MySQL

PART 1

일반적인 MySQL 사용법

01 MySQL을 가지고 시작하기

Chapter

이 장에서는 MySQL 관계형 데이터베이스 관리 시스템(RDBMS, Relational Database Management System)과 MySQL이 제공하는 SQL 언어(SQL, Structured Query Language)에 대해 소개하도록 하겠다. 여기서는 여러분들이 알아야만 하는 기본적인 항목과 개념들을 다루고, 이 책 전반에 걸쳐 사용하게 될 예제들에 대한 샘플 데이터베이스를 설명한다. 따라서 이 장을 MySQL을 이용해서 데이터베이스를 만들고 이 데이터베이스와 상호작용하는 방법을 보여주는 안내서로 삼아도 좋을 것이다.

만일 여러분이 데이터베이스를 처음 접하는 상황이고, 데이터베이스를 필요로 하는지 또는 사용할 수 있는지 잘 모르겠다면 이 장부터 시작하는 것이 바람직하다. 더구나 MySQL이나 SQL에 관해 아는 것이 없고 이에 대한 공부를 시작하기 위한 길잡이가 필요하다면 반드시 이 장을 읽어야만 한다. MySQL이나 다른 데이터베이스 시스템들에 경험이 있는 여러분들은 이 장을 대강 읽고 싶어할 지도 모르겠다. 하지만, 1.2절 "샘플 데이터베이스"만큼은 반드시 읽어서 이 책 전체에서 반복적으로 사용되는 sampdb 데이터베이스의 목적과 내용에 친숙해지기 바란다.

1.1 MySQL이 어떻게 도울 수 있는가?

이 절에서는 MySQL 데이터베이스 시스템을 유용하게 사용할 수 있는 상황에 대해 설명하고자 한다. MySQL이 할 수 있는 것들과 그 중에서 여러분에게 도움이 될 수 있는 경우들을 제시하게 된다. 데이터베이스 시스템의 유용성에 대해 이미 잘 알고 있는 사람들이라면, 아마도 마음속에 어떤 문제를 이미 가지고 있거나 단지 그 문제를 풀고자 하여 MySQL을 작동시키는 방법을 알고 싶은 경우이기 때문일 수 있는데, 그렇다면 이 장의 뒷부분에 나오는 1.2절 "샘플 데이터베이스"로 넘어가도 좋다.

데이터베이스 시스템이라는 것은 기본적으로는 단지 나열된 정보들을 관리하는 매우 좋은 방법이라 할 수 있다. 정보는 매우 다양한 곳에서 얻어 올 수 있다. 예를 들면, 연구 자료, 업무 기록들, 고객의 요구사항들, 스포츠 통계 자료, 판매 실적, 개인적인 취미 정보, 인사 기록, 버그 보고 자료들, 학생 성적 등이 모두 정보가 될 수 있다. 하지만, 데이터베이스 시스템이 광범위한 정보를 다룰 수 있어도, 개인이 사적인 용도로 이러한 시스템을 사용하지는 않는다. 어떤 업무가 이미 다루기 쉬운 경우라면, 굳이 그 업무에 데이터베이스 시스템을 도입해서 사용할 필요는 없을 것이다. 구매할 식료품 목록이 좋은 예가 될 수 있다. 구매할 품목을 작성하고, 쇼핑하면서 고른 것들은 목록에서 지우고, 쇼핑이 끝나면 목록은 던져버리는 것이다. 이러한 일에 데이터베이스를 도입한다는 것은 별로 타당해 보이지 않는다. 한 손에 쥘 수 있는 휴대용 컴퓨터를 지니고 있는 경우라 할지라도, 보통은 이 컴퓨터의 데이터베이스 기능보다는

노트패드 기능 정도를 사용해서 식료품의 구매 목록을 작성하게 될 것이다.

정리하고 관리하려고 하는 정보가 비대하고 복잡하게 되어서 기록하는 것이 손으로 처리하기에는 너무 부담스러울 정도로 힘들어질 때 비로소 데이터베이스 시스템의 능력이 빛을 발하게 된다. 하루에 수백만 건의 업무를 처리하는 대형 기업의 경우는 분명히 이에 해당한다. 이와 같은 환경에서는 데이터베이스 도입이 필수적이다. 하지만, 이를테면 개인이 사적인 관심에 대한 정보를 기록해 두려는 것과 같은 작은 규모의 경우에도 데이터베이스가 필요할 수 있다. 정보가 다루기 힘들 정도로 방대해지기 전에는 그다지 많은 양을 필요로 하지 않기 때문에 데이터베이스를 도입해서 크게 도움을 받을 수 있는 시나리오를 상정하는 것은 그리 어려운 일이 아니다. 다음과 같은 상황들을 고려해보자.

- 목공 사업을 하는데, 몇 명을 고용하고 있는 경우를 보자. 누구한테 언제 급여를 지불했는지 알 수 있도록 피고용인과 급여에 관한 기록을 유지해야 할 필요가 있고, 세무서에 수입에 대한 근거를 제공할 수 있도록 이러한 기록들을 정리해 놓고 있어야 한다. 게다가, 회사가 참여하고 있는 작업에 대한 것과 어떤 피고용인이 어떤 작업에 투입되는지에 대한 일정을 관리해야 할 것이다.
- 자동차 회사 부품 창고의 네트워크를 운영하는 사람은 고객의 주문이 들어올 때 재고 목록에서 어떤 부품이 어느 곳에 보유되어 있는지 말할 수 있어야 한다. 그래야 손님의 주문에 처리할 수 있게 된다.
- 다년간의 연구 과정을 거쳐서 쌓아온 대량의 연구 자료들을 논문으로 출간하려면 분석 작업을 거쳐야 할 것이다. 이렇게 해서 "발표하지 못하면 도태(역자 주: Publish or Perish — 미국 대학 사회의 치열한 경쟁을 보여주는 말로, 연구를 해서 좋은 논문을 발표하지 못하면 정년 보장 심사에서 탈락되는 등 대학 사회에서 도태된다는 뜻이다)"라는 격언에 자신의 경력이 걸리지 않게 하고자 할 것이다. 상당량의 원시 자료들을 요약해서 정리된 정보로 만들어서, 관찰된 것들 중에서 선택적으로 골라내어 더욱 상세한 통계 분석을 위한 자료로 정리할 필요가 있다.
- 선생님들에게는 학생들의 성적과 출석을 계속 기록해 두는 업무가 있다. 간단한 퀴즈나 시험이 있을 때마다 모든 학생들의 성적을 기록한다. 성적 기록부에 점수를 써 내려가는 것은 그다지 어려운 일은 아니지만, 나중에 이 점수들을 사용하는 일은 매우 귀찮은 일이 된다. 각 시험에 대한 점수를 정렬해서 성적 분포 곡선을 그리는 작업과 학기말에 최종 성적이 정해졌을 때 각 학생들의 성적을 추가하는 작업 등은 매우 하기 싫은 일들이다. 각 학생들의 결석일수를 세는 것도 재미있는 일은 아니다.
- 회원 명부를 관리하는 어떤 조직의 사무직원의 경우도 있다(여기서의 조직에는 교수 협회, 클럽, 레퍼토리 극단, 심포니 오케스트라, 또는 스포츠 클럽 등이 해당될 수 있다). 각 회원에 대한 것들을 매년 출력된 형태로 제작하는데, 회원 정보가 변할 때 워드 프로세스의 문서를 편집하는 식으로 작업한다. 회원 목록을 유지하는 일은 회원이 늘어감에 따라 점점 더 고단한 일이 되어 간다. 항목들을 여러 다른 방법으로 분류하는 것은 어려운 일이 되고, 각 항목의 특정한 부분을 선택하는 것은 쉽지 않게 된다(이름과 전화번호만 들어가는 목록과 같은 작업). 신규 회원을 곧바로 추가할 때 필요한 일들과 같은, 회원들의 특정 목록을 찾는 것도 쉽지 않다. 만일 가능하다면, 갱신된 유의사항들을 수신할 필요가 있는 회원들을 찾기 위해 매월 항목들을 뒤지는 일들이 추가될 수 있다. 또한 회원 명부 전체를 손수 편집하는 일은 정말로 하고 싶지 않지만, 협회의 예산은 그다지 많이 잡혀있지 않고 누군가를 고용하는 일은 꿈도 꿀 수 없다. 전자적으로 기록을 보관하는 "종이 없는 사무실"에 관해 들어본 적이 있었겠지만, 그로부터의 아무런 이득도 보지 못했다. 회원 기록은 전자적으로 이루어지지만, 아

이러니하게도 회원 명부를 출력해서 종이를 만들어내는 것을 제외하고는 아무것도 쉽게 사용되어질 수 있는 형태가 아니다!

이상의 시나리오들은 상대적으로 적은 양의 정보를 다루는 상황에서부터 광범위한 정보를 다루는 상황까지 포함하고 있다. 이 시나리오들에는 손으로 수행할 수 있는 업무에서와 공통된 특성들을 가지고 있지만 데이터베이스 시스템을 도입함으로써 보다 효과적으로 수행될 수 있는 것들이 된다.

MySQL과 같은 데이터베이스 시스템을 적용해서 기대할 수 있는 구체적인 이득에는 어떤 것이 있을까? 실제 당면한 특정한 필요성과 문제들에 따라 다르다. 그리고 이전의 예들에서 보인 것처럼, 꽤 많은 사례가 있다. 데이터베이스가 아주 적당하게 사용되는 빈번한 예를 알아보도록 하자. 데이터베이스 관리 시스템들은 사람들이 서류함을 사용할 때와 같은 식으로 업무를 처리하도록 사용될 때가 많다. 실제로, 데이터베이스는 어떤 면에서는 커다란 서류함과 같기도 하지만, 내장된 파일 시스템을 가지고 있다. 손수 서류들을 관리하는 것에 비해 전자적으로 서류들을 관리함으로써 얻는 몇 가지 중요한 이점들이 있다. 예를 들면, 고객들의 서류를 사무실 안에서 다루는 일을 하고 있는 경우라면, MySQL의 파일 처리 시스템 기능을 사용해서 얻을 수 있는 몇 가지 이점을 다음과 같이 쓸 수 있을 것이다.

- **줄어든 기록 파일 처리 시간** 새로운 기록을 어디에 추가할지 파악하기 위해 서류함 안의 서랍을 뒤져볼 필요가 없다. 그냥 단지 새로운 기록을 파일 처리 시스템에 넘겨주면 이 시스템이 알아서 해당 기록을 제대로 된 위치에 두게 된다.

- **감소된 기록을 찾아서 가져오는 시간** 여러 기록들을 찾을 때, 원하는 정보를 담고 있는 것들을 손수 하나씩 찾아볼 필요가 없다. 치과병원의 사무실에 근무한다고 해보자. 한동안 병원에 방문하지 않았던 모든 환자들에게 기념품을 보내고자 하는 경우에는 파일 처리 시스템에 적당한 기록을 찾을 수 있도록 요청한다. 물론, "지난 6개월 동안 한 번도 방문하지 않았던 환자들을 찾아주세요"라고 다른 사람에게 요청하는 식으로 간접적으로 일을 처리할 수도 있을 것이다. 데이터베이스를 사용하게 되면, 다음과 같은 문장은 굉장히 이상한 주문을 외우는 것처럼 보일지도 모르겠다.

```
SELECT last_name, first_name, last_visit FROM patient
WHERE last_visit < DATE_SUB(CURDATE(),INTERVAL 6 MONTH);
```

  이러한 문구를 보면 이전에 이와 비슷한 것을 본 적이 없다면 다소 겁을 먹을 수도 있겠지만, 기록들을 찾느라 뒤적거리는 데 한 시간 이상을 소모하지 않고 단 몇 초 안에 결과를 얻어낼 수 있다면 매력을 느낄 수밖에 없을 것이다(어떠한 경우에도, 걱정할 필요는 없다. 이렇게 괴상하게 보이는 까다로운 공문과 같은 느낌은 얼마 지나지 않아서 사라지게 될 것이다. 사실, 이 장을 모두 읽게 되는 시점이 되면 이러한 문장이 무엇을 의미하는지 정확하게 이해할 수 있을 것이다).

- **유연한 검색 순서** 기록들을 저장했을 때의 고정된 순서에 따라서 가져올 필요가 없다(예를 들면, 환자의 성을 기준으로). 파일 처리 시스템에 지시해서 좋아하는 순서대로 저장된 기록을 꺼내오도록 할 수 있다. 즉, 성이나 보험회사 이름, 마지막 방문일 등과 같은 순서가 될 수 있다.

- **유연한 출력 포맷** 원하는 자료들을 찾은 다음에, 그 정보를 손수 복사할 필요가 없다. 파일 처리 시스템이 알아서 목록을 만들어 주기 때문이다. 어떤 때는 해당 정보만을 출력하기도 한다. 어떤 때는 또 다른 프로그램이 그 자료를 사용할 수 있게 만들 수도 있다(예를 들면, 치과병원에 방문 기한이

지난 환자의 목록을 만든 뒤에, 해당하는 환자들에게 전송할 수 있는 통지사항들을 출력하는 워드 프로세서 안으로 이러한 정보를 입력시킬 수 있다). 또는 선택된 레코드의 개수와 같은 통계 정보에만 관심을 가질 수도 있다. 이러한 개수를 직접 일일이 셀 필요는 없다. 파일 처리 시스템이 대신해서 통계 수치를 만들어낼 수 있다.

- **동시에 복수의 사용자가 기록에 접근할 수 있다** 두 사람이 동시에 하나의 기록을 보고자 할 때, 종이에 기록된 형태라면, 두 번째 사람은 첫 번째 사람이 그 기록을 다 보고 돌려줄 때까지 기다려야 한다. MySQL은 기록을 동시에 두 사람 모두 접근할 수 있는 다중 사용자 기능을 제공해 준다.
- **원격으로 접속해서 기록을 전자적으로 전송한다** 종이로 된 레코드들을 사용할 때에는 레코드가 있는 곳으로 직접 가거나 다른 사람이 가서 복사해서 가져오도록 해야 한다. 전자 레코드들은 떨어진 곳에서 레코드에 접속하거나 이 레코드들을 전자적으로 전송할 수 있는 잠재력을 가지고 있다. 만일 치과 그룹이 지사 사무실들을 거느리고 있는 연합체라면, 이 연합체는 각기 자신의 위치에서 레코드들에 접근할 수 있는 것이다. 심부름꾼을 시켜서 복사본을 전달할 필요가 없는 것이다. 레코드가 필요한 누군가가 여러분과 같은 종류의 데이터베이스 소프트웨어를 가지고 있지 않지만 전자 메일은 가능하다면, 여러분이 그 누군가가 원하는 레코드를 검색해서 그 내용을 전자적으로 전송해줄 수 있다.

이전에 데이터베이스 관리 시스템을 사용해본 적이 있는 사람이라면, 이미 방금 설명한 장점들을 알고 있을 것이고, 일상적인 "서류함을 대신하는" 응용 프로그램 수준을 넘어서 그밖의 것을 알고자 할 것이다. 많은 기관들이 데이터베이스를 웹 사이트와 연동해서 사용하는 방식은 아주 좋은 예가 된다. 회사가 부품 창고 데이터베이스를 사용한다고 가정해보자. 이 부품 창고 데이터베이스는 고객들이 어떤 물품을 재고로 보유하고 있는지 그리고 비용이 얼마인지 알아봐 달라고 요청할 때 서비스 데스크 직원들이 사용한다. 이것이 비교적 전통적인 데이터베이스의 적용 사례가 된다. 하지만, 회사가 고객이 방문해서 볼 수 있도록 웹 사이트를 구축한다면, 여기에 추가적인 서비스를 제공할 수 있다. 즉, 고객이 스스로 물품 재고와 가격을 알아볼 수 있는 검색 페이지가 그것이다. 여기서 고객은 스스로 원하는 정보를 얻을 수 있는데, 궁금해 하는 품목에 대해서 데이터베이스 안에 저장된 재고 정보를 자동적으로 검색하는 방법을 사용하는 것이다. 고객은 즉시 정보를 얻게 된다. 고객은 짜증나는 음악을 들으면서 기다리거나 서비스 창구의 업무 시간에 맞춰서 방문 시간을 조절하는 등의 일을 할 필요가 없게 된다. 또한 고객들이 웹 사이트를 사용하게 됨으로 인해서, 서비스 창구에서 전화 응대를 하거나 고객 상담을 해줄 직원 수를 줄일 수 있게 된다(이것이 제일 큰 이유가 될지도 모르겠다).

그러나 데이터베이스를 그보다 더 잘 사용할 수도 있다. 웹 기반의 저장 창고 검색에서 수집된 요청들은 고객뿐 아니라 회사에도 유용한 정보가 된다. 이 요청들은 고객들이 주로 찾는 것이 무엇인지를 말해주고, 이러한 검색의 결과를 보면 회사가 고객들의 요청을 잘 만족시켜 주는지 여부를 알 수 있다. 고객들이 원하는 물품이지만 재고가 없는 경우가 많아지게 되면, 사업 실패의 원인이 될 수도 있는 것이다. 결론적으로, 저장 창고의 검색에 대한 정보를 기록해 두는 것은 현명한 일이다. 즉 고객들이 무엇을 찾는지, 창고에 그것에 대한 재고가 있는지에 관한 항목을 기록해 두는 것이다. 그러면 이러한 정보를 가지고 재고를 조절하고 고객들에게 더 나은 서비스를 제공할 수 있다. 또 다른 웹 기반의 데이터베이스 애플리케이션에는 웹 페이지 내에 배너 광고를 서비스하는 것이다. 필자 자신도 여러분 못지 않게 이러한 광고를 싫어하지만, 많은 광고 서비스에서 광고 문구를 저장하고 웹 서버가 표시하도록 검색해오는 도

구로 MySQL을 사용한다는 사실은 틀림없다. 게다가, MySQL은 제공되는 광고의 활성도와 같은 요소들을 추적하여 기록으로 보관할 수도 있다.

그러면 MySQL은 어떤 식으로 동작하는가? 알 수 있는 가장 좋은 방법은 직접 MySQL을 사용해보는 것이고, 이렇게 해보면 데이터베이스를 도입할 필요를 느끼게 될 것이다.

1.2 샘플 데이터베이스

이 절에서는 이 책의 전반에 걸쳐서 사용하게 될 샘플 데이터베이스를 설명한다. 이것은 MySQL을 가지고 작업하는 것을 배우게 될 때 해볼 수 있는 예제들의 소스가 된다. 주로 앞서 설명한 두 가지 상황에서 예제들을 구성해나갈 것이다.

- **어떤 단체의 간사로 일하는 경우** "단체"라는 개념보다 조금 더 자세한 정의가 필요할 것 같다. 따라서 여기서 다룰 단체의 특성을 더 자세히 설명해보자면 다음과 같다. 미국의 역사에 공통적인 관심을 가지고 함께 연구해 보는 사람들로 구성된 단체이다(별로 좋은 이름이 생각나지 않아서 미국 역사 연구회라고 이름을 지어보았다). 일정 기간 단위로 회비를 납부해서 회원 자격을 갱신한다. 회비는 소식지인 "미국의 지난 발자취"를 발간하는 등의 활동에 대한 비용으로 쓰인다. 또한 연구회는 작은 웹 사이트를 운영하지만 그다지 많이 개발된 상태는 아니다. 지금까지 이 사이트에서는 연구회에 관한 설명, 종사하는 사무직원들, 가입 정보 등과 같은 기본적인 정보만 제공해왔다.
- **성적 기록부의 경우** 학기 중에 퀴즈와 시험을 치르고, 성적들을 기록하고, 학점을 배정하는 등의 업무를 수행하는 직업을 생각해본다. 최종 학점을 결정하면, 이것을 출석 기록과 함께 교무처에 제출한다.

이제 이러한 상황들에 대해서 두 가지의 요구사항으로 좀더 밀접하게 검토해보도록 하자.

- 데이터베이스에서 꺼내고자 하는 것이 무엇인지를 결정해야만 한다. 이것이 성취하고자 하는 목표이다.
- 데이터베이스 안에 어떤 내용을 넣을 것인지 명확하게 알아야 한다. 이것은 보존하게 될 데이터를 말한다.

아마도 무엇을 저장할지 고민하기 전에 데이터베이스에서 무엇을 꺼내올 수 있는지를 거꾸로 생각하게 될 것이다. 무엇보다 먼저, 데이터를 입력해 두어야 그것을 검색할 수 있는 것이다. 하지만, 데이터베이스를 사용하는 방법은 목표에 따라 좌지우지되고, 이들은 데이터베이스에서 어떤 것을 입력하는가보다는 어떤 것을 가져오고 싶은가에 보다 밀접하게 연관되어 있다. 확실히, 이후에 어떤 문제를 위해 데이터베이스를 사용하게 되지 않는다면 시간을 허비하지 않고 데이터베이스 안으로 정보를 입력하고자 한다.

1.2.1 미국 역사 연구회

이 시나리오에 대한 초기 상황은 워드 프로세서로 문서 작업을 해서 회원 명부를 관리하는 연구회의 간사로 근무하는 것으로 설정해본다. 워드 프로세서로만 작업을 해도 명부를 출력해서 책자로 만드는 일은 아주 잘 되겠지만, 그 안의 정보를 가지고 할 수 있는 일에는 제한이 따른다. 다음과 같은 목표들을

명심한다.

- 연구회의 간사는 명부에서 특정 애플리케이션에 맞는 정보만을 사용한 여러 가지의 포맷으로 출력물을 만들어내기를 원한다. 한 가지 목표는 매년 인쇄된 명부를 만들어낼 수 있게 되는 것이다. 이것은 계속해서 수행하려고 계획한 과거에서부터 해온 연구회의 요구사항인 것이다. 명부 안의 정보를 다른 식으로 사용하는 것을 생각할 수 있다. 예를 들면, 배포한 인쇄된 프로그램에 대한 현재의 회원 목록을 리그 연례 연회의 참석자들에게 제공하기 위한 것이 있을 수 있다. 이러한 애플리케이션들은 또 다른 종류의 정보를 포함한다. 인쇄된 명부는 각 회원 항목의 전체 내용을 사용한다. 연회 프로그램에 경우에는, 단지 회원의 이름만 끄집어내면 된다(워드 프로세서를 사용하면 뭔가 쉽지 않은 작업이다).

- 여러 특징들을 만족시키는 항목들을 가지는 회원에 대한 명부를 검색하고자 한다. 예를 들면, 곧 회원 자격을 갱신해야 하는 회원들을 알기를 원한다. 각 회원들에 대해 유지하는 키워드가 있어서 이들 키워드로 검색할 수 있는 애플리케이션이 있었으면 한다. 이 키워드들은 각 회원이 특별히 관심을 가지는 미국 역사에서의 영역을 나타낸다(예를 들면, 시민전쟁, 경제 대공황, 시민권, 또는 토마스 제퍼슨의 일생 등). 회원들은 이따금씩 자신과 비슷한 관심을 가지고 있는 다른 회원들을 알기 원하는데, 이러한 요청을 만족시켜줄 수 있으면 좋다.

- 회원 명부를 연구회의 웹 사이트에 온라인으로 올리기를 원한다. 회원들과 사무직원 모두에게 좋은 작업이 된다. 어떤 자동화된 과정을 거쳐서 회원 명부를 웹 페이지로 변환하면, 회원 명부의 온라인 버전은 종이에 출력된 버전보다 항상 최신의 자료로 유지시킬 수 있다. 또한 온라인 명부를 검색 가능하도록 만들면, 회원들은 스스로 정보를 쉽게 찾아볼 수 있다. 예를 들면, 시민전쟁에 관심이 있는 다른 회원이 있는지 알아볼 때 사무실에서의 대기시간 없이 빠르게 알아볼 수도 있고, 스스로 알아볼 수도 있다.

필자는 데이터베이스라는 것이 세상에서 가장 흥미로운 것은 아니라는 것을 잘 알고 있기 때문에, 데이터베이스를 사용하는 것이 창조적인 사고를 가져온다고 강력하게 주장하지는 못하겠다. 그럼에도 불구하고, 정보라는 것이 반드시 낑낑대며 씨름해야 되는 어떤 것이라는 생각(워드 프로세서의 문서로 작업하는 것과 같은)을 그만두고 정보라는 것이 상대적으로 쉽게 다루어질 수 있는 어떤 것(MySQL로 처리하고 싶어 하는 것과 같은)이라고 여겨질 때, 여러분의 능력에 긍정적인 영향이 생겨서 그러한 정보를 사용하거나 나타내는 새로운 경지에 이를 수 있게 되는 것이다.

- 데이터베이스 내의 정보가 온라인 명부의 형태로 된 웹 사이트로 옮겨질 수 있다면, 정보 흐름을 다른 방식으로 만들어갈 수 있을 것이다. 예를 들면, 회원들이 온라인상에서 자신의 항목을 편집하여 데이터베이스를 갱신할 수 있으면, 간사 자신이 모든 편집을 할 필요가 없고 명부 내의 정보를 보다 더 정확하게 만드는 데 도움이 될 것이다. 실제로는 간사 자신이 모든 디렉토리에 대한 편집을 피하는 것이 좋지만, 연구회는 많은 예산을 가지고 있지 않기 때문에 어떤 다른 사람을 고용할지는 묻지 않아도 될 것이다.

- 데이터베이스 안에 전자 메일 주소를 저장하면, 한동안 항목을 갱신하지 않은 회원들에게 이를 이용해서 전자 메일을 보낼 수 있다. 메일의 메시지에는 회원들의 항목에 대한 현재의 내용이 들어 있고,

이것을 재고해볼 것인지를 묻고, 수정할 것이 있는 경우에는 웹 사이트에 준비된 기능을 사용하여 수정하는 방법이 들어 있다.

- 데이터베이스를 사용하여 회원 목록에 관련되지 않은 것도 포함시켜서 웹 사이트를 좀더 유용하게 만들 수 있다. 연구회는 미국 역사 이야기라는 뉴스레터를 발행하는데, 여기에는 어린이들을 위한 절이 있고, 각 주제는 역사에 기반한 퀴즈를 담고 있다. 최근 이슈 중 어떤 것은 미합중국 대통령에 관한 전기적인 사실들에 초점이 맞추어져 있다. 웹 사이트에서도 아이들을 위한 섹션도 제공할 수 있을 것이며, 퀴즈도 온라인상에서 제공될 것이다. 아마도 이 섹션은 대화식으로 만들어서 데이터베이스 안에 퀴즈 정보를 입력해 두고, 웹 서버가 데이터베이스에서 적당한 퀴즈를 검색해서 방문자에게 제공할 수 있도록 할 수 있을 것이다.

좋다! 지금 이 시점에서 데이터베이스를 가지고 해야 할 작업의 양을 생각해보면 다소 흥분되는 자신을 발견할지도 모르겠다. 잠시 한숨을 돌리고 나서, 몇 가지 실제적인 질문을 해보기로 하자.

- **이것이 조금은 야심찬 작업은 아닌가?** 이러한 것을 설정하는 데 많은 작업을 해야 하는가? 물론 실제 해보지 않고 단지 생각만 할 때에는 모든 것이 쉽게 느껴지겠지만, 이러한 것이 간단하게 이루어질 것이라고 속이지는 않겠다. 그럼에도 불구하고, 이 책의 마지막에 가면 방금 구상한 모든 것을 해낼 수 있을 것이다. 단지 하나만 명심하기 바란다. 즉, 한 번에 모든 것을 해낼 필요는 없다. 해당 업무가 있으면 각 부분을 조각으로 나누어서 한 번에 한 조각씩 해결해나갈 것이다.

- **MySQL이 이러한 모든 것을 해낼 수 있을까?** 아니, 최소한 그 자체만 가지고 한다면 그렇지는 않다. 예를 들면, MySQL은 직접 웹 프로그램을 작성할 수 있는 기능이 없다. 그렇지만 MySQL 자체로는 논의된 모든 것을 할 수 없을지 몰라도, 부족한 부분을 보상하고 그 기능을 확장시켜 주는 일을 하는 다른 도구들을 MySQL과 연동시킬 수 있다.

  Perl 스크립트 언어와 DBI(데이터베이스 인터페이스, DataBase Interface) Perl 모듈을 사용하여 MySQL 데이터베이스에 접근하는 스크립트를 작성할 것이다. Perl은 훌륭한 텍스트 처리 기능을 가지고 있어서, 고도로 유연한 방법으로 쿼리 결과를 처리해서 다양한 포맷으로 결과를 생성할 수 있다. 예를 들면, Perl을 사용하여 RTF(Rich Text Format)로 된 디렉터리를 만들 수 있는데, RTF는 모든 종류의 워드 프로세스들이 읽을 수 있는 포맷이다.

  그리고 또 다른 스크립트 언어인 PHP를 사용할 것이다. PHP는 특히 웹 애플리케이션을 작성하는 데 특화되어 있고, 데이터베이스와 쉽게 연동된다. 이 언어를 사용해서 웹 페이지에서 바로 MySQL 쿼리를 실행할 수 있도록 만들 수 있고 데이터베이스 쿼리의 결과를 포함하는 새로운 페이지를 생성할 수 있다. PHP는 Apache와도 잘 작동하는데(세상에서 가장 대중적인 웹 서버이다), 검색 창과 검색의 결과를 표시하는 것과 같은 작업을 쉽게 할 수 있다.

  MySQL은 이 툴들과 매우 잘 '통합' 시켜서, 마음먹은 결과를 성취할 수 있는 좋은 방법을 선택할 수 있도록 해준다. 필요하지 않은 부분까지 모두 하나로 통합된 형태를 사용할 필요 없이, 필요한 것만 잘 작동되도록 선택할 수 있다.

- **그리고 마지막으로, 진짜 중요한 질문 — 비용은 전부 얼마나 드는가?** 연구회의 예산은 한정되어 있다.

이해가 가지 않는 상황이기는 하지만, 더 이상 예산을 늘릴 것 같지는 않다.

통상적인 데이터베이스 시스템들의 가격은 대부분 상당히 높다. 이와는 달리, MySQL은 보통 무료이다. 심지어, 엔터프라이즈에서의 셋팅에 있어서 지원과 유지보수도 지원되기 때문에 MySQL은 다른 데이터베이스 시스템에 비해서 저렴하다. (자세한 내용은 www.mysql.com을 방문하라). 사용하게 될 다른 툴들(Perl, DBI, PHP, Apache)도 무료이기 때문에, 모든 것을 고려해봐도 큰 비용상의 부담 없이 쓸만한 시스템을 구비할 수 있게 된다.

데이터베이스를 개발할 운영체제를 고르는 것은 여러분에게 달렸다. 실제로 여기에서 논의할 모든 소프트웨어는 UNIX(여기서 BSD UNIX, Linux, Mac OS X 등을 통칭해서 UNIX라 부른다)와 Windows 양쪽 모두에서 실행한다. 몇 가지 예외사항들은 Unix 또는 Windows에 따른 쉘이나 배치 스크립트에 존재한다.

1.2.2 성적 기록부 프로젝트

지금 샘플 데이터베이스를 사용하는 다른 상황에 대해서 고려하도록 하자. 여기서의 초기 시나리오는 선생님이 되는 것인데, 성적 기록부를 다룰 책임이 있다. 성적 기록부를 사용하여 수작업으로 학점을 처리하는 방식을 바꾸어서 MySQL을 사용하여 전자적으로 처리하고자 한다. 이 경우에는 성적 기록부를 사용하는 방법에서 데이터베이스로부터 얻고자 하는 정보를 암시할 수 있다.

- 간이 퀴즈나 시험에서 나온 점수들을 기록한다. 시험에서 나온 점수 순으로 나열해서 이것을 보고 A, B, C, D, F와 같은 학점으로 구분할 수 있도록 한다.
- 학기말에는 각 학생들의 전체 점수를 계산하고, 이 값들을 정렬시켜 이에 기반해서 학점을 정하는 일을 한다. 퀴즈보다 시험에 좀더 무거운 비중을 두어야 하기 때문에 가중치를 포함시켜서 총점을 계산하게 된다.
- 학기말에는 출석 정보를 교무처에 제출한다.

점수를 분류하고 이에 대한 통계를 내는 것, 그리고 출석을 기록하는 것과 같은 작업들을 수작업으로 하지 않는다는 것이 목표이다. 다시 말하면, 학기 말에 점수를 분류하는 일과 각 학생들의 총점과 결석일을 계산하는 데 필요한 연산을 수행하는 일을 MySQL이 해주기를 바란다는 것이다. 이러한 목표들을 성취하려면, 학급 내의 학생들 목록, 각각의 퀴즈와 시험에 대한 점수들, 그리고 학생들이 결석한 날짜들이 필요하게 된다.

1.2.3 샘플 데이터베이스가 실제와 비슷한가?

역사 연구회나 성적 기록부 프로젝트 같은 것이 별로 재미없는 예라고 느껴진다면, 이것들을 가지고 무엇을 해야 할지 헤맬 수도 있겠다. 정답은 이렇게 예를 든 시나리오들은 그 자체로 마지막이 아니라는 것이다. 이들은 단지 MySQL과 이에 관련된 툴들을 가지고 여러분들이 무엇을 할 수 있는지 보기를 보여주는 장치를 제공할 뿐이다.

조금 생각을 해보면, 예제 데이터베이스 쿼리들이 풀고자 하는 특정한 문제들에 어떻게 적용되는지 알 수 있을 것이다. 앞서 언급했던 치과의사의 사무실에서 일하는 것을 가정해보자. 이 책에 있는 많은 치과에 관련된 쿼리들을 보지는 못하겠지만, 여기서 찾을 수 있는 환자 기록 관리, 사무실에서의 부기 작업 등에 적용되는 많은 쿼리들을 보게 될 것이다. 예를 들면, 조만간 회원 자격을 갱신할 필요가 있는 역사 연구회 회원들을 정하는 문제는 한동안 치과의사에게 방문하지 않은 환자들을 알아내는 문제와 비슷하다. 두 가지 모두 날짜에 기반한 쿼리들이기 때문에, 회원 자격 갱신 쿼리를 작성하는 법을 배우고 나면 그 기술을 미납-환자 쿼리를 작성하는 데 적용할 수 있다.

1.3 기본 데이터베이스 용어

어떤 데이터베이스 책자에서 몇 페이지 정도를 미리 읽었으면 몇 가지 자주 쓰이는 용어나 기술적인 단어들을 보았을 것이다. 다루게 될 샘플 데이터베이스의 대략적인 사양을 얘기하긴 했지만, 사실 이 책에서는 아직까지 "데이터베이스"라는 것이 실제 어떠한 것인지 전혀 언급하지 않았다. 하지만, 이제 샘플 데이터베이스를 설계해서 구현하기 시작할 때가 되었기 때문에, 전문용어를 더 이상 피할 수 없게 되었다. 이 절에서는 이러한 전문용어를 다루기로 한다. 이 책에서 자주 등장하는 용어들에 친숙해질 수 있도록 설명한다. 다행히도, 관계형 데이터베이스의 많은 개념들이 실제로 아주 간단하다. 사실상, 관계형 데이터베이스의 매력의 대부분은 기저에 깔린 단순함에서 비롯된다.

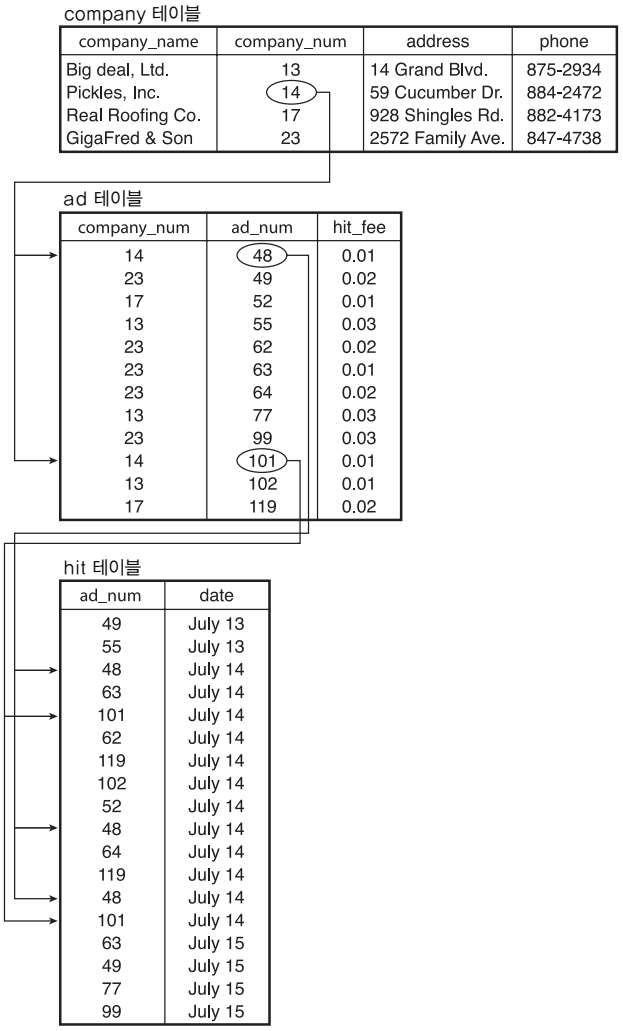
1.3.1 구조적인 용어

데이터베이스의 세계에서, MySQL은 관계형 데이터베이스 관리 시스템(RDBMS, Relational Database Management System)으로 분류된다. 이 문구를 쪼개어 보면 다음과 같이 설명된다.

- 데이터베이스(RDBMS에서 "DB"에 해당)는 정보를 단순하고 규칙적인 모양새로 구성해서 저장하고자 할 때 이 정보에 대한 저장소를 말한다.
  - 어떤 데이터베이스 안에 모아 놓은 데이터들은 테이블로 구성된다.
  - 각 테이블은 행과 칼럼으로 구성된다.
  - 테이블 안의 각 행을 레코드라고 한다.
  - 레코드들은 몇 조각의 정보로 이루어진다. 어떤 테이블 안의 각 칼럼은 이러한 조각 중 하나에 해당하는 것이다.
- 관리 시스템("MS"에 해당)은 레코드들을 삽입, 탐색, 수정, 또는 삭제할 수 있도록 해서 자료들을 사용할 수 있게 해주는 소프트웨어이다.
- "관계형"(Relational인 "R"에 해당)이라는 단어는 DBMS의 특정한 종류를 가리키는데, 이것은 하나의 테이블에 저장된 정보를 다른 곳에 저장된 정보에 연관(어떤 조건에 대응되는 것)시켜서 각각의 공통된 요소를 찾는 데 아주 편리하다. 테이블에서 데이터를 편리하게 가져오고, 여러 연관된 테이블들에서 정보를 연결하여 개별 테이블만 사용해서는 답을 얻을 수 없는 질문에 대한 답변을 만들어낼 때 관계형 DBMS의 진정한 능력이 빛을 발한다. (실제로, "관계형"은 필자가 사용하는 것과 다른 정

의를 가진다. 순수주의자에게 용서를 빌지만, 필자의 정의는 RDBMS의 유용성을 전달하는 데 더 많은 도움이 될 것이다.)

관계형 데이터베이스가 어떻게 데이터를 테이블 안으로 조직화시키며 하나의 테이블에서 또 다른 테이블로 정보를 어떻게 관련시키는지를 보여주는 예제들이 여기에 있다. 배너 광고 서비스를 포함하는 웹 사이트를 운영한다고 해보자. 이 사이트에 있는 페이지에 사람들이 방문할 때 광고가 표시되게 하고 싶은 회사들과 계약을 하는 것이다. 이 페이지에 방문객이 한 번 접속할 때마다, 방문객의 브라우저로 전송되는 페이지에 내포된 형태의 광고를 제공하고 계약된 회사에 적당한 요금을 부과한다. 이러한 정보를 나타내기 위해서, 관리자는 세 가지 테이블을 준비한다([그림 1.1] 참조). 한 테이블은 company인데,



[그림 1.1] 배너 광고 테이블

여기에는 회사 이름, 번호, 주소, 그리고 전화번호가 들어간다. 또 하나는 ad 테이블인데, 여기에는 광고 번호들, 광고를 "소유한" 해당 회사에 대한 번호, 그리고 히트(hit)할 때마다 부과하는 요금 등이 들어간다. 세 번째는 hit 테이블인데, 여기에는 히트된 광고의 번호와 해당 광고가 제공된 날짜가 기록된다.

어떤 질문들은 하나의 테이블을 사용한 이러한 정보로부터 답을 얻을 수 있다. 계약한 회사의 수를 알아보려면, company 테이블 안에 있는 행의 수를 세기만 하면 된다. 이와 비슷하게, 주어진 기간 동안 히트한 수를 알아보려면, 단지 hit 테이블을 조사해보면 된다. 이보다 좀더 복잡한 질문들이 있을 수 있는데, 이러한 문제에 대한 답을 얻으려면 복수의 테이블을 참고할 필요가 있다. 예를 들면, 7월 14일에 Pickles 주식회사의 광고가 몇 번 제공되었는지 알아보려면, 다음과 같이 이 세 가지 테이블 모두를 사용하게 된다.

1. company 테이블에서 회사 이름(즉, Pickles 주식회사)을 찾아서 회사 번호를 알아낸다(즉, 14번).

2. 회사 번호를 사용해서 ad 테이블 안에 대응되는 레코드들을 찾으면, 관련된 광고 번호들을 알아낼 수 있다. 여기서는 이러한 광고가 두 개 있는데, 48과 101이다.

3. ad 테이블 안에 대응되는 각각의 레코드에 대해, 레코드 안에 있는 광고 번호를 사용해서 hit 테이블 안의 목표하는 날짜의 범위 안에 들어가는 레코드들을 찾아내고, 대응되는 번호의 개수를 센다. 48번 광고에 대해서 세 개가 대응되고 101번 광고에 대해서는 두 개가 대응된다.

복잡하게 들리는가? 하지만 이것이야말로 바로 관계형 데이터베이스 시스템이 훌륭하다는 것을 보여주는 한 가지 예가 된다. 여기서 언급한 각 단계들은 실제로 단일 대응 작업, 즉 하나의 테이블의 행에 있는 값을 또 다른 테이블의 행들 내의 값들에 대응시킴으로써 하나의 테이블을 다른 테이블에 연관시키는 작업보다 많은 양이 되지 못하기 때문에, 복잡성은 다소 허상처럼 보일 수 있다. 이 동일한 간단한 조작을 여러 가지 방법으로 교묘히 이용해서 모든 종류의 질문에 대답할 수 있다. 즉, 얼마나 많은 다른 광고들을 각 회사가 하고 있는가? 어느 회사의 광고가 가장 인기가 좋은가? 각 광고가 만들어내는 수익은 얼마나 되는가? 현재 청구 기간 동안 각 회사에 대한 전체 요금은 얼마인가?

이제 이 책의 나머지 부분을 이해하기 위해 필요한 만큼의 관계형 데이터베이스 이론을 알게 되었는데, 3차 정규 형식, 엔티티-관계 다이어그램, 그리고 그밖의 여러 종류의 이론들까지 다룰 필요는 없다(그러한 것들에 대해 읽어보고 싶다면, C.J. Date 또는 E.F. Codd의 저서들을 읽어볼 것을 권한다).

1.3.2 쿼리 언어 용어

MySQL과 대화를 하려면, SQL(Structured Query Language)이라고 부르는 언어를 사용해야 한다. SQL은 이제는 표준 데이터베이스 언어가 되었고, 주류를 이루는 모든 데이터베이스 시스템에서 지원된다. SQL은 여러 가지 다른 종류의 문장들을 지원하는데, 모든 문장들은 데이터베이스를 흥미롭고 유용한 방법으로 다룰 수 있도록 설계되었다.

다른 모든 언어와 마찬가지로, SQL도 처음 배우는 동안에는 낯설게 느껴질 수 있다. 예를 들어, 어떤 테이블을 만들려면, MySQL에 그 테이블의 구조가 어떻게 되는지 알려야 한다. 어떤 사람이 다른 사람한테 그 구조를 알릴 때에는 그림이나 도형을 써서 할 수도 있겠지만, MySQL은 그런 것을 이해하지 못하

므로, 다음과 같은 문장을 써서 MySQL이 테이블을 만들도록 한다.

```
CREATE TABLE company
(
  company_name  CHAR(30),
  company_num   INT,
  address       CHAR(30),
  phone         CHAR(12)
);
```

위와 같은 문장들은 처음 SQL을 접하는 사람에게는 다소 벅찰 수도 있겠으나, SQL을 효과적으로 사용하는 방법을 배우기 위해 꼭 프로그래머일 필요는 없다. 이 언어에 친숙해져감에 따라서, CREATE TABLE을 다른 각도로 보게 될 것이다. 이상한 말로 횡설수설하는 것이 아니라 정보를 설명하는 데 도움을 주는 동반자가 될 것이다.

1.3.3 MySQL의 구조적인 것에 관련한 용어

MySQL을 사용할 때에는 MySQL 자체가 클라이언트/서버 구조를 사용하기 때문에, 실제로는 두 개의 프로그램을 사용하고 있는 것이다. mysqld라는 서버 프로그램은 데이터베이스들이 저장된 컴퓨터에 있다. 이 프로그램은 네트워크를 통해 들어오는 클라이언트들의 요청을 받아들이고, 이러한 요청들에 따라 데이터베이스의 내용에 접근하여 요청에 따른 정보를 클라이언트에 제공한다. 클라이언트는 데이터베이스 서버에 연결하여 쿼리를 보내서 클라이언트가 어떠한 정보를 원하는지 서버에 알리는 역할을 하는 프로그램이다.

MySQL의 배포본에는 데이터베이스 서버와 몇 가지의 클라이언트 프로그램이 들어 있다. (만약 Linux에서 RPM 패키지를 사용한다면, 서버와 클라이언트 RPM 패키지가 분리되어 있을 것이며, 둘 다 설치되어야 할 것이다.) 사용자는 달성하고자 하는 목적에 알맞은 클라이언트를 사용해야 한다. 가장 보편적으로 사용되는 것은 mysql인데, 이것은 대화식으로 쿼리를 보내고 결과를 볼 수 있게 해주는 클라이언트이다. 두 개의 운영·관리를 위한 클라이언트가 있는데, 하나는 테이블의 내용을 하나의 파일로 덤프시키는 mysqldump이고, 또 하나는 서버의 상태를 점검하고 서버를 중지시키도록 지시하는 것과 같은 운영·관리 작업을 수행하는 mysqladmin이다. 배포본에는 또한 그밖의 다른 클라이언트들도 포함되어 있다. 표준 클라이언트 프로그램으로는 불충분한 작업을 해야 하는 경우를 위해서, MySQL에서는 직접 프로그램을 작성할 수 있도록 클라이언트 프로그래밍 라이브러리를 제공하고 있다. 이 라이브러리는 C 프로그램에서는 직접 사용할 수 있게 되어 있다. C가 아닌 다른 언어를 사용하고자 하는 경우에는 기타 다른 프로그램들에 대한 인터페이스를 사용할 수 있는데, 몇 가지만 언급해본다면 Perl, PHP, Python, Java, Ruby 등에 대한 것이 있다.

이 책에서 언급된 클라이언트 프로그램은 커맨드 라인에서 실행된다. 만약 그래픽 기반의 사용자 인터페이스(GUI)를 사용하는 도구를 사용하고, 클릭해서 실행하고자 한다면 http://www.mysql.com/products/tools/를 방문해야 한다.

MySQL의 클라이언트/서버 구조로 인한 몇 가지 장점이 있다. 즉, 다음과 같다.

- 서버는 두 사용자가 동시에 같은 레코드를 수정할 수 없도록 동시성 제어(concurrency control)를 제공한다. 모든 클라이언트의 요청들은 서버로 전달되어, 서버는 누가 언제 무엇을 할 것인지에 대해서 순서를 정한다. 복수의 사용자들이 동시에 같은 테이블에 접근하려 한다면, 사용자들이 서로 알아보고 협상하는 것이 아니다. 사용자들은 단지 요청을 서버로 전송하고, 서버가 그 요청들이 수행될 순서를 결정하게 된다.
- 반드시 데이터베이스가 위치한 시스템상에 로그인할 필요는 없다. MySQL은 인터넷을 통해 작업할 수 있는 기능을 제공하고 있기 때문에 클라이언트 프로그램을 아무데서나 실행시켜서 네트워크를 통해 서버에 연결할 수 있다. 얼마나 떨어져 있는가 하는 것은 전혀 문제가 되지 못한다. 지구상의 어느 곳에서도 서버에 접근할 수 있다. 서버가 호주에 있는 컴퓨터상에서 동작할지라도, 아이스랜드로 여행 중일 때 노트북을 사용해서 데이터베이스에 접근하는 것이 가능한 일이다. 이러한 사실이 누구나 인터넷으로 연결하는 것만으로도 여러분의 데이터를 가져갈 수 있다는 것을 뜻하는 것일까? 그렇지 않다. MySQL에는 유연한 보안 시스템이 포함되어 있어서, 허락된 사람들만 접근이 가능하다. 그리고 이와 같은 사람들에게도 접근 범위를 제한할 수 있다. 회계 부서에 있는 샐리(Sally)는 레코드들을 읽고 갱신(수정)할 수 있지만, 서비스 창구에 있는 필(Phil)은 단지 그 데이터를 읽을 수만 있다. 이처럼 각 사람들의 권한을 설정할 수 있는 것이다. 상당히 폐쇄적인 시스템을 운영하는 경우라면, 고객들이 서버가 실행되는 호스트에서만 연결할 수 있도록 접근 권한을 설정한다.

mysqld 서버는 클라이언트/서버 구조로 실행되지만, MySQL은 서버를 라이브러리로도 제공하는데, 이 라이브러리는 libmysqld이고, 이것은 MySQL 기반의 독립 실행형 애플리케이션을 만들어내기 위해 프로그램에 링크될 수 있다. 이것을 "임베디드 서버 라이브러리"라고 부르는데, 개별 애플리케이션 안으로 내장(embedded)시키기 때문이다. 클라이언트/서버 식의 서버와는 대조적으로 임베디드 서버는 네트워크를 필요로 하지 않는다. 임베디드 라이브러리를 사용하면, 외부의 작동 환경에 대해 거의 신경 쓰지 않고 배포시킬 수 있는 자신만의 애플리케이션을 만들어서 배포하는 것을 쉽게 만들 수 있다. 한편, 오직 임베디드 애플리케이션으로만 서버가 관리하는 데이터베이스로 접근할 수 있는 상황에서 사용되어야 한다.

> MySQL과 **mysql** 간의 차이 TIP

혼란스럽지 않도록 하기 위해 지적해 두자면, MySQL은 MySQL RDBMS 전체를 가리키는 말이고 "mysql"은 특정한 클라이언트 프로그램의 이름인 것이다. 이 둘을 발음해본다면 똑같이 들리지만, 이 책에서는 대소문자로 이 둘을 구별하게 된다.

MySQL을 발음할 때에는 "마이-에스-큐우-엘"로 발음한다. MySQL 레퍼런스 매뉴얼에서 이렇게 말하고 있기 때문에, 그것을 따른다. 그리고 SQL은 "시쿠얼" 또는 "에스-큐우-엘"로 발음하는데, 맘에 드는 쪽으로 고르면 된다. 어느 쪽이 더 낫다고는 하지 않을 것이다. 이 책에서는 "에스-큐우-엘"로 발음하는데, 이는 "a SQL query(어 시큐얼 쿼리)"보다 "an SQL query(언 에스큐엘 쿼리)"를 사용하는 이유이기도 하다.

1.4 MySQL 튜토리얼

이제 필요한 모든 배경을 갖추게 되었다. 드디어 MySQL을 다루어 볼 때가 되었다!

이 절에서는 해볼 수 있는 튜토리얼을 제공해서 MySQL에 익숙해지는 데 도움이 되고자 한다. 이 튜토리얼을 통해 작업해나가게 되는데, 샘플 데이터베이스와 테이블 몇 개를 만들고 테이블들에 정보를 추가, 추출, 삭제, 수정 작업을 해보는 것으로 데이터베이스를 다루어 보게 된다. 샘플 데이터베이스를 가지고 작업하는 과정에서 다음과 같은 것들을 배워나가게 된다.

- **MySQL이 제공하는 SQL언어의 기본** 다른 RDBMS를 사용해본 경험으로 이미 SQL을 알고 있는 경우라면, 이 부분은 가볍게 읽고 잘 알고 있는 데이터베이스의 SQL과 MySQL의 SQL의 차이점만 알면 된다.
- **몇 개의 표준 MySQL 클라이언트 프로그램을 사용해서 MySQL 서버와 대화하는 방법** 앞의 절에서 언급한 대로, MySQL은 클라이언트/서버 구조를 사용해서 동작하는데, 여기서 서버는 데이터베이스를 지니고 있는 컴퓨터상에서 실행되고, 클라이언트는 이 서버에 네트워크를 통해서 연결하게 된다. 이 튜토리얼은 mysql 클라이언트 프로그램을 사용해서 진행하는 부분이 많은데, 이 프로그램은 사용자로부터 SQL 쿼리를 받아들여서, 이를 서버로 전송해서 실행되도록 하고, 그 결과를 화면에 표시해서 어떤 일이 일어났는지 볼 수 있도록 해준다. mysql은 MySQL이 지원하는 모든 플랫폼에서 동작하고 서버와 대화할 수 있는 가장 직접적인 방법을 제공하므로, 이 클라이언트 프로그램을 사용하여 시작하는 것이 좋은 선택이 된다. 몇 가지 예에서는 이 외에도 mysqlimport와 mysqlshow도 사용한다.

이 책에서는 샘플 데이터베이스 이름으로 sampdb를 사용하지만, 원한다면 다른 이름을 사용해서 진행해도 된다. 예를 들면, 누군가 다른 사람이 여러분의 시스템에서 이미 sampdb라는 이름을 그들 자신의 데이터베이스 이름으로 사용하고 있다면 MySQL 운영·관리자는 다른 데이터베이스 이름을 할당할 수 있다.

여러 사용자가 한 시스템에서 자신만의 샘플 데이터베이스를 가지고 있는 경우라 할지라도 테이블 이름들은 예제들에 나온 그대로 정확하게 사용할 수 있다. MySQL에서는 각 사용자가 자신의 데이터베이스를 사용하는 한, 동일한 테이블 이름들을 사용하는 것에 대해서는 상관하지 않는다. MySQL은 각 테이블을 잘 지켜서 서로간에 영향을 미치지 않도록 해준다.

1.4.1 샘플 데이터베이스 배포본 구하기

이 튜토리얼은 "샘플 데이터베이스 배포본"에 있는 파일들을 참조한다(여기서 샘플 데이터베이스의 이름을 sampdb로 사용하므로 sampdb 배포본이라고도 호칭한다). 이 파일들에는 샘플 데이터베이스를 설정하는 데 도움이 되는 쿼리들과 데이터가 들어 있다. 부록 A "소프트웨어를 구해서 설치하기"에는 배포본을 구하기 위한 지침이 들어 있다. 배포본을 풀어내면, 필요한 파일들이 들어 있는 sampdb라는 디렉터리가 있을 것이다. 샘플 데이터베이스에 속하는 예제들을 가지고 작업할 때마다 해당 디렉터리로 위치를 옮겨서 작업하기를 추천한다.

디렉토리에 상관없이 MySQL 프로그램을 사용하기 편하도록 한다면 커맨드 인터프리터의 검색 경로에 MySQL bin 디렉토리를 추가하는 것이다. 이렇게 함으로서 부록 A에 나와 있는 명령어를 사용하여 PATH 환경 변수 셋팅에 디렉토리 경로를 포함시킬 수 있다.

1.4.2 주요한 요구사항들

이 튜토리얼에 있는 예제들을 시험해보려면, 다음과 같은 몇 가지의 주요한 요구사항들이 만족되어야만 한다.

- MySQL 소프트웨어가 설치되어 있어야 한다.
- 해당 서버에 연결할 수 있는 MySQL 계정이 있어야 한다.
- 작업할 데이터베이스가 있어야 한다.

필요한 소프트웨어에는 MySQL 클라이언트와 MySQL 서버가 있다. 클라이언트 프로그램들은 반드시 작업할 컴퓨터상에 위치해야만 한다. 서버는 자신의 컴퓨터에 위치할 수도 있지만, 필수적인 것은 아니다. 서버에 연결할 수 있는 허가권만 가지고 있다면, 서버는 아무 곳에 위치해도 좋다. MySQL을 구하고자 한다면, 부록 A에 나온 지침을 따르기 바란다. 인터넷 서비스 제공업체(ISP, Internet Service Provider)를 통해 네트워크 서비스를 제공받는 상황이라면, 해당 업체가 MySQL을 서비스로 제공하는지 알아보는 것이 좋다. ISP가 MySQL 서비스를 제공하지 않고 또 설치할 계획도 없다면, MySQL을 제공하는 더 나은 다른 업체를 고르는 방법을 배우기 바란다.

MySQL 소프트웨어에 더하여, 서버에 연결해서 샘플 데이터베이스와 테이블을 만들 수 있게 하기 위한 MySQL 계정도 필요하게 된다(이미 MySQL 계정을 가지고 있는 경우에는 그것을 사용하면 되지만, 이 책에 있는 내용들을 따로 사용하기 위한 별도의 계정을 설정하고 싶은 경우도 있을 것이다).

이 시점에서, 닭이 먼저냐 달걀이 먼저냐 하는 식의 문제에 봉착하게 된다. 서버로 연결하기 위해 사용할 MySQL 계정을 설정하려면, 서버에 연결해서 작업을 해야 하기 때문이다. 서버가 실행되고 있는 호스트상에서 MySQL root 사용자로 연결해서 CREATE USER와 GRANT 문을 보내어 새로운 MySQL 계정을 생성하는 과정을 거치는 것이 보통이다. MySQL을 자신의 컴퓨터상에 설치해서 서버를 실행시키는 경우라면, 다음과 같은 과정처럼 서버에 연결해서 샘플 데이터베이스에 대한 새로운 샘플 데이터베이스 관리자 계정을 만드는데, 이름과 패스워드를 각각 sampadm과 secret로 한다(이 책 전체에 걸쳐서 다른 이름과 패스워드를 사용하고 싶다면 이 부분을 변경해서 실행하도록 한다).

```
% mysql -p -u root
Enter password: ******
mysql> CREATE USER 'sampadm'@'localhost' IDENTIFIED BY 'secret';
Query OK, 0 rows affected (0.04 sec)
mysql> GRANT ALL ON sampdb.* TO 'sampadm'@'localhost';
Query OK, 0 rows affected (0.01 sec)
```

mysql 프로그램의 명령에는 -p 옵션이 있어서 mysql이 root 사용자의 MySQL 패스워드에 대한 프롬프트를 표시해서 입력을 받을 준비를 하게 할 수 있다. 이 예에서는 ****** 로 되어 있는 곳에 패스워드를

입력한다. 여기서는 MySQL의 root 사용자에 대한 패스워드를 이미 설정해 두고 여러분이 이것이 무엇인지 알고 있다고 가정한다. 아직 패스워드를 배정하지 않은 상황이라면, password: 프롬프트에서 단순히 엔터키만 눌러서 패스워드 입력 항목을 지나간다. 하지만, root에 대한 패스워드가 없는 경우는 보안상 안전하지 않으며 가능한 한 빨리 패스워드를 설정해 두어야 한다. CREATE USER와 GRANT 문, MySQL 사용자 계정 설정 방법, 암호 변경 방법은 12장 "일반적인 MySQL 운영·관리"에서 더 자세히 설명하도록 한다.

방금 보았던 GRANT 문은 서버가 실행중인 장비에서 MySQL로 연결할 때 적절하다. 이것은 sampadm이란 이름과 secret라는 패스워드를 사용하여 서버로 연결할 수 있도록 해주고 sampdb 데이터베이스에 대한 완전한 접근을 허가한다. 그러나 GRANT는 데이터베이스를 만드는 문장은 아니다. 이에 대해서는 조금 후에 설명할 것이다.

동작하고 있는 서버와 다른 동일한 호스트로부터 MySQL에 접속하려는 계획이 아니라면, localhost를 작업하게 될 장비의 이름으로 변경한다. 예를 들면, asp.snake.net이라는 호스트에서 서버로 연결할 경우에는 GRANT 문은 다음과 같이 쓸 수 있다.

```
Mysql> CREATE USER 'sampadm'@'asp.snake.net' IDENTIFIED BY 'secret' ;
mysql> GRANT ALL ON sampdb.* TO 'sampadm'@'asp.snake.net' ;
```

서버에 대한 제어권이 없고 계정을 생성할 수 없는 경우에는 MySQL 운영·관리자에게 요청하여 계정을 설정하도록 한다. 그런 다음, 이 책의 예제에 나오는 sampadm, secret, 그리고 sampdb 자리에 운영자가 부여해주는 MySQL의 사용자 이름, 패스워드, 그리고 데이터베이스 이름으로 대체해서 사용하기 바란다.

1.4.3 서버로 연결을 수립하고 종료하기

서버로 연결하려면, 사용자의 커맨드 프롬프트(UNIX 쉘 프롬프트나 Windows에서는 DOS 콘솔)에서 mysql 프로그램을 호출한다. 명령은 다음과 같다.

```
% mysql options
```

이 책에서는 커맨드 프롬프트를 나타내는 기호로 %를 사용하기로 한다. 이것은 표준 UNIX 프롬프트들 중 한 가지이다. 또 다른 표준으로 $가 있다. Windows에서 보게 될 프롬프트의 모양은 C:\>처럼 생겼다.

mysql의 커맨드 라인의 *options* 부분은 없어도 되지만, 아마도 다음과 같이 보이는 명령들을 보내야 제대로 될 것이다.

```
% mysql -h host_name -p -u user_name
```

mysql을 호출할 때 이 옵션 전부를 더할 필요는 없지만, 최소한 이름과 패스워드는 지정해야 할 것이다. 이 옵션들의 의미하는 바는 다음과 같다.

- `-h host_name` (대체 형식: `--host=host_name`)
  연결하고자 하는 서버 호스트. MySQL 서버가 mysql을 실행하고 있는 것과 같은 컴퓨터상에서 실행되는 경우라면, 이 옵션은 보통 빼도 된다.

■ `-u user_name` (대체 형식: `--user=user_name`)
MySQL 사용자 이름. UNIX를 사용하는데 MySQL의 사용자 이름으로 로그인 이름과 동일한 것을 사용하는 경우라면, 이 옵션은 빼도 된다. mysql은 이 경우에 MySQL 이름으로 로그인 이름을 사용하게 된다.

Windows에서 디폴트 사용자 이름은 ODBC인데, 이것은 디폴트 이름으로는 그다지 쓸만한 것 같지는 않다. -u 옵션을 커맨드 라인상에서 지정하거나 환경 변수로 USER 변수를 설정해서 디폴트를 추가하도록 한다. 예를 들면, 사용자 이름을 sampadm으로 지정하려면 다음과 같은 set 명령을 사용할 수 있다.

```
C:\> set USER=sampadm
```

만약 제어판의 시스템 아이템을 사용하여 USER 환경 변수를 설정하면, 각 콘솔 윈도우에 영향을 미치게 되어 매번 이러한 명령을 수행할 필요가 없게 된다.

■ `-p` (대체 형식: `--password`)
이 옵션은 mysql이 MySQL 패스워드를 입력받기 위한 Enter password: 프롬프트를 표시하도록 한다. 예를 들면, 다음과 같다.

```
% mysql -h host_name -p -u user_name
Enter password:
```

Enter password: 프롬프트를 보게 되면, 패스워드를 타이핑한다(패스워드를 누군가가 어깨너머로 보는 것을 방지하기 위해서 입력하는 스크린상에는 패스워드가 나타나지 않는다). MySQL 패스워드는 UNIX나 Windows의 패스워드와 동일할 필요가 없다는 것에 주의하자.

-p 옵션을 빼면, mysql은 패스워드가 필요 없다고 가정하고 이에 대한 프롬프트를 표시하지 않는다.

이 옵션의 대체 형식(대체 형식: --password=*your_pass*)을 사용하면 *-pyour_pass*와 같이 옵션을 타이핑하게 되어 패스워드 값을 커맨드 라인상에서 직접 지정하게 된다. 하지만, 보안상의 이유로 이렇게 하지 않는 것이 좋다. 한 가지 예를 들어보면, 이렇게 하면 다른 사람들이 패스워드를 볼 수 있게 된다.

굳이 커맨드 라인상에서 패스워드를 지정해야겠다면, -p 옵션과 그 다음에 오는 패스워드 값 사이에 **공백이 없도록** 특별히 주의하도록 한다. 이러한 점이 -h와 -u 옵션들과는 다르기 때문에 자주 혼동되는데, h와 -u 옵션들에는 옵션과 옵션 값 사이에 공백이 있든지 없든지 상관이 없다.

MySQL의 사용자 이름과 패스워드가 각각 sampadm과 secret라고 가정한다. MySQL 서버가 동일한 호스트상에서 실행되면, -h 옵션은 사용하지 않아도 되므로 서버에 연결하는 mysql 명령은 다음과 같이 쓸 수 있다.

```
% mysql -p -u sampadm
Enter password: ******
```

명령을 입력하고 나면, mysql은 Enter password:를 표시해서 패스워드를 입력받기를 기다리는데, 이때 패스워드를 입력한다(secret를 입력하는 부분에서 ****** 가 표시된다).

모두 제대로 되면, mysql은 인사말을 표시하고 mysql> 프롬프트를 표시해서 쿼리를 보낼 수 있도록 대기하게 된다. 전체적인 시작 순서는 다음과 같다.

```
% mysql -p -u sampadm
Enter password: ******
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 13762
server version: 5.0.60-log

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql>
```

다른 컴퓨터상에서 실행하는 서버로 연결하려면, -h 옵션을 사용하여 호스트 이름을 지정하도록 해야 한다. 서버의 호스트가 cobra.snake.net이면, 명령은 다음과 같이 된다.

```
% mysql -h cobra.snake.net -p -u sampadm
```

mysql 커맨드 라인을 표시하는 다음의 대부분의 예제들에서, -h, -u, 그리고 -p 옵션은 생략해서 표시를 간단하게 하는데, 여러분들은 필요한 대로 그 옵션들을 지정해서 사용하기 바란다. Mysqlshow와 같은 다른 MySQL 프로그램을 수행할 때에는 동일한 옵션을 사용할 필요가 있다.

서버에 연결을 수립한 후에는 아무 때나 quit를 타이핑해서 해당 세션을 종료할 수 있다.

```
mysql> quit
Bye
```

\q를 타이핑하거나(UNIX 상에서) Ctrl-D를 눌러서 종료할 수도 있다.

막 MySQL을 배우려고 시작할 때에는 이러한 보안 시스템이 하고자 하는 것을 더욱 어렵게 만들기 때문에 짜증스러울 수도 있을 것이다(데이터베이스를 만들고 접근하기 위한 퍼미션이 필요하고, 서버에 연결할 때마다 사용자 이름과 패스워드를 지정해야만 한다). 그렇지만, 초보로서 이 책에서 사용된 샘플 데이터베이스를 넘어서 스스로의 레코드를 사용하기 시작하면, 바라보는 시각이 급격하게 변할 것이다. 그러면, 자신의 소중한 자료를 다른 사람이 유출(또는 더 나쁘게는 파괴)시키지 못하도록 MySQL이 지켜주는 방법에 매우 감사하게 될 것이다.

계정을 설정하는 방법에는 몇 가지가 있으므로 mysql을 실행할 때마다 연결 인자들을 일일이 타이핑할 필요는 없다. 이들에 대해서는 1.5절 "mysql로 대화식 작업할 때의 요령"에서 논의한다. 연결 과정을 단순화시키는 가장 일반적인 방법은 옵션 파일 안에 연결 인자들을 저장하는 것이다. 그러한 파일을 설정하는 방법을 알아보려고 지금 바로 그 절로 넘어가고 싶어할지도 모르겠다.

1.4.4 쿼리문 실행하기

서버에 연결하고 나면 쿼리들을 실행할 준비가 된 것이다. 이 절에서는 mysql과 대화식으로 작업을 할 때 알아야만 되는 일반적인 몇 가지 사항을 설명한다. mysql에서 어떤 쿼리를 입력하려면, 그냥 타이핑해서 써 넣으면 된다. 쿼리의 끝부분에는 세미콜론 문자(';')를 타이핑하고 엔터를 눌러야 된다. 세미콜론을 넣어줌으로써 mysql에 해당 쿼리의 작성이 완료되었음을 알리는 것이다. 쿼리를 입력하고 나면,

mysql은 이것을 서버로 전송하여 실행되도록 한다. 서버는 해당 쿼리를 처리하고 나서 결과를 다시 mysql로 돌려주는데, 이것을 mysql이 결과로 표시한다.

다음의 예는 현재의 날짜와 시간에 대해 요청하는 간단한 쿼리를 보여준다.

```
mysql> SELECT NOW();
+---------------------+
| NOW()               |
+---------------------+
| 2008-03-21 10:51:23 |
+---------------------+
1 row in set (0.00 sec)
```

SQL문을 종료하기 위한 다른 방법으로 세미콜론보다는 \g('go')를 사용하도록 한다.

```
mysql> SELECT NOW()\g
+---------------------+
| NOW()               |
+---------------------+
| 2008-03-21 10:51:28 |
+---------------------+
1 row in set (0.00 sec)
```

또는 \G를 사용할 수도 있는데, 이것은 결과를 라인당 하나씩 "수직적인" 형태로 표시하게 된다.

```
mysql> SELECT NOW(), USER(), VERSION()\G
*************************** 1. row ***************************
    NOW(): 2008-03-21 10:51:34
   USER(): sampadm@localhost
VERSION(): 5.0.60-log
1 row in set (0.03 sec)
```

짧은 출력 라인을 만드는 쿼리에 대해서, \G를 사용하는 것은 그다지 유용하지 않지만 라인의 길이가 화면 폭을 넘어갈 만큼 긴 경우에는 \G를 사용하면 출력이 좀 더 읽기 쉽게 될 수도 있다.

mysql은 SQL 문의 결과를 나타내고 결과는 문장을 처리하는 동안에 소요된 시간과 결과를 보여준다. 다음 예제에서, 행의 카운트 라인은 나타내지 않을 것이다.

mysql은 문장의 종료를 나타내는 세미콜론을 기다리기 때문에 문장을 하나의 라인에 입력하지 않아도 되며, 원한다면 여러 라인에 걸쳐서 작성해도 된다.

```
mysql> SELECT NOW(),
    -> USER(),
    -> VERSION()
    -> ;
+---------------------+-------------------+------------+
| NOW()               | USER()            | VERSION()  |
+---------------------+-------------------+------------+
| 2008-03-21 10:51:37 | sampadm@localhost | 5.0.60-log |
+---------------------+-------------------+------------+
```

쿼리의 첫 번째 라인을 입력한 다음, 프롬프트가 mysql>에서 ->로 바뀌는 것에 주목하기 바란다. 이것은 mysql이 아직 더 입력해야 할 쿼리가 남아 있다고 여긴다는 것을 말해 주며, 매우 중요한 반응인 것이다. 만일 쿼리의 끝부분에 세미콜론을 입력하는 것을 잊은 경우, 프롬프트가 변경된 것을 보고 mysql이 무엇인가를 더 기다리고 있다는 것을 알아차리게 된다. 프롬프트가 바뀌는 기능이 없다면, 세미콜론을 잊고 엔터를 누른 다음에 결과를 기다리게 되고, 왜 MySQL이 쿼리를 처리하는 데 상당히 오랜 시간이 걸리는지 의아해할 것이고, 또 mysql은 입력하는 쿼리가 끝나기를 끈기 있게 기다리기만 할 것이다. (mysql에는 여기에 더해서 두 개의 다른 프롬프트가 더 있다. 이들에 대해서는 부록 F "MySQL 프로그램 레퍼런스"에 설명되어 있다).

복수의 라인으로 된 쿼리를 타이핑하기 시작했는데, 이것이 실행되는 것을 원하지 않는 경우에는 \c를 타이핑해서 작성하던 쿼리를 취소하도록 한다. 즉, 다음과 같다.

```
mysql> SELECT NOW(),
    -> VERSION(),
    -> \c
mysql>
```

mysql이 새로운 쿼리를 위해 대기한다는 것을 나타내도록 다시 mysql>로 프롬프트가 돌아오는 것을 주목하기 바란다.

여러 라인에 하나의 문장을 입력하는 것과 반대로, 종료 지시어를 사용하여 하나의 라인에 여러 개의 문장을 입력하는 방법도 존재한다.

```
mysql> SELECT NOW();SELECT USER();SELECT VERSION();
+---------------------+
| NOW()               |
+---------------------+
| 2008-03-21 10:52:31 |
+---------------------+
+--------------------+
| USER()             |
+--------------------+
| sampadm@localhost  |
+--------------------+
+------------+
| VERSION()  |
+------------+
| 5.0.60-log |
+------------+
```

대부분의 경우, 대문자로, 소문자로, 또는 대소문자를 섞어서 쿼리를 입력하는 것에 대해서는 문제가 되지 않는다. 다음의 쿼리들은 모두 동일하다.

```
SELECT USER();
select user();
SeLeCt UsEr();
```

이 책의 예제들에서, SQL 키워드와 함수 이름들에 대해서는 대문자로, 그리고 데이터베이스, 테이블, 그리고 칼럼 이름들에 대해서는 소문자로 표시하기로 한다.

쿼리 안에서 어떤 함수를 호출할 때, 함수 이름과 그 다음에 오는 괄호 사이에는 공백이 있으면 안 된다. 어떤 경우에, 이 공백이 문법 오류를 발생시킬 수 있다.

쿼리를 파일 안에 저장해서 키보드를 통하지 않고 파일에 있는 쿼리를 mysql이 읽도록 할 수 있다. 이렇게 하려면 쉘의 입력 리디렉션 기능을 이용한다. 예를 들어, myfile.sql이라는 이름으로 된 파일에 쿼리들이 저장되어 있다고 하면, 이 쿼리들은 다음과 같은 명령으로 수행시킬 수 있다. (어떤 필요한 연결 인자 옵션을 사용해야 함을 기억해야 한다.)

```
% mysql < myfile.sql
```

파일 이름은 아무 것이든 원하는 대로 정하면 된다. 여기서는 파일 안에 SQL 문장들이 들어 있다는 것을 나타내도록 관례대로 .sql이라는 확장자를 사용하도록 한다.

mysql을 이러한 방식으로 실행시키는 것에 대해서는 1.4.7절 "새로운 행들을 추가하기"에서 sampdb 데이터베이스 안으로 데이터를 입력하게 될 때 다루게 된다. 각 문장을 손수 일일이 타이핑하는 것보다는 파일에서 INSERT 문을 mysql이 읽어서 테이블로 로드하는 것이 훨씬 더 편하다.

이 튜토리얼의 나머지 부분에서는 스스로 해볼 수 있는 많은 쿼리들을 보여주게 된다. 이들에 대해서는 쿼리 앞에 mysql> 프롬프트 표시가 있고, 이러한 예제들은 보통 쿼리의 출력도 같이 표시된다. 표시된 대로 이 쿼리들을 타이핑해서 입력할 수 있고, 그 결과로 나오는 출력은 같아야 한다. 프롬프트 없이 표시되는 쿼리들은 단지 개념만 보여주기 위한 것이므로 굳이 실행시킬 필요는 없다(원한다면 해볼 수도 있다. mysql을 사용해서 해보는 경우라면, 마지막에 세미콜론과 같은 종료 지시어를 포함하는 것을 잊지 말기 바란다).

1.4.5 데이터베이스 생성하기

샘플 데이터베이스인 sampdb를 생성하고, 이 데이터베이스 안에 테이블을 만들고, 이 테이블 안에 데이터를 로드하고, 이들 테이블 안에 있는 데이터에 대해서 몇 가지 간단한 쿼리를 수행해보는 것으로 시작한다. 데이터베이스를 사용하는 작업은 다음의 몇 가지 단계를 포함한다.

1. 데이터베이스를 만든다(초기화시킴).

2. 데이터베이스 안에 테이블들을 만든다.

3. 이 테이블들을 가지고 데이터를 삽입, 탐색, 수정, 또는 삭제 등의 작업을 진행한다.

기존의 데이터를 탐색하는 것은 데이터베이스에서 수행하는 작업 중에서 가장 보편적인 작업이다. 그 다음으로 일반적인 작업은 새로운 데이터를 삽입하고, 기존의 데이터를 갱신하거나 삭제하는 작업이 된다. 테이블을 만드는 작업은 그만큼 자주 있는 일이 아니고 데이터베이스를 만드는 작업은 가장 드문 작업이 된다. 하지만, 지금은 맨 처음부터 시작해보는 것이기 때문에, 데이터베이스를 만드는 것으로 시작해야 하며, 그 다음에 테이블을 만드는 작업을 해보고 초기 데이터를 삽입하고 난 뒤에 실제로 일반적인

작업, 즉 데이터를 검색해오는 데까지 이르게 된다.

새로운 데이터베이스를 만들려면, mysql을 사용하여 서버에 연결한 다음 CREATE DATABASE 문에 데이터베이스 이름을 지정해서 보낸다.

```
mysql> CREATE DATABASE sampdb;
```

sampdb 데이터베이스를 먼저 만든 다음에 이 안에서 작업을 진행하게 될 테이블들을 만들거나 이 테이블들의 내용으로 무엇인가를 하도록 한다.

어떤 데이터베이스를 만들게 되면, 이 데이터베이스가 기본(또는 현재의) 데이터베이스로 선택되는가? 그렇지 않다. 다음과 같은 쿼리를 수행해보면 알 수 있다. 즉, 다음과 같다.

```
mysql> SELECT DATABASE();
+------------+
| DATABASE() |
+------------+
| NULL       |
+------------+
```

NULL의 의미는 어떠한 데이터베이스도 선택되지 않았다는 것이다. sampdb를 기본 데이터베이스로 만들려면, USE 문을 보내야 한다.

```
mysql> USE sampdb;
mysql> SELECT DATABASE();
+------------+
| DATABASE() |
+------------+
| sampdb     |
+------------+
```

데이터베이스를 선택하는 또 다른 방법으로는 mysql을 호출할 때 커맨드 라인상에서 기본으로 선택할 데이터베이스의 이름을 지정하는 것이 있다. 즉, 다음과 같다.

```
% mysql sampdb
```

사실, 이것은 사용하고자 하는 데이터베이스를 지명하는 통상적인 방법이다. 어떤 연결 인자들이 필요하다면, 데이터베이스 이름 앞에 옵션으로 인자들을 지정하면 된다. 예를 들면, 다음과 같은 두 가지 명령은 sampadm이라는 사용자가 로컬 호스트의 sampdb라는 데이터베이스에 연결할 수 있도록 해준다.

```
% mysql -p -u sampadm sampdb
```

원격 호스트에서 동작하는 MySQL 서버에 연결하고자 하면, 커맨드 라인에 호스트 이름을 넣어야 한다.

```
% mysql -h cobra.snake.net -p -u sampadm sampdb
```

특별히 별도로 언급하지 않으면, 이 다음에 오는 모든 예제들은 mysql을 호출할 때 커맨드 라인상에서 sampdb 데이터베이스를 지명해서 이것을 현재의 기본 데이터베이스로 만든다는 것을 가정한다. mysql을 호출할 때 커맨드 라인상에서 데이터베이스를 지명하는 것을 잊은 경우에는 USE sampdb 문을 mysql> 프롬프트에서 보내주기만 하면 된다.

1.4.6 테이블 만들기

이 절에서는 샘플 데이터베이스인 sampdb에 필요한 테이블들을 구축해볼 것이다. 먼저, 역사 연구회에 대해 필요한 테이블들을 고려해본 다음, 성적 기록부 프로젝트에 대한 테이블들을 생각해보자. 이것은 다른 데이터베이스 책들에서 분석과 디자인, 엔티티-관계(E-R) 다이어그램, 정규화 과정 등에 대해 다루는 부분에 해당한다. 이들 모두에 대해서도 언급하면 좋겠지만, 필자는 우리가 다루려는 데이터베이스가 무엇처럼 보일 것인가에 대해서 생각할 필요가 있다고 말하는 것을 더 좋아한다. 즉, 데이터베이스에는 무슨 테이블이 있어야 하는가, 각 테이블에는 무슨 내용이 있어야 하는가, 그리고 데이터가 표현될 방법을 결정하는 데 필요한 문제들은 무엇인가 등의 질문들이다.

여기서 데이터 표현에 관한 선택은 절대적인 것이 아니다. 다른 상황에서는 애플리케이션의 요구사항과 데이터를 입력하는 방식에 따라서 비슷한 데이터일지라도 좀 다른 방식으로 고르는 것이 좋을 수도 있다.

역사 연구회에 대한 테이블

역사 연구회에 대한 테이블의 설계는 아주 단순하다. 즉, 다음과 같다.

- **president 테이블** 이 테이블에는 각각 미국의 대통령들에 대해 설명한 기록들을 담는다. 역사 연구회의 웹 사이트에서 온라인 퀴즈를 진행할 때 필요하게 된다(연구회의 소식지에서 어린이들을 위한 섹션에 나오는 인쇄된 형태의 퀴즈를 대화형으로 바꾼 형태가 된다).
- **member 테이블** 이 테이블은 연구회의 각 회원들에 관한 현재의 정보를 유지하는 데 사용된다. 회원 목록의 인쇄된 버전과 온라인 버전을 만들고, 자동적으로 회원 자격이 갱신되었음을 알리는 기념품을 전달한다던가 하는 등의 일에 사용된다.

president 테이블

president 테이블은 단순하므로, 이것을 먼저 다루어보기로 한다. 이 테이블에는 미국의 대통령들에 대한 생애에 관련된 기본적인 정보 몇 가지를 담게 된다.

- **이름** 테이블 안에 몇 가지 방법으로 이름들을 표현할 수 있다. 예를 들면, 전체 이름 모두를 단일 칼럼 안에 담도록 만들거나, first name과 last name을 별도의 칼럼에 둘 수 있다. 하나의 칼럼만 사용하는 것이 물론 더 간단하지만, 몇 가지 제약이 따르게 된다.
  - first name을 앞에 오도록 입력하면, last name만으로 정렬할 수 없게 된다.
  - last name이 앞에 오도록 입력하면, first name이 앞에 오도록 표시할 수 없게 된다.
  - 이름들을 검색하는 것이 더 어렵게 된다. 예를 들어, 특정한 last name을 찾으려면, 패턴을 사용해서 이 패턴에 이름이 대응되는지 알아보아야 할 것이다. 이와 같은 작업은 정확하게 last name만 찾는 것에 비해 비효과적이고 느리다.

  이와 같은 제약들을 피하기 위해, president 테이블은 last name과 first name을 분리해서 별도의 칼럼을 사용할 것이다.

  first name 칼럼은 middle name이나 이니셜을 담당한다. middle name으로 정렬하는 것은 흔한 일이 아니기 때문에, 이 칼럼으로 인해 정렬이 깨지는 일은 없을 것이다. 이름이 "Bush, George W." 포

맷인지 또는 "George W. Bush" 포맷인지의 여부와는 상관없이 first name 바로 뒤에 middle name이 오게 되므로, 이름 표시 또한 적절하게 이루어져야 한다.

이 외에도 좀 더 복잡한 상황이 있다. 어떤 대통령(Jimmy Carter)은 "Jr."이 이름 끝에 붙는다. 이 경우에는 어떻게 처리해야 하는가? 이름이 출력되는 포맷에 따라, 이 대통령의 이름은 "James E. Carter, Jr."로 표시될 수도 있고, "Carter, James E., Jr."로 표시될 수도 있다. "Jr."은 first name과 결합되는 것도 아니고 last name과도 결합되는 것이 아니므로, 이름 접미어를 취급하는 또 다른 칼럼을 만들게 된다. 여기서 데이터를 어떻게 나타낼 것인지를 결정하려고 할 때, 단일 값이 어떤 문제를 야기시키는지를 알 수 있게 된다. 또한 이것으로, 데이터베이스 안에 데이터를 입력하기 전에 다루게 될 데이터 값들의 타입에 관해 가능한 한 많이 알아야 하는 것이 왜 좋은지 알 수 있다. 다룰 데이터가 어떤 것인지 완전히 파악되지 않았다면, 일단 사용하기 시작한 뒤에 테이블 구조를 변경해야 할 때가 올 것이다. 이것이 재앙까지는 아니더라도, 일반적으로는 피해야 할 일임에는 분명하다.

- **출생지(도시와 주)** 이름에서와 같이, 이것 역시 단일 칼럼을 사용하거나 복수의 칼럼을 사용해서 나타낼 수 있다. 단일 칼럼을 사용하는 것이 더 단순하지만, 이름의 경우처럼 분리된 칼럼을 사용하면 단일 칼럼을 사용할 때 쉽게 할 수 없는 것들을 할 수 있다. 예를 들면, 도시와 주가 별도로 나열되면 특정 주에서 태어난 대통령에 대한 행들을 찾기가 쉬워진다. 두 가지 값으로 칼럼을 분리해서 사용해야할 것이다.

- **출생일과 사망일** 여기서의 특별한 문제는 여전히 생존해 있는 대통령들이 있기 때문에 사망일을 필수적으로 채워넣게 할 수 없다는 것뿐이다. MySQL은 "값 없음"을 의미하는 NULL이라는 특별한 값을 제공하므로, 사망일 칼럼에 이것을 사용하여 "아직 살아 있음"을 표시하도록 한다.

member 테이블

역사 연구회의 회원 목록에 대한 member 테이블은 각 행에 한 사람에 대해 기초적으로 설명하는 정보를 담는다는 점에서 president 테이블과 같다. 하지만, 각 member 행에는 좀더 많은 칼럼들이 포함된다.

- **이름** 여기서도 president 테이블에서와 마찬가지로 세 가지 칼럼을 사용하여 이름을 표현한다. 즉, last_name, first_name, 그리고 suffix가 그것이다.

- **ID 번호** 이것은 회원 자격이 처음 주어질 때, 각 회원에 배정되는 유일한 값이다. League는 ID 번호를 그 전에는 사용한 적이 없지만, 이제는 좀더 체계적으로 기록할 수 있게 되어서, 좋은 시작점이 되는 것이다(여러분들은 아마도 MySQL의 좋은 쓰임새를 찾을 것이고, 여기서 설명한 것과는 또 다른 방식으로 League의 기록들에 그 쓰임새를 적용해보는 것을 생각할지도 모르겠다. 이러한 경우가 생길 때, 이름이 아닌 숫자를 사용하는 경우라면 member 테이블 내의 행에 또 다른 회원에 관련된 테이블들을 연관시키는 것은 더욱 쉬운 일이 될 것이다).

- **기한 만료일** 회원들은 자격이 상실되지 않도록 주기적으로 회원 자격을 갱신해야만 한다. 어떤 애플리케이션에 대해서, 가장 최근에 갱신한 날짜를 사용할 수도 있겠지만, 이 연구회의 목적에는 적합하지 않다. 회원 자격은 임의의 햇수만큼 갱신될 수 있고(보통 1년, 2년, 3년, 또는 5년 단위로), 가장 최근에 갱신된 날짜로는 다음 갱신이 일어나야 하는 때를 알 수 없다. 게다가, 연구회는 평생회원 자격도 허용한다. 이를 먼 미래의 어느날을 정해서 표현할 수도 있겠지만, NULL이 좀더 적절하다고 여

겨지는데, 이는 논리적으로 "값이 없음"은 "절대로 기한이 만료되지 않음"에 부합하기 때문이다.

- **이메일 주소** 이메일 주소를 표시하게 되면 각 회원들 간에 보다 원활한 통신이 가능하게 된다. 연구회의 간사 업무를 보는 입장에서는 편지보다는 이메일을 통해서 회원 자격 갱신 통지 등을 전자적으로 전송할 수 있게 된다. 직접 우체국을 방문하는 것보다 훨씬 편하며 비용도 저렴하다. 또한 이메일을 사용해서 회원 명부의 현재 내용을 전송해서 갱신이 필요하면 정보를 갱신하도록 요청할 수도 있다.
- **우편 주소** 이메일을 가지고 있지 않은(또는 이메일에 응답이 없는) 회원들에게 연락하기 위한 수단으로서 필요한 항목이다. 여기서는 street 주소, city, state, 그리고 zip 코드에 대한 칼럼을 사용한다.

여기서 모든 연구회의 회원들은 미국 내에 거주하고 있다고 가정한다. 물론, 국제적인 관점에서 회원 자격을 고려해본다면, 이러한 가정은 너무 과도하게 단순화시킨 셈이 된다. 여러 국가를 고려한 주소체계를 다루고 싶다면, 여러 나라에서 사용되는 여러 가지 다른 주소 체계에 관한 복잡하고 지루한 문제들과 씨름하게 될 것이다. 예를 들면, Zip 코드는 국제 표준이 아니며, 몇몇 나라들에서는 주가 아닌 도(province)를 사용하기도 한다.

- **전화번호** 주소 필드들과 같이, 회원에게 연락할 방법으로 유용하게 사용된다.
- **특별 관심사항에 관한 키워드들** 모든 회원은 미국의 역사에 일반적인 관심을 가지고 있다고 가정하지만, 또한 특별한 지역에 관심을 가지는 회원들도 있을 것이다. 이 칼럼에 이러한 관심사항들을 기록한다. 회원들은 이것을 사용해서 비슷한 관심사항을 가지고 있는 다른 회원들을 찾을 수 있게 된다. (연관된 멤버에 대한 ID와 하나의 키워드를 가지고 있는 열들을 가지는 분리된 테이블을 사용하는 편이 낫다. 하지만, 여기서는 다루고자 하지 않는다.)

역사 연구회의 테이블 만들기

이제 역사 연구회에 대한 테이블들을 만들기 위한 준비가 되었다. 이 작업을 위해 CREATE TABLE 문장을 사용하는데, 이 문장은 다음과 같은 일반적인 형식을 가진다.

```
CREATE TABLE tbl_name (column_specs);
```

*tbl_name*은 해당 테이블에 부여하고자 하는 이름을 나타낸다. *column_specs*은 해당 테이블 내의 칼럼들에 대한 자세한 사항들을 제공하는데, 인덱스가 있으면 마찬가지로 여기서 인덱스를 지정한다. 인덱스는 탐색을 빠르게 만들어준다. 이에 대해서는 5장 "쿼리 최적화"에서 좀더 논의하게 된다.

president 테이블에서 CREATE TABLE 문장은 다음과 같이 만들 수 있다.

```
CREATE TABLE president
(
  last_name   VARCHAR(15) NOT NULL,
  first_name  VARCHAR(15) NOT NULL,
  suffix      VARCHAR(5) NULL,
  city        VARCHAR(20) NOT NULL,
  state       VARCHAR(2) NOT NULL,
  birth       DATE NOT NULL,
  death       DATE NULL
);
```

이 문장을 실행하기 위해서는 여러 가지 방법을 사용할 수 있다. 손수 타이핑해서 넣든지 sampdb의 create_president.sql에 있는 미리 작성된 문장을 사용하면 된다.

위의 문장을 손수 타이핑해서 입력하고 싶다면, 다음과 같이 sampdb가 현재의 데이터베이스가 되도록 mysql을 호출한다.

```
% mysql sampdb
```

그러고나서, 위에 나타난 대로 CREATE TABLE 문장을 입력하는데, mysql이 해당 문장의 끝이라는 것을 인식할 수 있도록 맨 마지막에 세미콜론을 꼭 포함시키도록 한다. 들여쓰기는 문제되지 않으며, 동일한 곳에서의 라인 브레이크도 또한 문제되지 않는다. 원한다면 길게 문장을 입력할 수도 있다.

미리 작성된 문서를 사용해서 president 테이블을 만들려면, sampdb 배포본에 있는 create_presidnet.sql 파일을 이용한다. 이 파일은 sampdb 디렉터리 안에 들어 있는데, 배포본을 풀어낼 때 만들어진다. sampdb 디렉터리로 위치를 바꾼 다음에 다음과 같은 명령을 실행한다.

```
% mysql sampdb < create_president.sql
```

어떤 방법으로 mysql을 호출하든지, 커맨드 라인상에서 데이터베이스 이름 뒤에 필요한 연결 인자들(호스트 이름, 사용자 이름, 또는 패스워드)을 지정해야 한다.

CREATE TABLE 문 안의 각 칼럼에 대한 지정사항에는 칼럼 이름, 데이터 타입(해당 칼럼이 취급하게 될 값들의 종류), 그리고 몇 가지의 칼럼 속성들이 있게 된다.

president 테이블에서는 두 가지 타입의 칼럼을 사용하게 되는데, VARCHAR과 DATE이다. VARCHAR(*n*)은 가변 길이 문자열 값을 담는 칼럼을 의미하는데, 최대 *n*문자 길이까지 허용된다. 값이 얼마나 길게 될 것인지 예측해서 *n*값을 정한다. state는 VARCHAR(2)와 같이 선언되는데, 각 주들이 약자로 두 개의 글자만 사용해서 입력되는 상황이라면 이것으로 충분한 것이 된다. 다른 문자열 값을 가지는 칼럼들은 더 긴 값들에 맞추려면 폭이 더 커져야 할 것이다.

사용하게 되는 또 다른 칼럼 타입은 DATE이다. 이 타입은 당연히 해당 칼럼이 날짜 값을 취급한다는 것을 가리킨다. 하지만, 날짜를 나타내는 포맷은 낯설지 모르겠다. MySQL에서는 '*CCYY-MM-DD*' 와 같은 날짜 포맷을 사용하는데, 여기서 *CC*, *YY*, *MM*, 그리고 *DD*는 각각 세기, 세기 내의 연도, 월, 그리고 일을 나타낸다. 이것은 날짜를 표현하는 ANSI SQL 표준이다("ISO 8601 포맷"으로도 알려져 있다). 예를 들면, "July 18, 2005"에 해당하는 날짜는 MySQL에서는 '2005-07-18' 과 같이 지정하는데, '07-18-2005' 또는 '18-07-2005' 와 다른 형식이다.

president 테이블에 있는 칼럼들에 대해 사용하는 속성들은 오직 NULL(값이 빠져도 된다)과 NOT NULL(값은 반드시 채워져야 한다) 뿐이다. 대부분의 칼럼들은 항상 값을 가지고 있어야 하므로 NOT NULL로 된다. NULL 값을 가질 수 있는 칼럼이 두 개가 있는데, 각각 suffix(대부분의 이름에는 이것이 없다)와 death(아직 생존하고 있는 대통령에 대해서는 사망일이 아직 없기 때문이다)이다.

member 테이블에 대해, CREATE TABLE 문장은 다음과 같이 된다.

```
CREATE TABLE member
(
```

```
    member_id   INT UNSIGNED NOT NULL AUTO_INCREMENT,
    PRIMARY KEY (member_id),
    last_name   VARCHAR(20) NOT NULL,
    first_name  VARCHAR(20) NOT NULL,
    suffix      VARCHAR(5) NULL,
    expiration DATE NULL,
    email       VARCHAR(100) NULL,
    street      VARCHAR(50) NULL,
    city        VARCHAR(50) NULL,
    state       VARCHAR(2) NULL,
    zip         VARCHAR(10) NULL,
    phone       VARCHAR(20) NULL,
    interests   VARCHAR(255) NULL
);
```

mysql 안에서 위의 문장을 타이핑하든지 또는 다음과 같은 명령을 실행시켜서 sampdb 배포본에 미리 작성된 파일을 사용한다. Sampdb 배포판의 member 테이블의 CREATE TABLE 문을 가지고 있는 파일은 create_member.sql이다. 이를 사용하기 위해서는 다음의 명령어를 실행시킨다.

```
% mysql sampdb < create_member.sql
```

칼럼 타입의 관점에서 보면, member 테이블 중 대부분의 칼럼들은 두 개를 제외하고는 가변 길이 문자열로 되기 때문에 그다지 흥미롭지 못하다. member_id와 expiration 칼럼이 예외적으로 각각 시퀀스 번호와 날짜를 취급하기 위한 칼럼이 된다.

member_id 회원 번호 칼럼에 대해 기본적으로 고려한 사항은 회원들 사이에서 혼동되지 않도록 이 칼럼의 값들이 중복되지 않도록 만든다는 것이다. 새로운 회원을 추가할 때 자동적으로 MySQL이 유일한 번호를 만들어 주도록 할 수 있기 때문에, 여기서 AUTO_INCREMENT 칼럼을 사용하는 것이 좋다. 이 칼럼이 단지 숫자만을 포함하더라도, member_id에 대한 선언은 여러 부분을 가진다.

- INT는 해당 칼럼이 정수를 취하는 것을 의미한다(지수부가 없는 수치 값).
- UNSIGNED는 음의 수를 허용하지 않는다.
- NOT NULL은 칼럼 값이 반드시 채워져야 함을 의미한다(ID 번호가 없는 회원은 없다는 것을 뜻하게 된다).
- AUTO_INCREMENT는 MySQL에 있는 특별한 속성이다. 이것은 해당 칼럼이 시퀀스 번호를 취한다는 것을 나타낸다. AUTO_INCREMENT의 작동 방식을 설명하자면 이렇다. 즉, 새로운 member 테이블 레코드를 생성할 때 member_id 칼럼에 대한 값이 누락(또는 NULL)되면, MySQL은 자동으로 다음 순서의 숫자를 만들어서 해당 칼럼에 대입시킨다. 이러한 작업은 NULL이라는 값을 칼럼에 인가할 때 발생한다. MySQL이 값을 자동으로 생성하기 때문에 AUTO_INCREMENT이 유일한 ID를 새로운 멤버에게 할당하는 일을 자동으로 해줄 수 있다.

PRIMARY KEY 절은 member_id 칼럼에 대한 인덱스를 만들어서 탐색을 빠르게 한다는 것과 해당 칼럼 내의 각 값이 유일(unique)해야만 한다는 것을 나타낸다. 값이 유일해야 한다는 속성은 실수로 동일한

ID를 두 번 이상 중복해서 사용하지 못하게 해주기 때문에, 회원 ID 값에 잘 어울린다. 한편으로, MySQL은 모든 AUTO_INCREMENT 칼럼에 몇 가지 종류의 유일한(unique) 인덱스 중 한 가지를 필수적으로 요구하므로, 이러한 것 없이 테이블을 정의하는 것은 적법하지 않다. (어떤 PRIMARY KEY 칼럼은 NOT NULL 값을 가져야만 한다. 만약 NOT NULL을 member_id 정의에서 빼먹었다면 MySQL이 자동으로 이를 추가할 것이다.)

AUTO_INCREMENT와 PRIMARY KEY에 대한 내용을 잘 이해하지 못한다면, 각 회원들에 대한 ID 번호를 만드는 신기한 방법쯤으로 생각해 두기 바란다. 이것은 그저 유일한 값을 지니는 값이란 것밖에는 그다지 특별한 것은 아니다. (AUTO_INCREMENT 칼럼을 사용하는 방법에 대해 더 많은 것을 배우게 될 텐데, 3장 "데이터 타입"에서 이러한 사항이 자세히 다루어진다).

expiration 칼럼은 DATE이다. 이 칼럼은 디폴트 값으로 NULL을 가진다. 날짜가 없음을 의미하는 NULL이 들어가게 된다. 이전에 언급한 바와 같이, 이러한 이유는 expiration 칼럼에 NULL을 채우게 되므로, 이 회원은 종신회원임을 나타내는 방법을 사용하는 것이다.

이제 MySQL이 두 개의 테이블을 만들도록 했으므로, 기대한 대로 작업이 잘 되었는지 점검해본다. mysql 프로그램으로는 다음과 같은 쿼리를 보내서 president 테이블의 구조를 볼 수 있다.

```
mysql> DESCRIBE president;
+------------+-------------+------+------+---------+-------+
| Field      | Type        | Null | Key  | Default | Extra |
+------------+-------------+------+------+---------+-------+
| last_name  | varchar(15) | NO   |      |         |       |
| first_name | varchar(15) | NO   |      |         |       |
| suffix     | varchar(5)  | YES  |      | NULL    |       |
| city       | varchar(20) | NO   |      |         |       |
| state      | varchar(2)  | NO   |      |         |       |
| birth      | date        | NO   |      |         |       |
| death      | date        | YES  |      | NULL    |       |
+------------+-------------+------+------+---------+-------+
```

만약 DESCRIBE member 문을 사용한다면, mysql은 member 테이블에 대한 정보들을 보여줄 것이다.

DESCRIBE는 테이블에서 칼럼의 이름을 잊었을 때 매우 유용하며, 얼마만큼의 길이를 가지는지 또는 데이터 타입이 어떤 것인지 알고자 할 때도 필요하다. 또한 MySQL이 테이블 행들에 칼럼들을 저장하는 순서를 알아내고자 할 때에도 유용하다. INSERT나 LOAD DATA 문을 사용할 때는 디폴트 칼럼 순서로 칼럼 값들을 나열해야 되기 때문에 이 순서가 중요하다.

DESCRIBE가 만들어 주는 정보는 다른 방법으로도 얻을 수 있다. DESC라는 약자로 써도 되고 EXPLAIN이나 SHOW 문으로 써도 된다. 다음의 문장들은 모두 동의어들이다.

```
DESCRIBE president;
DESC president;
EXPLAIN president;
SHOW COLUMNS FROM president;
SHOW FIELDS FROM president;
```

이 문장들은 또한 특정한 칼럼들로 출력을 제한할 수 있도록 해준다. 예를 들면, SHOW 문장의 마지막에 LIKE 절을 추가해서 주어진 패턴에 대응되는 칼럼 이름들에 대한 정보만 표시하도록 할 수 있다.

```
mysql> SHOW COLUMNS FROM president LIKE '%name';
+------------+-------------+------+-----+---------+-------+
| Field      | Type        | Null | Key | Default | Extra |
+------------+-------------+------+-----+---------+-------+
| last_name  | varchar(15) | NO   |     |         |       |
| first_name | varchar(15) | NO   |     |         |       |
+------------+-------------+------+-----+---------+-------+
```

여기서 사용된 '%' 문자는 특별한 와일드카드 문자인데, 이에 대해서는 1.4.9 절의 하위절에 나오는 "패턴 대응"에서 설명한다. SHOW FULL COLUMN 은 SHOW COLUMN 과 같다. 하지만, 다른 부가적인 칼럼 정보를 나타낸다.

MySQL 에서 또 다른 형태의 정보들을 얻는 데 유용하게 사용할 수 있는 다른 형식의 SHOW 문장이 있다. SHOW TABLES 는 현재의 데이터베이스 안에 있는 테이블들을 나열하므로, sampdb 데이터베이스 안에 지금까지 만든 두 개의 테이블에 대해, 다음과 같은 출력을 얻을 수 있다.

```
mysql> SHOW TABLES;
+------------------+
| Tables_in_sampdb |
+------------------+
| member           |
| president        |
+------------------+
```

SHOW DATABASES 는 연결된 서버가 관리하는 데이터베이스들을 나열해준다.

```
mysql> SHOW DATABASES;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| menagerie          |
| mysql              |
| sampdb             |
| test               |
+--------------------+
```

데이터베이스들의 목록은 서버마다 다르지만, 적어도 information_schema 와 sampdb .information _schema 만큼은 보여야 한다. 이는 항상 존재하는 특별한 데이터베이스이며, sampdb 는 손수 만든 데이터베이스이다. test 라는 데이터베이스는 MySQL 설치 과정에서 생성된다. 접근 권한에 의해서 mysql 이라는 이름으로 된 데이터베이스는 MySQL 의 누가 무엇을 할 수 있는지에 대한 권한들을 제어하는 권한 테이블들을 취급한다.

mysqlshow 클라이언트 프로그램은 SHOW 문이 표시하는 것과 동일한 정보를 위한 커맨드 라인 인터

페이스를 제공한다. Mysqlshow를 실행시키면 username, password, hostname을 위한 커맨드 라인 옵션을 제공한다. 이러한 옵션들은 mysql을 실행시켰을 때에도 동일하다. 아무런 인자 없이 호출하면, mysqlshow는 데이터베이스들의 목록을 표시한다.

```
% mysqlshow
+--------------------+
| Databases          |
+--------------------+
| information_schema |
| menagerie          |
| mysql              |
| sampdb             |
| test               |
+--------------------+
```

데이터베이스 이름을 써 주면, mysqlshow는 주어진 데이터베이스 안의 테이블들을 보여준다.

```
% mysqlshow sampdb
Database: sampdb
+-----------+
| Tables    |
+-----------+
| member    |
| president |
+-----------+
```

데이터베이스와 테이블 이름을 지명해서 mysqlshow를 실행하면, SHOW COLUMNS 명령을 사용했을 때와 아주 비슷하게 해당 테이블 안에 있는 칼럼들에 대한 정보를 볼 수 있다.

성적 기록부 프로젝트에 대한 테이블

성적 기록부 프로젝트에서 어떤 테이블들이 필요한지 알아보기 위해서, 우선 종이로 된 성적표를 사용할 때 점수를 어떤 식으로 기록하게 되는지를 생각해보도록 하자. [그림 1.2]는 성적 기록부 중의 한 페이지를 보여준다. 이 페이지의 주요 부분은 성적을 기록하는 표가 된다. 또한 이 점수들의 뜻이 통하도록 하기 위한 필수 정보가 더 있다. 학생의 이름과 ID 번호는 표의 왼편에 위쪽에서 아래쪽으로 나열된다(단순하게 하기 위해서, 네 명의 학생만 적었다). 표의 위쪽에는 퀴즈와 시험을 시행했던 날짜들을 왼

students		scores						
		Q	Q	T	Q	Q	T	
ID	name	9/3	9/6	9/9	9/16	9/23	10/1	. . .
1	Billy	14	10	73	14	15	67	. . .
2	Missy	17	10	68	17	14	73	. . .
3	Johnny	15	10	78	12	17	82	. . .
4	Jenny	14	13	85	13	19	79	. . .
. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .

[그림 1.2] 성적 기록부의 예

편에서 오른편으로 주욱 적는다. 이 그림에는 9월 3일, 6일, 16일, 그리고 23일에 퀴즈를 냈었고 9월 9일과 10월 1일에 시험을 치렀다는 것을 보여준다.

데이터베이스를 사용하여 이와 같은 정보를 보존하기 위해 score 테이블을 만든다. 이 테이블에는 어떤 행들이 포함되어야 할까? 답은 쉽게 나온다. 각 행에 대해, 학생의 이름, 퀴즈 또는 시험의 시행날짜, 그리고 점수가 필요하다. [그림 1.3]에서는 성적 기록부에 있는 점수들이 이와 같은 테이블 안에서 표시되는 모습을 보여준다(MySQL은 날짜를 표시할 때 '*CCYY-MM-DD*' 포맷을 사용해서 나타낸다).

하지만, 아직 몇 가지 정보가 빠져 있으므로 이런 식으로 테이블을 구성하는 것에는 문제가 있다. 예를 들어, [그림 1.3]에 있는 행들을 보면, 여기에 있는 점수들이 퀴즈에 대한 것인지 시험에 대한 것인지의 여부를 알 수가 없다. 퀴즈와 시험의 가중치가 다르기 때문에 최종적으로 학점을 정할 때 점수의 유형을 아는 것이 중요하다. 주어진 날짜에 비추어서 점수의 범위로부터 유형을 추론해볼 수는 있지만(시험이 퀴즈보다 더 비중 있게 계산된다), 이런 방법은 추론에 의존하고 데이터 자체에 명시적으로 표기된 것이 아니므로 지저분한 방법일 수밖에 없다.

score 테이블

name	date	score
Billy	2008-09-23	15
Missy	2008-09-23	14
Johnny	2008-09-23	17
Jenny	2008-09-23	19
Billy	2008-10-01	67
Missy	2008-10-01	73
Johnny	2008-10-01	82
Jenny	2008-10-01	79

[그림 1.3] 초기 score 테이블의 개요

각 행 안에 점수의 유형을 기록해둠으로써 점수들을 구별할 수 있는데 예를 들면, [그림 1.4]에서와 같이 각 행에 "시험"이나 "퀴즈"라는 것을 나타내기 위해 각각 'T' 나 'Q' 라는 문자를 가질 수 있는 점수 테이블을 위한 칼럼을 추가하게 되는 것이다. 이러한 작업은 데이터 안에 점수의 유형을 명시적으로 만드는 장점이 있으며, 반대로 이 정보로 인해 취급하게 되는 자료가 늘어나게 된다는 단점이 있다. 특정한 날짜에 해당하는 모든 행들에 대해서는 점수 유형 칼럼이 동일한 값을 가지게 된다는 것을 알 수 있다. 9월 23일에 대한 점수들은 모두 'Q' 의 유형을 가지고 있고, 10월 1일에 대한 점수들은 모두 'T' 의 유형을 가지게 된다. 이러한 점은 별로 탐탁치 않다. 이러한 식으로 퀴즈나 시험에 대한 일련의 점수들을 기록한다면, 각각의 새로운 행에 대해서 동일한 날짜에 기록하게 될 뿐 아니라 동일한 점수 유형도 계속해서 기록하게 된다. 누가 이렇게 장황하게 정보를 입력하고 싶겠는가?

다른 표현을 만들어보도록 하자. score 테이블 안에 점수 유형을 기록하지 말고, 대신에 날짜에서 유형을 찾을 수 있도록 해보겠다. 날짜들의 목록을 만들어 두고 이것을 사용해서 각 날짜에 "grade_event"(퀴즈 또는 시험)가 발생했는지 기록해 둘 수 있다. 그러면 날짜와 사건 목록에 있는 정보를 결합해서, 주어진 점수가 퀴즈의 점수인지 시험의 점수인지 결정할 수가 있다. 단지 score 테이블의 행에 있는 날

score 테이블

name	date	score	type
Billy	2008-09-23	15	Q
Missy	2008-09-23	14	Q
Johnny	2008-09-23	17	Q
Jenny	2008-09-23	19	Q
Billy	2008-10-01	67	T
Missy	2008-10-01	73	T
Johnny	2008-10-01	82	T
Jenny	2008-10-01	79	T

[그림 1.4] score 테이블의 개요. 점수 유형을 포함시켜서 다시 만듦

짜와 grade_event 테이블 안에 있는 날짜를 대응시키는 것으로 이벤트의 유형을 알아낼 수 있는 것이다. [그림 1.5]는 이러한 테이블의 개요를 보여주며 9월 23일이라는 날짜를 가진 score 테이블의 행에 대해 어떤 식으로 연결이 되는지를 설명하고 있다. grade_event 테이블 안에 해당되는 행을 가진 행을 대응시켜서, 퀴즈에서 나온 점수를 보게 된다.

score 테이블

name	date	score
Billy	2008-09-23	15
Missy	2008-09-23	14
Johnny	2008-09-23	17
Jenny	2008-09-23	19
Billy	2008-10-01	67
Missy	2008-10-01	73
Johnny	2008-10-01	82
Jenny	2008-10-01	79

grade_event 테이블

date	type
2008-09-03	Q
2008-09-06	Q
2008-09-09	T
2008-09-16	Q
2008-09-23	Q
2008-10-01	T

[그림 1.5] score와 grade_event 테이블. 날짜로 결합되어 있음.

이러한 방법이 추론에 의해서 점수 유형을 알아내는 것보다 훨씬 좋다. 그 대신, 데이터베이스 안에서 명시적으로 기록된 데이터에서 직접 유형을 가져오게 된다. score 테이블 안에서 점수의 유형들을 기록하면 점수 레코드마다 한 번씩이 아닌, 각 유형을 단 한 번만 기록하므로 이것이 더 좋다.

하지만, 이제는 복수의 테이블에 있는 정보들을 연결하고 있다. 만일 여러분이 필자와 비슷하다면, 이런 종류의 것을 처음 접하게 될 때, "와, 그것은 매우 괜찮은 생각인데요? 하지만 매번 찾으려 할 때마다 많은 작업을 해야 되지 않나요? 문제를 더 복잡하게 만드는 것 같은데요?" 라고 말할지 모르겠다.

어떤 면에서는 옳은 생각일 수도 있다. 좀더 많은 작업을 해야 한다. 두 목록의 레코드들을 취급하는 것은 하나의 목록만 취급하는 것보다 좀더 복잡하다. 그러나 성적 기록부를 조금 다른 시각으로 보도록 하자([그림 1.2] 참조). 이미 두 세트의 레코드를 취급하고 있지 않은가? 다음과 같은 사실들을 고려해보자.

- 점수표 안의 칸들을 사용해서 점수를 보관하는데, 여기서 각 칸에는 학생의 이름과 날짜로 인덱스가 되어 있다(표의 왼편의 아래쪽 방향과 맨 위쪽의 오른쪽 방향이 된다). 이것은 한 세트의 레코드를

나타낸다. 이것은 score 테이블의 내용과 닮아 있다.

- 각 날짜가 나타내는 이벤트의 종류가 무엇인지 어떻게 알 수 있는가? 해당 날짜 위에 'T' 나 'Q' 라고 적어 놓았으므로, 표의 상단에 따라 날짜와 점수 유형 간의 결합을 유지할 수 있다. 이것은 두 번째 세트의 레코드를 나타내는 것이 된다. grade_event 테이블 내용과 유사하다.

다시 말하면, 이에 대해 그런 식으로 생각하지는 않을지라도, 두 개의 테이블 안에 정보를 유지하는 방법으로 수행하는 제안보다는 실제로 성적 기록부와 다른 어떤 무엇도 하지 않을 것이다. 실제적인 차이점은 두 가지 종류의 정보가 종이에 기반한 성적 기록부 안에서 그다지 명시적으로 구분되지 않는다는 것이다.

성적 기록부 내의 페이지에는 우리가 생각하는 정보에 관한 것과 데이터베이스 안에 어떻게 정보를 넣는지 알아내는 데 대한 어려움을 보여준다. 우리는 다른 종류의 정보를 통합해서 전체로서 이들을 해석하려고 한다. 데이터베이스들은 그처럼 작업하지 않는데, 이러한 이유로 데이터베이스들이 때로 인공적이고 비자연적으로 보이는 것이다. 자연적으로 정보를 통합하려는 경향 때문에 그것이 몇 배나 더 어려워졌는데, 심지어 우리가 단지 하나 대신에 복수 타입의 데이터를 가졌을 때를 깨닫기 위해서조차 그렇다. 이런 이유 때문에, 여러분은 여러분의 데이터가 어떻게 표현되어야 하는지에 대해 "데이터베이스가 생각하는 대로 생각하라" 라는 명제에의 도전이라는 것을 알아챌 것이다.

[그림 1.5]에 나타낸 레이아웃에 의해 grade_event 테이블에 부과된 한 가지 요구사항은 각 날짜가 score 테이블과 grade_event 테이블에 있는 행을 함께 링크하는 데 사용되므로 이 날짜들이 유일해야 한다는 것이다. 다시 말하면, 같은 날에 퀴즈나 시험을 두 번 이상 치를 수 없다는 것이다. 그렇게 한다면, score 테이블에는 두 세트의 레코드가 있게 되고 grade_event 테이블에는 두 개의 레코드가 있게 되는데, 이들 모두는 동일한 날짜를 가지게 되고 score 행들을 grade_event 행들에 대응할 방도가 없게 되는 것이다.

하루에 두 번 이상의 퀴즈나 시험을 치르는 일만 없다면 이러한 문제는 절대로 발생하지는 않는다. 그러나 이렇게 두 번 이상 치르지 않을 것이라고 가정하는 것이 정말로 유효한 것인가? 그렇게 여겨지기도 할 것이다. 무엇보다도, 같은 날에 시험과 퀴즈를 치를 만큼 여러분 스스로를 가학적이라고 생각하지는 말라. 하지만, 필자가 의심이 많은 것이라면 필자를 용서해주기 바란다. 나는 종종 사람들로부터 그들의 데이터에 관한 불만을 듣는데, "이러한 별난 경우는 절대로 일어나지 않을 것이다" 라는 주장이다. 그러면, 이러한 괴상한 경우가 종종 일어난다는 것이고, 이러한 이상한 경우 때문에 생기는 문제들을 고치기 위해서 테이블을 다시 설계해야만 한다.

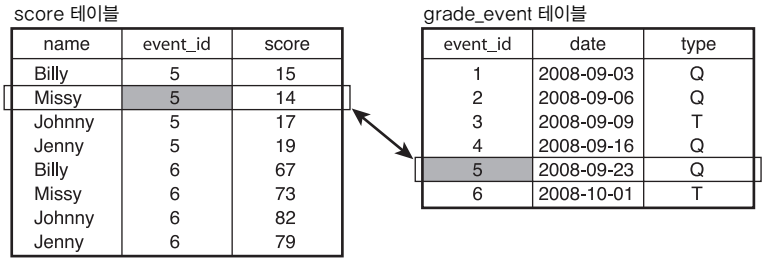
가능한 문제들을 앞서서 예측해보고 그런 경우를 어떻게 다룰 것인지 생각해보는 것이 좋다. 따라서 동일한 날에 두 세트의 점수들을 기록할 필요가 있다고 가정해보자. 이런 경우는 어떻게 처리해야 하는가? 밝혀진 대로, 이 문제는 그다지 다루기 어렵지는 않다. 데이터를 약간 다르게 배치하면, 주어진 날에 복수의 이벤트가 있어도 문제가 발생하지 않을 것이다.

1. 하나의 칼럼을 grade_event 테이블에 추가하고 이것을 사용하여 테이블 안에 각 행으로 유일한 숫자를 대입시킨다. 사실상, 이러한 작업으로 각 이벤트마다 자신만의 고유한 ID 번호를 부여하는 것이므로, 이 칼럼을 event_id 라고 하겠다(이렇게 하는 것이 조금 이상하게 보일지 모르겠지만, [그림 1.2]

에 있는 성적 기록부는 이미 이러한 속성을 가지고 있다는 것을 고려해보기 바란다. 이벤트 ID는 성적 기록부의 점수표에 있는 칼럼 번호에 해당하는 것이다. 이 번호는 점수표에서 명시적으로 "event ID"라고 분류해서 적어놓지는 않았지만, 바로 그것에 해당한다).

2. score 테이블에 점수들을 넣을 때, 날짜가 아닌 event ID를 기록한다.

이러한 변경으로 인한 결과는 [그림 1.6]에서 볼 수 있다. 이제, 날짜가 아니라 event ID를 사용하여 점수와 event 테이블을 함께 연결시키고, grade_event 테이블을 사용하여 각 점수의 유형뿐만 아니라 일이 발생한 날짜까지도 정할 수 있게 되었다. 또한 grade_event 테이블 안에서 유일해야 하는 것은 날짜가 아니라 event ID가 되는 것이다. 이렇게 해서, 같은 날에 시험과 퀴즈를 수십 번씩 치를 수 있고, 기록들을 직접 테이블에 보관할 수 있게 된 것이다(이러한 소식에 학생들은 떨게 될지도 모르겠다).



[그림 1.6] event ID로 연결된 score 테이블과 grade_event 테이블

불행히도, 인간적인 관점에서는 [그림 1.6]에 있는 테이블의 모양새가 이전 것보다는 덜 만족스럽게 보인다. score 테이블은 쉽고 분명한 의미를 가지는 칼럼이 거의 없기 때문에 더 추상적이다. 앞쪽의 [그림 1.4]에 나와 있는 테이블의 배치가 보고 이해하기에는 더 쉬운데, score 테이블에 날짜에 대한 칼럼과 점수 유형에 대한 칼럼이 모두 있기 때문이다. [그림 1.6]에 나와 있는 현재의 score 테이블은 두 칼럼 모두 가지고 있지 않다. "event ID"가 들어 있는 score 테이블을 살펴보고 싶은 사람이 있을까? 이것만 보면 무슨 뜻인지 알기 힘들다.

여기서 교차로에 도달했다. 전자적으로 학점을 기록할 수 있는 가능성과 학점을 배정할 때 손수 지루한 계산을 하지 않아도 된다는 유혹에 이끌렸다. 하지만, 데이터베이스 안에서 점수 정보를 실제로 어떻게 표현하는지 보게 된 다음에, 그 정보가 추상적이고 직관과 동떨어진 것처럼 보이게 되면 의욕이 사라지기도 할 것이다.

다음과 같은 질문을 자연스럽게 할 수도 있다: "데이터베이스를 사용하지 않는 것이 더 좋지 않겠어요? 아마 MySQL은 나와 맞지 않는 것 같습니다." 예상하겠지만, 필자는 그러한 질문에 부정적으로 대답할 텐데, 그렇지 않으면 이 책을 더 이상 진행할 필요가 없어져버리기 때문이다. 그러나 어떤 작업을 해나가는 방식에 대해 생각할 때는, 여러 대안을 고려해서 MySQL이나 다른 데이터베이스 시스템을 사용하는 것이 더 나은지, 아니면 스프레드시트 프로그램 등과 같은 그밖의 다른 프로그램을 이용하는 것이 나은지 질문해보는 것이 좋다.

- 성적 기록부는 행과 칼럼으로 구성되어서 마치 스프레드시트와 같다. 따라서 개념적으로, 그리고 실제적으로 성적 기록부와 스프레드시트는 매우 비슷하다.
- 스프레드시트 프로그램은 계산을 수행할 수 있으므로, 계산 필드를 사용해서 각 학생들의 점수들을 모두 합산할 수 있다. 퀴즈와 시험에 대해 다른 가중치가 적용되기 때문에 약간의 공식이 필요하지만 그리 어려운 것은 아니다.

한편, 단지 데이터의 일부분만 살펴보기를 원한다면(예를 들면, 퀴즈 부분만 또는 시험 부분만), "남학생 대 여학생"과 같은 비교를 수행하거나 또는 통계 정보를 유연한 방법으로 표시하는데, 이것은 또 다른 이야기가 된다. 이 같은 기능에 대해 스프레드시트가 그다지 잘 처리하지 못하는 반면, 관계형 데이터베이스 시스템은 이러한 작업을 쉽게 수행해나갈 수 있다.

고려해야 할 또 다른 점은 관계형 데이터베이스 안에서 표현되는 것과 같은 데이터의 추상적이고 분리된 특성은 어쨌거나 실제로는 그다지 큰 문제는 아니라는 것이다. 데이터베이스를 설정할 때 무의미한 방법으로 데이터를 배치하지 않도록 할 수 있는 표현에 대해 생각해보아야 한다. 하지만, 일단 표현 방법을 결정한 다음에는 데이터베이스 엔진을 잘 활용하여 의미 있는 정보로서 데이터를 나타내야 할 것이다. 데이터는 더 이상 한 다발의 분리된 조각처럼 보이지 않을 것이다.

예를 들면, score 테이블에서 점수들을 검색할 때, event ID들을 보는 것을 원하지는 않는다. 그것은 문제가 아니다. 데이터베이스는 event ID에 기초해서 grade_event 테이블에서 날짜들을 찾게 될 것이고 이것들을 여러분에게 보여준다. 더구나 이 점수들이 시험에 대한 것인지, 퀴즈에 대한 것인지의 여부도 알고 싶을 것이다. 이것도 별 문제가 아니다. 데이터베이스는 동일한 방법으로 event ID를 사용하여 점수 유형을 찾을 것이다. 기억할 것은 바로 이러한 점이 MySQL과 같은 관계형 데이터베이스 시스템의 탁월한 점이라는 것이다. 즉, 한 가지를 다른 것에 연관시켜서 복수의 소스로부터 정보를 뽑아내어, 정말로 알고자 하는 것을 사용자에게 제공해준다는 개념이다. 성적 기록부의 데이터를 다루는 이러한 경우에서, event ID들을 사용하여 적절한 정보를 꺼내오는 일을 MySQL이 하므로 사용자들이 그럴 필요가 없게 된다.

이제, 시험 삼아 MySQL이 한 가지를 다른 것에 연관시키는 작업을 한번 해보도록 하겠다. 2008년 9월 23일에 치러진 점수들을 찾아본다고 가정해보자. 특정한 날짜에 주어진 이벤트에 대한 점수들을 꺼내오는 쿼리는 다음과 같다.

```
SELECT score.name, grade_event.date, score.score, grade_event.category
FROM score, INNER JOIN grade_event
ON score.event_id = grade_event.event_id
WHERE grade_event.date = '2008-09-23';
```

조금 겁이 나는가? 이 쿼리는 학생의 이름, 날짜, 점수, 그리고 score 테이블 행을 grade_event 테이블 행에 결합시켜서(관련지어서) 점수의 유형을 검색한다. 결과는 다음과 비슷할 것이다.

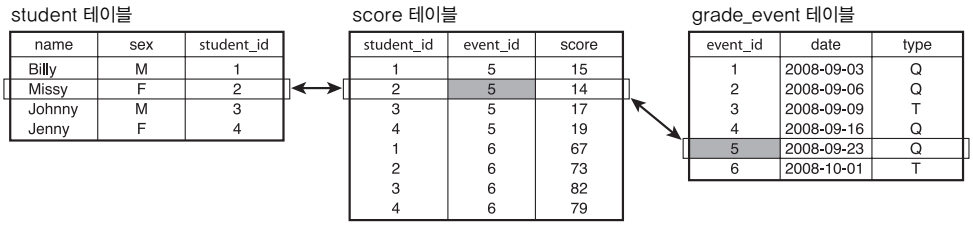
```
+--------+------------+-------+------+
| name   | date       | score | type |
+--------+----------- +-------+------+
| Billy  | 2008-09-23 |    15 | Q    |
```

```
| Missy   | 2008-09-23 |    14 | Q    |
| Johnny  | 2008-09-23 |    17 | Q    |
| Jenny   | 2008-09-23 |    19 | Q    |
+---------+------------+-------+------+
```

이러한 포맷으로 된 정보에 대해서 익숙해져야 하는가? 그래야만 한다. 이것은 [그림 1.4]에 있는 테이블의 모양새와 같다. 그리고 이러한 결과를 얻기 위해 event ID를 알아야 할 필요가 없다. 관심이 있는 날짜를 지정해주고, MySQL이 그 날짜에 해당하는 점수 행들을 찾도록 한다. 우리에게 의미 있는 형태로 데이터베이스에서 정보를 가져오게 될 때 이런 식으로 정보를 분리하고 단절시킨 것으로 인해 좌절할지 모르겠지만, 그럴 필요는 없다.

물론, 그러한 쿼리를 바라보고 난 후에도, 그밖에 또 이상한 생각이 들지도 모르겠다. 다시 말해서, 길고 복잡한 듯 보인다. 단지 주어진 날짜에 대한 점수들을 찾기 위해 해야 할 것이 그리도 많은 것인가? 맞다, 그렇다. 하지만, 쿼리를 보내고자 할 때마다 매번 몇 줄에 걸친 SQL 문장을 전부 타이핑하지 않도록 하는 방법은 있다. 일반적으로, 이러한 쿼리를 수행하는 방법을 한 번 알고 나면 필요할 때마다 쉽게 반복할 수 있도록 쿼리를 저장한다. 1.5절 "mysql로 대화식 작업할 때의 요령"에서 이러한 작업을 어떻게 하는지 보게 될 것이다.

필자는 실제로 조준점을 다소 바꾸어서 그러한 쿼리를 보여줄 것이다. 그것은 믿거나 말거나, 점수들을 검색해내는 데 실제로 사용하게 되는 것보다는 조금 더 단순하다. 이렇게 하는 이유는 우리들의 테이블 모양새를 한 번 더 변경할 필요가 있기 때문이다. score 테이블 안에 있는 학생 이름을 기록하는 대신, 유일한 student ID를 사용할 것이다(이것은 성적 기록부의 "Name" 칼럼이 아닌 "ID" 칼럼에 있는 값을 사용한다는 것이다). 그러고나서, student라고 하는 또 다른 테이블을 만드는데, 여기에는 name과 student_id 칼럼이 포함된다([그림 1.7]).



[그림 1.7] student ID와 event ID에 연결된 student, score, 그리고 grade_event 테이블

왜 이렇게 수정하는가? 동명이인인 학생이 있을 수 있다는 것이 한 가지 이유가 된다. 유일한 학생 ID 번호를 사용하면 이들의 점수를 따로 관리할 수 있다(이것은 날짜가 아니라 유일한 event ID를 사용하여 같은 날에 치러진 시험과 퀴즈에 대해서 점수를 따로따로 분리해서 얘기할 수 있는 것과 매우 유사한 상황이다). 테이블의 얼개에 이러한 변경을 가하고 난 뒤, 주어진 날짜에 대해 점수를 꺼내오는 데 실제로 사용하게 되는 쿼리는 좀 더 복잡해질 것이다.

```
SELECT student.name, grade_event.date, score.score, grade_event.category
FROM grade_event INNER JOIN score INNER JOIN student
ON grade_event.event_id = score.event_id
AND score.student_id = student.student_id
WHERE grade_event.date = '2008-09-23';
```

이 쿼리의 의미가 즉각적으로 파악되지 않아서 고민 중이라면, 그럴 필요 없다. 대부분의 사람들도 그렇다. 이 튜토리얼을 좀더 진행해가다 보면 이 쿼리를 다시 보게 되는데, 지금과 그때의 차이는, 그 때에는 이것이 무슨 의미인지 알아볼 수 있게 된다는 것이다. 지금 농담하는 것이 아니다.

[그림 1.7]에서 student 테이블에 성적 기록부에 있지 않은 무엇인가를 추가한 것에 주목하기 바란다. 성별을 위한 칼럼이 그것이다. 이 칼럼을 이용해서 학급 내의 남학생과 여학생의 수를 세는 것과 같은 간단한 일들을 할 수 있고, 또 남학생과 여학생의 성적을 비교하는 것과 같은 좀 더 복잡한 일을 할 수도 있다.

이제 성적 기록부 프로젝트에 대한 테이블을 설계하는 일은 거의 다 끝났다. 출석 점검에 대해 결석을 기록하기 위한 테이블이 하나 더 필요할 뿐이다. 이것의 내용은 상대적으로 직관적이다. 학생 ID 번호와 날짜가 내용이 된다([그림 1.8] 참조). 테이블 안의 각 행은 주어진 날에 결석한 학생을 나타낸다. 학기말에 MySQL의 계산 기능을 이용하여 테이블의 내용을 통계 처리해서 각 학생들의 결석 횟수를 정리할 것이다.

absence 테이블

student_id	date
2	2008-09-02
4	2008-09-15
2	2008-09-20

[그림 1.8] absence 테이블

student 테이블

이제 학점을 기록하는 테이블이 어떻게 생겼는지 알게 되었으므로, 이를 생성할 준비가 된 셈이다. student 테이블에 대한 CREATE TABLE 문은 다음과 같다.

```
CREATE TABLE student
(
  name         VARCHAR(20) NOT NULL,
  sex ENUM     ('F','M') NOT NULL,
  student_id INT UNSIGNED NOT NULL AUTO_INCREMENT,
  PRIMARY KEY (student_id)
) ENGINE=InnoDB;
```

여기서 CREATE TABEL 문에 대한 새로운 것을 추가하고자 한다. 그리고 간단히 그것의 목적에 대해서 설명하도록 하겠다.

위의 문장을 mysql 프로그램 안에서 타이핑해서 입력하든지 또는 다음의 명령을 실행시킨다.

```
% mysql sampdb < create_student.sql
```

CREATE TABLE 문은 student라는 이름의 테이블을 만드는데, 여기에는 name, sex, 그리고 student_id라는 세 개의 칼럼이 포함된다.

name은 20 문자까지 취급할 수 있는 가변 길이 문자열 칼럼이다. 여기서의 name은 역사 연구회에서의 테이블에 사용된 것보다 훨씬 간단하게 나타내었다. first name과 last name을 분리시키지 않고 단일 칼럼만 사용한다. 이렇게 하는 이유는 성적을 기록하는 쿼리 예제에서는 분리된 칼럼을 사용해서 더 좋아질 일이 없기 때문이다(속는 느낌이 드는가? 인정한다. 실제로는 다중 칼럼을 쓰도록 하라).

sex는 어떤 학생이 남학생인지 여학생인지의 여부를 나타낸다. 이것은 ENUM(열거형, enumeration) 칼럼인데, 이는 칼럼 지정에서 나열한 값들 중에서 단 하나만 취할 수 있다는 것을 뜻한다. 즉, 남자에 대해서는 'F'이고 여자에 대해서는 'M'만을 고를 수 있다. ENUM은 어떤 칼럼이 취할 수 있는 제한된 세트의 값들을 가지고 있을 때 유용하다. 대신에 CHAR(1)을 사용할 수도 있지만, ENUM은 해당 칼럼이 가질 수 있는 값을 좀더 명확하게 만들어준다. 어떤 값을 고를 수 있는지 잘 모르겠다면, DESCRIBE 문을 쓴다. ENUM 칼럼에 대해서, MySQL은 적법한 열거형 값의 목록을 표시해줄 것이다.

```
mysql> DESCRIBE student 'sex';
+-------+---------------+------+-----+---------+-------+
| Field | Type          | Null | Key | Default | Extra |
+-------+---------------+------+-----+---------+-------+
| sex   | enum('F','M') | NO   |     |         |       |
+-------+---------------+------+-----+---------+-------+
```

ENUM 칼럼 내의 값들은 단지 하나의 문자로 되어야 할 필요는 없다. 해당 타입 칼럼은 그 대신에 ENUM('female', 'male')과 같이 선언되어도 좋다.

student_id는 유일한 학생 ID 번호를 포함하는 정수 칼럼이다. 보통, 교무실과 같은 원천 소스로부터 학생에 대한 ID 번호를 얻어오게 될 것이다. 여기서는 간단하게 AUTO_INCREMENT를 사용하여 ID 번호를 부여하는데, 앞서 만든 member 테이블의 일부인 member_id 칼럼에서와 상당히 비슷한 방법으로 선언한다.

실제로, 학생 ID 번호를 자동으로 생성하지 않고 교무실에서 가져오게 되면, student_id 칼럼은 AUTO_INCREMENT 속성 없이 선언해야 함에 주의하기 바란다. 그러나 중복되거나 NULL ID 값이 있어서는 안 되므로 PRIMARY KEY 절은 그대로 둔다.

이제 CREATE TABLE 문의 끝에 있던 ENGINE 지정자가 무엇인지를 살펴보도록 한다. 이 지정자가 존재한다면, 이는 MySQL이 테이블을 생성하기 위해 사용되어야 하는 저장 엔진의 이름을 알려준다. 저장 엔진이라는 것은 어떠한 종류의 테이블을 다룰 수 있는 일종의 핸들러이다. MySQL은 여러 개의 저장 엔진을 가지고 있으며, 각각은 고유의 특성을 가지고 있으며 2.6.1절 "저장 엔진 특성"에서 설명하도록 한다.

만약 ENGINE 지정자를 설정하지 않는다면, MySQL은 MyISAM이라는 기본 엔진을 사용하게 된다.

"ISAM"은 인덱스가 있는 순차 접근 방법이며, MySQL의 고유 접근 방법에 기초한다. 초기에, 우리는 역사 연구회 테이블(president와 member)을 생성할 때 ENGINE 지정자를 사용하지 않았기 때문에 MyISAM 테이블이 될 것이다. (만약 서버를 다른 기본 엔진을 사용하도록 재설정하지 않은 경우에 해당한다.) 그 대신, 성적 기록부 프로젝트에서는 명백하게 InnoDB 엔진을 사용할 것을 나타내었다. InnoDB 엔진은 외부 키를 사용하여 "참조 무결성"이라는 동작을 수행할 수 있다. 이는 MySQL을 통해서 테이블간의 상호 연관성에 제약을 가한다는 것을 의미한다.

- Score 행들은 성적 이벤트 및 학생과 연관되어 있다. 만약 student ID와 grade event ID를 student, grade_event 테이블에서 알 수 없다면 score 테이블로 행의 엔트리들이 참조되는 것을 허용하지 않는다.
- 비슷하게, absence 레코드들이 학생과 연관관계를 가지고 있다. student id가 student 테이블에서 알 수 없는 경우라면 absence 테이블에서 레코드 엔트리들이 참조될 수 없다.

이러한 제약사항을 강제화하기 위해서 외부 키 관계를 설정해야 한다. 여기서 "외부"의 의미는 "다른 테이블"을 의미하며, "외부 키"는 다른 테이블에 있는 키 값과 매칭되어야 하는 키를 의미한다. 성적 기록 프로젝트 테이블의 나머지를 만들기 위해서는 이러한 개념들에 대해서 명확하게 정의되어야 한다.

grade_event 테이블

grade_event 테이블은 다음과 같이 만든다.

```
CREATE TABLE grade_event
(
  date      DATE NOT NULL,
  category  ENUM('T','Q') NOT NULL,
  event_id  INT UNSIGNED NOT NULL AUTO_INCREMENT,
  PRIMARY KEY (event_id)
) ENGINE = InnoDB;
```

이 테이블을 만들려면, mysql에서 CREATE TABLE 문을 타이핑해서 입력하든가 아니면 다음과 같은 명령을 실행한다.

```
% mysql sampdb < create_grade_event.sql
```

Date 칼럼은 표준 MySQL DATE값을 취하는데, '*CCYY-MM-DD*' 포맷이다(연도가 맨 먼저 나와야 한다).

category는 점수 유형을 나타낸다. student 테이블 안에 있는 sex와 같이, type도 열거형 칼럼이다. 허용되는 값은 'T' 와 'Q' 인데, 각각"시험(test)"과 "퀴즈(quiz)"를 의미한다.

event_id는 PRIMARY KEY로 선언된 AUTO_INCREMENT 칼럼인데, student 테이블 안에 있는 student_id와 유사하다. AUTO_INCREMENT를 사용하면, 유일한 event ID를 쉽게 만들어낼 수 있게 된다. student 테이블 내의 student_id 칼럼에서처럼, ID가 어떤 특정한 값이 되는 것보다 ID가 유일해야 하는 것이 더 중요하다.

모든 칼럼들은 NOT NULL로 선언되는데, 이들 중 어느 것도 누락되어서는 안 되기 때문이다.

score 테이블

score 테이블은 다음과 같이 작성할 수 있다.

```
CREATE TABLE score
(
  student_id  INT UNSIGNED NOT NULL,
  event_id    INT UNSIGNED NOT NULL,
  score       INT NOT NULL,
  PRIMARY KEY (event_id, student_id),
  INDEX (student_id),
  FOREIGN KEY (event_id) REFERENCES grade_event (event_id),
  FOREIGN KEY (student_id) REFERENCES student (student_id)
) ENGINE = InnoDB;
```

여기서도 다시 테이블을 정의할 때 FOREIGN KEY라는 새로운 것을 사용하였다.

위의 문장을 mysql 안에서 타이핑하여 입력하든가 아니면 다음과 같은 명령을 실행한다.

```
% mysql sampdb < create_score.sql
```

student_id와 event_id 칼럼은 INT(정수) 칼럼으로 지정해서 각 점수가 적용되는 학생들과 이벤트를 가리키게 된다. 이 칼럼들을 사용해서 student 테이블과 grade_event 테이블에 연결하면, 학생의 이름과 시험 또는 퀴즈를 치른 날짜를 알 수 있게 된다. 여기서 student_id와 event_id 칼럼에 대해서 주목할 만한 중요한 점이 두 가지 있다.

- PRIMARY KEY를 사용하여 두 칼럼의 조합을 만들 수도 있다. 이렇게 하면 주어진 퀴즈나 시험에 대해 특정 학생의 성적이 중복되지 않는다. student_id와 event_id 칼럼의 조합은 유일하다는 것을 명심해 두자. score 테이블에서는 두 개의 값이 유일하지 않다. 각 event_id 값에 대해서 복수의 점수 행이 있을 수 있고(각 학생당 하나씩), 각 student_id 값에 대해서 복수의 행이 있을 수 있다(시험과 퀴즈마다 하나씩).
- 각 ID 칼럼에서 FOREIGN KEY는 제약을 정의한다. FOREIGN KEY 절에서 REFERENCE는 score 칼럼이 참조하는 테이블과 칼럼을 나타낸다. event_id가 가지고 있는 제약은 칼럼의 각 값은 grade_event 테이블의 event_id 값과 매칭되어야 한다. 이와 비슷하게, score 테이블의 각 student_id 값도 student 테이블의 student_id 값과 매칭되어야 한다.

PRIMARY KEY는 score 행들이 중복해서 생성되지 못하도록 보장해 준다. FOREIGN KEY는 grade_event와 student 테이블에 존재하지 않는 위조된 ID 값을 가지는 행들을 가지지 않도록 해준다.

그런데, 왜 student_id의 인덱스가 필요한 것일까? 이에 대한 이유는 FOREIGN KEY 정의에서 어떤 칼럼들에 대해서는 인덱스가 존재하든지, 다중 칼럼 인덱스에 맨 처음으로 나열된 칼럼이 존재해야 하기 때문이다. event_id의 FOREIGN KEY에 대해서, 칼럼은 PRIMARY KEY에서 맨 처음으로 나열된다. student_id에서 FOREIGN KEY에 대해서, student_id는 맨 처음으로 나열될 수 없기 때문에 PRIMARY KEY는 사용될 수 없다. 그 대신, student_id에 대해서는 인덱스를 독립적으로 생성한다.

InnoDB는 실제로 외부 키 정의에서 다수의 칼럼들에 대한 인덱스를 생성하는데, 동일한 인덱스 정의

를 사용할 수는 없다(이에 대한 더 많은 내용은 2.14.1 절 "외부 키의 생성 및 사용"에서 살펴보자). 인덱스의 정의에 대해서는 이번에는 살펴보지 않는다.

absence 테이블

출석 기록을 위한 absence 테이블은 다음과 같이 작성할 수 있다.

```
CREATE TABLE absence
(
  student_id  INT UNSIGNED NOT NULL,
  date        DATE NOT NULL,
  PRIMARY KEY (student_id, date),
  PRIMARY KEY (student_id) REFERENCE student (student_id)
) ENGINE = InnoDB;
```

이 문장을 mysql에서 타이핑하여 입력하거나 다음과 같은 명령을 실행한다.

```
% mysql sampdb < create_absence.sql
```

student_id 칼럼과 date 칼럼은 둘 다 NOT NULL로 선언해서 누락되는 값이 없도록 한다. 이 두 칼럼의 조합을 주 키로 만들어서, 뜻밖에 중복된 레코드가 만들어지지 않도록 한다. 무엇보다도, 한 학생이 같은 날에 두 번 이상 결석으로 처리되는 것은 공정하지 못한 일이 아닌가!

absence 테이블도 외부 키 관계를 포함하고, 각 student_id 값이 student 테이블의 student_id 값과 매칭되는지를 보장한다.

성적 기록부 테이블에서 외부 키 관계를 가진다는 것은 데이터 입력 시 제약사항을 수행한다는 것을 의미한다. 적합한 성적 이벤트와 학생 ID 값을 포함하는 행들을 입력해야 한다. 외부 키 관계는 그 외의 또 다른 영향을 줄 수 있으며, 테이블의 생성과 삭제 순서에 제한을 가할 수 있다.

- score 테이블은 grade_event, student 테이블을 참조하기 때문에 score 테이블을 생성하기 이전에 먼저 두 테이블이 생성되어야 한다. 이와 비슷하게, absence는 student를 참조하므로, student는 absence를 생성하기 이전에 먼저 존재해야만 한다.
- 만약 테이블을 삭제하려면, 이와 반대로 되어야 한다. 먼저 score 테이블을 삭제하지 않으면 grade_event 테이블을 삭제할 수 없고, 먼저 score, absence를 삭제하지 않으면 student를 삭제할 수 없다.

NOTE

어떤 이유에 의해서 MySQL 서버가 InnoDB를 지원하지 않는다면, 대신에 MyISAM 테이블과 같은 성적 기록 프로젝트 테이블을 생성할 수 있다. 각 CREATE TABLE 문에서 MyISAM을 InnoDB 대신에 사용하거나 ENGINE 절을 생략하도록 한다. 그러나, 만약 MyISAM 테이블을 사용한다면, 외부 키의 동작을 나타내는 테이블에 대한 이 책의 예들이 제대로 동작하지 않을 것이다.

1.4.7 새로운 행들을 추가하기

이 시점에서, 데이터베이스와 테이블이 만들어졌다. 이제 이 테이블들에 몇 가지 행들을 넣을 필요가 있다. 하지만, 무엇인가를 테이블 안에 넣은 다음에 이 테이블에 무엇이 들어 있는지 점검하는 방법을 사용하는 것이 유용하므로, 비록 1.4.9절 "정보 검색하기"에서 다루기 이전까지 탐색하는 방법이 자세히 다루어지지 않더라도, 최소한 다음과 같은 문장을 사용해서 테이블의 *tbl_name*의 내용을 탐색하여 가져올 수 있다는 것 만큼은 알아두기 바란다.

```
SELECT * FROM tbl_name;
```

예를 들면, 다음과 같다.

```
mysql> SELECT * FROM student;
Empty set (0.00 sec)
```

지금, mysql은 테이블에 아무것도 없는 빈 상태임을 나타내지만, 이 절에 있는 예제들을 따라해보고 난 다음에는 다른 결과를 보게 될 것이다.

어떤 데이터베이스에 데이터를 추가하는 방법에는 여러 가지가 있다. INSERT 문장들을 보냄으로써 수동적으로 테이블에 행들을 삽입할 수도 있고, 파일에서 내용을 읽어서 행들을 추가할 수도 있는데, 파일 안에 mysql로 보낼 수 있도록 INSERT 문장들을 미리 작성해 놓은 형태일 수도 있고 LOAD DATA 문을 이용해서 로드하거나 mysqlimport 유틸리티로 로드하게 되는 가공되지 않은 데이터 값이 들어 있는 형태일 수도 있다.

이 절에서는 테이블 안으로 행들을 삽입하는 각 방법들의 예를 보이도록 하겠다. 여기서 여러분이 반드시 해야만 하는 것은 이것들을 다루어봄으로써 스스로 이 방법들에 익숙해지고 이 작업이 어떤 식으로 이루어지는지 잘 파악하는 것이다. 이 방법들을 실행해보고 난 다음에는 1.4.8절 "sampdb 데이터베이스의 리셋"에 가서 그곳에 있는 명령들을 실행해서 테이블을 삭제하고, 다시 만들고, 알고 있는 일련의 데이터들을 테이블에 로드해 본다. 이렇게 해보는 것으로, 다음에 오는 절들을 작성하는 동안에 필자가 작업하는 것과 동일한 행들이 테이블들이 포함하는 것을 확인할 수 있게 되고, 이 절들에서 나타나는 것과 동일한 결과를 얻게 될 것이다. (행들을 삽입하는 방법을 이미 알고 있고 단지 테이블들을 설정시키기만 원하는 경우라면, 그 절로 직접 건너뛰기를 바랄 것이다.)

INSERT를 사용하여 행들을 추가하기

INSERT를 사용해서 행들을 추가하는 작업을 시작해보자. INSERT 문은 어떤 테이블을 지정해서 그 안으로 데이터의 행을 삽입하고 그 행 안으로 값을 넣을 때 사용되는 SQL 문장이다. INSERT 문장에는 여러 형식이 있다.

- **다음과 같이 모든 칼럼들에 대한 값들을 지정할 수 있다.** 문법은 다음과 같다.

  ```
  INSERT INTO tbl_name VALUES(value1,value2,...);
  ```

  예제:

  ```
  mysql> INSERT INTO student VALUES('Kyle','M',NULL);
  ```

```
mysql> INSERT INTO event VALUES('2008-09-03','Q',NULL);
```

이 문법을 사용할 때는 VALUES 목록에 테이블 안의 각 칼럼에 대한 값들이 들어가야 하는데, 이때 각 값들은 각 칼럼이 테이블에 저장된 순서대로 나열되어야 한다(보통은 해당 테이블을 만들 때 사용한 CREATE TABLE 문장에서 사용한 칼럼들의 순서가 된다). 칼럼의 순서가 어떻게 되는지 잘 모르겠다면, DESCRIBE *tbl_name* 문장을 보내서 알아낼 수 있다.

MySQL에서, 문자열과 날짜 값들은 단일 인용 부호 또는 이중 인용 부호를 써서 인용문으로 만들 수 있다. student와 event 테이블 안의 AUTO_INCREMENT 칼럼들에 대해서는 NULL 값을 사용한다. AUTO_INCREMENT 칼럼에 "빠진 값"을 삽입하면, MySQL이 해당 칼럼에 대한 다음 순서의 숫자를 자동으로 만들어 넣게 된다.

MySQL은 단일 INSERT 문장에 값 목록을 복수로 지정해서 한 테이블에 여러 행들을 한번에 삽입할 수 있도록 해준다.

```
INSERT INTO tbl_name VALUES(...),(...),... ;
```

예제:

```
mysql> INSERT INTO student VALUES('Avery','F',NULL),('Nathan','M',NULL);
```

이러한 형식을 사용하면 여러 INSERT 문장을 사용하지 않게 되므로 타이핑 횟수를 줄일 수 있고, 서버가 좀 더 효과적으로 실행될 수 있다. 괄호는 각 행에 대한 칼럼 셋을 감싸고 있다. 따라서 다음의 문장은 괄호 안에 여러 값들의 개수가 맞지 않기 때문에 잘못되어 있다.

```
mysql> INSERT INTO student VALUES('Avery','F',NULL,'Nathan','M',NULL);
ERROR 1136 (21S01): Column count doesn't match value count at row 1
```

- **값을 할당할 칼럼만 따로 지명한 다음에 값들을 나열할 수 있다.** 이 형식은 초기에 단지 몇 개의 칼럼만 설정하면 되는 레코드를 생성할 때 유용하게 쓰인다.

```
INSERT INTO tbl_name (col_name1,col_name2,...) VALUES(value1,value2,...);
```

예제:

```
mysql> INSERT INTO member (last_name,first_name) VALUES('Stein','Waldo');
```

MySQL는 이러한 형식의 INSERT 문장에도 값들의 목록을 복수로 사용할 수 있다. 즉, 다음과 같다.

```
mysql> INSERT INTO student (name,sex) VALUES('Abby','F'),('Joseph','M');
```

칼럼 목록 안에 지명되지 않은 그밖의 칼럼에는 디폴트 값이 대입된다. 예를 들면, 이전의 문장들에서는 member_id 칼럼이나 student_id 칼럼에 대해 아무런 값도 지정해 주지 않았으므로, MySQL은 디폴트 값인 NULL을 이들 칼럼에 대입하게 된다(member_id와 student_id는 둘 다 AUTO_INCREMENT 칼럼이므로, 명시적으로 NULL을 대입했을 때와 마찬가지로, 실제적으로는 각 경우에 다음 순서의 번호를 만들어서 대입하게 되는 것이다).

- **MySQL는 칼럼과 값을 묶어서 지정할 수 있다.** 이 문법은 value( ) 리스트보다는 *col_name=value* 할당을 포함하는 SET 절을 사용한다.

```
INSERT INTO tbl_name SET col_name1=value1, col_name2=value2, ... ;
```

예제:

```
mysql> INSERT INTO member SET last_name='Stein',first_name='Waldo'
```

SET 절에 지명되지 않은 칼럼에는 디폴트 값이 대입된다. 이러한 형식의 INSERT 문을 사용해서 복수의 행들을 삽입할 수는 없다.

지금 INSERT가 어떻게 동작하는지 알아보도록 하며, 이를 통해서 외부 키 관계가 score, absence 테이블에서 잘못된 행들의 입력을 어떻게 방지하는지 살펴보도록 한다. grade_event 또는 student 테이블에 존재하지 않는 ID 값들을 포함하고 있는 행들을 추가하도록 해보자.

```
mysql> INSERT INTO score (event_id,student_id,score) VALUES(9999,9999,0);
ERROR 1452 (23000): Cannot add or update a child row: a foreign key
constraint fails (`sampdb`.`score`, CONSTRAINT `score_ibfk_1` FOREIGN
KEY (`event_id`) REFERENCES `grade_event` (`event_id`))
mysql> INSERT INTO absence SET student_id=9999, date='2008-09-16';
ERROR 1452 (23000): Cannot add or update a child row: a foreign key
constraint fails (`sampdb`.`absence`, CONSTRAINT `absence_ibfk_1`
FOREIGN KEY (`student_id`) REFERENCES `student` (`student_id`))
```

제약사항들이 제대로 동작하고 있음을 에러 메시지를 통해서 알 수 있다.

파일로부터 새로운 행들을 추가하기

테이블 안으로 행들을 로딩하는 또 다른 방법으로는 파일에서 직접 읽어오는 것이 있다. 예를 들면, sampdb 배포판은 presidnet 테이블에 새로운 행들을 추가하는 INSERT 문장들을 포함하는 insert_president.sql을 가진다. 파일이 위치한 동일한 디렉터리에서 다음과 같은 문장을 수행할 수 있다.

```
% mysql sampdb < insert_president.sql
```

mysql을 이미 실행시키고 있다면, SOURCE 명령을 사용해서 해당 파일을 읽어올 수 있다.

```
mysql> SOURCE insert_president.sql;
```

파일 안에 저장된 행의 형태가 INSERT 문장으로 만들어진 것이 아니고 가공되지 않은 데이터 값의 형태라면, LOAD DATA 문장이나 mysqlimport 유틸리티를 사용하여 로드할 수 있다.

LOAD DATA 문은 파일에서 데이터를 통째로 로드해오도록 되어 있다. mysql 안에서 사용한다.

```
mysql> LOAD DATA LOCAL INFILE 'member.txt' INTO TABLE member;
```

member.txt라고 하는 데이터 파일이 클라이언트 호스트 측의 현재 디렉터리에 있다고 가정하면, 이 문장은 그 파일을 읽어서 내용을 서버로 전송시켜 member 테이블로 로드되게 한다. (member.txt 파일은 sampdb 배포판에서 찾을 수 있다.)

기본적으로, LOAD DATA 문은 칼럼 값들이 탭 문자로 구분되고 라인의 마지막이 개행 문자로 끝난다는 것을 가정한다("라인피드" 문자로도 알려져 있는 문자이다). 그리고 각 값들은 칼럼이 테이블에 저장된 순서대로 표현되어야 한다는 것을 가정한다. 다른 포맷으로 파일을 읽거나 다른 칼럼 순서를 지정할

수도 있다. 자세한 사항은 부록 E에 있는 "SQL 문법 레퍼런스"에서 LOAD DATA에 대한 항목을 보기 바란다.

LOAD DATA 문의 키워드 LOCAL은 클라이언트 프로그램이 데이터 파일을 읽을 수 있도록 한다. LOCAL 키워드가 없으면, 서버 호스트에 위치한 파일을 읽게 되고, 따라서 이 경우에는 대부분의 MySQL 사용자가 가지고 있지 않은 서버의 접근 권한이 필요하게 된다. 서버가 찾을 수 있는 파일의 전체 경로이름을 정해야만 한다.

만약 LOAD DATA LOCAL 사용 시 다음과 같은 에러를 얻게 되면, LOCAL은 기본적으로 사용하지 못하게 된다.

```
ERROR 1148 (42000): The used command is not allowed with this MySQL version
```

그러면, --local-infile 옵션을 가지고 mysql을 다시 한 번 호출해본다. 예를 들면, 다음과 같다.

```
% mysql --local-infile sampdb
mysql> LOAD DATA LOCAL INFILE 'member.txt' INTO TABLE member;
```

이것도 제대로 동작하지 않는다면, 서버에 LOCAL이 허용되도록 지시해야 할 필요가 있다. 이에 대한 정보는 12장을 보기 바란다.

데이터 파일을 불러오기 위한 다른 방법은 mysqlimport 클라이언트 프로그램을 사용하는 것이다. 명령어 프롬프트에서 mysqlimport를 호출하면, 이 유틸리티는 LOAD DATA 문을 생성하게 된다.

```
% mysqlimport --local sampdb member.txt
```

mysql 프로그램에서와 마찬가지로, 연결 인자들을 지정할 필요가 있는 경우에는 이 인자들을 데이터베이스 이름 앞에 오도록 해서 써주면 된다.

방금 나타난 명령에 대해서, mysqlimport는 LOAD DATA 문장을 생성해서 member.txt 파일을 member 테이블 안으로 로드한다. mysqlimport는 인자로 정해진 파일 이름에서 첫 번째로 나오는 점('.')까지의 문자열을 테이블 이름으로 삼는다. 예를 들면, mysqlimport는 member.txt와 president.txt를 각각 member 테이블과 president 테이블 안으로 로드된다. 이것은 파일 이름을 신중하게 선택하지 않으면 mysqlimport가 제대로 된 파일 이름을 사용하지 않게 됨을 의미한다. 예를 들어, member1.txt와 member2.txt가 member 테이블로 로드되는 것을 생각하고 있을 수도 있겠지만, mysqlimport는 이것이 각각 member1 테이블과 member2 테이블로 로드되는 것으로 간주하게 되는 것이다. 만약 member 테이블로 모든 파일을 로드하길 원한다면, member.1.txt와 member.2.txt 또는 member.txt1과 member.txt2와 같은 이름을 사용할 수는 있다.

1.4.8 sampdb 데이터베이스의 리셋

방금 설명한 레코드 추가 방법을 해보았으므로, 다음 절에서 가정하는 것과 동일한 내용이 되도록 sampdb 데이터베이스 테이블을 다시 만들어서 로드해야 한다. sampdb 배포 파일이 있는 디렉터리에서 mysql 프로그램을 사용할 때, 다음의 문장들을 실행하도록 한다.

```
% mysql sampdb
```

```
mysql> source create_member.sql;
mysql> source create_president.sql;
mysql> source insert_member.sql;
mysql> source insert_president.sql;
mysql> DROP TABLE IF EXISTS absence, score, grade_event, student;
mysql> source create_student.sql;
mysql> source create_grade_event.sql;
mysql> source create_score.sql;
mysql> source create_absence.sql;
mysql> source insert_student.sql;
mysql> source insert_grade_event.sql;
mysql> source insert_score.sql;
mysql> source insert_absence.sql;
```

이 명령들을 하나씩 타이핑하는 것이 귀찮다면(재미있는 일은 아니다), UNIX에서는 다음과 같이 해본다.

```
% sh init_all_tables.sh sampdb
```

또한 Windows에서는 다음과 같이 한다.

```
C:\> init_all_tables.bat sampdb
```

연결 인자를 지정해야 하는 경우에는 커맨드 라인상에서 데이터베이스 이름 앞쪽에 인자들을 나열한다.

1.4.9 정보 검색하기

이제 테이블이 만들어져서 데이터가 로드되었기 때문에, 이 데이터로 할 수 있는 일이 무엇인지 알아보기로 하자. SELECT 문은 테이블에서 정보를 검색해서 표시할 수 있도록 해주는데, 취향대로 일반적인 방법이나 특정한 방법을 사용할 수 있다. 다음과 같이 테이블의 전체 내용을 표시할 수 있다.

```
SELECT * FROM president;
```

또한 다음과 같이 하나의 행에서 하나의 칼럼만 검색할 수도 있다.

```
SELECT birth FROM president WHERE last_name = 'Eisenhower';
```

SELECT 문은 여러 개의 절(또는 부분)이 있는데, 이들을 필요한 만큼 연결해서 관심이 있는 정보를 검색하게 된다. 이들 각각의 절들은 단순할 수도 있고 복잡할 수도 있어서, SELECT 문장 전체가 단순하거나 복잡하게 될 수 있다. 하지만, 이 책 안에서는 이해하는 데 한 시간 이상 걸리는 한 페이지가 넘는 길이의 쿼리는 없으므로 안심해도 좋다(나 자신도 읽고 있던 문서에서 엄청난 길이의 쿼리를 보게 되면 그냥 그것을 넘겨 버리고 말텐데, 여러분들도 마찬가지일 것이라고 생각한다). SELECT 문의 일반적인 형식은 다음과 같다.

```
SELECT what to retrieve
FROM table or tables
WHERE conditions that data must satisfy;
```

SELECT 문을 작성하려면, 검색하고자 하는 것을 지정한 다음에 몇 가지 선택적인 절을 지정한다. 위에 나타난 절들(FROM과 WHERE)은 아주 일반적인 것들이고, 이와 별도로 GROUP BY, ORDER BY, 그리

고 LIMIT와 같은 또 다른 절들도 지정할 수 있다. SQL은 형식에 구애받지 않는 언어라는 것을 기억하기 바란다. 따라서 자신이 직접 SELECT 쿼리를 작성할 때에는 이 책의 여러 곳에서 사용하는 선 구분 문자를 넣지 않아도 상관없다.

보통 FROM절을 사용하게 되지만, 테이블 이름을 지정하지 않는 경우에는 이것을 사용할 필요가 없다. 예를 들면, 다음과 같은 쿼리는 단순히 어떤 표현식의 값들을 표시한다. 이러한 쿼리문은 어떠한 테이블도 참조하지 않고 계산될 수 있기 때문에, FROM절이 필요 없는 것이다.

```
mysql> SELECT 2+2, 'Hello, world', VERSION();
+-----+--------------+----------------+
| 2+2 | Hello, world | VERSION()      |
+-----+--------------+----------------+
| 4   | Hello, world | 5.0.60-log     |
+-----+--------------+----------------+
```

FROM절을 사용하여 검색할 데이터가 있는 테이블을 지정하면, 또한 보고자 하는 칼럼들도 지정하게 될 것이다. 가장 "일반적인" 형태의 SELECT는 칼럼 지정자로 *를 사용하는데, 이것은 "모든 칼럼들"이란 의미를 짧게 쓴 것이다. 다음과 같은 쿼리는 student 테이블에 있는 모든 칼럼들을 검색해서 이들을 표시한다.

```
mysql> SELECT * FROM student;
+-----------+-----+------------+
| name      | sex | student_id |
+-----------+-----+------------+
| Megan     | F   |          1 |
| Joseph    | M   |          2 |
| Kyle      | M   |          3 |
| Katie     | F   |          4 |
...
```

MySQL이 테이블에 저장해 둔 순서대로 칼럼들이 표시된다. DESCRIBE student 문장을 보낼 때 나열한 칼럼들의 순서와 동일한 순서가 된다(예문의 마지막에 있는 "..."은 쿼리 결과가 더 있지만 생략한다는 뜻이다).

보고자 하는 칼럼만 명시적으로 지명할 수도 있다. 학생의 이름만 검색하고자 한다면, 다음과 같이 한다.

```
mysql> SELECT name FROM student;
+-----------+
| name      |
+-----------+
| Megan     |
| Joseph    |
| Kyle      |
| Katie     |
...
```

하나 이상의 칼럼을 지명할 때는 콤마로 칼럼 이름들을 구분한다. 다음의 문장은 SELECT * FROM student와 동일하지만, 각 칼럼을 명시적으로 지명한 문장이다.

```
mysql> SELECT name, sex, student_id FROM student;
+-----------+-----+------------+
| name      | sex | student_id |
+-----------+-----+------------+
| Megan     | F   |          1 |
| Joseph    | M   |          2 |
| Kyle      | M   |          3 |
| Katie     | F   |          4 |
...
```

칼럼은 어떤 순서로도 지명할 수 있다.

```
SELECT name, student_id FROM student;
SELECT student_id, name FROM student;
```

심지어, 원하기만 한다면 하나의 칼럼을 여러 번 지명할 수도 있는데, 일반적으로 이렇게 하지는 않는다.

MySQL은 한 번에 하나 이상의 테이블에 있는 칼럼들을 검색할 수 있도록 해준다. 이를 테이블의 "join"이라고 한다. 이에 대해서는 1.4.9절의 하위절 "복수의 테이블에서 정보 검색하기"에서 자세히 알아보게 된다.

MySQL에서는 칼럼 이름에서 대소문자 구별을 하지 않으므로, 다음의 쿼리들은 모두 같은 것이 된다.

```
SELECT name, student_id FROM student;
SELECT NAME, STUDENT_ID FROM student;
SELECT nAmE, sTuDeNt_Id FROM student;
```

이와는 달리, 데이터베이스 이름과 테이블 이름은 대소문자 구별을 할 수도 있다. 이는 서버 호스트상의 파일 시스템에 따라 달라진다. Windows의 파일 이름은 대소문자 구별을 하지 않으므로, Windows 상에서 실행되는 서버는 데이터베이스 이름과 테이블 이름에서 대소문자 구별을 하지 않는다. UNIX 상에서 실행되는 서버는 UNIX의 파일 시스템이 대소문자 구별을 하므로 데이터베이스 이름과 테이블 이름에서 대소문자 구별을 하게 된다. HFS+와 UFS 파일 시스템을 모두 지원하는 Mac OS X에서는 예외가 있는데, HFS+ 파일 시스템 상의 파일들은 대소문자 구별을 하지 않지만, UFS 파일 시스템상에 있는 파일 이름들은 대소문자 구별을 한다.

MySQL이 데이터베이스 이름과 테이블 이름들에서 대소문자 구별을 하지 않도록 하고 싶다면, 11.2.5절 "운영체제가 데이터베이스와 테이블의 이름에 가하는 제약"을 보기 바란다.

검색 범위 지정하기

SELECT 문으로 검색되는 행들을 제한하려면, WHERE 절을 사용해서 검색할 행들에 대한 범위를 지정한다. 여러 조건을 만족시키는 칼럼 값을 탐색해서 행들을 검색할 수 있고, 특정 타입의 값을 검색해올 수도 있다. 예를 들면, 다음과 같이 숫자에 대해 검색할 수 있다.

```
mysql> SELECT * FROM score WHERE score > 95;
+------------+----------+-------+
| student_id | event_id | score |
+------------+----------+-------+
```

```
|          5 |         3 |    97 |
|         18 |         3 |    96 |
|          1 |         6 |   100 |
|          5 |         6 |    97 |
|         11 |         6 |    98 |
|         16 |         6 |    98 |
+------------+-----------+-------+
```

또는 문자열 값을 찾아볼 수 있다. 문자열 비교는 보통 대소문자를 비교하지 않는다.

```
mysql> SELECT last_name, first_name FROM president
    -> WHERE last_name='ROOSEVELT';
+-----------+-------------+
| last_name | first_name  |
+-----------+-------------+
| Roosevelt | Theodore    |
| Roosevelt | Franklin D. |
+-----------+-------------+

mysql> SELECT last_name, first_name FROM president
    -> WHERE last_name='roosevelt';
+-----------+-------------+
| last_name | first_name  |
+-----------+-------------+
| Roosevelt | Theodore    |
| Roosevelt | Franklin D. |
+-----------+-------------+
```

또한 날짜에 대해서도 검색할 수 있다.

```
mysql> SELECT last_name, first_name, birth FROM president
    -> WHERE birth < '1750-1-1';
+------------+------------+------------+
| last_name  | first_name | birth      |
+------------+------------+------------+
| Washington | George     | 1732-02-22 |
| Adams      | John       | 1735-10-30 |
| Jefferson  | Thomas     | 1743-04-13 |
+------------+------------+------------+
```

여러 값의 조건을 결합해서 검색하는 것도 가능하다.

```
mysql> SELECT last_name, first_name, birth, state FROM president
    -> WHERE birth < '1750-1-1' AND (state='VA' OR state='MA');
+------------+------------+------------+-------+
| last_name  | first_name | birth      | state |
+------------+------------+------------+-------+
| Washington | George     | 1732-02-22 | VA    |
| Adams      | John       | 1735-10-30 | MA    |
| Jefferson  | Thomas     | 1743-04-13 | VA    |
+------------+------------+------------+-------+
```

WHERE 절에 있는 표현식에는 산술 연산자([표 1.1]), 비교 연산자([표 1.2]), 그리고 논리 연산자([표 1.3]) 등을 사용할 수 있다. 표현식의 일부분에 괄호를 사용해서 묶을 수도 있다. 상수, 테이블 칼럼, 그리고 함수 호출을 이용하여 연산이 수행된다. 이 튜토리얼에서는 쿼리 안에서 MySQL의 함수 몇 가지를 이따금씩 사용하겠지만, 여기서 보는 것보다 훨씬 더 많은 함수들이 있다. 완전한 목록을 보려면 부록 C "연산자와 함수 레퍼런스"를 참조하기 바란다.

논리 연산자가 필요한 쿼리를 작성할 때, 통상적으로 "~과(and)"라고 말하는 것을 염두에 두고 논리적인 AND 연산자의 의미를 혼동하지 않도록 해야 한다. "버지니아에서 태어난 대통령들과 메사추세츠에서 태어난 대통령들"을 찾으려 한다고 해보자. 이와 같은 명제에서는 "~과(and)"를 사용한 구문으로 생각해서 다음과 같은 쿼리를 작성해야 하는 것처럼 생각되기도 한다. 즉, 다음과 같다.

```
mysql> SELECT last_name, first_name, state FROM president
    -> WHERE state='VA' AND state='MA';
Empty set (0.36 sec)
```

[표 1.1] 산술 연산자

연산자	의미
+	더하기
−	빼기
*	곱하기
/	나누기
DIV	정수형 나누기
%	Modulo(나머지)

[표 1.2] 비교 연산자

연산자	의미
<	오른쪽보다 작은
<=	오른쪽보다 작거나 같은
=	오른쪽과 같은
<=>	오른쪽과 같은 (NULL에도 쓸 수 있음)
<> 또는 !=	오른쪽과 같지 않은
>=	오른쪽보다 크거나 같은
>	오른쪽보다 큰

[표 1.3] 논리 연산자

연산자	의미
AND	논리 AND
OR	논리 OR
XOR	배타적 논리 OR
NOT	논리 부정

하지만, 이 쿼리는 제대로 동작하지 않고 빈 결과만 얻을 것이다. 왜 제대로 안 될까? 위 쿼리의 실제 뜻은 "버지니아와 메사추세츠 양쪽 모두에서 태어난 대통령들을 선택하라"라는 것이기 때문인데, 따라서 아무 의미도 없는 쿼리가 되어버린다. 일상 생활에서 말할 때는 "~과 (and)"를 사용하여 쿼리를 표현하기도 하지만, SQL에서 이 두 조건은 OR로 연결해야 한다.

```
mysql> SELECT last_name, first_name, state FROM president
    -> WHERE state='VA' OR state='MA';
+------------+-------------+-------+
| last_name  | first_name  | state |
+------------+-------------+-------+
| Washington | George      | VA    |
| Adams      | John        | MA    |
| Jefferson  | Thomas      | VA    |
| Madison    | James       | VA    |
| Monroe     | James       | VA    |
| Adams      | John Quincy | MA    |
| Harrison   | William H.  | VA    |
| Tyler      | John        | VA    |
| Taylor     | Zachary     | VA    |
| Wilson     | Woodrow     | VA    |
| Kennedy    | John F.     | MA    |
| Bush       | George H.W. | MA    |
+------------+-------------+-------+
```

이와 같이 자연어와 SQL 간의 차이점은 자신의 쿼리를 작성할 때 뿐만 아니라, 다른 사람들이 의뢰한 쿼리를 작성해 줄 때에도 유념해야 한다. 동일한 논리 연산자들을 사용하여 의뢰인의 설명을 SQL로 그대로 베끼지 않고, 의뢰인이 검색하고자 하는 것에 대해 설명하는 것을 주의 깊게 경청하는 것이 가장 좋다. 방금 설명한 예에 대해서도, 그 쿼리에 대한 것과 동등한 일상 언어로 표현하게 되면, "버지니아 또는 메사추세츠 중 한 곳에서 태어난 대통령들을 검색하라" 정도가 될 것이다.

이와 같이 조건을 찾으려고 하는 쿼리를 작성할 때 IN( ) 연산자를 사용하게 되면 더욱 더 쉽게 찾을 수 있다. 이전의 쿼리를 IN( )을 사용해서 다시 작성하면 다음과 같이 된다.

```
SELECT last_name, first_name, state FROM president
WHERE state IN('VA','MA');
```

IN( )은 특히 어느 한 칼럼을 많은 값과 비교하려고 할 때 편리하다.

NULL 값

NULL 값은 특별한 값이다. 이것은"값이 없음" 또는 "모르는 값"을 뜻하기 때문에, 알려진 두 개의 값을 서로 평가하는 방식과 같은 방식으로 알려진 값들에 대해 NULL 값을 평가할 수는 없다. 일반적인 산술 비교 연산자에 NULL을 사용하는 경우에는 결과가 정의되지 않는다.

```
mysql> SELECT NULL < 0, NULL = 0, NULL <> 0, NULL > 0;
+----------+----------+-----------+----------+
| NULL < 0 | NULL = 0 | NULL <> 0 | NULL > 0 |
+----------+----------+-----------+----------+
|     NULL |     NULL |      NULL |     NULL |
+----------+----------+-----------+----------+
```

사실, 두 개의 알 수 없는 값들을 비교하는 것의 결과는 정의될 수 없기 때문에, NULL끼리는 비교조차 할 수 없는 것이다.

```
mysql> SELECT NULL = NULL, NULL <> NULL;
+-------------+--------------+
| NULL = NULL | NULL <> NULL |
+-------------+--------------+
|        NULL |         NULL |
+-------------+--------------+
```

NULL 값들에 대한 검색을 수행하려면 특별한 문법을 사용해야만 한다. 같음 또는 다름에 대한 검사를 하는 경우, =, <>, 또는 != 를 사용하지 않고 그 대신에 IS NULL 또는 IS NOT NULL을 사용한다. 예를 들면, president 테이블에서 사망 날짜를 NULL로 나타낸다고 정했기 때문에 다음의 쿼리를 사용해서 아직 생존해 있는 대통령들을 검색할 수 있다.

```
mysql> SELECT last_name, first_name FROM president WHERE death IS NULL;
+-----------+-------------+
| last_name | first_name  |
+-----------+-------------+
| Carter    | James E.    |
| Bush      | George H.W. |
| Clinton   | William J.  |
| Bush      | George W.   |
+-----------+-------------+
```

NULL이 아닌 값을 찾기 위해서 IS NOT NULL을 사용해야 한다. 이 쿼리는 접미어 부분이 있는 이름을 찾는다.

```
mysql> SELECT last_name, first_name, suffix
    -> FROM president WHERE suffix IS NOT NULL;
+-----------+------------+--------+
| last_name | first_name | suffix |
+-----------+------------+--------+
```

```
| Carter     | James E.    | Jr.     |
+-----------+------------+--------+
```

MySQL의 특별한 비교 연산자인 <=>는 NULL 대 NULL 비교에도 true가 된다. 앞서의 두 개의 쿼리는 이 연산자를 사용해서 다음과 같이 다시 작성할 수 있다.

```
SELECT last_name, first_name FROM president WHERE death <=> NULL;

SELECT last_name, first_name, suffix
FROM president WHERE NOT (suffix <=> NULL);
```

쿼리 결과를 정렬하기

테이블을 만들어서 여기에 몇 개의 행을 로드하고 난 다음에 SELECT * FROM *tbl_name* 문을 제출하는 경우, 이 레코드들은 삽입시켰을 때와 같은 순서대로 검색되는 경향이 있다는 것을 MySQL 사용자들은 쉽사리 알게 된다. 이러한 것은 어떤 직관적인 감을 주는데, 결과적으로 삽입 순서대로 행이 검출된다는 것이 믿을만한 원리라는 가정을 자연스럽게 하게 되는 것이다. 그러나 이것은 틀린 가정이 된다. 예를 들면, 초기에 테이블을 로딩한 후에 행을 삭제하고 삽입하면, 그러한 작업은 서버가 테이블의 행을 리턴하는 순서를 변경시키기도 한다. (행들을 삭제하면 테이블 안에 사용되지 않는 공간이 "구멍"처럼 생기는데, MySQL은 나중에 새로운 행을 삽입할 때 이곳을 채우게 된다.)

행 검색 순서에 대해서 기억해야 할 것은 다음과 같다. 즉, 특별히 따로 지정하지 않으면, 서버가 리턴하는 행들의 순서에 대해서는 **아무런 보장을 할 수 없다**. 그렇게 하려면, ORDER BY 절을 추가하여 원하는 정렬 순서를 정의하는 문장을 만들면 된다. 다음의 쿼리는 대통령의 이름을 리턴하는데 last name을 알파벳 순서로 정렬하게 된다.

```
mysql> SELECT last_name, first_name FROM president
    -> ORDER BY last_name;
+------------+---------------+
| last_name  | first_name    |
+------------+---------------+
| Adams      | John          |
| Adams      | John Quincy   |
| Arthur     | Chester A.    |
| Buchanan   | James         |
...
```

오름차순은 ORDER BY 절에서 기본 정렬 방법이 된다. ASC 또는 DESC 키워드를 ORDER BY 절 안의 칼럼 이름 뒤에 써 주면 칼럼을 오름차순 또는 내림차순으로 정렬하도록 지정할 수 있다. 예를 들어, 대통령들의 이름을 역순(내림차순)으로 정렬하려면 다음과 같이 DESC를 사용한다.

```
mysql> SELECT last_name, first_name FROM president
    -> ORDER BY last_name DESC;
+------------+---------------+
| last_name  | first_name    |
+------------+---------------+
| Wilson     | Woodrow       |
```

```
| Washington | George        |
| Van Buren  | Martin        |
| Tyler      | John          |
...
```

쿼리 출력은 복수의 칼럼들에 대해 정렬될 수 있는데, 각 칼럼은 다른 칼럼에 독립적으로 오름차순 또는 내림차순으로 정렬될 수 있다. 다음의 쿼리는 president 테이블에서 행들을 검색하는데, 태어난 주(state)는 역순으로, 각 주(state) 안에서는 last name을 오름차순으로 정렬시킨다.

```
mysql> SELECT last_name, first_name, state FROM president
    -> ORDER BY state DESC, last_name ASC;
+------------+---------------+-------+
| last_name  | first_name    | state |
+------------+---------------+-------+
| Arthur     | Chester A.    | VT    |
| Coolidge   | Calvin        | VT    |
| Harrison   | William H.    | VA    |
| Jefferson  | Thomas        | VA    |
| Madison    | James         | VA    |
| Monroe     | James         | VA    |
| Taylor     | Zachary       | VA    |
| Tyler      | John          | VA    |
| Washington | George        | VA    |
| Wilson     | Woodrow       | VA    |
| Eisenhower | Dwight D.     | TX    |
| Johnson    | Lyndon B.     | TX    |
...
```

칼럼을 정렬할 때 NULL 값이 오름차순의 맨 처음 또는 내림차순의 맨 마지막에서 나타날 수도 있다. NULL 값이 주어진 정렬 순서의 마지막 쪽에 항상 오도록 만들고 싶다면, NULL 값과 NULL이 아닌 값을 구별해주는 여분의 정렬 칼럼을 추가한다. 예를 들면, 사망 날짜를 기준으로 대통령들을 정렬시키지만 현재 생존해 있는 대통령들(사망날짜 값이 NULL이다)은 정렬 순서에서 마지막으로 위치하게 될 것이다. 만약 처음으로 위치하도록 하고 싶다면, 다음과 같은 쿼리를 사용한다.

```
mysql> SELECT last_name, first_name, death FROM president
    -> ORDER BY IF(death IS NULL,0,1), death DESC;
+------------+---------------+------------+
| last_name  | first_name    | death      |
+------------+---------------+------------+
| Clinton    | William J.    | NULL       |
| Bush       | George H.W.   | NULL       |
| Carter     | James E.      | NULL       |
| Bush       | George W.     | NULL       |
| Ford       | Gerald R.     | 2006-12-26 |
| Reagan     | Ronald W.     | 2004-06-05 |
| Nixon      | Richard M.    | 1994-04-22 |
| Johnson    | Lyndon B.     | 1973-01-22 |
```

```
...
| Jefferson  | Thomas         | 1826-07-04 |
| Adams      | John           | 1826-07-04 |
| Washington | George         | 1799-12-14 |
+------------+----------------+------------+
```

IF() 함수는 첫 번째 인수에 의한 표현식을 평가해서 표현식이 참 또는 거짓이냐에 의해서 두 번째 인수 또는 세 번째 인수의 값을 리턴한다. 쿼리에서 나타낸 바와 같이, IF()는 NULL 값에 대해서 0으로 평가하고 NULL이 아닌 값에 대해서는 1로 평가한다. 이는 모든 NULL 값들을 NULL이 아닌 값들의 앞으로 위치시킨다.

쿼리 결과를 제한하기

쿼리가 많은 행들을 리턴하지만, 이 중에서 단지 몇 개의 행만 보고자 한다면 LIMIT 절을 사용하는데, 특히 ORDER BY와 결합해서 사용하면 좋다. MySQL은 어떤 쿼리의 리턴될 결과 중에서 처음 *n*개의 행만 출력하도록 제한하는 기능이 있다. 다음은 생년월일이 빠른 순서대로 5명의 대통령을 검색하는 쿼리이다.

```
mysql> SELECT last_name, first_name, birth FROM president
    -> ORDER BY birth LIMIT 5;
+------------+------------+------------+
| last_name  | first_name | birth      |
+------------+------------+------------+
| Washington | George     | 1732-02-22 |
| Adams      | John       | 1735-10-30 |
| Jefferson  | Thomas     | 1743-04-13 |
| Madison    | James      | 1751-03-16 |
| Monroe     | James      | 1758-04-28 |
+------------+------------+------------+
```

역순으로 정렬하고자 한다면, ORDER BY birth DESC를 사용하는데, 이렇게 하면 가장 최근에 태어난 5명의 대통령을 검색하게 된다.

```
mysql> SELECT last_name, first_name, birth FROM president
    -> ORDER BY birth DESC LIMIT 5;
+-----------+-------------+------------+
| last_name | first_name  | birth      |
+-----------+-------------+------------+
| Clinton   | William J.  | 1946-08-19 |
| Bush      | George W.   | 1946-07-06 |
| Carter    | James E.    | 1924-10-01 |
| Bush      | George H.W. | 1924-06-12 |
| Kennedy   | John F.     | 1917-05-29 |
+-----------+-------------+------------+
```

또한 LIMIT를 사용하면, 결과 세트의 중간 부분에서 행들을 골라낼 수도 있다. 이렇게 하려면, 두 개의 값을 지정해야 한다. 처음 값은 결과 세트의 시작에서부터 건너뛰어야 할 행의 수가 되고, 두 번째 값은

리턴할 행의 수가 된다. 다음의 쿼리는 앞서의 쿼리와 비슷하지만 처음 10개의 행을 건너뛰고 5개의 행을 리턴한다.

```
mysql> SELECT last_name, first_name, birth FROM president
    -> ORDER BY birth DESC LIMIT 10, 5;
+-----------+-------------+------------+
| last_name | first_name  | birth      |
+-----------+-------------+------------+
| Truman    | Harry S.    | 1884-05-08 |
| Roosevelt | Franklin D. | 1882-01-30 |
| Hoover    | Herbert C.  | 1874-08-10 |
| Coolidge  | Calvin      | 1872-07-04 |
| Harding   | Warren G.   | 1865-11-02 |
+-----------+-------------+------------+
```

president 테이블에서 임의적으로 행들을 선택하려면, ORDER BY RAND( )를 LIMIT와 결합해서 사용한다.

```
mysql> SELECT last_name, first_name FROM president
    -> ORDER BY RAND() LIMIT 1;
+-----------+------------+
| last_name | first_name |
+-----------+------------+
| Johnson   | Lyndon B.  |
+-----------+------------+
mysql> SELECT last_name, first_name FROM president
    -> ORDER BY RAND() LIMIT 3;
+-----------+-------------+
| last_name | first_name  |
+-----------+-------------+
| Harding   | Warren G.   |
| Bush      | George H.W. |
| Jefferson | Thomas      |
+-----------+-------------+
```

출력되는 칼럼 값들을 연산하고 명명하기

앞에 제시된 쿼리들 대부분은 테이블에서 값을 검색해서 출력을 만들었다. MySQL은 또한 출력되는 칼럼 값을 표현식의 결과로 계산할 수 있도록 해준다. 표현식은 단순할 수도 복잡할 수도 있다. 다음의 쿼리는 단순한 표현식(상수) 하나와 몇 가지 산술 연산과 한 쌍의 함수 호출을 포함하는 좀 더 복잡한 표현식 하나를 평가한다.

```
mysql> SELECT 17, FORMAT(SQRT(25+13),3);
+----+-------------------------+
| 17 | FORMAT(SQRT(25+13),3)   |
+----+-------------------------+
| 17 | 6.164                   |
+----+-------------------------+
```

표현식은 테이블 칼럼에 대한 참조가 될 수도 있다.

```
mysql> SELECT CONCAT(first_name,' ',last_name),CONCAT(city,', ',state)
    -> FROM president;
+-----------------------------------+-------------------------+
| CONCAT(first_name,' ',last_name)  | CONCAT(city,', ',state) |
+-----------------------------------+-------------------------+
| George Washington                 | Wakefield, VA           |
| John Adams                        | Braintree, MA           |
| Thomas Jefferson                  | Albemarle County, VA    |
| James Madison                     | Port Conway, VA         |
...
```

이 쿼리는 first name과 last name 사이에 공백 문자를 하나 넣어서 병합시킨 단일 문자 스트링을 대통령의 이름 포맷으로 사용하고, 태어난 도시와 주 사이에 콤마를 하나 넣어서 대통령의 태어난 장소의 포맷으로 사용한다.

표현식을 사용하여 하나의 칼럼 값을 계산할 때, 해당 표현식은 해당 칼럼의 이름이 되고 출력 표에서 표제로 쓰인다. 이전의 쿼리에서 볼 수 있는 것처럼, 표현식이 긴 경우에는 매우 폭이 큰 칼럼을 가진 표가 출력되기도 한다. 이러한 문제를 다루려면, *AS name* 구조를 사용해서 칼럼에 다른 이름을 대입하면 된다. 이러한 이름을 "칼럼 에일리어스(column alias)"라고 부른다. 칼럼 에일리어스를 사용하게 되면, 다음과 같이 이전의 쿼리에서의 출력이 좀 더 의미 있게 만들어지게 된다.

```
mysql> SELECT CONCAT(first_name,' ',last_name) AS Name,
    -> CONCAT(city,', ',state) AS Birthplace
    -> FROM president;
+-----------------------+-------------------------+
| Name                  | Birthplace              |
+-----------------------+-------------------------+
| George Washington     | Wakefield, VA           |
| John Adams            | Braintree, MA           |
| Thomas Jefferson      | Albemarle County, VA    |
| James Madison         | Port Conway, VA         |
...
```

칼럼 에일리어스에 공백 문자를 포함시키려면, 칼럼 에일리어스를 인용 부호로 감싸야 한다.

```
mysql> SELECT CONCAT(first_name,' ',last_name) AS 'President Name',
    -> CONCAT(city,', ',state) AS 'Place of Birth'
    -> FROM president;
+-----------------------+-------------------------+
| President Name        | Place of Birth          |
+-----------------------+-------------------------+
| George Washington     | Wakefield, VA           |
| John Adams            | Braintree, MA           |
| Thomas Jefferson      | Albemarle County, VA    |
| James Madison         | Port Conway, VA         |
...
```

키워드 AS는 칼럼 에일리어스를 사용할 때 선택할 수 있다.

```
mysql> SELECT 1, 2 AS two, 3 three;
+---+-----+-------+
| 1 | two | three |
+---+-----+-------+
| 1 |   2 |     3 |
+---+-----+-------+
```

필자는 AS를 사용하는 편이다. 그것 없이 쿼리를 작성하는 것은 쉽지만, 의도한 결과를 얻기는 쉽지 않다. 예를 들면, 대통령 이름을 찾는 쿼리를 작성할 때 first_name과 last_name 사이에 콤마를 잊으면 안 된다.

```
mysql> SELECT first_name last_name FROM president;
+---------------+
| last_name     |
+---------------+
| George        |
| John          |
| Thomas        |
| James         |
...
```

결국 쿼리는 두 칼럼을 나타내지 못했다. 그 대신, 오직 first_column만을 표시했을 뿐이고, 컬럼 에일리어스로서 last_name이 사용되었다. 만약 찾기를 원하는 칼럼들이 검색되지 않았고 예측한 것 이외의 다른 칼럼 이름들만 나온다면, 칼럼들 사이에 빠진 콤마를 찾아야 할 것이다.

날짜를 가지고 작업하기

MySQL에서 날짜를 다루게 될 때 기본적으로 명심해야 할 것은 MySQL은 항상 연도가 먼저 나오도록 날짜를 표현한다는 것이다. 2008년 7월 27일은 '2008-07-27'로 나타낸다. 이전에 흔히 쓰는 표현처럼, '07-27-2008' 또는 '27-07-2008'로 나타내지 않는다. 실행할 수 있는 기능 몇 가지를 나열해보면 다음과 같다.

- 날짜를 기준으로 정렬시킴(이것은 이미 몇 차례 해보았다)
- 특정한 날짜나 날짜의 범위에 대해서 검색함
- 연, 월, 일과 같이 날짜 값의 부분을 추출함
- 날짜들 간의 차(빼기 연산)를 계산함
- 어떤 날짜에서 기간을 더하거나 뺌으로써 날짜를 계산함

이들 연산의 몇 가지 예를 든다면 다음과 같다.

정확한 값으로 어떤 특정한 날짜를 찾거나 다른 값에 대해 비교되는 값으로 어떤 특정한 날짜를 찾고자 한다면, 관심이 있는 값에 DATE 칼럼을 비교한다.

```
mysql> SELECT * FROM event WHERE date = '2008-10-01';
+------------+----------+----------+
| date       | category | event_id |
+------------+----------+----------+
| 2008-10-01 | T        |        6 |
+------------+----------+----------+
mysql> SELECT last_name, first_name, death
    -> FROM president
    -> WHERE death >= '1970-01-01' AND death < '1980-01-01';
+-----------+------------+------------+
| last_name | first_name | death      |
+-----------+------------+------------+
| Truman    | Harry S.   | 1972-12-26 |
| Johnson   | Lyndon B.  | 1973-01-22 |
+-----------+------------+------------+
```

날짜의 일부분을 시험하거나 추출하려면, YEAR(), MONTH(), 또는 DAYOFMONTH() 등과 같은 함수들을 사용하면 된다. 예를 들어, 월의 값이 3인 날짜들을 찾아보면 필자의 태어난 달(3월)과 같은 달에 태어난 대통령들을 찾을 수 있는 것이다.

```
mysql> SELECT last_name, first_name, birth
    -> FROM president WHERE MONTH(birth) = 3;
+-----------+------------+------------+
| last_name | first_name | birth      |
+-----------+------------+------------+
| Madison   | James      | 1751-03-16 |
| Jackson   | Andrew     | 1767-03-15 |
| Tyler     | John       | 1790-03-29 |
| Cleveland | Grover     | 1837-03-18 |
+-----------+------------+------------+
```

또한 이 쿼리는 그 달의 이름을 쓸 수도 있다.

```
mysql> SELECT last_name, first_name, birth
    -> FROM president WHERE MONTHNAME(birth) = 'March';
+-----------+------------+------------+
| last_name | first_name | birth      |
+-----------+------------+------------+
| Madison   | James      | 1751-03-16 |
| Jackson   | Andrew     | 1767-03-15 |
| Tyler     | John       | 1790-03-29 |
| Cleveland | Grover     | 1837-03-18 |
+-----------+------------+------------+
```

날짜 수준까지 맞추어서 좀 더 특정하게 지정해보면, 필자의 생일과 같은 대통령들을 찾기 위해 MONTH( )와 DAYOFMONTH( )를 같이 사용해서 검사할 수 있다.

```
mysql> SELECT last_name, first_name, birth
    -> FROM president WHERE MONTH(birth) = 3 AND DAYOFMONTH(birth) = 29;
```

```
+-----------+------------+------------+
| last_name | first_name | birth      |
+-----------+------------+------------+
| Tyler     | John       | 1790-03-29 |
+-----------+------------+------------+
```

이러한 종류의 쿼리는 아마도 소식지의 연예/오락 섹션에서 종종 보게 되는 "오늘 태어난 저명한 인사들"의 목록을 만들어내는 데 사용될 수 있을 것이다. 하지만, 현재의 날짜에 대해, 이전의 쿼리에서 했던 방법에 특정한 날짜를 끼워 넣을 필요는 없다. 오늘과 같은 날짜에 태어난 대통령들을 검사해 보려면, 그들의 생일을 CURDATE( )의 월과 날 부분에 비교하는데, 이 함수는 항상 현재의 날짜를 리턴한다.

```
SELECT last_name, first_name, birth
FROM president WHERE MONTH(birth) = MONTH(CURDATE())
AND DAYOFMONTH(birth) = DAYOFMONTH(CURDATE());
```

어떤 날짜에서 하루를 뺄 수도 있는데, 이것은 날짜들 간의 간격을 알아낼 수 있도록 해준다. 예를 들어, 어느 대통령이 가장 장수했는지 알아보려면, 사망날짜에서 태어난 날짜를 빼면 된다. 이러한 경우에는 TIMESTAMPDIFF() 함수가 유용하게 사용된다.

```
mysql> SELECT last_name, first_name, birth, death,
    -> TIMESTAMPDIFF(YEAR, birth, death) AS age
    -> FROM president WHERE death IS NOT NULL
    -> ORDER BY age DESC LIMIT 5;
+-----------+------------+------------+------------+------+
| last_name | first_name | birth      | death      | age  |
+-----------+------------+------------+------------+------+
| Reagan    | Ronald W.  | 1911-02-06 | 2004-06-05 |   93 |
| Ford      | Gerald R.  | 1913-07-14 | 2006-12-26 |   93 |
| Adams     | John       | 1735-10-30 | 1826-07-04 |   90 |
| Hoover    | Herbert C. | 1874-08-10 | 1964-10-20 |   90 |
| Truman    | Harry S.   | 1884-05-08 | 1972-12-26 |   88 |
+-----------+------------+------------+------------+------+
```

날짜 간의 차이를 계산하는 또 다른 방법으로 TO_DAYS() 함수를 사용할 수 있다. 또한 날짜들 간의 간격을 계산하는 것은 어떤 기준일로부터 며칠이나 떨어져 있는지 알아볼 때 유용하다. 예를 들면, 조만간 회원 자격을 갱신해야 할 필요가 있는 역사 연구회의 회원이 누구인지 알아낼 때에도 사용할 수 있다. 현재 날짜와 회원 자격 만료일의 차이를 계산하고, 만일 어떤 기준 값에 미달하게 되면, 곧 자격 갱신이 필요하게 되는 것이다. 다음은 60일 이내에 자격 갱신을 해야 하는 회원들을 찾는 쿼리이다.

```
SELECT last_name, first_name, expiration FROM member
WHERE (TO_DAYS(expiration) - TO_DAYS(CURDATE())) < 60;
```

TIMESTAMPDIFF()를 사용한 동일한 문장을 만들면 다음과 같다.

```
SELECT last_name, first_name, expiration FROM member
WHERE TIMESTAMPDIFF(DAY, CURDATE(), expiration) < 60;
```

어떤 날에서 다른 날을 계산하려면, DATE_ADD()나 DATE_SUB()를 사용하면 된다. 이 함수들은 날짜

와 간격을 인수로 받아서 새로운 날짜를 만들어내는데, 예를 들면 다음과 같다.

```
mysql> SELECT DATE_ADD('1970-1-1', INTERVAL 10 YEAR);
+---------------------------------------+
| DATE_ADD('1970-1-1', INTERVAL 10 YEAR) |
+---------------------------------------+
| 1980-01-01                            |
+---------------------------------------+
mysql> SELECT DATE_SUB('1970-1-1', INTERVAL 10 YEAR);
+---------------------------------------+
| DATE_SUB('1970-1-1', INTERVAL 10 YEAR) |
+---------------------------------------+
| 1960-01-01                            |
+---------------------------------------+
```

이 절의 앞부분에 1970년대에 사망한 대통령들을 골라내는 쿼리가 있었는데, 그때에는 선택 영역의 양쪽 끝 시점에 대한 날짜를 문자적으로 사용했다. 그 쿼리는 문자로 된 시작 날짜, 그리고 시작 날짜와 시간 간격에서 계산된 마지막 날짜를 사용하여 재작성할 수 있다.

```
mysql> SELECT last_name, first_name, death
    -> FROM president
    -> WHERE death >= '1970-1-1'
    -> AND death < DATE_ADD('1970-1-1', INTERVAL 10 YEAR);
+-----------+------------+------------+
| last_name | first_name | death      |
+-----------+------------+------------+
| Truman    | Harry S.   | 1972-12-26 |
| Johnson   | Lyndon B.  | 1973-01-22 |
+-----------+------------+------------+
```

회원 자격 갱신 쿼리는 DATE_ADD( )를 사용하여 작성할 수 있다.

```
SELECT last_name, first_name, expiration FROM member
WHERE expiration < DATE_ADD(CURDATE(), INTERVAL 60 DAY);
```

만약 expiration 칼럼이 인덱스되면 5장에서 언급되었던 원래의 쿼리보다 좀 더 효율적으로 사용될 수 있다.

이 장의 앞부분에 치과병원에 한동안 방문하지 않았던 환자들을 알아내기 위한 쿼리가 소개되었었다.

```
SELECT last_name, first_name, last_visit FROM patient
WHERE last_visit < DATE_SUB(CURDATE(),INTERVAL 6 MONTH);
```

이 쿼리가 의미하는 것을 그때에는 잘 파악하지 못했을 수도 있다. 이제는 그 의미를 좀더 쉽게 알 수 있지 않은가?

패턴 대응

MySQL은 패턴 대응 기능을 지원하는데, 이 기능은 정확한 비교 값을 주지 않고 행들을 검색할 수 있도록 해준다. 패턴 대응을 시킬 때는 특별한 연산자(LIKE와 NOT LIKE)들을 사용하고, 와일드카드 문자들

을 포함하는 문자열을 지정한다. 문자 '_'는 단일 문자 아무것에나 대응되며, '%'는 문자들의 시퀀스에 대응된다(공백 시퀀스도 포함된다).

다음의 패턴은 'W'나 'w' 문자로 시작하는 last name을 대응시킨다.

```
mysql> SELECT last_name, first_name FROM president
    -> WHERE last_name LIKE 'W%';
+------------+------------+
| last_name  | first_name |
+------------+------------+
| Washington | George     |
| Wilson     | Woodrow    |
+------------+------------+
```

이 쿼리는 일반적인 에러를 보여준다. 패턴 매치는 LIKE를 사용하지 않았기 때문에 에러가 발생하는데, 이것은 산술 비교 연산자에 패턴을 적용했을 때 발생한다.

```
mysql> SELECT last_name, first_name FROM president
    -> WHERE last_name = 'W%';
Empty set (0.00 sec)
```

이러한 비교가 성공되는 조건은 문자열이 정확하게 'W%' 또는 'w%'를 포함하는 경우이다.

다음의 패턴은 last_name 안에서 반드시 시작 부분이 아니라 아무 곳에나 'W' 또는 'w'가 포함되는 경우에 대응된다.

```
mysql> SELECT last_name, first_name FROM president
    -> WHERE last_name LIKE '%W%';
+------------+------------+
| last_name  | first_name |
+------------+------------+
| Washington | George     |
| Wilson     | Woodrow    |
| Eisenhower | Dwight D.  |
+------------+------------+
```

다음의 패턴은 정확하게 네 개의 문자를 포함하는 last name에 대응된다.

```
mysql> SELECT last_name, first_name FROM president
    -> WHERE last_name LIKE '____';
+-----------+-------------+
| last_name | first_name  |
+-----------+-------------+
| Polk      | James K.    |
| Taft      | William H.  |
| Ford      | Gerald R    |
| Bush      | George H.W. |
| Bush      | George W.   |
+-----------+-------------+
```

또한 MySQL은 정규 표현식과 REXEXP 연산자에 기반한 또 다른 형태의 패턴 대응을 지원하는데, 이에 대한 것은 부록 C와 3.5.1 절의 하위절 "연산자 타입"에서 설명된다.

사용자 정의 변수들을 설정하고 사용하기

MySQL은 자기 자신의 변수를 정의할 수 있도록 해준다. 이러한 기능에 의해서 쿼리 결과를 사용하여 변수들을 설정할 수 있어서, 편리하게 그 다음의 쿼리들에서 사용하기 위한 변수들을 저장할 수 있다. Andrew Jackson보다 생년월일이 빠른 대통령들을 찾아내는 작업을 한다고 해보자. 작업 절차를 생각해보면, 그 사람의 생년월일을 검색하여 어떤 변수에 넣은 다음, 이 변수의 값보다 앞선 생년월일을 가지는 다른 대통령들을 검색한다.

```
mysql> SELECT @birth := birth FROM president
    -> WHERE last_name = 'Jackson' AND first_name = 'Andrew';
+-----------------+
| @birth := birth |
+-----------------+
| 1767-03-15      |
+-----------------+
mysql> SELECT last_name, first_name, birth FROM president
    -> WHERE birth < @birth ORDER BY birth;
+------------+------------+------------+
| last_name  | first_name | birth      |
+------------+------------+------------+
| Washington | George     | 1732-02-22 |
| Adams      | John       | 1735-10-30 |
| Jefferson  | Thomas     | 1743-04-13 |
| Madison    | James      | 1751-03-16 |
| Monroe     | James      | 1758-04-28 |
+------------+------------+------------+
```

변수의 이름은 *@var_name* 문법을 사용해서 정하고, 변수에 값을 대입할 때는 SELECT 문 안에서 *@var_name := value* 형식의 표현식을 사용한다. 그러므로 위의 두 쿼리 중 첫 번째 쿼리는 Andrew Jackson에 대한 생년월일을 찾아서 @birth 변수로 대입시키는 것이다(그래도 SELECT의 결과는 표시된다. 어떤 변수에 쿼리의 결과를 대입시킨다고 해서 해당 쿼리의 출력이 생략되도록 하지는 않는다). 그 다음, 두 번째 쿼리는 해당 변수를 참조하고 그 변수의 값을 사용해서 이보다 이른 생년월일을 가지는 대통령들의 행을 찾는다.

이전의 문제는 조인(join) 또는 서브쿼리를 사용한 한 개의 쿼리를 가지고 풀 수 있지만, 조인에 관해 논의할 시점에 아직 이르지 않았다. 조인을 사용하면 변수를 사용하는 것에 비해 좀더 쉬워질 수 있다.

또한 변수에는 SET 문을 사용하여 값을 대입시킬 수 있는데, 이 경우에는 대입 연산자로 = 또는 := 중 어느 것을 사용해도 된다.

```
mysql> SET @today = CURDATE();
mysql> SET @one_week_ago := DATE_SUB(@today(), INTERVAL 7 DAY);
mysql> SELECT @today(), @one_week_ago;
```

```
+------------+---------------+
| @today     | @one_week_ago |
+------------+---------------+
| 2008-03-21 | 2008-03-14    |
+------------+---------------+
```

통계 만들기

MySQL을 가지고 할 수 있는 일 중에서 가장 유용한 것들 중의 한 가지는 많은 양의 가공되지 않은 데이터를 요약해서 통계를 내는 일이다. 통계를 내는 일은 손으로 작업을 한다면 매우 지루하고, 시간이 많이 들고, 에러를 내기 일쑤인 일이기 때문에 MySQL을 사용하여 이러한 일들을 하는 방법을 배우게 된다면 MySQL은 여러분의 강력한 동반자가 되어줄 것이다.

통계를 내는 일 중에서 한 가지 단순한 형태로, 어떤 종류의 값들 중에서 나타난 값들을 중복시키지 않고 하나씩만 나오도록 하는 것이 있다. 어떤 결과 세트에서 중복된 행들을 제거하려면 DISTINCT 키워드를 사용하면 된다. 예를 들어, 각 대통령들을 배출한 주를 나열하려면, 다음과 같이 하면 된다.

```
mysql> SELECT DISTINCT state FROM president ORDER BY state;
+-------+
| state |
+-------+
| AR    |
| CA    |
| CT    |
| GA    |
| IA    |
| IL    |
| KY    |
| MA    |
| MO    |
| NC    |
| NE    |
| NH    |
| NJ    |
| NY    |
| OH    |
| PA    |
| SC    |
| TX    |
| VA    |
| VT    |
+-------+
```

또 다른 형태의 통계 예로, COUNT( ) 함수를 이용하여 세는 것이 있다. COUNT(*)를 사용하면, 쿼리에 의해 검색된 행의 수를 알 수 있다. WHERE 절이 없는 쿼리라면, COUNT(*)는 테이블 내의 행의 수를 말해준다. 다음의 쿼리는 역사 연구회의 member 테이블에서 나열되는 회원의 총 수를 보여준다.

```
mysql> SELECT COUNT(*) FROM member;
+----------+
| COUNT(*) |
+----------+
|      102 |
+----------+
```

쿼리에 WHERE 절이 있으면, COUNT(*)는 해당 절에 대응되는 행의 수를 의미하게 된다. 다음의 쿼리는 학급에서 지금까지 얼마나 많은 퀴즈풀이를 시행했는지를 보여준다.

```
mysql> SELECT COUNT(*) FROM grade_event WHERE type = 'Q';
+----------+
| COUNT(*) |
+----------+
|        4 |
+----------+
```

COUNT(*)는 검색된 모든 행을 센다. 이와는 대조적으로, COUNT(*col_name*)은 오로지 NULL이 아닌 값들만 센다. 다음의 쿼리는 이들 간의 차이가 어떠한지를 보여주고 있다.

```
mysql> SELECT COUNT(*),COUNT(email), COUNT(expiration) FROM member;
+----------+--------------+-------------------+
| COUNT(*) | COUNT(email) | COUNT(expiration) |
+----------+--------------+-------------------+
|      102 |           80 |                96 |
+----------+--------------+-------------------+
```

이 예제는 member 테이블에 102개의 행이 있지만, 이 중에서 80개만 email 칼럼에 값을 가지고 있다는 것을 보여준다. 평생 회원권을 가진 사람은 6명이라는 것도 알 수 있다(expiration 칼럼에 있는 NULL 값은 평생회원이라는 것을 나타내고, 102개의 행 중에서 96개가 NULL이 아니므로 나머지는 6이 남는다).

COUNT()와 DISTINCT를 같이 연결하여 사용하면 결과 안의 구별된 NULL이 아닌 값들의 수를 셀 수 있다. 예를 들면, 어떤 주(state)에 몇 명의 대통령이 태어났는지를 알아보려면, 다음과 같이 한다.

```
mysql> SELECT COUNT(DISTINCT state) FROM president;
+------------------------+
| COUNT(DISTINCT state) |
+------------------------+
|                     20 |
+------------------------+
```

어떤 칼럼 안의 값의 전체 개수를 세거나 카테고리별로 셀 수 있다. 예를 들면, 다음의 쿼리를 실행하면 학급 내의 모든 학생수를 알 수 있게 된다.

```
mysql> SELECT COUNT(*) FROM student;
+----------+
| COUNT(*) |
+----------+
```

```
|       31 |
+----------+
```

하지만, 이 학생들 중에서 남학생과 여학생의 수는 어떻게 되는가? 알아볼 수 있는 한 가지 방법은 각 성(sex)을 따로 구별해서 세는 것이다.

```
mysql> SELECT COUNT(*) FROM student WHERE sex='f';
+----------+
| COUNT(*) |
+----------+
|       15 |
+----------+

mysql> SELECT COUNT(*) FROM student WHERE sex='m';
+----------+
| COUNT(*) |
+----------+
|       16 |
+----------+
```

그러나 이러한 방법이 잘 통한다고 하더라도, 어떤 경우에는 매우 따분한 작업이 될 수도 있고 몇 가지 다른 값들을 가지게 되는 칼럼들에 대해서 실제로 그다지 썩 잘 맞지 않는 수도 있다. 이 방법으로 각 주(state)에서 태어난 대통령들의 수를 세려고 한다고 해보자. 대통령이 태어난 주들을 일단 찾고 난 다음(SELECT DISTINCT state FROM president), 각 주에 대해 SELECT COUNT(*) 쿼리를 실행해야만 한다. 분명히 이런 식으로 작업을 하려는 사람은 없을 것이다.

다행히도, MySQL에서는 하나의 단일 쿼리를 사용해서, 어떠한 칼럼 안에 각각의 구별된 값이 몇 번 나오게 되는지 알 수 있다. 여기서 다루는 학생들의 목록에 대해서, 남학생과 여학생의 수를 세는 것은 다음과 같이 하면 된다.

```
mysql> SELECT sex, COUNT(*) FROM student GROUP BY sex;
+-----+----------+
| sex | COUNT(*) |
+-----+----------+
| F   |       15 |
| M   |       16 |
+-----+----------+
```

이와 동일한 형태의 쿼리를 사용해서 각 주에 태어난 대통령들의 수를 세려면 다음과 같이 한다.

```
mysql> SELECT state, COUNT(*) FROM president GROUP BY state;
+-------+----------+
| state | COUNT(*) |
+-------+----------+
| AR    |        1 |
| CA    |        1 |
| CT    |        1 |
| GA    |        1 |
```

```
| IA    |         1 |
| IL    |         1 |
| KY    |         1 |
| MA    |         4 |
| MO    |         1 |
| NC    |         2 |
| NE    |         1 |
| NH    |         1 |
| NJ    |         1 |
| NY    |         4 |
| OH    |         7 |
| PA    |         1 |
| SC    |         1 |
| TX    |         2 |
| VA    |         8 |
| VT    |         2 |
+-------+-----------+
```

이런 식으로 값들을 셀 때에는 GROUP BY 절이 필수적으로 들어간다. 이 절은 MySQL이 값들을 세기 전에 그룹 단위로 묶는 방법을 정해주는 것이다. 이 절을 누락시키게 되면 에러가 발생한다.

값들을 세기 위해 GROUP BY와 COUNT(*)를 이용하게 되면, 각각 구분된 칼럼 값에 개별적으로 수를 세는 것에 비해 몇 가지 장점을 가지게 된다.

- 통계를 내는 칼럼 안에 어떤 값들이 들어 있는지 몰라도 된다.
- 복수의 쿼리를 작성할 필요 없이, 단 하나의 쿼리만 작성해도 된다.
- 하나의 쿼리로 모든 결과를 얻을 수 있기 때문에, 출력을 정렬할 수 있다.

위에서 나열한 것 중 처음 두 개의 장점은 쿼리를 더욱 쉽게 표현하고자 할 경우에 중요한 사항이 된다. 세 번째의 장점은 결과를 매우 유연하게 표시할 수 있다는 점에서 중요하다. 기본적으로, MySQL은 결과를 정렬할 때 GROUP BY에 지명된 칼럼들을 사용하지만, ORDER BY 절을 지정해서 이와는 다른 순서로 정렬할 수 있다. 예를 들면, 각 주(state)에 태어난 대통령의 수를 구할 때 많이 배출한 순서대로 주를 정렬하려면, 다음과 같이 ORDER BY 절을 사용하면 된다.

```
mysql> SELECT state, COUNT(*) AS count FROM president
    -> GROUP BY state ORDER BY count DESC;
+-------+-------+
| state | count |
+-------+-------+
| VA    |     8 |
| OH    |     7 |
| MA    |     4 |
| NY    |     4 |
| NC    |     2 |
| VT    |     2 |
| TX    |     2 |
```

```
| SC    |     1 |
| NH    |     1 |
| PA    |     1 |
| KY    |     1 |
| NJ    |     1 |
| IA    |     1 |
| MO    |     1 |
| CA    |     1 |
| NE    |     1 |
| GA    |     1 |
| IL    |     1 |
| AR    |     1 |
| CT    |     1 |
+-------+-------+
```

정렬하고자 하는 칼럼이 통계 함수에 의해 정해지게 될 때, ORDER BY 절에서 이 함수를 참조할 수 없다. 그 대신, 칼럼을 에일리어스로 만들고, 이렇게 해서 참조를 할 수 있다. 앞에 있는 쿼리에서 이와 같은 사항을 볼 수 있는데, 여기서 COUNT(*) 칼럼은 count로 에일리어스를 만들었다. 이와 같이 ORDER BY 절의 칼럼을 참조하는 또 다른 방법으로는 출력 안에서 위치를 이용하는 것이다. 이전의 쿼리는 다음과 같이 다시 작성할 수 있다.

```
SELECT state, COUNT(*) FROM president
GROUP BY state ORDER BY 2 DESC;
```

위치에 의해 칼럼들을 참조하는 것은 MySQL에서 사용될 수 있는 방법이지만 문제가 될 수도 있다.

- 숫자가 이름보다 의미를 알기에는 부족하므로 칼럼 위치를 이용하는 방법은 그다지 이해하기 쉬운 형태의 쿼리가 아니라고 여겨진다.
- 출력 칼럼을 더하고, 삭제하고, 또는 재정렬하게 되면, 반드시 ORDER BY 절을 점검해보는 것을 기억해서, 칼럼 번호가 변경되면 그것을 수정해야 한다.
- ORDER BY 절에서 칼럼 위치를 참조하는 문법은 표준 SQL이 아니며, 이러한 방법은 지양되어야 한다.

에일리어스를 사용할 때에는 이러한 문제가 없다.

계산된 칼럼에 GROUP BY를 사용하여 결과를 그룹으로 묶고 싶으면, ORDER BY로 했던 것처럼 에일리어스 또는 칼럼 위치를 사용하여 참조할 수 있다. 다음은 특정 연도의 각 월마다 태어났던 대통령의 수를 알아보는 쿼리이다.

```
mysql> SELECT MONTH(birth) AS Month, MONTHNAME(birth) AS Name,
    -> COUNT(*) AS count
    -> FROM president GROUP BY Name ORDER BY Month;
+-------+-----------+-------+
| Month | Name      | count |
+-------+-----------+-------+
|     1 | January   |     4 |
|     2 | February  |     4 |
```

```
|     3 | March     |     4 |
|     4 | April     |     4 |
|     5 | May       |     2 |
|     6 | June      |     1 |
|     7 | July      |     4 |
|     8 | August    |     4 |
|     9 | September |     1 |
|    10 | October   |     6 |
|    11 | November  |     5 |
|    12 | December  |     3 |
+-------+-----------+-------+
```

COUNT()는 ORDER BY와 LIMIT를 같이 사용해서 president 테이블에서 대통령을 가장 많이 배출한 네 개의 주를 찾는 쿼리를 작성할 수 있다.

```
mysql> SELECT state, COUNT(*) AS count FROM president
    -> GROUP BY state ORDER BY count DESC LIMIT 4;
+-------+-------+
| state | count |
+-------+-------+
| VA    |     8 |
| OH    |     7 |
| MA    |     4 |
| NY    |     4 |
+-------+-------+
```

LIMIT 절을 사용하지 않고 특정한 COUNT() 값을 찾는 것으로 쿼리의 출력을 제한하고자 하는 경우에는 HAVING 절을 사용한다. HAVING은 출력 행들이 만족해야 하는 조건을 지정한다는 점에서는 WHERE와 비슷하다. COUNT()와 같은 통계 함수들의 결과를 참조할 수 있다는 점이 WHERE와는 다르다. 다음의 쿼리는 2인 이상의 대통령을 배출한 주를 출력한다.

```
mysql> SELECT state, COUNT(*) AS count FROM president
    -> GROUP BY state HAVING count > 1 ORDER BY count DESC;
+-------+-------+
| state | count |
+-------+-------+
| VA    |     8 |
| OH    |     7 |
| MA    |     4 |
| NY    |     4 |
| NC    |     2 |
| VT    |     2 |
| TX    |     2 |
+-------+-------+
```

좀더 일반적으로 말하면, 이러한 형식의 쿼리는 칼럼 내에서 중복된 값을 찾을 때 실행하는 쿼리이다. 또는 중복되지 않은 값을 찾기 위해서 HAVING count =1을 사용한다.

COUNT() 이외에도 여러 통계 함수들이 있다. MIN(), MAX(), SUM(), 그리고 AVG() 함수들은 어떤 칼럼 내에서 각각 최소값, 최대값, 총합, 그리고 평균값을 구할 때 사용된다. 이들은 모두 동시에 같이 사용할 수도 있다. 다음의 쿼리는 주어진 각 퀴즈와 시험에 대한 여러 가지 수치적인 특성을 보여준다. 이 쿼리는 또한 얼마나 많은 점수들을 가지고 각 값들을 계산했는지 보여준다(어떤 학생들은 결석했기 때문에 셈에서 빠지는 수도 있다).

```
mysql> SELECT
    -> event_id,
    -> MIN(score) AS minimum,
    -> MAX(score) AS maximum,
    -> MAX(score)-MIN(score)+1 AS span,
    -> SUM(score) AS total,
    -> AVG(score) AS average,
    -> COUNT(score) AS count
    -> FROM score
    -> GROUP BY event_id;
+----------+---------+---------+-------+-------+---------+-------+
| event_id | minimum | maximum | span  | total | average | count |
+----------+---------+---------+-------+-------+---------+-------+
|        1 |       9 |      20 |    12 |   439 | 15.1379 |    29 |
|        2 |       8 |      19 |    12 |   425 | 14.1667 |    30 |
|        3 |      60 |      97 |    38 |  2425 | 78.2258 |    31 |
|        4 |       7 |      20 |    14 |   379 | 14.0370 |    27 |
|        5 |       8 |      20 |    13 |   383 | 14.1852 |    27 |
|        6 |      62 |     100 |    39 |  2325 | 80.1724 |    29 |
+----------+---------+---------+-------+-------+---------+-------+
```

물론, 이와 같은 정보는 event_id 값이 퀴즈인지 시험인지의 여부를 알았다면 상당히 의미 있는 것이 된다. 하지만, 이러한 정보를 만들기 위해서는 grade_event 테이블에도 질의할 필요가 있는 것이다. 이 쿼리는 1.4.9 절의 하위절 "복수의 테이블에서 정보 검색하기"에서 다시 다루게 된다.

"통계 요약"을 제공하기 위해서 추가 출력 라인을 생성하길 원한다면, WITH ROLLUP 절을 사용하면 된다. 이는 그룹으로 묶여진 행들의 위한 "슈퍼 집합" 값들을 계산한다. 각 성별 학생의 수를 계산하는 이전 쿼리 문장에 기반한 예를 살펴보도록 하자. WITH ROLLUP 절은 두 성별의 총 합을 요약하는 다른 라인을 생성한다.

```
mysql> SELECT sex, COUNT(*) FROM student GROUP BY sex WITH ROLLUP;
+-----+----------+
| sex | COUNT(*) |
+-----+----------+
| F   |       15 |
| M   |       16 |
| NULL|       31 |
+-----+----------+
```

그룹으로 묶여진 칼럼들에서 NULL에 해당하는 숫자가 그룹의 통계 값이라는 것을 알려준다.

또한 WITH ROLLUP 절은 다른 총계 함수를 사용할 수 있다. 다음의 문장은 이전에 나타내었던 성적 통계를 계산하는 것이며, 역시 별도의 슈퍼 집합 라인을 제공한다.

```
mysql> SELECT
    -> event_id,
    -> MIN(score) AS minimum,
    -> MAX(score) AS maximum,
    -> MAX(score)-MIN(score)+1 AS span,
    -> SUM(score) AS total,
    -> AVG(score) AS average,
    -> COUNT(score) AS count
    -> FROM score
    -> GROUP BY event_id
    -> WITH ROLLUP;

+----------+---------+---------+------+-------+---------+-------+
| event_id | minimum | maximum | span | total | average | count |
+----------+---------+---------+------+-------+---------+-------+
|        1 |       9 |      20 |   12 |   439 | 15.1379 |    29 |
|        2 |       8 |      19 |   12 |   425 | 14.1667 |    30 |
|        3 |      60 |      97 |   38 |  2425 | 78.2258 |    31 |
|        4 |       7 |      20 |   14 |   379 | 14.0370 |    27 |
|        5 |       8 |      20 |   13 |   383 | 14.1852 |    27 |
|        6 |      62 |     100 |   39 |  2325 | 80.1724 |    29 |
|     NULL |       7 |     100 |   94 |  6376 | 36.8555 |   173 |
+----------+---------+---------+------+-------+---------+-------+
```

이 결과에서 마지막 라인은 모든 이전 통계 값에 기반하여 계산된 집합에서 나타내고자 하는 값들을 표시한다.

WITH ROLLUP은 다른 쿼리를 수행하지 않고도 추가적인 정보를 제공하기 때문에 매우 유용하다. 서버가 데이터를 두 번 검사할 필요가 없어지기 때문에 단일 쿼리가 더욱 더 효율적이게 된다. 만약 GROUP BY 절이 한 개 이상의 칼럼을 표시하게 되면, WITH ROLLUP이 보다 높은 수준의 통계 값을 포함하는 슈퍼 집합 라인을 추가적으로 제공하게 된다.

통계 함수들은 매우 강력하기 때문에 가지고 놀기에 재밌는 것들이지만, 쉽게 질리기도 한다. 다음과 같은 쿼리를 생각해보자.

```
mysql> SELECT
    -> state AS State,
    -> AVG(TIMESTAMPDIFF(YEAR, birth, death)) AS Age
    -> FROM president WHERE death IS NOT NULL
    -> GROUP BY state ORDER BY Age;
+-------+---------+
| State | Age     |
+-------+---------+
| KY    | 56.0000 |
| VT    | 58.5000 |
```

```
| NC    | 59.5000 |
| OH    | 62.2857 |
| NH    | 64.0000 |
| NY    | 69.0000 |
| NJ    | 71.0000 |
| TX    | 71.0000 |
| MA    | 72.0000 |
| VA    | 72.3750 |
| PA    | 77.0000 |
| SC    | 78.0000 |
| CA    | 81.0000 |
| MO    | 88.0000 |
| IA    | 90.0000 |
| NE    | 93.0000 |
| IL    | 93.0000 |
+-------+---------+
```

이 쿼리는 태어난 주를 기준으로 사망한 대통령들을 그룹으로 만들고, 사망 당시의 나이를 계산하고, 평균 나이를 계산하고(각 주에 대하여), 평균 나이를 기준으로 정렬한다. 다시 말하면, 이 쿼리는 생존하지 않는 대통령들에 대해서 출신 주를 기준으로 사망 당시의 평균 연령을 알아내는 것이다.

그러면 이것이 무엇을 보여주는 것인가? 이것은 그저 이러한 쿼리를 작성할 수 있다는 것을 보여줄 뿐인 것이다. 분명히, 이와 같은 쿼리는 작성할 가치가 있는 것으로 여겨지지는 않는다. 데이터베이스를 가지고 할 수 있는 것이라고 해서 모두 동등한 가치를 지니는 것은 아니다. 그럼에도 불구하고, 때때로 사람들은 자신의 데이터베이스를 가지고 무엇인가를 할 수 있다는 것을 알아냈을 때, 그때 사용한 쿼리를 생각하면서 행복한 생각이 들기도 한다. 이것을 보면서 지난 몇 년간 중계된 스포츠 경기에 관한 혼자만의 비법으로서의 통계가 점점 더 늘어나고 있는 것을 설명해줄 수도 있겠다. 스포츠 통계원은 알고 싶어하는 모든 것들과 절대로 알고 싶어하지 않는 모든 것들까지도 자신의 데이터베이스를 사용하여 알아낸다. 3군 경기에서 자기 팀이 두 번째 쿼터의 마지막 2분에 15야드 라인 안쪽에서 14점 이상을 앞서고 있을 때, 세 번째 다운에서 가장 많은 인터셉트를 기록하고 있는 쿼터백은 누구인가를 실제 알고 싶어하는 여러분이 정말 있기나 한 것일까?

복수의 테이블에서 정보 검색하기

지금까지 작성해온 쿼리들은 단일 테이블에서 데이터를 가져오는 것이었다. 그러나 MySQL은 이보다 더 힘든 작업을 할 수 있는 능력을 가지고 있다. 개별 테이블만 가지고는 해답을 얻을 수 없는 문제들을 복수의 테이블에서 정보를 결합해서 풀 수 있기 때문에, 하나의 테이블을 또 다른 테이블에 연관시킬 수 있는 능력이 관계형 DBMS의 강력함이라고 앞서서 언급했었다. 이 절에서는 이러한 작업을 하는 쿼리를 작성하는 방법을 설명한다.

복수의 테이블에서 정보를 검색할 때, "조인(join)"이라고 하는 작업을 수행하게 된다. 이 명칭은 하나의 테이블에 있는 정보를 또 다른 테이블에 있는 정보에 결합시켜서 결과를 얻는다는 뜻에서 붙여졌다. 조인 작업은 또 다른 SELECT 안에 중첩된 SELECT를 사용하여 여러 테이블 안에 있는 공통된 값들을 대

응시킴으로써 이루어진다. 이렇게 중첩된 SELECT를 "서브쿼리"라고 한다. 이 절에서는 두 가지 모든 연산자에 대해서 살펴보도록 한다.

예제를 통해 작업을 해보도록 하자. 앞에 나온 1.4.6절의 하부절 "성적 기록부 프로젝트에 대한 테이블"에서, 어떤 주어진 날짜에 대한 퀴즈 점수 또는 시험 점수를 가져오기 위한 쿼리가 별도의 설명 없이 주어졌다. 지금이 그 설명을 할 때이다. 이 쿼리는 실제로 3중의 조인을 포함하므로, 두 단계에서 구성을 할 것이다. 첫 번째 단계에서는 다음과 같이 주어진 날짜에 대한 점수들을 검색하는 쿼리를 구성한다.

```
mysql> SELECT student_id, date, score, category
    -> FROM grade_event INNER JOIN score
    -> ON grade_event.event_id = score.event_id
    -> WHERE date = '2008-09-23';
+------------+------------+-------+----------+
| student_id | date       | score | category |
+------------+------------+-------+----------+
|          1 | 2008-09-23 |    15 | Q        |
|          2 | 2008-09-23 |    12 | Q        |
|          3 | 2008-09-23 |    11 | Q        |
|          5 | 2008-09-23 |    13 | Q        |
|          6 | 2008-09-23 |    18 | Q        |
...
```

이 쿼리는 주어진 ('2008-09-23') 데이터를 가지는 grade_event 행을 찾는 것으로 작업한 다음, 찾은 행 안의 event ID를 사용하여 동일한 event ID를 가지는 점수(score)를 알아낸다. 각각의 대응되는 grade_event 행과 score 행의 조합에 대하여, 학생의 ID, 점수, 날짜, 그리고 이벤트 타입이 표시된다.

이 쿼리는 지금까지 작성해온 쿼리들과는 다른 두 가지의 중요한 측면이 있다.

- 하나 이상의 테이블에서 데이터를 검색하기 때문에 하나 이상의 테이블의 이름을 FROM 절에 써준다.

  ```
  FROM grade_event INNER JOIN score
  ```

- on 절이 지정하는 것은 grade_event 테이블과 score 테이블이 각 테이블 안에 있는 event_id 값들을 대응시킴으로써 결합된다.

  ```
  ON grade_event.event_id = score.event_id
  ```

참조하고 있는 테이블이 어떤 것인지 MySQL이 알 수 있도록 *tbl_name.col_name* 문법을 사용하여 grade_event.event_id로서 event_id 칼럼들을 참조하고 있는 것을 주의해서 보기 바란다. event_id는 두 테이블 모두 가지고 있으므로, 테이블 이름을 쓰지 않고 event_id를 얘기하게 되면 어느 쪽에 해당되는 것인지 모호하게 되는 것이다. 쿼리 안의 다른 칼럼들(date, score, category)은 테이블을 나타내는 수식어를 사용하지 않아도 되는데, 이 칼럼들은 단 하나의 테이블에서만 사용하므로 테이블 수식어 없이 사용해도 모호하지 않기 때문이다.

이 책에서는 일반적으로 조인(join) 문장에서는 모든 칼럼에 테이블 수식어를 사용해서, 각 칼럼이 어느 테이블에 속하는가를 좀더 분명하게(더욱 명시적으로) 만들도록 할 것이다. 지금부터는 이런 식으로 조인을 작성해나갈 것이다.

```
SELECT score.student_id, grade_event.date, score.score, grade_event.category
FROM grade_event INNER JOIN score
ON grade_event.event_id = score.event_id
WHERE grade_event.date = '2008-09-23';
```

첫 번째 단계의 쿼리는 grade_event 테이블을 사용하여 어떤 날짜를 특정 이벤트의 ID에 대응시키고 난 다음, 그 ID를 사용하여 score 테이블에서 대응되는 점수들을 찾는다. 이 쿼리에서의 출력에는 student_id 값들이 포함되지만, 이름을 사용할 때 더 잘 알아볼 수 있게 된다. student 테이블을 사용하면 student ID들을 실제 이름에 대응시킬 수 있는데, 이것이 두 번째 단계가 된다. score 테이블과 student 테이블은 모두 student_id 칼럼을 가지고 있고, 이 칼럼들 안의 행들을 연결할 수 있다는 사실을 이용하면 이름을 표시할 수 있는 것이다. 그 결과로 다음과 같은 쿼리를 작성할 수 있다.

```
mysql> SELECT
    -> student.name, grade_event.date, score.score, grade_event.category
    -> FROM grade_event INNER JOIN score INNER JOIN student
    -> ON grade_event.event_id = score.event_id
    -> AND score.student_id = student.student_id
    -> WHERE grade_event.date = '2008-09-23';
+-----------+------------+-------+----------+
| name      | date       | score | category |
+-----------+------------+-------+----------+
| Megan     | 2008-09-23 | 15    | Q        |
| Joseph    | 2008-09-23 | 12    | Q        |
| Kyle      | 2008-09-23 | 11    | Q        |
| Abby      | 2008-09-23 | 13    | Q        |
| Nathan    | 2008-09-23 | 18    | Q        |
...
```

이 쿼리는 이전의 쿼리와는 다음과 같은 점에서 다르다.

- student 테이블이 FROM절에 추가되는데, grade_event 테이블과 score 테이블 외에도 student 테이블을 사용해야 되기 때문이다.
- student_id 칼럼 이전에는 모호하지 않았기 때문에, 지정자가 없는 형태(student_id)와 지정자를 사용하는 형태(score.student_id) 모두 참조가 가능했다. 지금은 score 테이블과 student 테이블 양쪽 모두에 student_id가 있기 때문에 모호한 상황이 되어, 반드시 score.student_id 또는 student.student_id와 같이 지정자를 사용하여 어떤 테이블을 사용할 것인지를 명확하게 만들어야 한다.
- student ID를 기반으로 하여 score 테이블의 행들이 student 테이블 행에 대응된다는 점을 지정하는 별도의 항목이 ON절에 들어 있다.

  ```
  ON ... score.student_id = student.student_id
  ```

- 이 쿼리는 student ID가 아니고 학생의 이름을 표시한다(물론 원한다면 둘 다 표시하게 할 수 있다. 출력 칼럼의 리스트에 student_id를 추가한다).

이 쿼리에 어떠한 날짜라도 대입시켜 그 날짜에 대한 점수를 다시 얻어서, 학생의 이름과 점수 유형으로 완료할 수 있다. student ID나 event ID에 대해서는 몰라도 되는 것이다. MySQL이 적절한 ID값들을 찾아내서, 이 값을 가지고 자동적으로 테이블 행들을 맞춰나가게 된다.

또한 성적 기록부 프로젝트에는 학생들의 결석에 대한 통계를 내는 업무가 있다. absense 테이블 안에 student ID와 날짜를 넣어서 absence 기록을 남긴다. 학생들의 이름을 얻으려면(ID만 얻는 것이 아닌), student_id값에 기반하여 absence 테이블을 student 테이블에 조인(join)시켜야 한다. 다음의 쿼리는 결석일수 순서대로 학생의 ID 번호와 이름을 나열한다.

```
mysql> SELECT student.student_id, student.name,
    -> COUNT(absence.date) AS absences
    -> FROM student INNER JOIN absence
    -> ON student.student_id = absence.student_id
    -> GROUP BY student.student_id;
+------------+-------+----------+
| student_id | name  | absences |
+------------+-------+----------+
|          3 | Kyle  |        1 |
|          5 | Abby  |        1 |
|         10 | Peter |        2 |
|         17 | Will  |        1 |
|         20 | Avery |        1 |
+------------+-------+----------+
```

NOTE

여기서는 GROUP BY 절 안에 테이블을 지정하는 지정자를 사용했지만, 이 쿼리에서 반드시 필요한 것은 아니다. GROUP BY는 출력 칼럼들을 참조하며, 출력 칼럼에는 이름이 student_id인 칼럼 하나뿐이므로, MySQL은 어느 테이블에 있는 칼럼인지 혼동하지 않기 때문이다.

어느 학생들이 결석을 했는지 알고자 하는 경우에는 이 쿼리가 꽤 좋은 출력을 만들어낸다. 그러나 이 목록을 교무처에 제출하게 되면, 교무 담당이 "다른 학생들에 대한 자료는요? 모든 학생들에 대한 자료가 필요한데요" 라고 할지도 모르겠다. 이것은 약간 다른 문제이다. 결석하지 않은 학생들에 대한 자료까지도 포함시켜서 결석일수를 정리해야 한다는 것이다. 다른 문제가 되기 때문에 쿼리 또한 달라진다.

내부 조인(join)이 아닌 LEFT JOIN을 사용하면 이러한 문제를 해결할 수 있다. MySQL에서 LEFT JOIN을 쓰면 조인 관계 안에서 처음 지명된 테이블에서 검색된 각 행에 대한 출력을 하나의 행으로 만들어낸다(이것은 LEFT JOIN 키워드의 왼쪽에 명명된 테이블이다). student 테이블을 우선 명명함으로써, absense 테이블 안에 나타나지 않은 학생이더라도 모든 학생에 대한 출력을 얻을 것이다. 이 쿼리를 작성할 때에는 FROM 절 안에서 지명된 테이블들 간에 LEFT JOIN을 사용하고(콤마로 테이블들을 분리하지 않고), 두 테이블 안에 있는 행을 어떻게 대응시키는지 말해 주는 ON 절을 추가한다. 이 쿼리는 다음

과 같이 작성된다.

```
mysql> SELECT student.student_id, student.name,
    -> COUNT(absence.date) AS absences
    -> FROM student LEFT JOIN absence
    -> ON student.student_id = absence.student_id
    -> GROUP BY student.student_id;
+------------+-----------+----------+
| student_id | name      | absences |
+------------+-----------+----------+
|          1 | Megan     |        0 |
|          2 | Joseph    |        0 |
|          3 | Kyle      |        1 |
|          4 | Katie     |        0 |
|          5 | Abby      |        1 |
|          6 | Nathan    |        0 |
|          7 | Liesl     |        0 |
...
```

앞서 1.4.9절의 하위절 "통계 만들기"에서 score 테이블 안에 있는 데이터의 수치적인 특성을 만들어내는 쿼리를 실행했다. 그 쿼리에서는 event ID의 목록이 출력되지만 event의 날짜나 유형은 포함되지 않았는데, 그때에는 score 테이블을 grade_event 테이블에 조인시켜서 날짜와 유형에 대해 ID들을 대응시키는 방법을 몰랐기 때문이다. 이제는 그렇게 할 수 있다. 다음의 쿼리는 앞서서 실행한 것과 같지만, 단순하게 수치적인 event ID들이 아니라 날짜와 유형을 모두 보여주게 된다.

```
mysql> SELECT
    -> grade_event.date,grade_event.category,
    -> MIN(score.score) AS minimum,
    -> MAX(score.score) AS maximum,
    -> MAX(score.score)-MIN(score.score)+1 AS span,
    -> SUM(score.score) AS total,
    -> AVG(score.score) AS average,
    -> COUNT(score.score) AS count
    -> FROM score INNER JOIN grade_event
    -> ON score.event_id = grade_event.event_id
    -> GROUP BY grade_event.date;
+------------+----------+---------+---------+------+-------+---------+-------+
| date       | category | minimum | maximum | span | total | average | count |
+------------+----------+---------+---------+------+-------+---------+-------+
| 2008-09-03 | Q        |       9 |      20 |   12 |   439 | 15.1379 |    29 |
| 2008-09-06 | Q        |       8 |      19 |   12 |   425 | 14.1667 |    30 |
| 2008-09-09 | T        |      60 |      97 |   38 |  2425 | 78.2258 |    31 |
| 2008-09-16 | Q        |       7 |      20 |   14 |   379 | 14.0370 |    27 |
| 2008-09-23 | Q        |       8 |      20 |   13 |   383 | 14.1852 |    27 |
| 2008-10-01 | T        |      62 |     100 |   39 |  2325 | 80.1724 |    29 |
+------------+----------+---------+---------+------+-------+---------+-------+
```

다른 테이블에 있는 칼럼들인 경우라도, 복수의 칼럼들에 대해 통계 계산을 하고 싶다면, COUNT()와 AVG() 같은 함수들을 사용하면 된다. 다음의 쿼리는 event 날짜와 학생들의 성별의 각 조합에 대해 점수들의 수와 평균 점수를 알아내는 것이다.

```
mysql> SELECT grade_event.date, student.sex,
    -> COUNT(score.score) AS count, AVG(score.score) AS average
    -> FROM grade_event INNER JOIN score INNER JOIN student
    -> ON grade_event.event_id = score.event_id
    -> AND score.student_id = student.student_id
    -> GROUP BY grade_event.date, student.sex;
+------------+-----+-------+---------+
| date       | sex | count | average |
+------------+-----+-------+---------+
| 2008-09-03 | F   |    14 | 14.6429 |
| 2008-09-03 | M   |    15 | 15.6000 |
| 2008-09-06 | F   |    14 | 14.7143 |
| 2008-09-06 | M   |    16 | 13.6875 |
| 2008-09-09 | F   |    15 | 77.4000 |
| 2008-09-09 | M   |    16 | 79.0000 |
| 2008-09-16 | F   |    13 | 15.3077 |
| 2008-09-16 | M   |    14 | 12.8571 |
| 2008-09-23 | F   |    12 | 14.0833 |
| 2008-09-23 | M   |    15 | 14.2667 |
| 2008-10-01 | F   |    14 | 77.7857 |
| 2008-10-01 | M   |    15 | 82.4000 |
+------------+-----+-------+---------+
```

이와 비슷한 쿼리를 사용해서 성적 기록부 프로젝트 중 하나인, 학기말에 학생에 대한 총점을 계산하는 업무를 수행할 수 있다. 쿼리는 다음과 같이 작성할 수 있다.

```
SELECT student.student_id, student.name,
SUM(score.score) AS total, COUNT(score.score) AS n
FROM grade_event INNER JOIN score INNER JOIN student
ON grade_event.event_id = score.event_id
AND score.student_id = student.student_id
GROUP BY score.student_id
ORDER BY total;
```

다른 테이블들 사이에서 수행되는 조인에는 요구사항은 없다. 처음에는 이것이 이상하게 보일지 모르겠지만, 어떤 테이블을 그 자체에 조인을 설정할 수도 있다. 예를 들면, 각 대통령들의 고향을 모든 다른 대통령들의 고향에 대해서 조사해보는 방법으로, 같은 도시에서 대통령들이 태어났는지의 여부를 알 수 있다.

```
mysql> SELECT p1.last_name, p1.first_name, p1.city, p1.state
    -> FROM president AS p1 INNER JOIN president AS p2
    -> ON p1.city = p2.city AND p1.state = p2.state
    -> WHERE (p1.last_name <> p2.last_name OR p1.first_name <> p2.first_name)
    -> ORDER BY state, city, last_name;
```

```
+-----------+-------------+-----------+-------+
| last_name | first_name  | city      | state |
+-----------+-------------+-----------+-------+
| Adams     | John Quincy | Braintree | MA    |
| Adams     | John        | Braintree | MA    |
+-----------+-------------+-----------+-------+
```

이 쿼리에는 두 가지의 기법이 들어 있다.

- 동일한 테이블의 인스턴스 두 개를 참조할 필요가 있기 때문에, 이 쿼리에서는 테이블 에일리어스(p1과 p2)를 만들어서, 이 에일리어스들을 이용하여 테이블의 칼럼들에 대한 참조가 모호해지지 않도록 한다.
- 모든 대통령들의 행은 그 자신에 대응하게 되는데, 출력에는 이것이 보이지 않는 것이 좋다. WHERE 절의 두 번째 라인은 강제로 다른 대통령들에 대해서 비교되도록 해서 자기 자신에는 대응되지 않도록 만든다.

생일이 같은 대통령들을 찾는 쿼리도 이와 비슷하게 작성될 수 있다. 그러나 출생 연월일을 직접 비교하는 것은 다른 해에 태어난 대통령들이 누락될 수 있기 때문에 부적당하다. 그 대신, MONTH()와 DAYOFMONTH() 함수를 사용하여 출생 연월일의 달과 날짜를 비교한다.

```
mysql> SELECT p1.last_name, p1.first_name, p1.birth
    -> FROM president AS p1 INNER JOIN president AS p2
    -> WHERE MONTH(p1.birth) = MONTH(p2.birth)
    -> AND DAYOFMONTH(p1.birth) = DAYOFMONTH(p2.birth)
    -> AND (p1.last_name <> p2.last_name OR p1.first_name <> p2.first_name)
    -> ORDER BY p1.last_name;
+------------+------------+------------+
| last_name  | first_name | birth      |
+------------+------------+------------+
| Harding    | Warren G.  | 1865-11-02 |
| Polk       | James K.   | 1795-11-02 |
+------------+------------+------------+
```

MONTH()와 DAYOFMONTH()를 결합해서 사용하지 않고 DAYOFYEAR()를 사용하면 더욱 단순한 쿼리로 만들 수 있지만, 윤년인 해의 날짜와 그렇지 않은 해의 날짜를 비교하게 되면 결과가 잘못될 수 있다.

서브쿼리를 사용하는 또 다른 다중 테이블 검색 방법은 중첩된 SELECT를 사용하는 것이다. 2.9절의 "서브쿼리를 사용한 복수 테이블의 검색 수행하기"에서 여러 가지 형태의 서브쿼리를 살펴볼 수 있다. 여기서는 두 개의 예제가 제공될 것이다. 결석하지 않은 학생을 찾는 경우를 가정할 때, 이는 아래에서 나타낸 바와 같이 결석 테이블에 나타나지 않는 학생을 찾는 것과 같다.

```
mysql> SELECT * FROM student
    -> WHERE student_id NOT IN (SELECT student_id FROM absence);
+-----------+-----+------------+
| name      | sex | student_id |
```

```
+-----------+-----+------------+
| Megan     | F   |          1 |
| Joseph    | M   |          2 |
| Katie     | F   |          4 |
| Nathan    | M   |          6 |
| Liesl     | F   |          7 |
...
```

중첩 SELECT는 absence 테이블에 존재하는 student_id 값을 얻을 수 있고, 외부 SELECT는 ID와 매칭되지 않는 student 행들을 검색할 수 있다.

서브쿼리는 1.4.9절의 하위절 "사용자 정의 변수들을 설정하고 사용하기"에서 제기된 Andrew Jackson 이전에 태어난 대통령에 대한 질문에 대한 단일 문장을 사용한 해결 방안을 제공한다. 원래의 해결 방안은 두 개의 문장과 사용자 변수를 사용하지만, 다음과 같은 서브쿼리에 의해서 해결될 수 있다.

```
mysql> SELECT last_name, first_name, birth FROM president
    -> WHERE birth < (SELECT birth FROM president
    -> WHERE last_name = 'Jackson' AND first_name = 'Andrew');
+------------+------------+------------+
| last_name  | first_name | birth      |
+------------+------------+------------+
| Washington | George     | 1732-02-22 |
| Adams      | John       | 1735-10-30 |
| Jefferson  | Thomas     | 1743-04-13 |
| Madison    | James      | 1751-03-16 |
| Monroe     | James      | 1758-04-28 |
+------------+------------+------------+
```

내부 SELECT를 통해서 Andrew Jackson의 생일이 언제인지를 알 수 있고, 외부 SELECT를 사용하여 그보다 먼저 태어난 대통령이 누구인지 알 수 있다.

1.4.10 기존의 행들을 삭제하거나 갱신하기

이따금씩 행을 없애거나 내용을 수정하고 싶을 때가 있다. 이러한 작업에는 DELETE와 UPDATE 문이 사용된다. 이 절에서는 이 문장들을 사용하는 방법에 대해 설명한다.

DELETE 문은 다음과 같은 형식을 갖는다.

```
DELETE FROM tbl_name
WHERE which rows to delete;
```

WHERE 절은 선택적으로 사용되는데, 삭제되어야 하는 행들을 지정한다. 그러나 이 부분을 비워두면 테이블 내의 모든 행이 삭제된다. 다시 말하면, 가장 단순한 DELETE 문은 가장 위험한 문장이 되는 것이다.

```
DELETE FROM tbl_name;
```

이 쿼리는 테이블의 모든 내용을 깨끗이 지워버리기 때문에 매우 조심해서 다루어야 한다! 특정한 행만 삭제하려면, WHERE 절을 사용해서 원하는 행만 선택하면 된다. 이것은 마치 SELECT 문에서 WHERE 절

을 사용해서 전체 테이블을 선택하지 않는 것과 비슷하다. 예를 들면, president 테이블에서 오하이오(Ohio)에서 출생한 대통령들만 선택적으로 삭제하려면, 다음과 같은 쿼리를 사용한다.

```
mysql> DELETE FROM president WHERE state='OH';
Query OK, 7 rows affected (0.00 sec)
```

DELETE 문이 그러한 조건의 레코드들을 확실하게 삭제하는지 확신이 서지 않는다면, 시험삼아 WHERE 절을 SELECT 문에 먼저 적용시켜서 대응되는 행들을 찾아보는 것도 좋은 생각이다. 이러한 작업으로 의도하는 행들을 실제로 그러한 행들만 삭제하는 데 도움을 받을 수 있다. Teddy Roosevelt에 대한 행만 지우고 싶다고 해보자. 다음과 같은 쿼리로 이러한 작업을 할 수 있겠는가?

```
DELETE FROM president WHERE last_name='Roosevelt';
```

마음먹고 있던 행을 지운다는 점에서는 맞다. 그러나 Franklin Roosevelt에 대한 행도 지워진다는 점에서 보면 틀리다. 다음과 같이, SELECT 문에 WHERE 절을 먼저 적용시켜서 점검하면 좀더 안전해진다.

```
mysql> SELECT last_name, first_name FROM president
    -> WHERE last_name='Roosevelt';
+-----------+-------------+
| last_name | first_name  |
+-----------+-------------+
| Roosevelt | Theodore    |
| Roosevelt | Franklin D. |
+-----------+-------------+
```

이름에 대한 조건을 더 추가함으로써 좀 더 상세하게 해야겠다는 생각이 들게 된다.

```
mysql> SELECT last_name, first_name FROM president
    -> WHERE last_name='Roosevelt' AND first_name='Theodore';
+-----------+------------+
| last_name | first_name |
+-----------+------------+
| Roosevelt | Theodore   |
+-----------+------------+
```

이제 원하는 행만 선택하는 적절한 WHERE 절을 알게 되었으므로 DELETE 쿼리를 올바르게 구성할 수 있게 되었다.

```
mysql> DELETE FROM president
    -> WHERE last_name='Roosevelt' AND first_name='Theodore';
```

행을 삭제하는 작업이 좀 번거로운 듯이 보이지만, 이를 반드시 기억해 두자. 나중에 미안한 마음을 가지는 것보다는 차라리 안전한 것이 낫지 않을까? 하지만, 역시 이를 반드시 기억해 두자. 여기서 번거로움을 줄이려면, 복사해서 붙이는 방법이나 입력 라인 편집 기술 등을 통해서 타이핑 횟수를 최소화시키면 될 것이다. 좀 더 자세한 내용은 1.5절 "mysql로 대화식 작업할 때의 요령"을 보기 바란다.

기존의 행들을 수정하려면 UPDATE를 사용하는데, 이것은 다음과 같은 형식을 가진다.

```
UPDATE tbl_name
```

```
SET which columns to change
WHERE which records to update
```

WHERE 절은 DELETE에 대한 것과 똑같이 사용한다. 이것은 없어도 되므로, 만일 지정하지 않는다면 테이블 내의 모든 행이 수정된다. 예를 들면, 다음과 같은 쿼리는 모든 학생들의 이름을 "George"로 바꾸어 놓는다.

```
mysql> UPDATE student SET name='George';
```

이와 같은 분명한 쿼리들은 아주 조심스럽게 취급해야 하므로, 보통 WHERE 절을 추가해서 갱신하는 행들에 관해 더욱 상세하게 지정해서 사용한다. 최근에 Historical League에 새로운 회원을 추가했지만, 회원 정보 항목은 몇 가지만 채웠다고 가정해보자.

```
mysql> INSERT INTO member (last_name,first_name)
    -> VALUES('York','Jerome');
```

그러면, 회원 기한 만료 날짜를 설정하는 것을 잊었다는 것을 알게 된다. 적절한 WHERE 절이 포함된 UPDATE 문을 가지고 어떤 행을 변경할 것인지를 지정해서 고칠 수 있다.

```
mysql> UPDATE member
    -> SET expiration='2009-7-20'
    -> WHERE last_name='York' AND first_name='Jerome';
```

단일 문장으로 복구의 칼럼을 갱신할 수도 있다. 다음의 UPDATE는 Jerome의 전자 메일과 우편 주소를 수정한다.

```
mysql> UPDATE member
    -> SET email='jeromey@aol.com', street='123 Elm St',
    -> city='Anytown', state='NY', zip='01003'
    -> WHERE last_name='York' AND first_name='Jerome';
```

칼럼에 NULL 값으로 설정하면 칼럼을 초기화시킬 수 있다(해당 칼럼에 NULL 값이 허용된다고 가정함). 미래의 어느 시점에 Jerome이 평생회원으로 회원 등급을 올리기로 결정한다면, Jerome의 기록에서 기한 만료 날짜를 NULL("만료되지 않음")로 설정할 수가 있다.

```
mysql> UPDATE member
    -> SET expiration=NULL
    -> WHERE last_name='York' AND first_name='Jerome'
```

DELETE에서와 같이, UPDATE 문에 대해서도 SELECT 문을 사용하여 WHERE 절을 테스트하는 방법으로 올바른 행을 선택해서 갱신하는지 확인하는 것은 좋은 생각이다. 선택 영역이 너무 좁거나 넓으면, 너무 적은 또는 너무 많은 행을 갱신하게 된다.

이 절에 있는 쿼리들을 실행해보면, sampdb 테이블 안에 있는 행들을 삭제하고 수정하게 된다. 다음 절로 진행하기에 앞서, 이들 변경사항들을 되돌려 놓도록 한다. 1.4.8절 "sampdb 데이터베이스의 리셋"의 끝부분에 있는 지침을 사용하여 테이블들을 다시 로딩하기 바란다.

1.5 mysql로 대화식 작업할 때의 요령

이 절에서는 mysql 클라이언트 프로그램을 보다 적은 타이핑 수로 좀더 효과적으로 사용하는 방법을 생각해보기로 한다. 또한 좀 더 쉽게 서버에 연결하는 방법, 손수 일일이 타이핑하지 않고 쿼리를 입력하는 방법을 설명한다.

1.5.1 연결 과정을 간단히 하기

mysql을 호출할 때 지정하는 연결 인자로는 호스트 이름, 사용자 이름, 패스워드 등이 쓰이게 된다. 단지 프로그램을 실행시키는 작업임에도 타이핑 수가 많고 이로 인해 일찍 따분하게 되어버린다. MySQL 서버에 연결을 수립하는 데 필수적인 타이핑의 양을 줄여주는 몇 가지 방법이 있다.

- 옵션 파일 안에 연결 인자를 저장한다.
- 쉘의 명령 히스토리 기능의 장점을 이용하여 명령어를 반복한다.
- 쉘 에일리어스 또는 스크립트를 사용하여 mysql 커맨드 라인을 짧게 정의한다.

옵션 파일 사용하기

MySQL은 옵션 파일 안에 연결 인자들을 저장할 수 있게 되었다. 그러므로 mysql을 실행할 때마다 커맨드 라인상에서 입력할 때 방금 사용한 인자들을 타이핑할 필요가 없다. 이러한 기법의 커다란 장점은 이렇게 저장된 인자들을 mysqlimport 또는 mysqlshow와 같은 다른 MySQL 클라이언트들도 사용하게 된다는 것에 있다. 다시 말하면, 옵션 파일은 mysql뿐만 아니라 다른 많은 프로그램들에도 마찬가지로 쉽게 사용되도록 해준다는 것이다. 이 절에서는 간략하게 클라이언트 프로그램에 의해서 사용되는 옵션 파일을 어떻게 셋업하는지를 나타낸다. F.2.2절 "옵션 파일들"에서는 더 자세한 사항들에 대해서 나타내도록 한다.

UNIX에서는 ~/.my.cnf라는 이름의 파일을 만들어서 옵션 파일을 설정한다(이것은 홈 디렉터리 안에 .my.cnf라는 이름의 파일을 뜻한다). Windows에서는 C 드라이브의 루트 디렉터리(C:\my.ini) 안 또는 MySQL 설치 디렉터리에 my.ini라는 이름으로 옵션 파일을 만든다. 옵션 파일은 평이한 텍스트 파일이므로, 텍스트 에디터이면 아무거나 사용해서 만들 수 있다. 파일의 내용은 다음과 같이 만들면 된다.

```
[client]
host=server_host
user=your_name
password=your_pass
```

[client] 라인은 client 옵션 그룹의 시작임을 알린다. 이 뒤에 오는 라인들은 파일의 마지막이나 다른 옵션 그룹의 시작이 나올 때까지 MySQL의 클라이언트 프로그램들이 읽어들인다. *server_host*, *your_name*, 그리고 *your_pass* 자리에 서버에 연결할 때 지정하는 호스트 이름, 사용자 이름, 그리고 패스워드를 대신 써주면 된다. 예를 들면, 호스트 *cobra.snake.net*에서 서버가 돌아가고, MySQL 사용자 이름, 패스워드가 sampadm, secret이면, .my.cnf파일에 다음과 같이 쓸 수 있다.

```
[client]
host=cobra.snake.net
user=sampadm
password=secret
```

[client] 라인은 이 옵션 그룹이 시작한다는 것을 정의하기 위해 꼭 필요하지만, 인자 값들을 정의하는 라인들은 선택적이므로 있어도 되고 없어도 된다. 단지 필요한 것만 지정해줄 수 있다. 예를 들면, UNIX를 사용하고 MySQL의 사용자 이름이 UNIX의 로그인 이름과 동일한 경우에는 user 라인을 포함시킬 필요가 없다. 만약 로컬 호스트에서 서버가 동작하고 필요한 host 라인이 없다면, 기본 호스트는 localhost가 된다.

옵션 파일을 만든 다음, 부가적으로 조심해야 할 것은 UNIX에서 아무도 이 옵션 파일을 읽거나 수정할 수 없도록 파일의 접근 모드를 제한적인 값으로 설정해야만 한다는 것이다. 다음의 명령 중 한 가지를 사용해서 파일을 자신만 접근할 수 있도록 만든다.

```
% chmod 600 .my.cnf
% chmod u=rw,go-rwx .my.cnf
```

쉘의 명령 히스토리 기능을 사용하기

tcsh와 bash와 같은 쉘들은 히스토리 목록에 사용한 명령들을 기억시켜 놓고, 이 목록에서 명령들을 반복해서 사용할 수 있도록 해준다. 이러한 쉘들 중 한 가지를 사용하는 경우라면, 히스토리 목록을 사용해서 전체 명령문을 전부 다 타이핑하지 않을 수 있다. 예를 들면, 최근에 mysql을 호출했다면, 다음과 같이 이것을 다시 실행할 수 있다.

```
% !my
```

'!' 문자는 쉘에게 명령 히스토리에서 "my"로 시작하는 가장 최근의 명령을 찾아서 그것을 직접 타이핑해서 입력한 것처럼 수행하라고 지시하는 것이다. 어떤 쉘들은 아래 위 화살표 키를 가지고 히스토리 목록에서 위 아래로 선택할 수 있게 되어 있다(또는 Ctrl-P와 Ctrl-N). 이런 방법으로 원하는 명령을 고른 다음에 엔터키를 눌러서 그것을 실행시키게 된다. tcsh와 bash는 이러한 기능을 가지고 있고, 다른 쉘들도 마찬가지로 이러한 기능을 가지고 있을 것이다. 히스토리 목록을 사용하는 방법에 대해서 더 많은 것을 알고 싶으면, 쉘에 대한 문서를 살펴보도록 한다.

쉘 에일리어스와 스크립트를 사용하기

쉘이 에일리어스 기능을 지원한다면, 긴 명령어에 대응되는 짧은 명령어 이름을 설정할 수 있다. 예를 들면, csh과 tcsh에서, alias 명령어를 사용해서 sampdb로 명명된 에일리어스를 다음과 같이 설정할 수 있다.

```
alias sampdb 'mysql -h cobra.snake.net -p -u sampadm sampdb'
```

bash에 대한 문법은 약간 다르다. 즉, 다음과 같다.

```
alias sampdb='mysql -h cobra.snake.net -p -u sampadm sampdb'
```

에일리어스를 정의하게 되면, 다음의 두 명령어는 동일한 것이 된다.

```
% sampdb
% mysql -h cobra.snake.net -p -u sampadm sampdb
```

분명히, 두 번째 것보다 첫 번째 것을 타이핑하는 것이 더 쉽다. 매번 로그인할 때마다 이 에일리어스가 효과를 가지게 하려면, 쉘의 시작 파일 중 하나에 에일리어스 명령을 포함하도록 한다(예를 들면, tcsh에 대해서는 .tcshrc이거나, bash에 대해서는 .bashrc 또는 .bash_profile가 해당된다).

Windows에서는 비슷한 기법으로 mysql 프로그램을 가리키는 바로가기(shortcut)를 만드는 것이 있다. 바로가기의 속성을 편집하여 적절한 연결 인자를 포함시키면 된다.

타이핑을 적게 하면서 명령어들을 호출하는 또 다른 방법으로는 스크립트를 만들어서 적절한 옵션으로 mysql을 실행하게 만드는 것이 있다. UNIX에서는 sampdb 에일리어스에 해당하는 쉘 스크립트는 다음과 같이 작성할 수 있다.

```
#!/bin/sh
exec mysql -h cobra.snake.net -p -u sampadm sampdb
```

이 스크립트의 이름을 sampdb로 하고 실행 가능하도록(chmod +x sampdb 명령을 사용) 만든다면, 명령 프롬프트상에서 sampdb라고 타이핑해서 mysql를 실행해서 데이터베이스에 연결할 수 있다.

Windows에서는 배치(batch) 파일을 이용해서 위와 동일한 일을 할 수 있다. 파일 이름을 smapdb.bat라고 하고, 다음과 같은 라인을 파일 안에 써 넣는다.

```
mysql -h cobra.snake.net -p -u sampadm sampdb
```

이 배치 파일은 DOS 콘솔 창의 프롬프트에서 sampdb라고 타이핑하거나Windows 아이콘으로 만들어서 더블클릭을 해서 실행시키면 된다.

복수의 데이터베이스에 접근하거나 복수의 호스트에 연결하는 경우에는 에일리어스나 바로가기(shortcut), 또는 스크립트를 그 수에 맞추어서 정의해 놓으면 되는데, 이들은 모두 각기 다른 옵션으로 mysql을 호출하도록 만들어 놓는다.

1.5.2 쿼리를 보낼 때 타이핑 횟수를 줄이기

mysql은 데이터베이스와 대화식으로 작업하게 해주는 아주 유용한 프로그램이지만, 이것의 인터페이스는 짧고, 한 줄로 된 쿼리에 가장 알맞게 되어 있다. mysql 자체로는 쿼리가 여러 줄에 걸쳐서 길게 되던지 상관하지 않는 것이 사실이지만, 긴 쿼리들을 타이핑해서 써넣기가 그다지 즐겁지는 않다. 짧은 쿼리라 할지라도 문법 에러가 있어서 다시 타이핑해 넣어야 할 때도 역시 재밌지는 않은 상황이다. 불필요한 타이핑과 다시 타이핑하는 일을 줄여주는 데 유용한 몇 가지 기법이 있다.

- mysql의 라인 입력 편집 기능을 사용한다.
- 복사하기와 붙이기 기능을 사용한다.
- mysql을 배치(batch) 모드에서 실행한다.

mysql의 라인 입력 편집기를 사용하기

mysql은 라인 입력 편집을 허용하도록 해주는 GNU Readline 라이브러리를 내장하고 있다. 현재 입력 중인 라인을 다루거나, 지금 혹은 나중에 수정할 때처럼 이전에 입력된 라인들을 불러와서 다시 입력할 수 있도록 해준다. 이 기능은 어떤 라인을 입력하다가 오타가 났을 때 사용하면 편리하다. 해당 라인 안에 백업을 해서 엔터키를 누르기 전에 문제를 바로잡을 수 있다. 잘못 작성된 쿼리를 입력했다면, 그 쿼리를 다시 불러와서 이 쿼리를 편집해 문제를 고치고, 그 다음에 다시 이것을 보낼 수 있다(전체 쿼리를 한 라인에 타이핑한 경우가 가장 쉽게 된다).

편집 시퀀스 중에서 쓸만하게 보이는 것 몇 가지를 [표 1.4]에 나타내었다. 하지만, 테이블에 나타나 있지 않은 입력 편집 명령어들도 많이 있다. GNU 프로젝트 웹 사이트에 있는 http://www.gnu.org/manual/에서 온라인으로 제공하는 bash 매뉴얼의 명령 편집(command editing)에 관한 장에서 이러한 명령어들에 관해 읽을 수 있다.

[표 1.4] mysql의 입력 편집 명령어

키 시퀀스	의미
Up 화살표 또는 Ctrl-P	이전 라인을 다시 불러옴
Down 화살표 또는 Ctrl-N	다음 라인을 다시 불러옴
Left 화살표 또는 Ctrl-B	커서를 왼쪽으로 옮김(뒤쪽)
Right 화살표 또는 Ctrl-F	커서를 오른쪽으로 옮김(앞쪽)
Esc b	한 단어 뒤로 옮김
Esc f	한 단어 앞으로 옮김
Ctrl-A	커서를 라인의 시작으로 옮김
Ctrl-E	커서를 라인의 끝으로 옮김
Ctrl-D	커서가 위치한 곳의 문자를 삭제
Del	커서의 왼쪽 문자를 삭제
Esc D	커서 위치부터 그 단어의 마지막까지 삭제
Esc Backspace	백스페이스 커서 위치부터 그 단어의 시작까지 삭제
Ctrl-K	커서 위치부터 그 줄의 마지막까지 삭제
Ctrl-_	마지막 변경사항을 되돌림, 반복 실행 가능

Windows에서는 입력 라인의 편집이 불가능하기 때문에, [표 1.5]에서 나타낸 바와 같은 명령어를 지원하며, mysql에 적용할 수 있다.

[표 1.5] Windows의 입력 편집 명령어

키 시퀀스	의미
Up 화살표	이전 라인을 다시 불러옴
Down 화살표	다음 라인을 다시 불러옴
Left 화살표	커서를 왼쪽으로 옮김(뒤쪽)
Right 화살표	커서를 오른쪽으로 옮김(앞쪽)
Control + Left 화살표	한 단어 뒤로 옮김
Control + Right 화살표	한 단어 앞으로 옮김
Home	커서를 라인의 시작으로 옮김
End	커서를 라인의 끝으로 옮김
Delete	커서가 위치한 곳의 문자를 삭제
Backspace	커서의 왼쪽 문자를 삭제
Esc	현재 라인 전체를 삭제
Page up	입력되었던 첫 번째 명령어를 호출
Page down	입력되었던 마지막 명령어를 호출
F3	입력되었던 마지막 명령어를 호출
F7	명령어 팝업 표시. Up 화살표, Down 화살표로 선택
F9	명령어 팝업 표시. 명령어 번호로 선택
F8, F5	명령어 리스트를 보여줌

다음의 예에서 입력 편집의 사용에 관해 간단하게 설명한다. mysql을 사용하면서 다음과 같은 쿼리를 입력했다고 가정한다.

```
mysql> SHOW COLUMNS FROM persident;
```

엔터키를 누르기 전에 "president"를 "persident"라고 잘못 적은 것을 알아챘다면, 다음과 같은 방법으로 쿼리를 수정할 수 있다.

1. 왼쪽 화살키나 Ctrl-B를 몇 번 눌러서 커서를 persident의 "s" 위에 올 때까지 옮긴다.

2. "er"을 지우기 위해서 Delete를 누르거나, 백스페이스를 두 번 누른다(시스템에서 커서의 왼쪽으로 문자가 지워지는 경우에 해당한다).

3. 오류를 수정하기 위해서 "re"를 타이핑한다.

4. 엔터를 눌러서 쿼리를 보낸다.

잘못 적은 것을 인지하기 전에 엔터를 눌렀어도, 별 문제가 안 된다. mysql이 에러 메시지를 표시한 이후, 그 라인을 다시 부르기 위해서 위쪽 화살키를 누르고, 그런 다음 방금 설명한 대로 편집한다.

쿼리를 보낼 때 복사해서 붙이기 방법을 사용하기

윈도우가 지원되는 환경에서 작업한다면, 쓸만하다고 생각되는 쿼리들의 텍스트를 파일 안에 저장해 복사해서 붙이기 작업으로 다시 불러올 수 있다.

1. 터미널 윈도우에서 mysql을 호출한다.

2. 도큐멘트 윈도우 안에서 쿼리가 들어 있는 파일을 연다(예를 들면, UNIX에서는 vi, Windows에서는 gvim를 사용한다).

3. 파일 안에 저장된 쿼리를 실행시키려면, 해당 도큐멘트 안에서 실행시키고자 하는 쿼리를 선택해서 복사한다. 그 다음에 터미널 윈도우로 전환해서 mysql 안으로 쿼리를 붙이기 기능으로 붙인다.

이러한 과정을 위와 같이 적어보면 귀찮은 작업으로 보일지 모르겠지만, 실제로 그대로 해보면, 타이핑하지 않고 빠르게 쿼리를 입력하는 방법이 되는 것을 알 수 있을 것이다.

또한 다른 방향으로 복사해서 붙이기 방법을 사용할 수도 있다(터미널 윈도우에서 쿼리 파일로). Unix의 mysql에서 여러 라인들을 입력할 때, 이들은 홈 디렉터리의 .mysql_history라는 이름의 파일 안에 저장된다. 나중에 참조하기 위해 저장해 두고 싶은 쿼리를 손으로 입력하는 경우, mysql을 종료시키고, 에디터에서 .mysql_history를 호출한 이후에 .mysql_history에 있는 쿼리를 자신의 쿼리 파일 안으로 복사해서 붙인다.

스크립트 파일들을 실행하기 위한 mysql 사용 방법

mysql을 항상 대화식으로만 실행시킬 필요는 없다. mysql은 비대화식인 배치 모드로 파일에 있는 내용을 읽어서 입력받을 수 있다. 주기적으로 실행하는 쿼리들은 실행시킬 때마다 매번 같은 쿼리를 다시 타이핑하는 것은 번거롭기 때문에 이러한 경우에는 배치 모드가 유용하다. 파일에 쿼리를 한 번 써 넣은 다음 필요할 때마다 그 파일의 내용을 실행하도록 만드는 것은 쉬운 일이다.

미국 역사의 특정한 영역에 관심이 있는 역사 연구회의 회원을 찾기 위해서 member 테이블의 interests 칼럼을 찾아보고자 하는 쿼리가 있다고 해보자. 예를 들면, 대공황(Great Depression)에 관심이 있는 회원을 찾으려면, 쿼리는 다음과 같이 작성될 수 있다.

```
SELECT last_name, first_name, email, interests FROM member
WHERE interests LIKE '%depression%'
ORDER BY last_name, first_name;
```

interest.sql 파일 안에 쿼리를 써놓고 난 다음, mysql에 다음과 같이 적용시켜서 이 쿼리를 실행한다.

```
% mysql sampdb < interests.sql
```

기본적으로, mysql은 배치 모드로 동작할 때 탭 문자로 구분된 포맷으로 출력을 만들어낸다. mysql을 대화식으로 실행할 때 동일한 종류의 표 형식의 출력을 원한다면, -t 옵션을 사용한다.

```
% mysql -t sampdb < interests.sql
```

출력이 저장되게 하려면, 이것을 파일로 리디렉션시킨다.

```
% mysql -t sampdb < interests.sql > interests.out
```

만약에 이미 mysql을 실행시켰다면, source 명령을 사용하여 파일 내용을 실행시킬 수 있다.

```
mysql> source interests.sql
```

이 쿼리를 사용해서 Thomas Jefferson에 관심이 있는 회원을 찾으려면, 이 쿼리 파일을 편집해서 depression을 Jefferson으로 바꾸고 나서 mysql을 다시 실행시키면 된다. 이러한 쿼리를 매우 자주 사용하지만 않는다면 이것은 잘 작동될 것이다. 자주 사용하는 경우에는 더 좋은 방법이 필요하다. 이러한 쿼리를 더욱 유연하게 만드는 방법 중 한 가지는 스크립트의 커맨드 라인에서 인자를 취하도록 쉘 스크립트를 만들어서 이 스크립트를 사용하여 쿼리의 텍스트를 변경할 수 있도록 하는 것이다. 이 방법은 스크립트를 실행시킬 때 원하는 interests 값을 지정할 수 있도록 쿼리를 인자화시키는 것이다. 이것이 어떻게 작동하는지를 보기 위해, 작은 쉘 스크립트인 interests.sh를 작성해보자.

```
#!/bin/sh
# interests.sh - 특정한 관심사항을 가진 USHL 회원을 찾는다.
if [ $# -ne 1 ]; then echo 'Please specify one keyword'; exit; fi
mysql -t sampdb <<QUERY_INPUT
SELECT last_name, first_name, email, interests FROM member
WHERE interests LIKE '%$1%'
ORDER BY last_name, first_name;
QUERY_INPUT
```

위 스크립트에서 세 번째 라인은 커맨드 라인상에서 하나의 인자가 있는지 확인하는 일을 한다. 인자가 없으면 간략한 메시지를 출력하고 프로그램을 끝낸다. <<QUERY_INPUT과 마지막에 있는 QUERY_INPUT 라인 사이에 있는 모든 것들은 mysql로 입력되게 된다. 쿼리 문장 내에서, 이 쉘은 $1에 대한 참조를 커맨드 라인에서 지정된 인자로 교체하게 된다. (쉘 스크립트에서는 $1, $2 등으로 명령 인자를 참조하게 된다.) 이렇게 해서 스크립트를 실행할 때 커맨드 라인상에서 지정한 어떠한 키워드라도 쿼리에 반영되는 것이다.

이 스크립트를 실행시키기 전에, 이것을 실행 가능하도록 만들어야 한다.

```
% chmod +x interests.sh
```

이제, 스크립트를 실행시킬 때마다 편집할 필요가 없게 되었다. 단지 커맨드 라인상에서 무슨 키워드를 찾을 것인지만 지정해주면 된다.

NOTE

이와 같은 방법을 사용하는 경우에는 인자에서 보안을 체크할 수 없고, SQL 문을 통한 공격이 이루어질 수 있기 때문에 스크립트를 이와 같은 방법으로 사용하지 않기를 권고한다. 어떤 사람이 다음과 같은 스크립트를 사용한다고 가정해보자.

```
% ./interests.sh "Jefferson';DROP DATABASE sampdb;"
```

이 스크립트를 사용함으로써 DROP DATABASE 문이 mysql으로 입력되어 실행될 수 있게 된다.

```
% ./interests.sh depression
% interests.sh Jefferson
```

interests.sh 스크립트는 sampdb 배포본의 misc 디렉터리에서 찾을 수 있다. 이와 같은 일을 하는 Windows의 배치(batch) 파일은 interests.bat이고 마찬가지로 같은 디렉터리에 있다.

1.6 이제 이 책의 어느 부분을 읽어갈 것인가?

이제 MySQL에 대해 약간 알 수 있게 되었다. 데이터베이스를 셋업하고 테이블을 생성할 수 있게 된 것이다. 테이블에 행들을 추가할 수 있고, 다양한 방법으로 그것들을 검색하며 변화시키고 지울 수 있다. 이 장에서 나타내고자 한 것은 전반적인 내용을 습득한 것이지만, MySQL에 대해서는 알아야 할 것들이 아직도 많이 남아 있다. 현재 사용하고 있는 sampdb 데이터베이스의 현재 상태에 의해서도 이를 알 수 있다. 이 데이터베이스와 테이블을 생성하고 초기 데이터를 추가하였다. 이러한 과정 중에 데이터베이스의 정보에 대한 질문에 답변할 수 있는 쿼리들이 어떻게 쓰여지는지 알 수 있었지만, 아직도 많은 것들이 남아 있다. 예를 들면, 성적 기록 프로젝트에서 새로운 점수 행들을 넣거나, 역사 연구회의 디렉터리에 새로운 멤버를 넣기 위한 대화식의 편리한 방법이 아직은 없다. 현재의 행에 대한 편리한 편집 방법도 없다. 이와 같은 일들에 대해서 그리고 그밖의 일들에 대해서는 앞으로 언급될 장들, 특히 8장의 "Perl DBI를 사용하여 MySQL 프로그램 작성하기"와 9장의 "PHP를 사용하여 MySQL 프로그램 작성하기"에서 나타낼 것이다.

관심사에 의해서 이 책의 원하는 곳으로 가면 된다. 만약 역사 연구회와 성적 기록부 프로젝트를 끝내고 싶다면, Part 2의 MySQL 기반 프로그램 만드는 방법으로 보면 된다. 만약 사이트에 대한 MySQL 관리자를 수행해야 한다면, Part 3에서 관리와 관련된 일에 대해서 살펴보면 된다. 그러나 필자는 Part 1의 나머지 장들을 읽고, 먼저 MySQL을 사용함에 있어서 일반적인 백그라운드 지식을 추가적으로 얻기를 추천한다. 이러한 장들은 SQL 문의 사용과 문법에 대한 추가적인 정보를 제공하고, 어떻게 MySQL이 데이터를 다루는지 나타내며, 어떻게 쿼리를 더 빠르게 실행할 수 있는지를 보여준다. 이러한 주제들에 대해서 좋은 배경을 갖추게 되면, mysql을 실행할 때나 자신만의 프로그램을 작성할 때, 또는 데이터베이스 관리자로서 작업을 할 때, 어느 때이든지 자신 있게 MySQL을 사용할 수 있는 자신을 발견하게 될 것이다.