.

```
insert into customer
values(600, 2, 'James', 'Dean', 'jamesdean@sakilacustomer.org',
       604, 1, now(), now());
```

이 질의를 수행하면 단순히 다음과 같은 메시지가 출력된다.

```
1 row(s) affected
```

이제 원하는 투플이 삽입되었는지 확인해 보기로 하자.

```
select *
from customer
where customer_id = 600;
```

위의 SQL 질의문을 수행하여 보자.

```
> select *
from customer
where customer_id = 600
+ ---------------- + ------------- + --------------- + -------------- + ----------
+ --------------- + ------------ + ---------------- + ---------------- +
| customer_id      | store_id      | first_name      | last_name      | email      | address_id
| active      | create_date      | last_update      |
+ ---------------- + ------------- + --------------- + -------------- + ----------
+ --------------- + ------------ + ---------------- + ---------------- +
| 600                    | 2                  | James              | Dean              |
jamesdean@sakilacustomer.org | 604                | 1             | 2011-11-10 17:20:52 |
2011-11-10 17:20:52 |
+ ---------------- + ------------- + --------------- + -------------- + ----------
+ --------------- + ------------ + ---------------- + ---------------- +
1 rows
```

우리가 삽입하기를 원했던 속성 값이 잘 입력된 것을 확인할 수 있다.

SQL 질의문에서 now() 함수는 현재 날짜와 시간을 반환하는 함수이다.

만일 삽입하려는 투플에서 일부 속성 값을 알지 못 하는 경우에는 NULL을 입력해야 할 것이다.

다음과 같은 SQL 질의문을 수행해 보자.

```
insert into customer
values(600, 2, 'James', 'Dean', 'jamesdean@sakilacustomer.org',
        604, 1, NULL, now());
```

이 경우 다음과 같은 메시지를 얻게 된다.

```
Error Code: 1064. You have an error in your SQL syntax; check
the manual that corresponds to your MySQL server version for
the right syntax to use near '@' at line 2
```

에러 메세지의 의미를 확인하기 위하여 다음과 같은 명령어를 입력해 보자.

```
desc customer;
```

이 명령어는 customer 릴레이션의 스키마를 검색하라는 명령어이다.

이 명령어의 수행 결과는 다음과 같다.

```
> desc customer

+ ---------- + --------- + --------- + -------- + ------------ + ---------- +
| Field      | Type      | Null      | Key      | Default      | Extra      |
+ ---------- + --------- + --------- + -------- + ------------ + ---------- +
| customer_id | smallint(5) unsigned | NO        | PRI      |              | auto_increment |
| store_id    | tinyint(3) unsigned | NO        | MUL      |              |            |
| first_name | varchar(45) | NO        |          |              |            |
| last_name   | varchar(45) | NO        | MUL      |              |            |
| email       | varchar(50) | YES       |          |              |            |
| address_id | smallint(5) unsigned | NO        | MUL      |              |            |
| active      | tinyint(1) | NO        |          | 1            |            |
| create_date | datetime   | NO        |          |              |            |
| last_update | timestamp | NO           |              | CURRENT_TIMESTAMP | on update
CURRENT_TIMESTAMP |
+ ---------- + --------- + --------- + -------- + ------------ + ---------- +
9 rows
```

수행 결과를 살펴보면 email을 제외한 모든 속성이 NULL을 가질 수 없도록 설정되어 있는 것을 확인 할 수 있다.

즉, 위에서 확인한 에러 메시지는 NULL이 입력될 수 없는 속성에 NULL을 삽입하였기 때문에 발생한 에러 메시지인 것이다.

그러므로 이번에는 NULL을 삽입할 수 있는 email 속성이 NULL인 투플을 삽입해 보기로 하자.

```
insert into customer
values(600, 2, 'James', 'Dean', NULL, 604, 1, now(), now());
```

```
> select *
from customer
where customer_id = 600
```

```
+ ---------------- + ------------- + --------------- + -------------- + ----------
+ --------------- + ----------- + ---------------- + ---------------- +
| customer_id       | store_id       | first_name       | last_name       | email       | address_id
| active       | create_date       | last_update       |
+ ---------------- + ------------- + --------------- + -------------- + ----------
+ --------------- + ----------- + ---------------- + ---------------- +
| 600               | 2              | James            | Dean            |             | 604
 | 1           | 2011-11-28 20:02:41 | 2011-11-28 20:02:41 |
+ ---------------- + ------------- + --------------- + -------------- + ----------
+ --------------- + ----------- + ---------------- + ---------------- +
1 rows
```

질의 결과를 살펴보면 email 속성에 값이 입력되지 않은 것을 확인 할 수 있다.

어떤 속성에 NULL을 입력하고자 할 때에는 INSERT문에서 속성 이름을 나열할 때에 NULL이 입력될 속성의 이름을 생략하는 방법도 있다. 다음과 같은 SQL 질의문을 실행해 보자.

```
insert into customer(customer_id, store_id, first_name,
                     last_name, address_id, active,
                     create_date, last_update)
values(601, 1, 'Tom', 'Cruise', 600, 1, now(), now());
```

이제 결과를 확인해 보자.

```
> select *
from customer
where customer_id = 601
+ ---------------- + ------------- + --------------- + -------------- + ----------
+ --------------- + ----------- + ---------------- + ---------------- +
| customer_id       | store_id       | first_name       | last_name       | email       | address_id
| active       | create_date       | last_update       |
+ ---------------- + ------------- + --------------- + -------------- + ----------
```

```
+ --------------- + ----------- + ---------------- + ---------------- +
| 601             | 1           | Tom              | Cruise           |            | 600
 | 1          | 2011-11-28 20:13:40 | 2011-11-28 20:13:40 |
+ ---------------- + ------------- + --------------- + -------------- + ----------
+ --------------- + ----------- + ---------------- + ---------------- +
1 rows
```

질의 수행 결과에서 email 속성에 NULL이 입력되어 있는 것을 확인할 수 있다.

만일 속성 이름의 리스트를 기술할 때에 이름이 잘못 입력되거나 속성의 개수가 맞지 않는 경우에는 다음과 유사한 에러 메시지가 출력된다.

Error Code: 1136. Column count doesn't match value count at row 1v

또는

Error Code: 1054. Unknown column 'address_ic' in 'field list'

그러므로 위와 같은 메시지가 출력될 때에는 속성 이름 리스트를 다시 확인해 보는 것이 필요하다.

또한, 새로운 투플을 삽입하는 경우 참조 무결성 제약 조건을 위배하는 현상이 발생할 수 있다.

외래키로 설정된 속성의 경우 참조되는 릴레이션에 입력된 속성 값을 가진 투플이 없으면 참조 무결성 제약 조건이 위배된다.

다음과 같은 SQL 질의문을 수행해 보자.

```
insert into customer(customer_id, store_id, first_name,
                     last_name, address_id, active,
                     create_date, last_update)
values(602, 1, 'Tom', 'Cruise', 700, 1, now(), now());
```

이 SQL 질의문에서 삽입되는 투플은 address_id의 속성 값이 700이다.

address_id는 address 릴레이션을 참조하는 속성이지만 address 릴레이션에는 address_id의 속성값이 700인 투플이 존재하지 않는다.

그러므로 위와 같은 질의문을 수행하는 것은 참조 무결성 제약 조건을 위배하게 된다.

이 경우 아래와 같은 에러 메시지가 출력된다.

```
Error Code: 1452. Cannot add or update a child row: a foreign key constraint fails (`sakila`.`customer`, CONSTRAINT `fk_customer_address` FOREIGN KEY (`address_id`) REFERENCES `address` (`address_id`) ON UPDATE CASCADE)
```

즉, 참조 무결성 제약 조건을 지키기 위해 투플의 삽입이 거부되는 것을 알 수 있다.

새로운 투플이 삽입되는 경우 입력되는 투플의 기본 키 속성값과 동일한 값을 가진 투플이 이미 존재할 수 있다.

이 경우에는 키 제약조건을 위배하게 된다.

다음과 같은 SQL 질의문을 수행해 보자.

```
insert into customer(customer_id, store_id, first_name,
                     last_name, address_id, active,
                     create_date, last_update)
values(500, 1, 'Tom', 'Cruise', 600, 1, now(), now());
```

이 질의문에서 새로 입력되는 투플의 키 속성인 customer_id 속성 값은 500이다.

그러나 customer 릴레이션에서 customer_id의 속성 값이 500인 고객의 이름은 'Reginald Kinder'로서 이미 존재한다.

이 경우 투플을 삽입하려는 질의문을 수행하면 다음과 같은 에러 메시지가 출력된다.

```
Error Code: 1062. Duplicate entry '500' for key 'PRIMARY'
```

에러 메시지의 의미는 기본 키 속성 값 500은 중복되므로 삽입이 거부된 것을 의미한다.

일반적으로 기본 키 속성에 NULL을 입력하는 경우에는 엔티티 무결성 제약 조건에 의해 삽입이 거부된다.

그러나 MySQL에서는 기본 키 속성에 NULL을 입력하는 경우 자동으로 중복되지 않는 키 값을 생성하여 입력해 주는 경우도 있다.

이 경우에 대해서는 개별적으로 확인해 보기로 한다.

이제 다른 릴레이션으로부터 투플의 속성 값을 가져와서 여러 투플을 한꺼번에 삽입하는 방법에 대하여 설명한다.

설명을 위하여 먼저 다음과 같은 릴레이션 하나를 생성하기로 하자.

릴레이션 이름 : film_actor_category

속성 : actor_id

actor_first_name

actor_last_name

film_id

film_title

category_id

category_name

이를 위하여 다음과 같은 SQL 질의문을 수행하도록 하자.

이 질의문은 삽입 질의문을 연습하기 위한 임시 릴레이션 film_actor_category를 생성하는 질의문이다.

이 릴레이션은 배우 이름, 배우의 출련 영화 이름, 그리고 그 영화의 장르를 기록한 릴레이션이다.

기본키는 actor_id, film_id, category_id이다.

```
CREATE  TABLE `sakila`.`film_actor_category` (
  `actor_id` SMALLINT NOT NULL ,
  `actor_first_name` VARCHAR(45) NOT NULL ,
  `actor_last_name` VARCHAR(45) NOT NULL ,
  `film_id` SMALLINT NOT NULL ,
  `film_title` VARCHAR(255) NOT NULL ,
  `category_id` TINYINT NOT NULL ,
  `category_name` VARCHAR(25) NOT NULL ,
  PRIMARY KEY (`film_id`, `actor_id`, `category_id`));
```

이제 다른 릴레이션으로부터 투플의 속성 값을 가져와서 여러 투플을 한꺼번에 삽입하는 SQL 질의문을 작성해보자.

질의 : 모든 영화에 대하여 출연한 배우와 영화 장르를 표현하는 film_actor_category 릴레이션의 상태를 만들어라.

```
INSERT INTO film_actor_category
select actor.actor_id, actor.first_name, actor.last_name,
       film.film_id, film.title, category.category_id, category.name
from actor, film, category, film_actor, film_category
where actor.actor_id = film_actor.actor_id
      and film.film_id = film_actor.film_id
      and film.film_id = film_category.film_id
      and film_category.category_id = category.category_id;
```

이제 결과를 확인하기 위하여 다음과 같은 SQL 질의를 수행해 보자.

전체 릴레이션의 상태를 확인하면 좋겠지만 지면 관계상 일부 내용만을 확인해 보기로 하자.

```
select *
from film_actor_category
where film_id between 100 and 106;
```

질의 결과에서 원래는 film, actor, category에 나뉘어 저장되어 있던 데이터들이 하나의 릴레이션에 모여 있는 것을 확인 할 수 있다.

film_actor_category 릴레이션을 생성하고 데이터를 삽입하는 질의어는 다음과 같이 하나의 문장으로도 작성할 수 있다.

```
create table as film_actor_category
select actor.actor_id, actor.first_name, actor.last_name,
        film.film_id, film.title, category.category_id, category.name
from actor, film, category, film_actor, film_category
where actor.actor_id = film_actor.actor_id
       and film.film_id = film_actor.film_id
       and film.film_id = film_category.film_id
       and film_category.category_id = category.category_id;
```

이와 같이 SQL 질의문을 작성하여 수행해도 동일한 내용을 가진 릴레이션이 생성된다.

지면 관계상 수행 결과는 생략한다.

```
> select *
from film_actor_category
where film_id between 100 and 106
```

actor_id	actor_first_name	actor_last_name	film_id	film_title	category_id	category_name
41	JODIE	DEGENERES	100	BROOKLYN DESERT	9	Foreign
62	JAYNE	NEESON	100	BROOKLYN DESERT	9	Foreign
90	SEAN	GUINESS	100	BROOKLYN DESERT	9	Foreign
125	ALBERT	NOLTE	100	BROOKLYN DESERT	9	Foreign
172	GROUCHO	WILLIAMS	100	BROOKLYN DESERT	9	Foreign
16	FRED	COSTNER	101	BROTHERHOOD BLANKET	6	Documentary
48	FRANCES	DAY-LEWIS	101	BROTHERHOOD BLANKET	6	Documentary
57	JUDE	CRUISE	101	BROTHERHOOD BLANKET	6	Documentary
62	JAYNE	NEESON	101	BROTHERHOOD BLANKET	6	Documentary
129	DARYL	CRAWFORD	101	BROTHERHOOD BLANKET	6	Documentary
158	VIVIEN	BASINGER	102	BUBBLE GROSSE	15	Sports
170	MENA	HOPPER	102	BUBBLE GROSSE	15	Sports
188	ROCK	DUKAKIS	102	BUBBLE GROSSE	15	Sports
26	RIP	CRAWFORD	103	BUCKET BROTHERHOOD	16	Travel
32	TIM	HACKMAN	103	BUCKET BROTHERHOOD	16	Travel
51	GARY	PHOENIX	103	BUCKET BROTHERHOOD	16	Travel

89	CHARLIZE	DENCH	103	BUCKET BROTHERHOOD	16	Travel
92	KIRSTEN	AKROYD	103	BUCKET BROTHERHOOD	16	Travel
193	BURT	TEMPLE	103	BUCKET BROTHERHOOD	16	Travel
82	WOODY	JOLIE	104	BUGSY SONG	9	Foreign
92	KIRSTEN	AKROYD	104	BUGSY SONG	9	Foreign
2	NICK	WAHLBERG	105	BULL SHAWSHANK	1	Action
23	SANDRA	KILMER	105	BULL SHAWSHANK	1	Action
29	ALEC	WAYNE	105	BULL SHAWSHANK	1	Action
43	KIRK	JOVOVICH	105	BULL SHAWSHANK	1	Action
116	DAN	STREEP	105	BULL SHAWSHANK	1	Action
123	JULIANNE	DENCH	105	BULL SHAWSHANK	1	Action
144	ANGELA	WITHERSPOON	105	BULL SHAWSHANK	1	Action
150	JAYNE	NOLTE	105	BULL SHAWSHANK	1	Action
1	PENELOPE	GUINESS	106	BULWORTH COMMANDMENTS	10	Games
65	ANGELA	HUDSON	106	BULWORTH COMMANDMENTS	10	Games
124	SCARLETT	BENING	106	BULWORTH COMMANDMENTS	10	Games
173	ALAN	DREYFUSS	106	BULWORTH COMMANDMENTS	10	Games

33 rows

사) 삭제를 위한 SQL 질의문

본 절에서는 투플을 삭제하는 SQL 질의문에 대하여 설명한다.

삭제를 위한 SQL 질의문의 기본 형식은 다음과 같다.

DELETE FROM 릴레이션 이름
WHERE <조건 리스트>

DELETE문을 실행하면 지정한 릴레이션에서 조건을 만족하는 모든 투플들이 삭제된다.

다음 질의어를 살펴보자.

질의 : film_actor_category 릴레이션에서 영화 BROOKLYN DESERT에 대한 투플을 모두 삭제하라.

이를 위하여 다음과 같은 SQL 질의문을 수행할 수 있다.

```
delete from film_actor_category
where film_title = 'Brooklyn Desert';
```

SQL 질의문에 문제가 dqjtdma에도 불구하고 에러가 발생할 것이다.

이는 MySQL에서 사용자가 실수로 데이터를 삭제하는 것을 방지하기 위하여 safe update mode라는 것을 설정해 두었기 때문이다.

safe update mode란 Delete문의 WHERE절에서 키 속성을 이용하지 않고 조건을 기술하는 경우 삭제가 수행되지 않도록 설정한 옵션이다.

그림 35와 같이 safe update 옵션을 해제하고 다시 SQL 질의문을 수행해 보자.

다음과 같은 메시지가 출력되며 성공적으로 투플이 삭제되었음을 알려준다.

5 row(s) affected

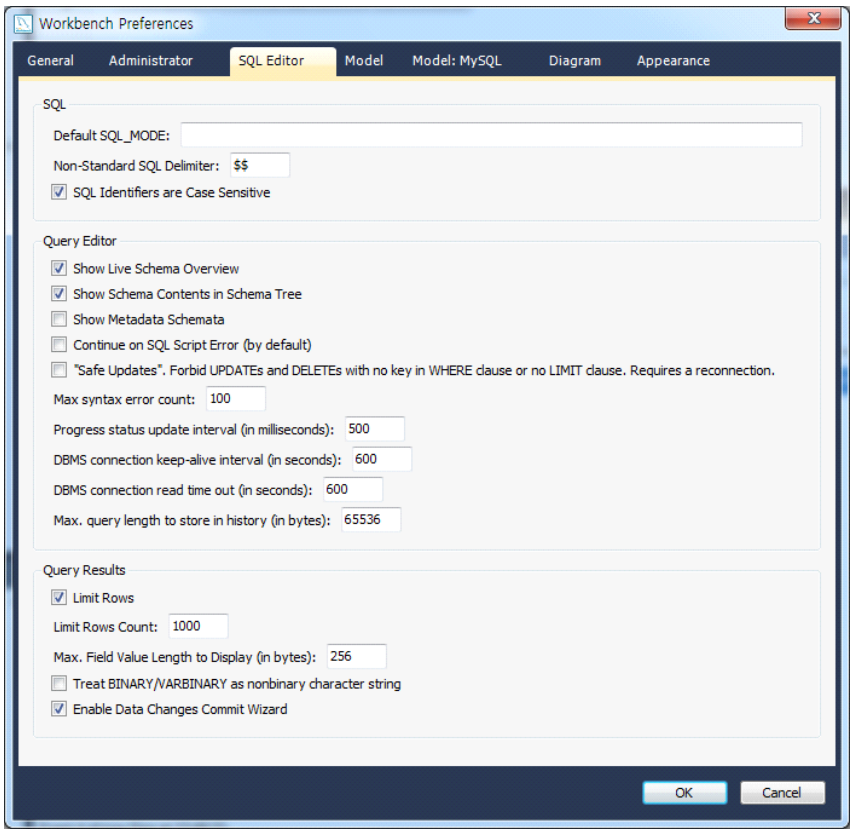

그림 35. safe update mode의 해제

이제 다음 SQL 질의문을 수행하여 삭제가 잘 이루어졌는지 확인해 보자.

```
select *
from film_actor_category
where film_title = 'Brooklyn Desert';
```

이 질의의 수행 결과로는 아무 투플도 검색되지 않는 것을 확인할 수 있다.

즉, Brooklyn Desert라는 영화에 대한 투플이 모두 삭제된 것을 의미한다.

이제 어떤 릴레이션에 속한 모든 투플을 삭제하는 SQL 질의문을 작성해 보자.

```
delete from film_actor_category;
```

이 질의문을 수행하면 film_actor_category 릴레이션의 모든 투플이 삭제된다.

투플을 삭제하면 참조 무결성 제약 조건을 위배하는 결과가 초래될 수 있다.

다음의 질의를 생각해보자.

질의 : film 릴레이션에서 영화 BROOKLYN DESERT에 대한 투플을 삭제하라.

이를 위하여 다음 SQL 질의문을 실행해 보자.

```
delete from film
where title = 'Brooklyn Desert';
```

그러면 다음과 같은 에러 메시지가 출력된다.

Error Code: 1451. Cannot delete or update a parent row: a foreign key constraint fails (`sakila`.`film_actor`, CONSTRAINT `fk_film_actor_film` FOREIGN KEY (`film_id`) REFERENCES `film` (`film_id`) ON UPDATE CASCADE)

이 메시지는 위의 질의문을 수행하면 film_actor 릴레이션에 있는 외래키 속성인 film_id에 대하여 참조 무결성 제약 조건이 위배되므로 투플을 삭제할 수 없다는 의미의 메시지이다.

아) 갱신을 위한 SQL 질의문

본 절에서는 투플을 갱신하는 SQL 질의문에 대하여 설명한다.

갱신을 위한 SQL 질의문의 기본 형식은 다음과 같다.

UPDATE 릴레이션 이름
SET <'속성 이름 = 속성 값'의 리스트>
WHERE <조건 리스트>

UPDATE문을 실행하면 지정한 릴레이션에서 조건을 만족하는 모든 투플들의 속성 값이 지정한 값으로 변경된다.

다음 질의어를 살펴보자.

질의 : 영화 ALICE FANTASIA의 대여료를 $2.99로 수정하라.

이 질의를 SQL 질의문으로 작성하면 다음과 같다.

```
update film
set rental_rate = 2.99
where title = 'Alice Fantasia';
```

이제 SQL 질의문의 실행 결과를 확인하기 위해 다음의 SQL 질의문을 수행해 보자.

```
select title, rental_rate
from film
where title = 'Alice Fantasia';
```

질의 결과

```
> select title, rental_rate
from film
where title = 'Alice Fantasia'
+ ---------- + ---------------- +
| title      | rental_rate      |
+ ---------- + ---------------- +
| ALICE FANTASIA | 2.99           |
+ ---------- + ---------------- +
1 rows
```

갱신 연산을 수행할 때에 산술 연산자를 사용할 수 있다.

다음과 같은 질의를 생각해보자.

질의 : 'A'로 시작하는 이름을 가진 영화 중 'G' 등급 영화의 대여료를 10% 인상하라.

이 질의를 SQL 질의문으로 작성하면 다음과 같다.

```
update film
set rental_rate = rental_rate * 1.1
where rating = 'G'
        and title like "A%";
```

SQL 질의문의 실행 결과를 확인하기 위해 update를 수행하기 전과 수행한 후에 각각 다음의 SQL 질의문을 수행해 보자.

질의 결과에서 조건을 만족하는 모든 투플들의 대여료가 10%씩 인상된 것을 확인할 수 있다.

```
select title, rental_rate
from film
where rating = 'G'
      and title like "A%";
```

Update 수행 전

```
> select title, rental_rate
from film
where rating = 'G'
      and title like "A%"
+ ---------- + ---------------- +
| title      | rental_rate      |
+ ---------- + ---------------- +
| ACE GOLDFINGER | 4.99            |
| AFFAIR PREJUDICE | 2.99          |
| AFRICAN EGG | 2.99             |
| ALAMO VIDEOTAPE | 0.99          |
| AMISTAD MIDSUMMER | 2.99        |
| ANGELS LIFE | 2.99             |
| ANNIE IDENTITY | 0.99          |
| ARMAGEDDON LOST | 0.99         |
| ATLANTIS CAUSE | 2.99          |
| AUTUMN CROW | 4.99             |
+ ---------- + ---------------- +
10 rows
```

Update 수행 후

```
> select title, rental_rate
from film
where rating = 'G'
      and title like "A%"
+ ---------- + ---------------- +
| title      | rental_rate      |
+ ---------- + ---------------- +
| ACE GOLDFINGER | 5.49            |
| AFFAIR PREJUDICE | 3.29          |
| AFRICAN EGG | 3.29             |
| ALAMO VIDEOTAPE | 1.09          |
| AMISTAD MIDSUMMER | 3.29        |
| ANGELS LIFE | 3.29             |
| ANNIE IDENTITY | 1.09          |
| ARMAGEDDON LOST | 1.09         |
| ATLANTIS CAUSE | 3.29          |
| AUTUMN CROW | 5.49             |
+ ---------- + ---------------- +
10 rows
```

투플을 갱신하면 여러가지 관계 제약 조건을 위배하는 결과가 초래될 수 있다.

1) 도메인 제약조건의 위배

투플에 대한 갱신 연산을 도메인 제약조건을 위배하는 상황을 발생시킬 수 있다.

다음의 질의를 생각해보자.

질의 : film 릴레이션에서 film_id가 1인 영화의 id를 -1로 변경하라.

film_id는 양의 정수로 도메인이 설정되어 있으므로, -1로 갱신하는 것은 도메인을 벗어난 값으로의 변경을 의미한다.

이를 위하여 다음 SQL 질의문을 실행해 보자.

```
update film
set film_id = -1
where film_id = 1;
```

그러면 다음과 같은 에러 메시지가 출력된다.

Error Code: 1264. Out of range value for column 'film_id' at row 1

이 메시지는 위의 질의문을 수행하면 film 릴레이션의 film_id 속성의 도메인을 벗어나는 값으로 갱신하려는 시도이므로 갱신할 수 없다는 의미의 메시지이다.

즉, 위의 질의는 도메일 제약 조건을 위배하므로 수행될 수 없다.

2) 키 제약조건의 위배

투플의 갱신은 키 제약조건을 위배하는 현상을 발생시킬 수 있다.

다음의 질의를 생각해보자.

질의 : film_id의 속성값이 1인 투플의 film_id를 NULL로 갱신하라.

film_id는 기본 키 속성이므로 그 값을 NULL로 변경하는 것은 키 제약 조건의 위배를 발생시키는 것이다.

```
update film
set film_id = NULL
where film_id = 1;
```

이 경우, 다음과 같은 에러 메시지가 출력된다.

Error Code: 1048. Column 'film_id' cannot be null

이 메시지는 film_id는 NULL이 입력될 수 없도록 설정된 속성이므로 갱신할 수 없다는 의미의 메시지이다.

즉, 위의 질의는 키 제약 조건을 위배하므로 수행될 수 없다.

3) 엔티티 무결성 제약조건의 위배

투플의 갱신은 엔티티 무결성 제약조건을 위배하는 현상을 발생시킬 수 있다.

다음의 질의를 생각해보자.

질의 : film_id의 속성값이 1인 투플의 film_id를 2로 갱신하라.

film_id는 기본 키 속성이고, 이미 그 값이 2인 투플이 존재하는 상태에서 film_id의 속성값이 1인 투플의 값을 2로 변경하는 것은 키 속성값의 중복을 발생시킨다.

이는 엔티티 무결성 제약 조건의 위배를 발생시키는 것이다.

```
update film
set film_id = 2
where film_id = 1;
```

이 경우, 다음과 같은 에러 메시지가 출력된다.

Error Code: 1062. Duplicate entry '2' for key 'PRIMARY'

이 메시지는 film_id는 기본 키 속성이므로 중복된 값이 입력될 수 없어서 갱신할 수 없다는 의미의 메시지이다.

즉, 위의 질의는 엔티티 무결성 제약 조건을 위배하므로 수행될 수 없다.

4) 참조 무결성 제약조건의 위배

투플의 갱신은 참조 무결성 제약조건을 위배하는 현상을 발생시킬 수 있다.

이를 확인하기 위하여 먼저 데이터를 확인하기 위해 다음 질의를 수행해 보자.

질의 : 영화 'Baby Hall'에 출연한 배우를 찾아, film_id, title, actor_id를 검색하라.

```
select film.film_id, title, actor_id
from film, film_actor
where film.film_id = film_actor.film_id
      and title = "Baby Hall";
```

질의 결과

```
> select film.film_id, title, actor_id
from film, film_actor
where film.film_id = film_actor.film_id
      and title = "Baby Hall"
+ ------------ + ---------- + ------------- +
| film_id      | title      | actor_id      |
+ ------------ + ---------- + ------------- +
| 47           | BABY HALL  | 2             |
| 47           | BABY HALL  | 8             |
| 47           | BABY HALL  | 89            |
```

```
| 47          | BABY HALL  | 95          |
| 47          | BABY HALL  | 127         |
| 47          | BABY HALL  | 143         |
| 47          | BABY HALL  | 153         |
| 47          | BABY HALL  | 158         |
+ ----------- + ---------- + ----------- +
8 rows
```

질의 결과에서 영화 'Baby Hall'의 film_id는 47이고, 총 8명의 배우가 출연한 것을 확인할 수 있다.

이 질의는 film 릴레이션과 film_actor 릴레이션을 조인하여 얻은 결과로서, film_actor 릴레이션에는 film 릴레이션을 참조하는 film_id 속성이 외래키로 포함되어 있다.

이제 film 릴레이션에서 영화 Baby Hall의 film_id를 10000 변경해 보자.

질의 : 영화 'Baby Hall'에 출연한 배우를 찾아, film_id, title, actor_id를 검색하라.

이를 위하여 다음 SQL 질의문을 수행한다.

```
update film
set film_id = 10000
where title = "Baby Hall";
```

film_id는 film_actor 릴레이션에서 참조하는 속성이므로 이 속성 값이 변경되면 참조 무결성 제약 조건을 위배할 수 있다.

즉, film_actor 릴레이션에서 film_id의 속성값으로 47을 가지고 있던 투플들에 대해 참조 무결성 제약 조건이 위배될 수 있는 것이다.

위의 갱신 연산을 수행한 후 다음 SQL 질의어를 수행하여 참조 무결성 제약 조건이 위배 되었는지 확인해 보자.

```
select film.film_id, title, actor_id
from film, film_actor
where film.film_id = film_actor.film_id
      and title = "Baby Hall";
```

질의 결과

```
> select film.film_id, title, actor_id
from film, film_actor
where film.film_id = film_actor.film_id
      and title = "Baby Hall"
+ ------------ + ---------- + ------------ +
| film_id      | title      | actor_id     |
+ ------------ + ---------- + ------------ +
| 10000        | BABY HALL  | 2            |
| 10000        | BABY HALL  | 8            |
| 10000        | BABY HALL  | 89           |
| 10000        | BABY HALL  | 95           |
| 10000        | BABY HALL  | 127          |
| 10000        | BABY HALL  | 143          |
```

```
| 10000       | BABY HALL  | 153          |
| 10000       | BABY HALL  | 158          |
+ ----------- + ---------- + ------------ +
8 rows
```

질의 결과 film 릴레이션에 대한 갱신 연산이 정확하게 수행되어 영화 Baby Hall의 film_id가 47에서 10000으로 갱신된 것을 확인할 수 있다.

또한, film 릴레이션과 film_actor 릴레이션의 조인도 정확하게 수행되어 출연한 배우가 정상적으로 검색된 것을 확인할 수 있다.

이는 갱신 연산으로 인하여 참조 무결성 제약 조건에 대한 위배가 발생하였으나 릴레이션을 생성할 때에 참조 무결성 제약 조건이 위배되는 경우 참조하는 릴레이션의 속성값도 함께 변경되도록 설정해 두었기 때문에 결과적으로 참조 무결성 제약 조건이 지켜진 것이다.

데이터베이스 관리 시스템의 활용

■ 본 교재는 2011 한국콘텐츠진흥원 산업계 맞춤형 인력양성사업 지원을 받았음

데이터베이스 관리 시스템의 활용

발 행 일 : 2012. 2. 1.
지 은 이 : 장 지 웅
발 행 인 : 고 봉 기
발 행 처 : EGO expertgroup
등　　록 : 제2008-000275호 (2008. 10. 22.)
주　　소 : 서울시 강남구 테헤란로 70길 14-8 세왕개발빌딩 8층
전　　화 : 070-7527-2250

ISBN 978-89-97499-01-4 93560
값 20,000원