O'Reilly Media, Inc. 介绍

为了满足读者对网络和软件技术知识的迫切需求，世界著名计算机图书出版机构O'Reilly Media, Inc.授权东南大学出版社，翻译出版一批该公司久负盛名的英文经典技术专著。

O'Reilly Media, Inc.是世界上在 Unix、X、Internet 和其他开放系统图书领域具有领导地位的出版公司，同时也是联机出版的先锋。

从最畅销的《The Whole Internet User's Guide & Catalog》（被纽约公共图书馆评为20世纪最重要的50本书之一）到GNN（最早的Internet门户和商业网站），再到WebSite（第一个桌面PC的Web服务器软件），O'Reilly Media, Inc.一直处于Internet发展的最前沿。

许多书店的反馈表明，O'Reilly Media, Inc.是最稳定的计算机图书出版商 —— 每一本书都一版再版。与大多数计算机图书出版商相比，O'Reilly Media, Inc.具有深厚的计算机专业背景，这使得O'Reilly Media, Inc.形成了一个与其他出版商迥然不同的出版方针。O'Reilly Media, Inc.所有的编辑人员以前都是程序员，或者是顶尖级的技术专家。O'Reilly Media, Inc.还有许多固定的作者群体 —— 他们本身是相关领域的技术专家、咨询专家，而现在编写著作，O'Reilly Media, Inc.依靠他们及时地推出图书。因为O'Reilly Media, Inc.紧密地与计算机业界联系着，所以O'Reilly Media, Inc.知道市场上真正需要什么图书。

深入浅出SQL

Lynn Beighley 著

O'Reilly Taiwan公司 编译

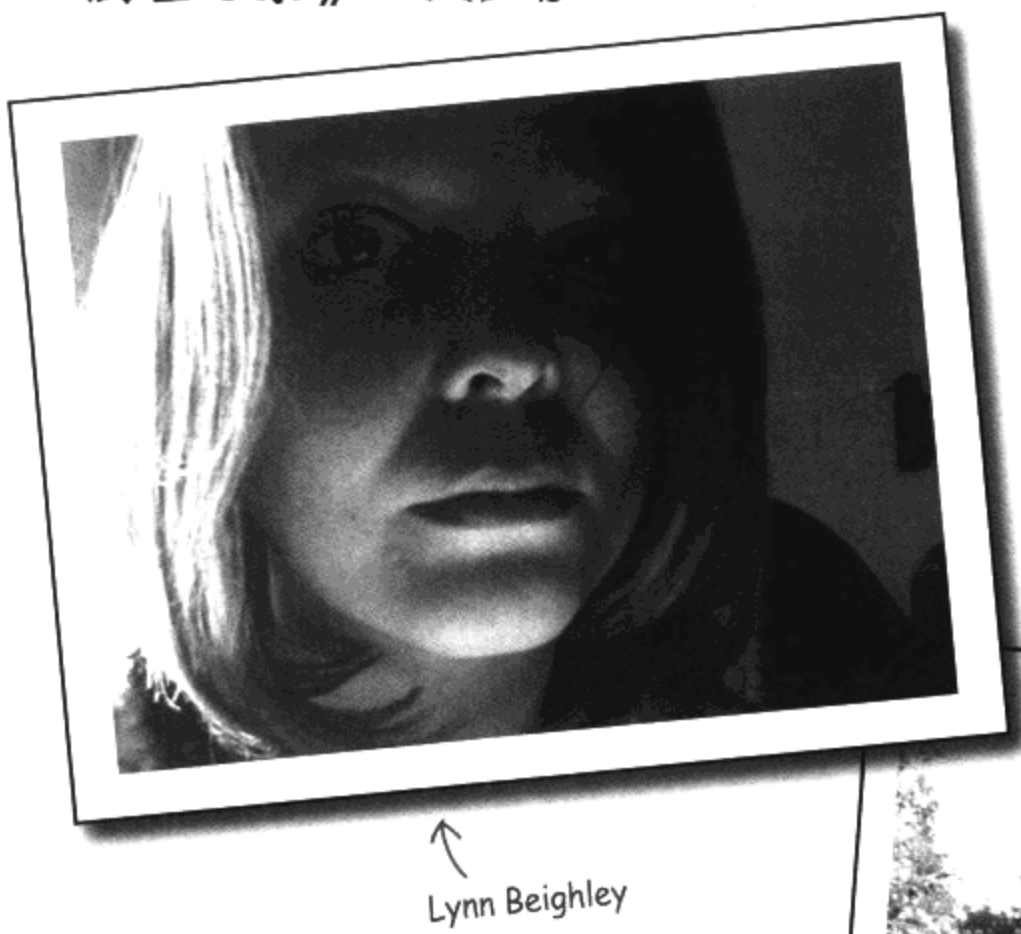
O'REILLY®

Beijing • Cambridge • Köln • Sebastopol • Taipei • Tokyo

O'Reilly Media, Inc.授权东南大学出版社出版

东南大学出版社

《深入浅出SQL》的作者



Lynn Beighley

Lynn 是位困在技术撰稿人身体里的小说家。当她发现技术书籍可以带来实际收入后，终于慢慢地学着接受并享受这方面的工作。

重返校园取得计算机科学硕士学位后，她为 NRL 和 LNAL 工作。然后她发现了 Flash 的存在，并写出她的第一本畅销书。

Lynn 选择移居硅谷的时机实在不太好，没过多久网络泡沫就发生了。接下来的几年，她在 Yahoo! 工作，同时也写了几本书并培训课程。最后她决定转到创意写作行业，因而搬到纽约，取得创意写作的MFA学位。

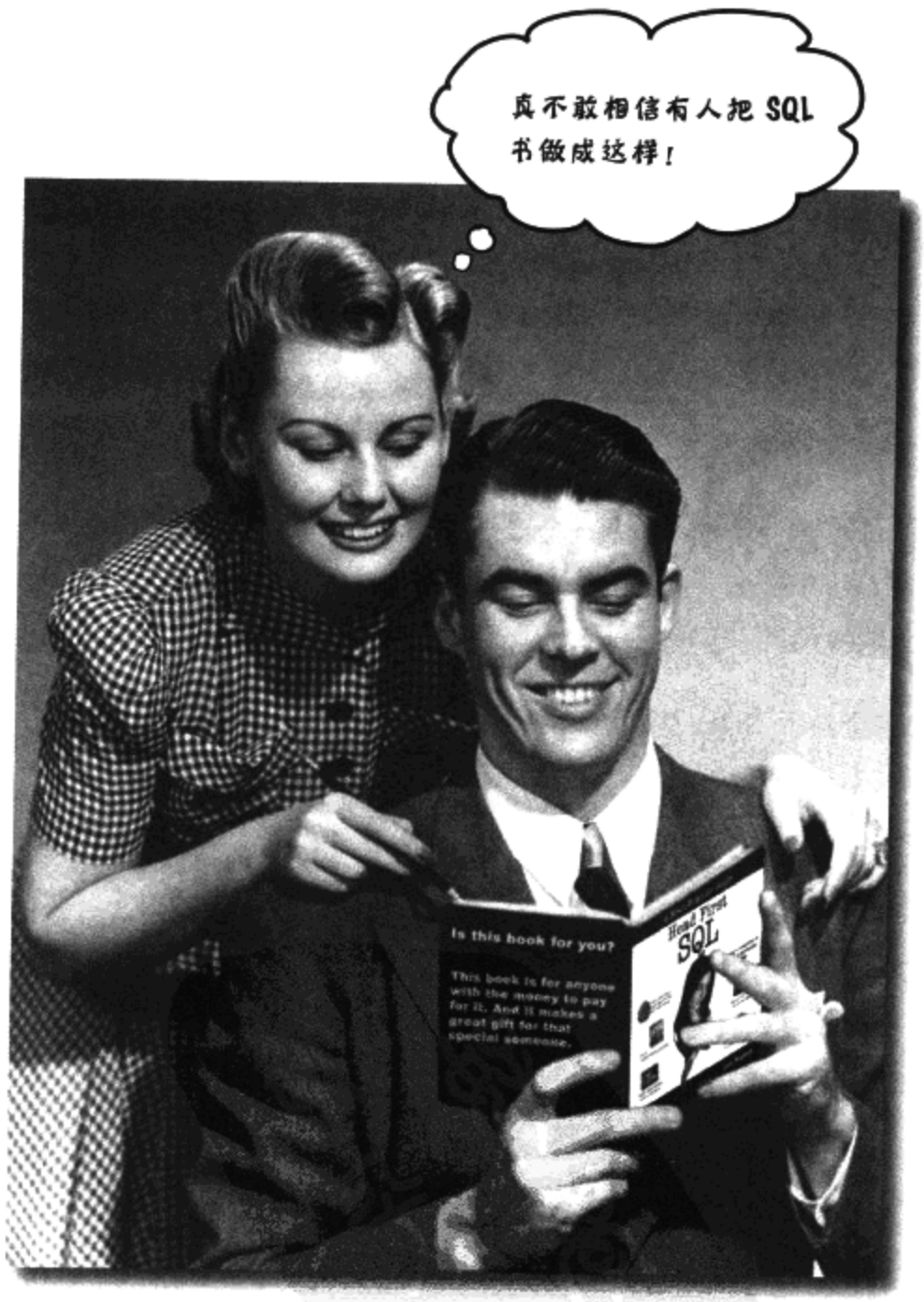
对着满屋子的教授和同学，她的 Head First 风格的论文获得了极高的评价，她拿到了学位，也完成了《深入浅出 SQL》，等不及要投入下一本书的创作中去。

Lynn 喜欢旅游、烹饪、为完全不认识的人编织详细的背景故事。对了，她还有点小害羞。

Lynn的窗外风光。

如何使用本书

序



本章回答了热门话题：

“为什么要把这些东西放进 SQL 书籍里？”

这本书适合谁？

请回答下列问题：

1. 你在安装了 RDBMS（例如 Oracle、MS SQL、MySQL）的机器上有访问权限吗？还是你有一台可以安装 MySQL 或其他 RDBMS 的机器？
2. 想要**学习、了解、牢记**如何**创建表、数据库并使用最新标准设计查询**吗？
3. 比较喜欢**刺激的晚宴对话**，而不喜欢**枯燥乏味的学术演讲**？

我们将以更容易理解及实际应用的方式协助大家学习SQL的概念和语法，差不多就是我们需要使用SQL的方式。

如果上述问题你都回答“是”，这本书就是为你而写。

谁或许应该远离这本书？

请回答下列问题：

1. 你已经**完全熟悉** SQL 基础语法，正在寻找帮助你更好设计数据库的理论吗？
2. 你已经是一位很有经验的 SQL 设计师，正在寻找 SQL 的**参考书籍**吗？
3. 你**害怕尝试不同的事物**吗？宁可接受根管治疗（抽神经）也不愿意混搭格子衫与条纹裤？你认为技术书籍若为 SQL 概念赋予人性则不够认真严肃吗？

如果你想回顾一下从前学过的SQL，或是一直都不算很了解的规范化、一对多关联、左外联接等问题，本书对你也是有帮助的。

如果上述问题你都能回答“是”，那这本书就不符合你的期望。

[销售部门补充：本书适合所有有信用卡的人。]

我们知道你在想什么。

“这怎么可能是一本正经的 SQL 书籍？”

“这一堆图是干什么的？”

“这样真能让我学到东西吗？”

我们也知道你的脑袋在想什么。

你的脑袋渴望新奇的事物，它总是在搜寻、扫描、期待着不寻常的事物。人类的大脑生来如此，正是这样的特质帮助我们常保活力，在竞争激烈的生命树上存活至今。

那么，对于那些每天都要面对的一成不变、平淡无奇的事物，你的脑袋又作何反应？它会尽量阻止这些事去干扰大脑的真正工作——记录真正重要的事。大脑不会浪费脑细胞去保存无聊的事，它们绝对无法通对“这显然不重要”的过滤器。

你的脑袋究竟怎样知道什么是重要的？假设你去郊游，突然有只老虎跳到你的眼前，你的脑袋和身体会做出怎样的反应？

神经紧绷、情绪激动、肾上腺素激增！

这就是脑袋“知道”的方式……

这绝对重要！别忘了！

但是，想象一下你在家里或图书馆。这里安全、温暖而且没有老虎出没。你正在读书、为考试做准备或者研究某个技术难题——你的老板认为需要1周，最多10天就能完成的难题。

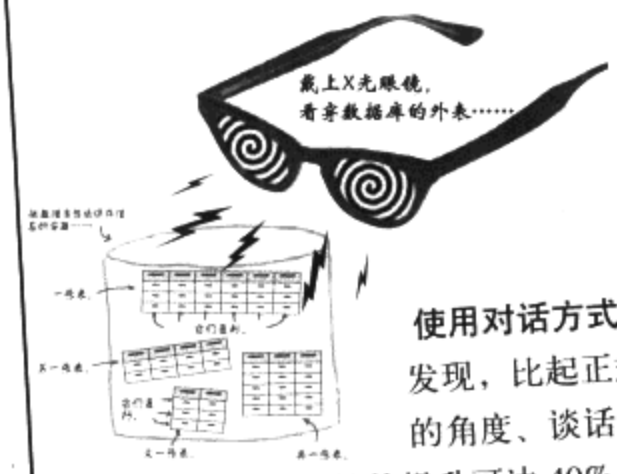
但是有个问题。你的脑袋正试图帮忙，它试着确保这件显然不重要的事不会占用有限的资源。毕竟，资源最好用来保存真正的大事，如遇到老虎、火灾的危险或绝对不应该穿短裤玩滑雪板。

而且也没有简单的方法可以告诉你的脑袋：“脑袋呀！拜托你啊……不管这本书多么枯燥，多么让我昏昏欲睡，还是请你把这些内容全都记住。”

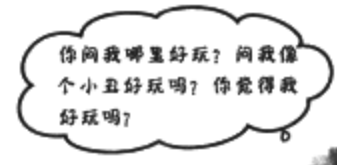
我们将“Head First”的读者视为学习者。

那么，该怎么学习呢？首先，你必须理解它，然后确定不会忘记它。我们不会用填鸭的方式来对待你。根据认知科学、神经生物学、教育心理学最新的研究，学习过程所需要的绝对不只是页面上的文字。我们知道如何开启你的脑袋。

Head First 学习守则：



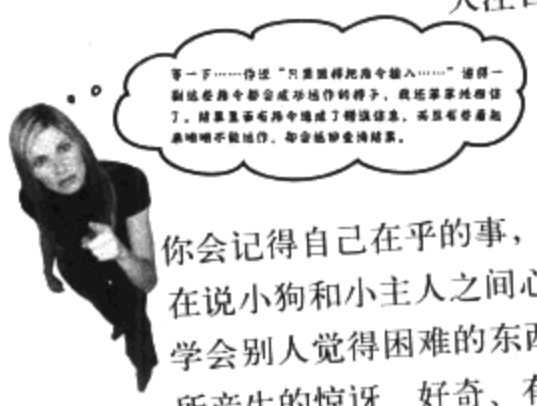
视觉化。图像远比文字更容易记忆，让学习更有效率（可让知识的回想和转换的效率提升89%）。图像也能让事情更容易理解，**将文字放进或靠近相关联的图像中**，而不是把文字放在图像下或后一页，可让学习者在解决相关内容的问题时达到事半功倍的效果。



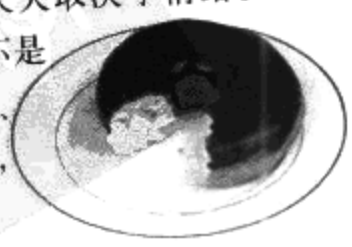
使用对话方式与拟人化风格。最新的研究发现，比起正式的叙述方式，改以第一人称的角度、谈话式风格直接与读者对话，学员课后测试成绩的提升可达 40%。用故事代替论述，以轻松的口语取代正式的演说，别太严肃。你觉得晚宴伴侣的耳边细语和课堂上的演说，哪一种更能引起你的注意力？

让学习者更深入地思考。换句话说，除非你主动刺激你的神经，否则大脑就不会有所作为。读者必须被激发，亲自参与，产生好奇心，自发去解决问题，作出结论，最后产生新知识。为达此目的，你必须接受挑战、勤做练习、用问题诱导思想、用活动活化左右脑并触发多重的感知。

引起——并保持——读者的注意力。我们都有这样的经验：“我真的很想学会这个东西，但是还没翻过第一页就已经昏昏欲睡了。”你的脑袋只会注意到特殊、有趣、怪异、引人注目以及超乎预期的东西。新颖、困难、技术主题学起来未必枯燥乏味，如果不觉得无聊，你的大脑就会学得快得多。



触动心弦。现在，我们知道记忆能力大大取决于情绪。你会记得自己在乎的事，当你心有所感时，你就会记住。不！我不是在说小狗和小主人之间心有灵犀的故事，而是在说当你解出迷题、学会别人觉得困难的东西或发现自己比工程部的Bob更懂技术时，所产生的惊讶、好奇、有趣以及“我好棒”这类的情绪与感觉。



元认知：想一想如何思考（译注1）

如果你真的想学习，想学得更快、更深入，那么请注意你是如何集中注意力，想想如何思考，学学如何学习。

大多数人在成长过程中没有修过元认知或学习理论的课程，我们希望学习，却又不知道如何学习。

我们假设大家拿着这本书是为了学习 SQL，而且可能不想花费太多时间。因为你可能很快就要试验 SQL 的操作，你必须记住读过的东西。为此目的，你必须先理解它。想要从本书（或任何书籍与学习经验）得到最多的知识，就请好好照料你的大脑，让你的大脑好好注意这些内容。

秘诀就在于让你的大脑认为你正在学习的新知识确实很重要，与你的生死存亡有关，就像跳到你面前的食人虎。否则，你就会不断陷入与大脑的苦战，老是记不住新知识。

那么，该如何让大脑把SQL视为一只饥饿的大老虎？

有既慢且繁琐的方法，也有快且有效的方法。慢的方法就是多读几次。你很清楚地知道勤能补拙，即使再乏味的知识，你也能够学会并记住。只要重复的次数够多，你的大脑就会说：“这虽然感觉不怎么重要，但他却一而再，再而三地苦读这部分，所以我想这应该是重要的吧！”

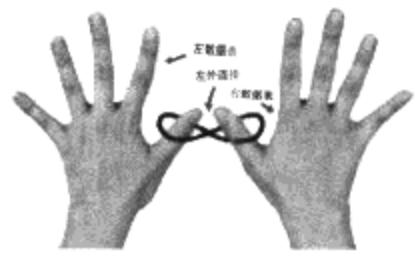
快的方法则是想办法**增加大脑活动**，特别是不同类型的大脑活动。前页出现的素材是解决办法的一种，已经被证实有助于大脑运作。例如，研究显示将文字放在它所描述的图片内（而不是置于页面内其他地方，如图解或正文），有助于大脑将两者联系起来，可以触发更多的神经元。越多的神经活动＝大脑越容易把这部分内容视为值得注意的信息，也越可能将它们记录下来。

对话式风格也很有帮助，因为在意识到自己身处对话中时，人们会付出更多的关注，因为他们必须竖起耳朵，注意整个对话的进行，跟上双方的谈话内容。神奇的是，你的大脑根本不在乎那是你与本书之间的“对话”！另一方面，如果写作风格既正式又枯燥，你的大脑会以为正在聆听一场演讲，自己只是一个被动的听众，根本不需要保持清醒。

然而，图片和对话式风格只不过是一个开端。

▸译注1：元认知，metacognition，教育心理学上的专有名词。管理学习和认知的过程，提高学习的效率。

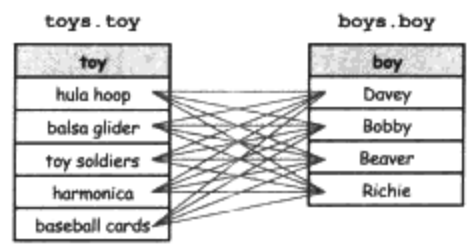
这是我们的做法：



我们使用图片，因为你的大脑对视觉化效果比较有感觉，而非文字。对你的大脑而言，一张图片胜过千言万语。当文字和图片需要合作时，我们将文字嵌入图片中，因为文字若不是在图解或正文中的某处，而是位于相关图片中，大脑会运作得更有效率。

我们重复表现相同的内容，以不同的表现方式、不同的媒介、多重的感知叙述相同的事物。之所以这么做，是为了增加机会将该内容烙印在大脑的不同区域。

我们以超乎预期的方式使用概念和图，因为大脑遇到新鲜有趣的事，波长才会同调。我们使用的图片与概念或多或少都具有情绪内容，也是因为大脑会注意情绪带来的化学反应。对于让我们有感觉的事物，自然较容易记住，即使那些感觉不过是幽默、惊讶、有趣等。



我们使用拟人化、对话式的风格，因为当大脑相信你处于对话过程中，而不是被动地聆听演说时会付出更多注意力，即使你的交谈对象是一本书。也就是说，虽然你是在“阅读”对话，但大脑还是会这么做。

我们用了超过80个的活动，因为当你做事情时的学习效果会比读东西时的效果更佳。我们让习题维持在具有挑战性，但又可以完成的程度，因为大多数人喜欢接受挑战。

我们使用了多种学习风格，因为你可能比较喜欢按部就班，有些人则喜欢先了解大方向，还有一些人则喜欢直接看程序代码范例。然而，不管你是哪一种人，都能受益于本书以不同方式表现相同内容的手法。

本书的设计同时考虑到左右脑，因为大脑中有越多脑细胞参与，你就越容易学会并记住这些东西，而且能保持更长时间的专注。使用一边的大脑，往往意味着另一边的大脑有机会休息，你就可以学得更久且更有效率。

我们也会用故事和练习呈现多个角度的看法，因为当大脑被迫进行评估或判断时会学习得更深入。

书中也有相当多的挑战习题，通过问题，而答案不见得都很直接。我们的用意是让大脑努力工作，才能学得更多、记得更牢。你想想看，只是看别人运动，你有办法达成帮自己塑身的效果吗？同时，我们尽量确保大脑往正确的方向努力，以免浪费大量脑力用于处理难以理解的范例或难以剖析、充满行话、咬文嚼字的论述。

我们还会使用人物。在故事、图片和范例中，处处都是人物。因为你也是人！你的大脑对于人会比对事物更加注意。

让你的大脑顺从你的方法

好吧，该做的我们都做了，剩下的就靠你了。这里介绍一些技巧，但只是一个开端，你应该听从你的大脑，看看哪些对你的大脑有效，哪些无效。试试看吧！

沿虚线剪下，用Hello Kitty磁铁贴在冰箱上。

① 慢慢来，理解越多，需要强记的就越少。

别只顾着翻页，记得停下来，好好思考。书中提出问题时，别完全不思考就直接看答案。想象有另外一个人面对面地向你提问，如果能够迫使大脑思考得更深入，你就有机会理解并记得更多的知识。

② 勤做练习，写下你的心得笔记。

我们在书中安排了习题，如果你只看不做，就好像看着别人做你想做的塑身运动，那是不会有效果的。**使用铅笔作答**。大量证据显示，学习中的实质活动可增强学习的效果。

③ 认真阅读“没有蠢问题”单元。

详细阅读所有的“没有蠢问题”。这可不是无关紧要的说明，而是**核心内容的一部分**！千万别略过了。

④ 把阅读本书作为睡前最后一件事，或者至少当作睡前最后一件具有挑战的事。

学习中的一部分反应发生在放下书本之后（特别是转化为长期记忆的过程）。你的大脑需要进一步处理新知识的时间。如果你在处理期间塞进其他新知识，某些刚学过的东西就会遗失。

⑤ 喝水，多喝水。

你的大脑需要浸泡在充分的液体内（译注2）才能运作良好，脱水（往往发生在感觉口渴之前）会减缓认知功能。

⑥ 说出来，大声说出来。

说话驱动大脑的不同部位。如果你需要理解某项事物或试图增强记忆力，请大声说出来。大声地解释给别人听，效果更佳。你会学得更快，甚至触发许多新的想法，这是光凭读书做不到的。

⑦ 倾听大脑的声音。

注意你的大脑是否负荷过重，如果你发现自己开始漫不经心，或者过目即忘，就到了应该休息的时候。当你错过某些重点时，放慢脚步，否则你将失去更多。

⑧ 用心感受！

必须让脑袋知道这一切都很重要。试着融入故事情境，为照片加上自己的说明，即使抱怨笑话太冷，都比毫无感觉要好，任何感觉对学习效果都有帮助。

⑨ 动手设计！

将所学内容应用到你的日常工作或项目决策中。反正就是尽量运用知识获取本书习题与活动之外的实践经验。你需要一个有待解决的难题……找一个能够运用本书技术的问题，试着解决它。

▸ 译注2：　医学报告指出，大脑的 80% 由水组成。

读我

这是一段学习经验，而不是一本参考书。所有阻碍学习的东西，我们都会刻意排除。第一次阅读时，你必须从头开始，因为本书对读者的知识背景做了一些假设。

我们从 SQL 基础语法开始，然后是 SQL 数据库设计概念，接下来是高级查询。

虽然说创建设计良好的数据库和数据表的确很重要，不过在到达这个境界前，我们需要了解 SQL 的语法。所以我们先提供大家可以动手尝试的 SQL 语句。你可以用这种方式以 SQL 做出成品，这样才有兴趣进一步接触 SQL。接触较多后，我们再引入良好的数据库设计实践。届时，各位已经很熟悉语法的运用，就可以专心地学习新概念。

我们并未涉及每一个 SQL 语句、函数或关键字。

虽然我们可以非常详细地涵盖所有 SQL 语句、函数和关键字，但是各位应该更希望拿到一本重量还算可以接受，但又能从中学到重要语句、函数、关键字的书籍吧。我们收录使用 SQL 时 95% 常用的必备知识。当你理解本书内容后，可以带着自信寻求高深查询所需要的深奥函数。

我们并未加入 RDBMS 的每一种特色。

市面上有 Standard SQL、MySQL、Oracle、MS SQL Server、PostgreSQL、DB2等众多 RDBMS 系统，如果本书试图囊括每个指令在各种系统上的变形，那么页数绝不只现在这么多。我们热爱珍贵的树木，所以只利用 MySQL 表达 Standard SQL语法。本书大多数范例均可用 MySQL 运行，多半也能在前面提到的 RDBMS 上通行无阻。有需要时再购买 RDBMS 的专属参考书吧!

不要略过任何活动。

习题与活动并非附加的装饰品，而是本书核心内容的一部分。有些可以帮助记忆，有些可以帮助理解，还有些可以帮助应用。所以，请不要略过这些练习。填字游戏是唯一非必要的部分，但是它们提供了不同情境来帮助大脑回顾学过的关键字与术语，中文版虽然翻译了提示，但答案还是英文哦。

重复是刻意且有必要的。

我们希望“Head First”系列书籍能让你真正学到东西，希望你读完此书之后能够记住你所读过的内容。大部分参考用书的目标并不包括知识记忆的保存和触发，但本书的重点是学习，所以重要内容会一再出现以加深你的印象。

程序范例尽量精简。

我们的读者告诉我们，不希望看到书中列出200行的程序代码，而其中和主题有关的关键程序代码却只有两行。本书尽量把程序代码缩短，让学习的过程清晰简单。不要期待所有的程序代码都很牢靠或完整，毕竟我们的程序代码是辅助学习之用，不见得一定功能完整。

我们把很多指令放在网站上以方便大家复制及粘贴到数据库软件或终端上。网址是`http://www.headfirstlabs.com/books/hfsql/`。

“动动脑”习题没有答案。

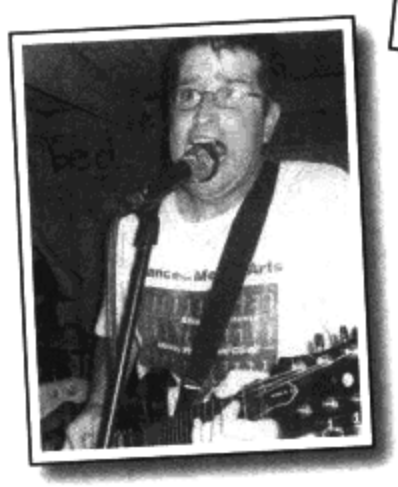
对于某些人来说，这类习题没有一定的答案;对于其他人来说，“动动脑”习题所启发的学习经验在于自我判断答案是否正确以及答案正确的时机。在某些习题中，我们会提供暗示来为你指引正确的方向。

技术审阅团队

Cary Collett

狗狗Chaucer也有不少贡献。

Steve Milano



Shelley Rheams

LuAnn Mazza

Jamie Henderson

惊天动地的审阅团队：

Cary Collett 15年来呆过新创企业、政府研究单位，目前服务于某个财政部门并审阅本书，他打算有空时要好好地享受工作以外的生活乐趣，像烹饪、登山健行、阅读以及陪狗玩。

人在伊利诺斯州的 **LuAnn Mazza**，从极度繁忙的专业软件设计与分析工作中神奇地抽出时间详细审阅本书，我们非常高兴听到她现在有空享受骑自行车、摄影、玩电脑、听音乐与打网球等休闲爱好。

如果**Steve Milano** 并未因为工作而埋首于十几种不同语言里，也没有为《深入浅出 SQL》贡献他的一流审阅功力，甚至不是跟摇滚乐团 Onion Flavored Rings 一起在没有空调的地下室练习，那他应该是在家陪他的爱猫 Ralph 和 Squeak。

"**Shelley" Moira Michelle Rheams**，MEd、MCP、MCSE 的讲师，并于新奥尔良 West Band 学院的 Delgado 社区大学执行早期儿童教育计划。她现在乐于配合Katrina飓风灾后重建的需要，把课程放到网上，我们特别感谢她从满载的行程中抽出时间配合我们。

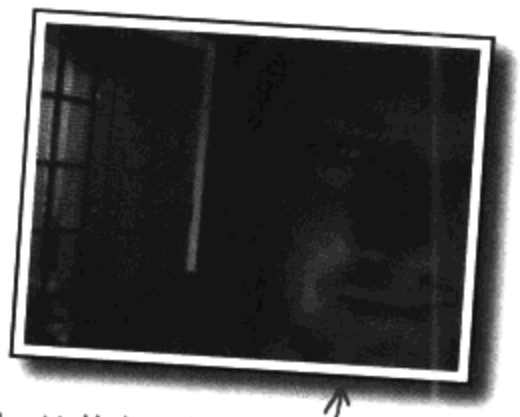
Jamie Henderson 是位有着紫色头发的资深系统构造师，她的休息时间都分给了大提琴、阅读、电子游戏与 DVD。

审阅团队是本书的程序代码与习题能够使用的原因，也是让大家在读完本书后成为一名自信 SQL 设计师的重要推手。他们对细节的重视也让我们的文字不会可爱过头、不会太过沾沾自喜，有时候甚至还阻止我们的冷笑话讲过头。

致谢

给我的编辑：

首先，我要谢谢我的编辑 **Brett McLaughlin**，他不仅指导了一本书，而且指导我这个新手写出两本“Head First”系列书籍。Brett 不只是我的编辑，他更是试金石与精神导师的综合体，如果没有他的指导、支持与关心，我绝对写不完本书。除了推动我参加一开始的作家甄选，Brett 还能够觉得我的超级冷笑话可以令人发笑，让这段时间成为我体验过的最佳写作过程。他还提供了许多建议与提示，也给了我非常多的指导。感谢你，Brett!



Brett McLaughlin

编辑 **Catherine Nolan** 最近得了溃疡，都是我在编辑过程快结束前发生的意外所害。因为有她，这本书才没有拖到 2008 年，或许她就是这本书存在的原因。截稿日期的控管就像把小猫抓在手上，Catherine 总是能阻止小猫落地或被小猫抓伤。我非常需要进度表，而 Catherine 是我遇过的最佳进度安排管理人，或许我也是她遇过的最不会安排时间的人吧！我衷心希望她的下一个项目能顺利一点。

Catherine Nolan

给O'Reilly小组：

设计编辑 **Louise Barr** 既是很好相处的朋友，也是非常出色的图片设计师，她总是有办法把我胡思乱想的想法转化为惊艳不凡的图片，让难懂的概念变得清晰易懂。所有卓越的版面设计都要归功于她，我相信各位也会想感谢她的设计。

Lou Barr

如果没有技术审阅的过程，我们就会生出一本错误百出的书，编辑 **Sanders Kleinfeld** 在这方面有很大贡献。他还指出一些非常需要补上“桥梁”的概念深谷，这远远超出他的职责了。谢谢你，Sanders!

最后，我想谢谢**Kathy Sierra**和**Bert Bates**，他们开创了这么美好的系列书籍，让我得以参加最能挑战智力的 Head First 新手培训。没有那三天的培训，我根本难以想象“Head First”系列书籍多么难以创作。还有，Bert 提出的最终编辑建议一针见血，大幅提升了这本书的品质。

目录（摘要版）

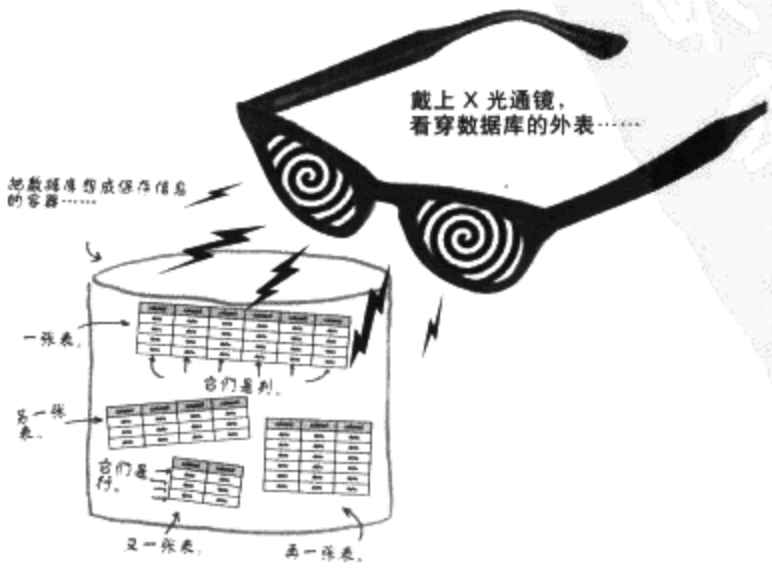
目录（具体版）

序

1 数据和表

保存所有东西的地方

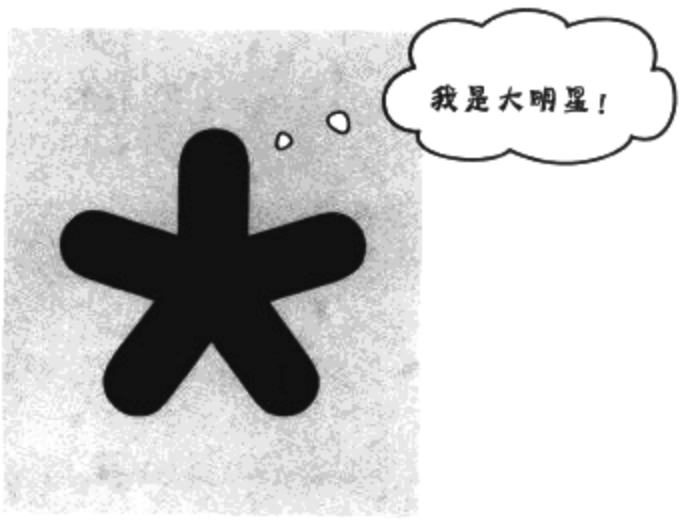
你是否也很讨厌总是找不到东西？找不到你的车钥匙、Urban Outfitter 75 折优惠券以及应用程序的数据吗？在最需要的时候，偏偏**找不到需要的东西**，还有什么比这更糟？提到应用程序，存储程序重要信息的最佳场所非**表**莫属。所以请翻到下一页，**踏入关系数据库**的世界吧。



×

2 SELECT语句 取得精美包装里的数据

施予真的胜过取得吗？在数据库的世界里，**取得数据**的需求很可能与插入数据的需求一样多。这就是本章的目的：各位将见识到功能非常强大的**SELECT**语句，并且学习到**如何取得放在表里的重要信息**。我们还会学到如何利用 **WHERE**、**AND**、**OR** 选择数据，甚至可以过滤掉不想选择的数据。



3 DELETE和UPDATE 改变是件好事

一直在改变你的心意吗？现在没有问题了！有了接下来会提到的命令— **DELETE** 和 **UPDATE**，我们不再受限于6个月前所做的决定，当时可能适合捕捞鳊鱼，但现在已经不是季节了。有了 UPDATE，我们可以**改变数据**，而 DELETE 则可**删除不需要的数据**。这一章不只是给你鱼竿，还会教你如何选择性地使用这些新能力，避免舍弃了需要的数据。

聪明的表设计

4 为什么要规范化？

你已经创建了一些表，但都没有经过仔细考虑。没有关系，这些表都可以用。你可以从中 SELECT、INSERT、DELETE、UPDATE列，但随着**取得的数据越来越多**，你一定希望以前能多考虑一点，好让现在的 WHERE 子句简单一点。我们需要让表更正常、更*规范*。

5 ALTER 改写历史

你可曾希望更正以前年少无知犯下的错误？现在，你的机会来了。使用 **ALTER 命令**，你能对几天前、几个月前，甚至是几年前设计的表套用新学到的设计方法。更好的是，套用时不会影响现有数据。随着熟悉本章的过程，各位还会学到**规范化**的真正意义，并且能让你的所有表都符合规范化，无论它是过去的还是未来的产物。

6 SELECT 进阶

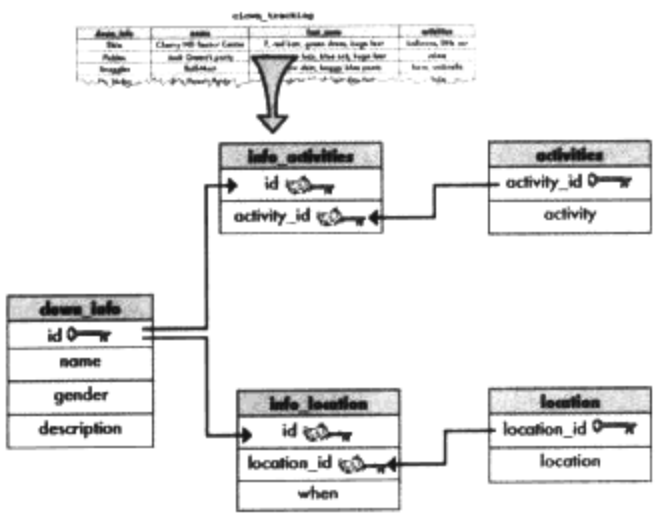
以新视角看你的数据

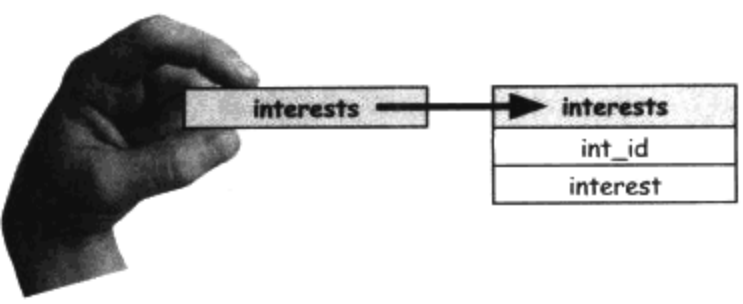
现在该为SQL工具包添加一些功能了。我们已经知道如何用SELECT和WHERE子句选出数据，但有时候我们需要比SELECT加上WHERE子句更**精确**的选取工具。在本章中，我们将学习如何给数据**排序**和**归组**，还会学习如何对查询结果套用**数学运算**。

7 多张表的数据库设计

拓展你的表

到了某个时候，只有一张表就不够了。数据变得越来越复杂，你所使用的**唯一一张表**实在装不下了。表里充满了多余的数据，既浪费存储空间，又会拖慢查询的速度。一张表的负荷已经接近极限了，但是外面的世界还很宽广。我们将用**不只一张表**来记录数据、控制数据，最后它将成为你的数据库的主人。





8 联接与多张表的操作

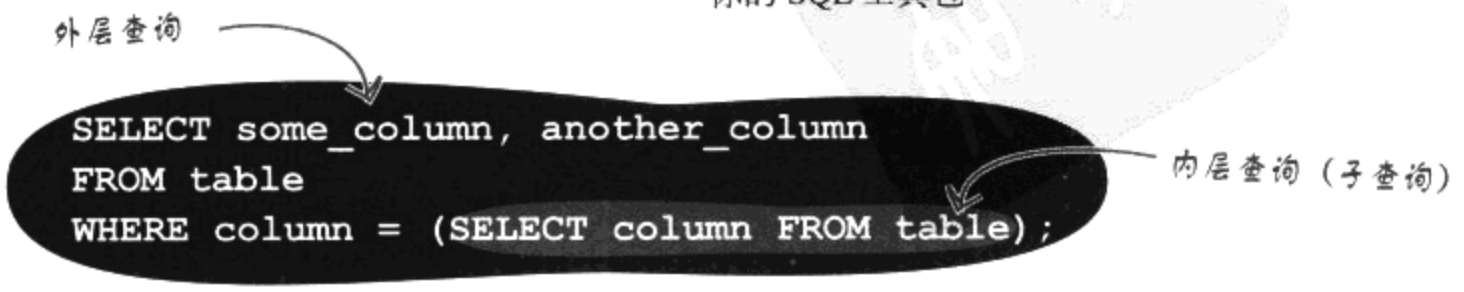
不能单独存在吗？

欢迎来到多张表的世界。数据库中有**多张表**是件好事，但我们也需要学习一些操控多张表的**新技术与工具**。混乱状态与多张表一起出现，所以你需要**别名**来让表更清楚简单。**联接**则有助于联系表，取得分布在各张表里的内容。准备好，再度**取回数据库的控制权**吧！

9 子查询

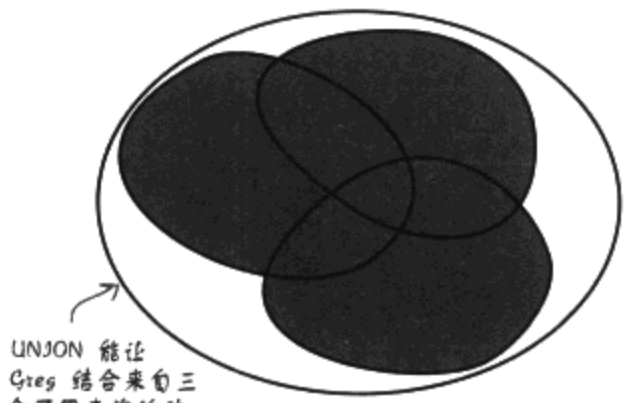
查询中的查询

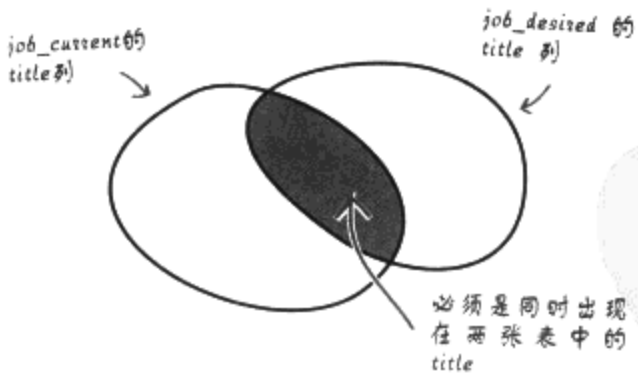
Jack，请给我被分成两部分的问题，谢谢。有了联接的确很好，但有时**要问数据库的问题不只一个。**或者需要**把甲查询的结果作为乙查询的输入。**这时候就该是**子查询**出场了。子查询有助于**避免数据重复，让查询更加动态灵活**，甚至能引入高端概念。（最后一项不一定会成真，不过三项好处中有两项是真的也很好嘛！）

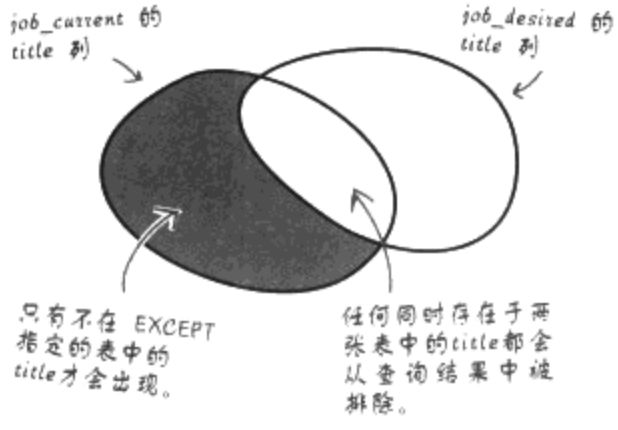


10 外联接、自联接与联合新策略

关于联接，我们只认识了一半。我们已经看过返回每个可能行的交叉联接，返回两张表中相符记录的内联接。但我们还没见过**外联接**，它可以在表中没有匹配记录的情况下返回记录；**自联接**（光听名称就很奇怪了），它可以联接表本身；还有**联合**，它可以合并查询结果。学会这些技巧后，你就可以采用自己需要的方式取得所有数据（而且本章也没有忘记探讨关于子查询的真相！）



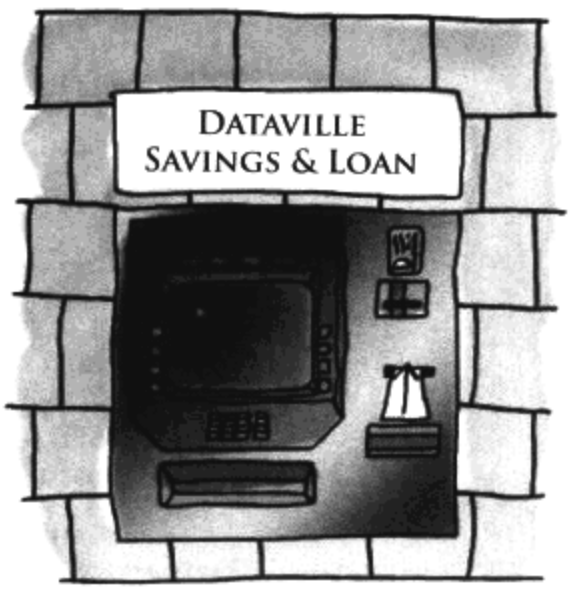




11 约束、视图与事务

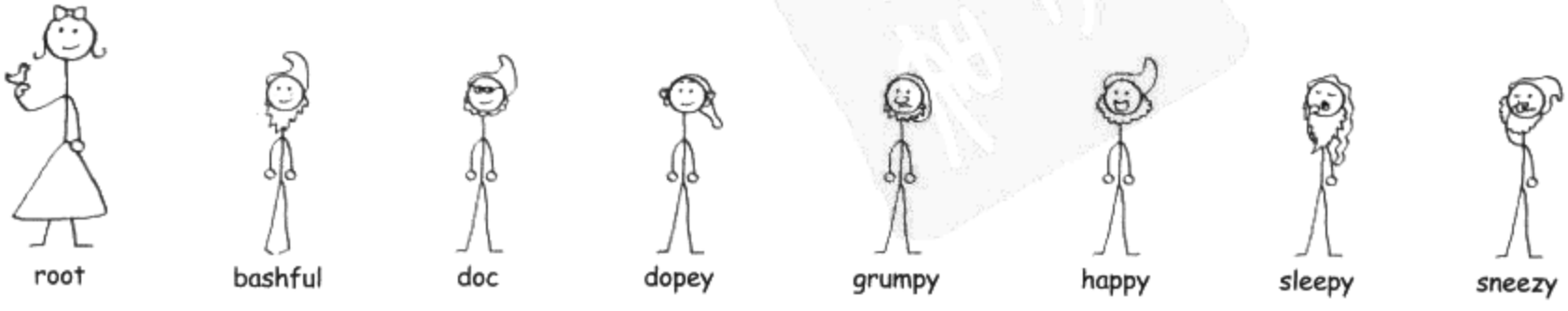
人多手杂，数据库受不了

你的数据库成长到一定规模了，出现了其他需要使用数据库的人。问题是，其他人的 SQL 技术可能不像各位这么娴熟。结果我们需要**防止其他人输入错误的数据**，需要让其他人**只看到部分数据**的技术，还需要**防止大家同时输入数据时互相踩到别人的地盘**。在本章中，我们开始保护数据，以免其他人对数据进行错误的操作。欢迎来到“数据库自卫术 Part 1”。



12 安全性 保护你的资产

为了创建数据库，大家已经花了许多时间与精力。如果数据库受到了什么伤害，你一定会崩溃吧！而且虽然让其他人**访问你的数据**有其必要性，但真的会忍不住担心有人插入或更新数据时的操作不正确，甚至发生更糟的情况：**删错了数据**。我们即将要学到如何把数据库和其中的对象变得更安全，以及如何全面控制**谁对数据进行什么操作**。



附录 1

十大遗珠

尽管刚刚结束一场 SQL 盛宴，但总会有剩下的东西。我想各位还需要知道一些其他补充事项，就算只能简短提一下也好……忽略了就是觉得怪怪的。放下本书前，希望大家稍微看一下本章的 SQL 小花絮。

另外，本章结束后还有两篇附录……可能还有几篇广告……然后就真的结束了，真的，我保证！

A	ABSOLUTE ACTION ADD ADMIN AFTER AGGREGATE ALIAS ALL ALLOCATE ALTER AND ANY ARE ARRAY AS ASC ASSERTION AT AUTHORIZATION
B	BEFORE BEGIN BINARY BIT BLOB BOOLEAN BOTH BREADTH BY
C	CALL CASCADE CASCADED CASE CAST CATALOG CHAR CHARACTER CHECK CLASS CLOB CLOSE COLLATE COLLATION COLUMN COMMIT COMPLETION CONNECT CONNECTION CONSTRAINT CONSTRAINTS CONSTRUCTOR CONTINUE CORRESPONDING CREATE CROSS CUBE CURRENT CURRENT_DATE CURRENT_PATH CURRENT_ROLE CURRENT_TIME CURRENT_TIMESTAMP CURRENT_USER CURSOR CYCLE
D	DATA DATE DAY DEALLOCATE DEC DECIMAL DECLARE DEFAULT DEFERRABLE DEFERRED DELETE DEPTH DEREF DESC DESCRIBE DESCRIPTOR DESTROY DESTRUCTOR DETERMINISTIC DICTIONARY DIAGNOSTICS DISCONNECT DISTINCT DOMAIN DOUBLE DROP DYNAMIC
E	EACH ELSE END END_EXEC EQUALS ESCAPE EVERY EXCEPT EXCEPTION EXEC EXECUTE EXTERNAL
F	FALSE FETCH FIRST FLOAT FOR FOREIGN FOUND FROM FREE FULL FUNCTION
G	GENERAL GET GLOBAL GO GOTO GRANT GROUP GROUPING
H	HAVING HOST HOUR
I	IDENTITY IGNORE IMMEDIATE IN INDICATOR INITIALIZE INITIALLY INNER INOUT INPUT INSERT INT INTEGER INTERSECT INTERVAL INTO IS ISOLATION ITERATE
J	JOIN
K	KEY
L	LANGUAGE LARGE LAST LATERAL LEADING LEFT LESS LEVEL LIKE LIMIT LOCAL LOCALTIME LOCALTIMESTAMP LOCATOR
M	MAP MATCH MINUTE MODIFIES MODIFY MODULE MONTH
N	NAMES NATIONAL NATURAL NCHAR NCLOB NEW NEXT NO NONE NOT NULL NUMERIC
O	OBJECT OF OFF OLD ON ONLY OPEN OPERATION OPTION OR ORDER ORDINALITY OUT OUTER OUTPUT
P	PAD PARAMETER PARAMETERS PARTIAL PATH POSTFIX PRECISION PREFIX PREORDER PREPARE PRESERVE PRIMARY PRIOR PRIVILEGES PROCEDURE PUBLIC
Q	
R	READ READS REAL RECURSIVE REF REFERENCED REFERENCING RELATIVE RESTRICT RESULT RETURN RETURNS REVOKE RIGHT ROLE ROLLBACK ROLLUP ROUTINE ROW ROWS
S	SAVEPOINT SCHEMA SCROLL SCOPE SEARCH SECOND SECTION SELECT SEQUENCE SESSION SESSION_USER SET SETS SIZE SMALLINT SOME SPACE SPECIFIC SPECIFICTYPE SQL SQLEXCEPTION SQLSTATE SQLWARNING START STATE STATEMENT STATIC STRUCTURE SYSTEM_USER
T	TABLE TEMPORARY TERMINATE THAN THEN TIME TIMESTAMP TIMEZONE_HOUR TIMEZONE_MINUTE TO TRAILING TRANSACTION TRANSLATION TREAT TRIGGER TRUE
U	UNDER UNION UNIQUE UNKNOWN UNNEST UPDATE USAGE USER USING
V	VALUE VALUES VARCHAR VARIABLE VARYING VIEW
W	WHEN WHENEVER WHERE WITH WITHOUT WORK WRITE
X	
Y	YEAR
Z	ZONE

```
File Edit Window Help
> SELECT CURRENT_DATE;
+--------------+
| CURRENT_DATE |
+--------------+
| 2007-07-26   |
+--------------+
1 row in set (0.00 sec)
```

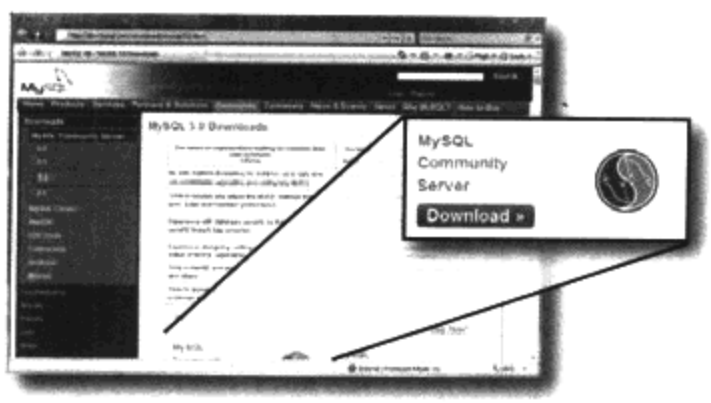
```
File Edit Window Help
> SELECT CURRENT_TIME;
+--------------+
| CURRENT_TIME |
+--------------+
| 11:26:48     |
+--------------+
1 row in set (0.00 sec)
```

```
File Edit Window Help
SELECT CURRENT_USER;
+----------------+
| CURRENT_USER   |
+----------------+
| root@localhost |
+----------------+
1 row in set (0.00 sec)
```

ii 附录 2 安装 MySQL

自己动手试

各位学到这么多 SQL 技巧，如果没有施展的场所，岂非英雄无用武之地？本篇附录说明了安装 MySQL 的方式，让大家有地方自我琢磨技艺。



iii 附录 3 SQL 工具总整理

崭新的 SQL 工具包

所有SQL工具初次齐聚一堂，仅限今夜哦！（**开玩笑啦！**）这篇附录收集了所有我们提到的 SQL 工具，花一点时间浏览一下，感受一下那份**成就感** —— **你已经完全学会这些工具了哦！**

“有些书籍会激发我们的购买欲，有些书籍会让我们把它带在身边，有些书籍我们则只会把它放在书桌上当摆设，感谢O'Reilly和Head First制作小组的出现，从他们手中诞生了翻到书页卷边、被画得乱七八糟、令人爱不释手的“Head First”系列。《Head First SQL》 现在在我家书架上排第一位。”

—— Bill Sawyer，ATG 课程管理人，Oracle

“这本书不是简化版的SQL教科书，而是充满挑战的SQL、让人感兴趣的SQL、让人觉得好玩的SQL。它甚至回答了长久以来的疑问：‘应该如何讲解非关联子查询而不会失去正面积极的心态？’风格明快又不严肃，而且阅读过程美妙无比——这才是正确的学习方式。”

—— Andrew Cumming，《SQL Hacks》的作者，Zoo Keeper at sqlzoo.net

“真是不敢相信我的眼睛！SQL 不是一种计算机语言吗？一本关于 SQL 的书应该是写给计算机看的书，不是吗？可是《Head First SQL》却是写给人类的书！怎么会发生这种事？！”

—— Dan Tow，《SQL Tuning》的作者

对 Head First 系列书籍的赞誉

“《深入浅出设计模式》条理清晰、幽默风趣、真材实料，甚至能帮助非程序员来好好思考问题解决之道。”

—— Cory Doctorow，Boing Boing 的共同编辑、
《Down and Out in the Magic Kingdom》与
《Someone Comes to Town, Someone Leaves Town》的作者

“如果你认为 Ajax 是相当复杂的技术，《深入浅出Ajax》就是为你而写。本书让每一位 Web 编程人员都能体验到无与伦比的动态感和吸引力。”

—— Jesse James Garrett，Adaptive Path

“我昨天刚收到这本书，在回家的路上便开始阅读，简直欲罢不能，于是我把书带到健身房，一边运动一边阅读，脸上堆满笑容。这真是太棒了！不仅有趣、涵盖许多基础知识，而且观点正确，给我留下了深刻的印象。”

—— Erich Gamma，IBM 杰出工程师、《Design Patterns》的共同作者

“本书融乐趣、捧腹大笑、洞察力、技术深度、非常实用的建议于一体，成为一本寓教于乐的书籍。不管是初次学习设计模式，或者已经具有多年的使用设计模式的经验，你都可以在访问对象村（Objectville）的过程中学到东西。”

—— Richard Helm，《Design Patterns》的共同作者

“这是我所读过的关于软件设计最有趣、最聪明的书籍之一。”

—— Aaron LaBerge，技术副总，ESPN.com

“我刚读完《深入浅出面向对象分析与设计》，已经深深爱上它！我最喜欢这本书把焦点放在为什么要实践 OOA&D 上 —— 为了写出美妙的软件！”

—— Kyle Brown，IBM 杰出工程师

“我衷心喜欢《深入浅出HTML与CSS、XHTML》这本书 —— 它以妙趣横生的形式讲述了需要学习的一切。”

—— Sally Applin，UI 设计师及精致艺术工作者，sally.com

“《深入浅出Java》明快、轻松、优雅并且充满乐趣，你自然而然就能从中学到东西。”

—— Ken Arnold，Sun Microsystem前任资深工程师、
《The Java Programming Language》的共同作者

“（读完《深入浅出设计模式》）我觉得好像刚刚把一本千斤重的书举过头顶。”

—— Ward Cunningham，Wiki 发明者、Hillside Group创始人

“《深入浅出设计模式》近乎完美，它在提供专业知识的同时，还保有相当高的可读性，口吻权威、阅读轻松。它是我所读过的软件书籍中极少数让我觉得不可或缺的一本。”

—— David Gelernter，耶鲁大学计算机科学系教授、
《Mirror World》和《Machine World》的作者

“《深入浅出设计模式》的内容正适合我们这些喜欢新技术的人。本书为实际的开发策略提供正确的参考，让我的头脑运转顺畅，不会被专家枯燥乏味的用语搞得头昏脑胀。”

—— Travis Kalanick，MIT TR100 的 Scour and Red Swoosh Member 的创始人

“运用‘Head First’/‘Head Rush’系列惯有的诙谐幽默风格，本书单刀直入地教你如何编写给服务器发送请求以及在返回时更新网页的JavaScript……本书最大的好处是除了对程序代码如何运作有绝妙诠释外，也顾及安全防护的主题。假如你通过这本书学习Ajax，就不太可能会忘记你所学过的一切。”

—— Stephen Chapman，JavaScript.About.com

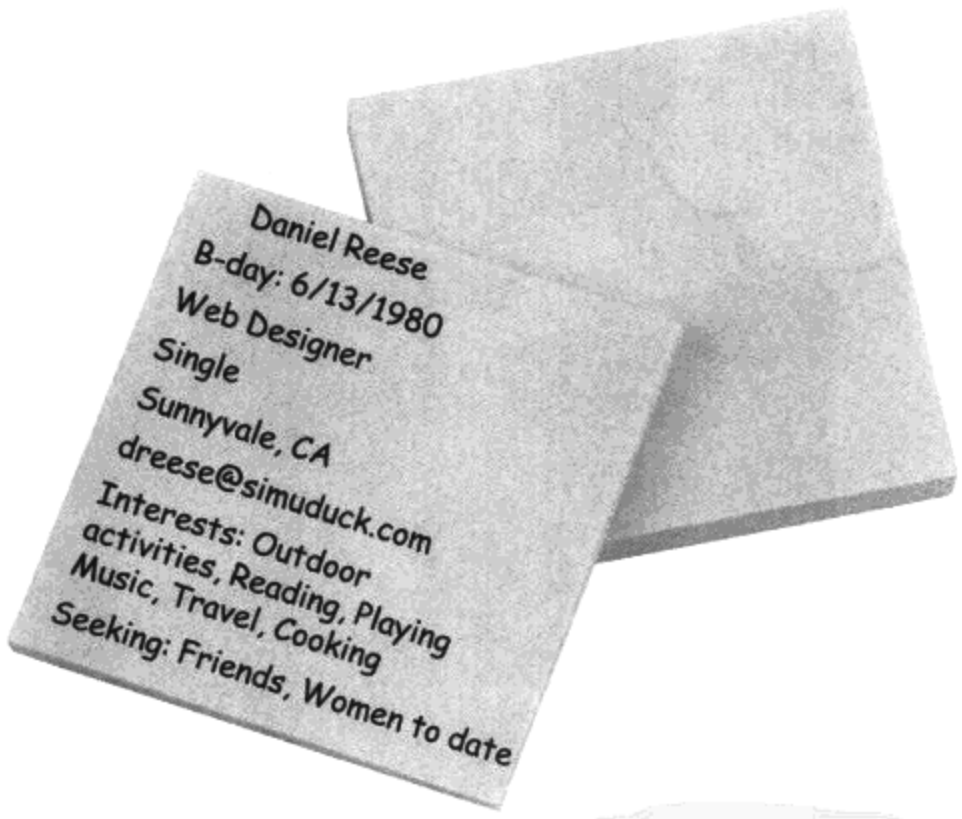
1 数据和表

保存所有东西的地方

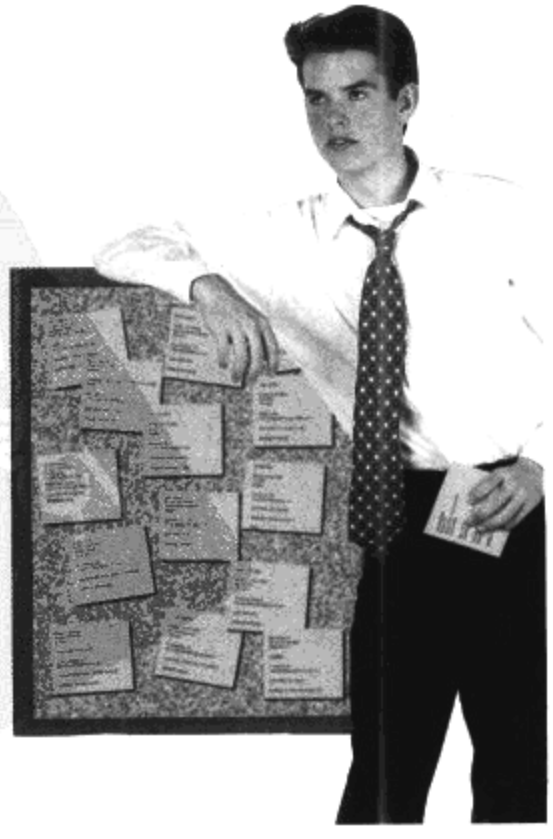
你是否也很讨厌总是找不到东西？找不到你的车钥匙、Urban Outfitter 75 折优惠券以及应用程序的数据吗？在最需要的时候，偏偏**找不到需要的东西**，还有什么比这更糟？提到应用程序，存储程序重要信息的最佳场所非**表**莫属。所以请翻到下一页，踏入**关系数据库**的世界吧。

定义数据

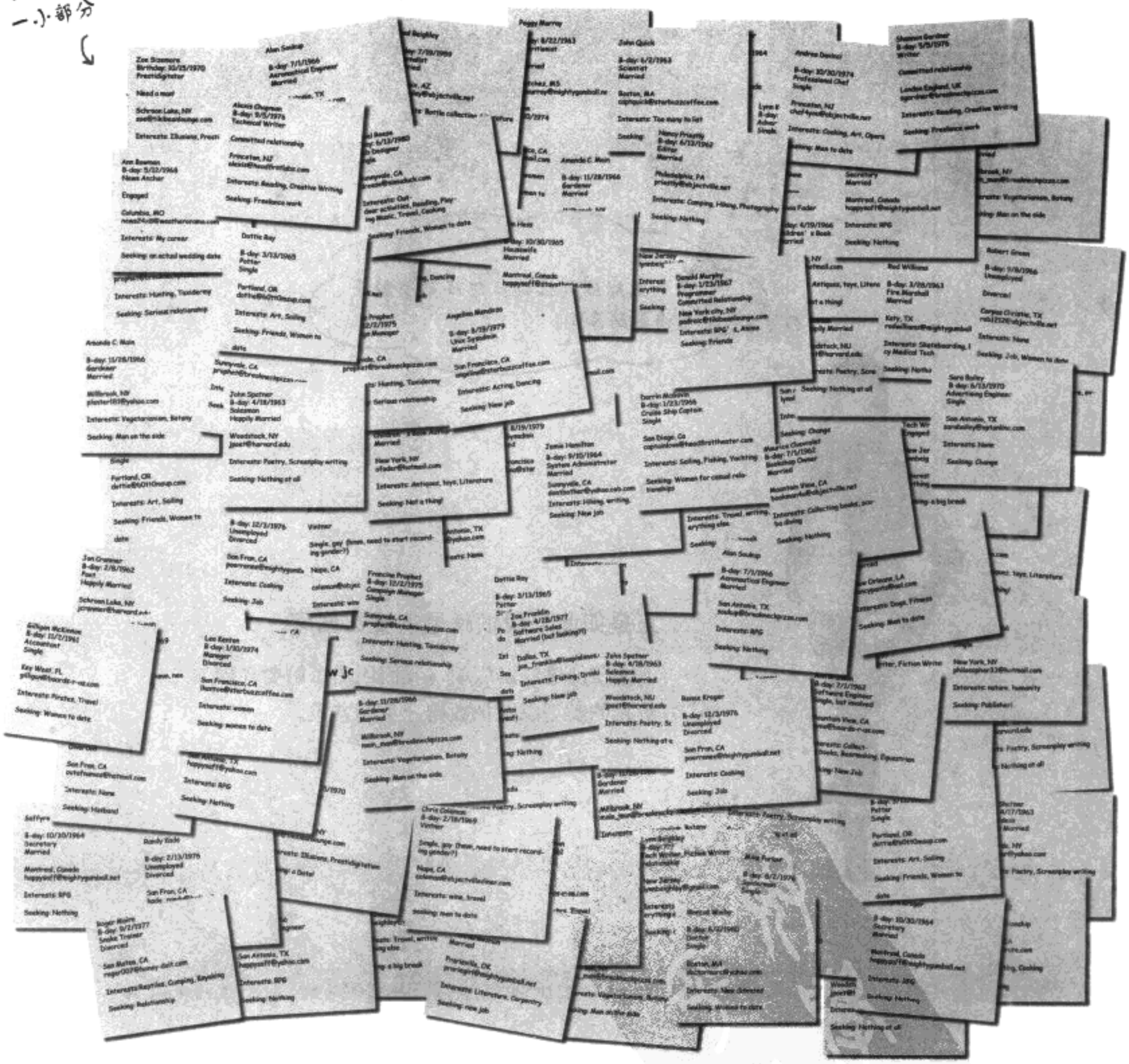
Greg 认识许多单身贵族，他最喜欢记录每位朋友的兴趣爱好，并为有缘人从中撮合。他把每个人的信息写在便笺上，就像这样：



Greg的这套系统已经行之有年。不过，他上星期刚把为人找新工作也加入自己的便笺中，结果他的人才列表飞快增长，非常非常快……



只是 Greg 的便笺的一小部分



动动脑

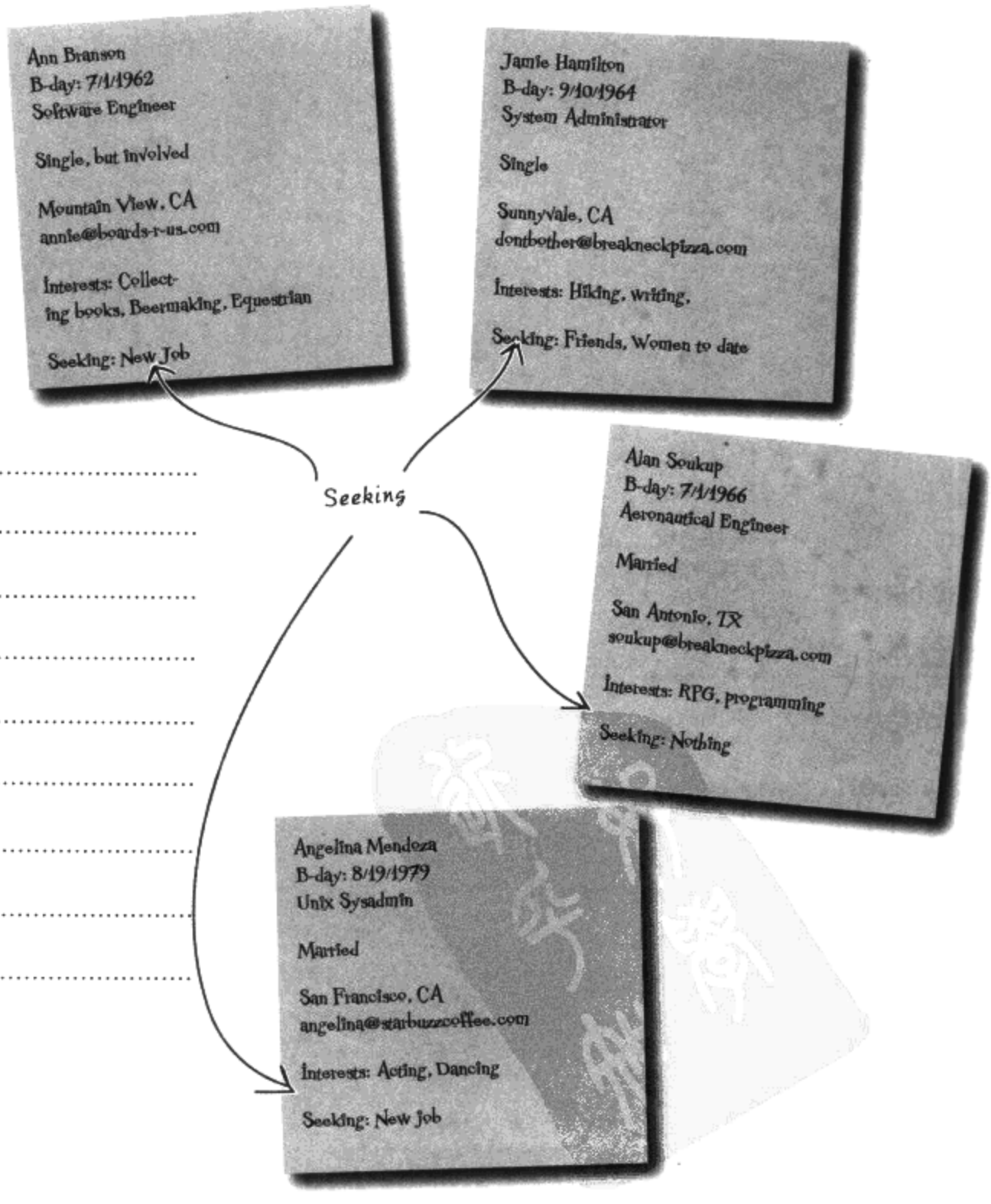
有更好的方式来组织信息吗？你会怎么做？

正是如此。我们就是需要数据库。

不过在创建数据库前，我们还需了解要存储数据的类型以及把数据分类的方式。

磨笔上阵

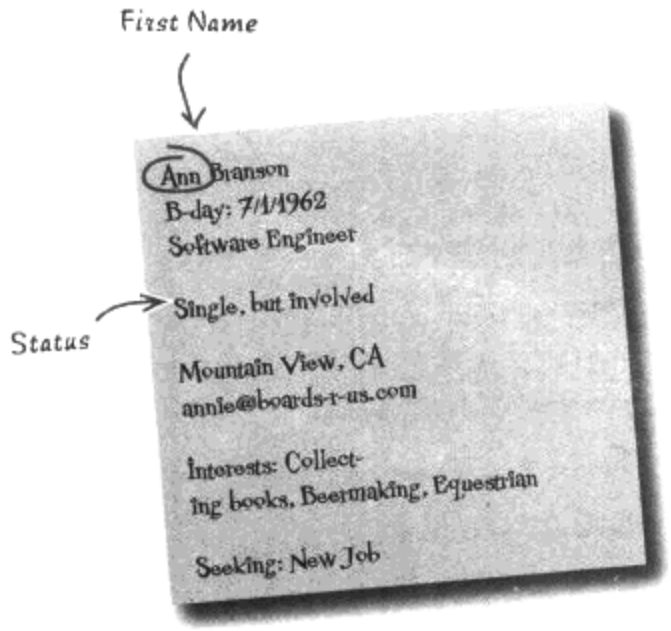
你看，下面是 Greg 的便笺。看看你能从这些便笺中找出哪些类似信息。为每种相似信息贴上标签来描述它的信息分类，然后把列出的标签写在本页的空白处。



磨笔上阵 解答

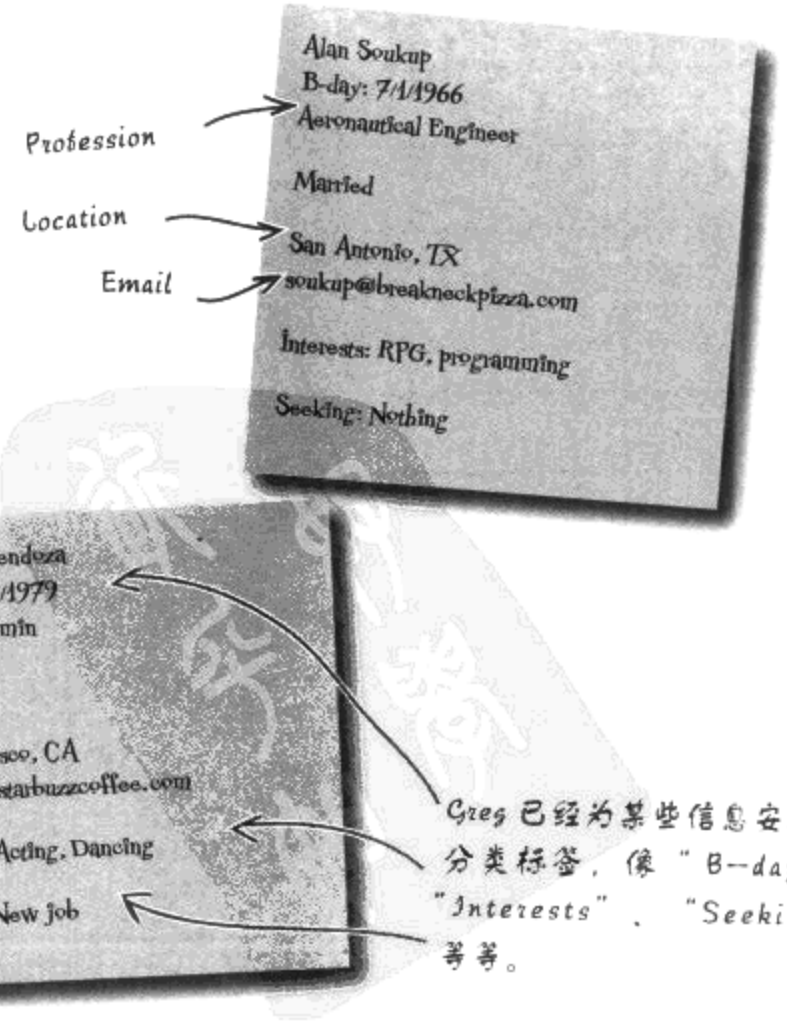
你看，下面是 Greg 的便利贴。看看你能从这些便利贴中找出哪些类似信息。为每种相似信息贴上标签来描述它的信息分类，然后把列出的标签写在本页的空白处。

既然创建了信息分类，我们就可以用分类来组织数据。



我们把姓名分成“姓”（Last Name）和“名”（First Name），这对稍后的数据排序非常有用。

- First Name
- Last Name
- Birthday
- Profession
- Status
- Location
- Email
- Interests
- Seeking

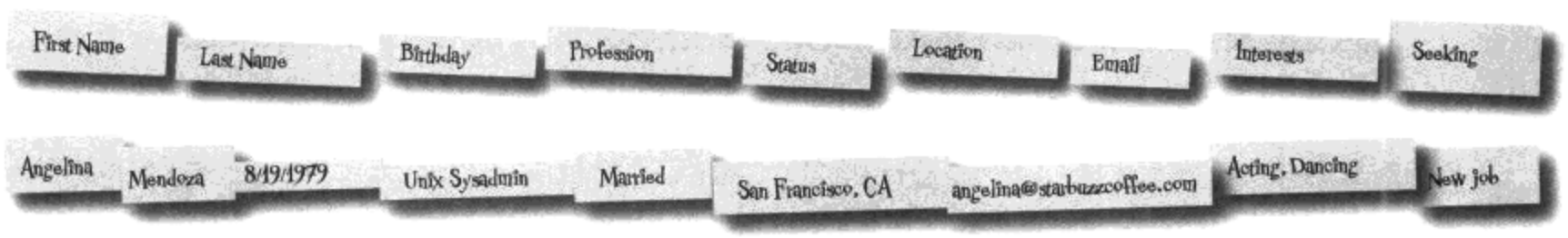


Greg 已经为某些信息安排了分类标签，像“B-day”、“Interests”、“Seeking”等等。

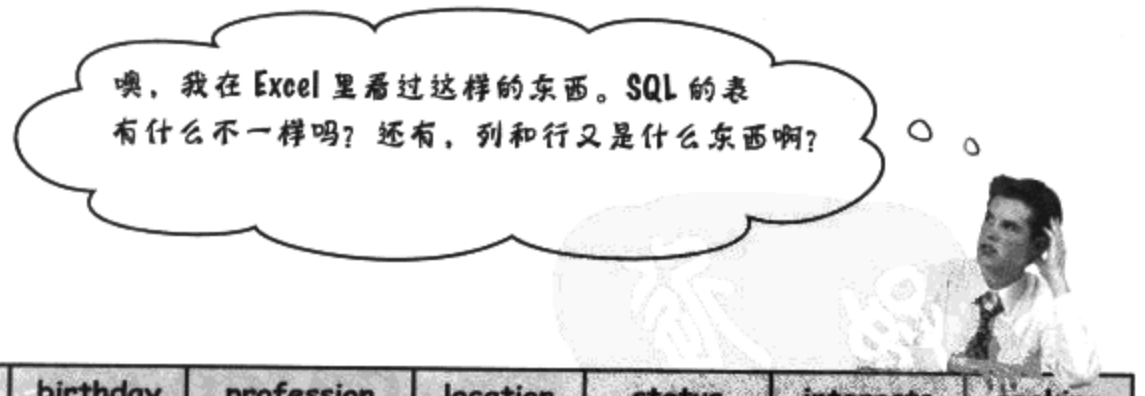
从分类的角度看数据

让我们以不同方式来看数据。如果剪开每张便笺，并把每条信息水平排列，你会看到如下所示的东西：

如果再裁切另一张便笺，而且在每张纸片上写下刚刚所说的分类，你会得到如下所示的东西：



这就是在表（table）中用列（column）和行（row）整齐呈现出的信息。



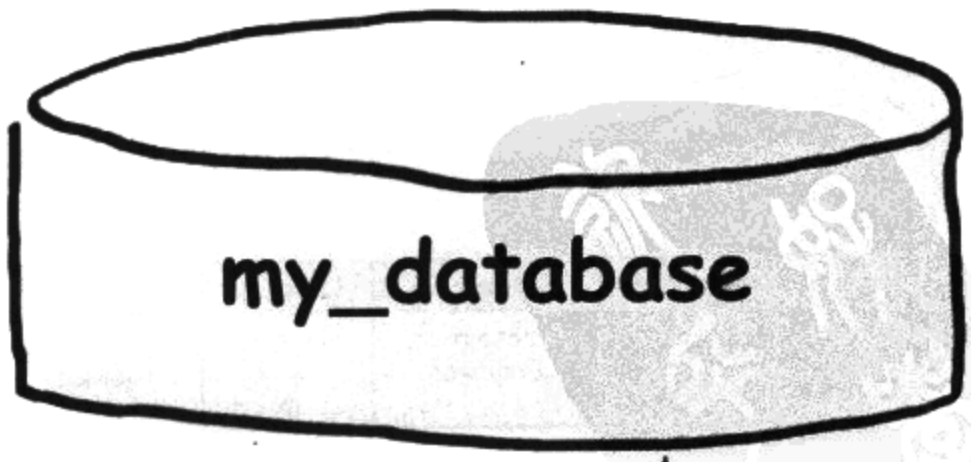
last_name	first_name	email	birthday	profession	location	status	interests	seeking
Branson	Ann	annie@boards-r-us.com	7-1-1962	Aeronautical Engineer	San Antonio, TX	Single, but involved	RPG, Programming	New Job
Hamilton	Jamie	dontbother@breakneckpizza.com	9-10-1966	System Administrator	Sunnyvale, CA	Single	Hiking, Writing	Friends, Women to date
Soukup	Alan	soukup@breakneckpizza.com	12-2-1975	Aeronautical Engineer	San Antonio, TX	Married	RPG, Programming	Nothing
Mendoza	Angelina	angelina@starbuzzcoffee.com	8-19-1979	Unix System Administrator	San Francisco, CA	Married	Acting, Dancing	New Job

什么是数据库？

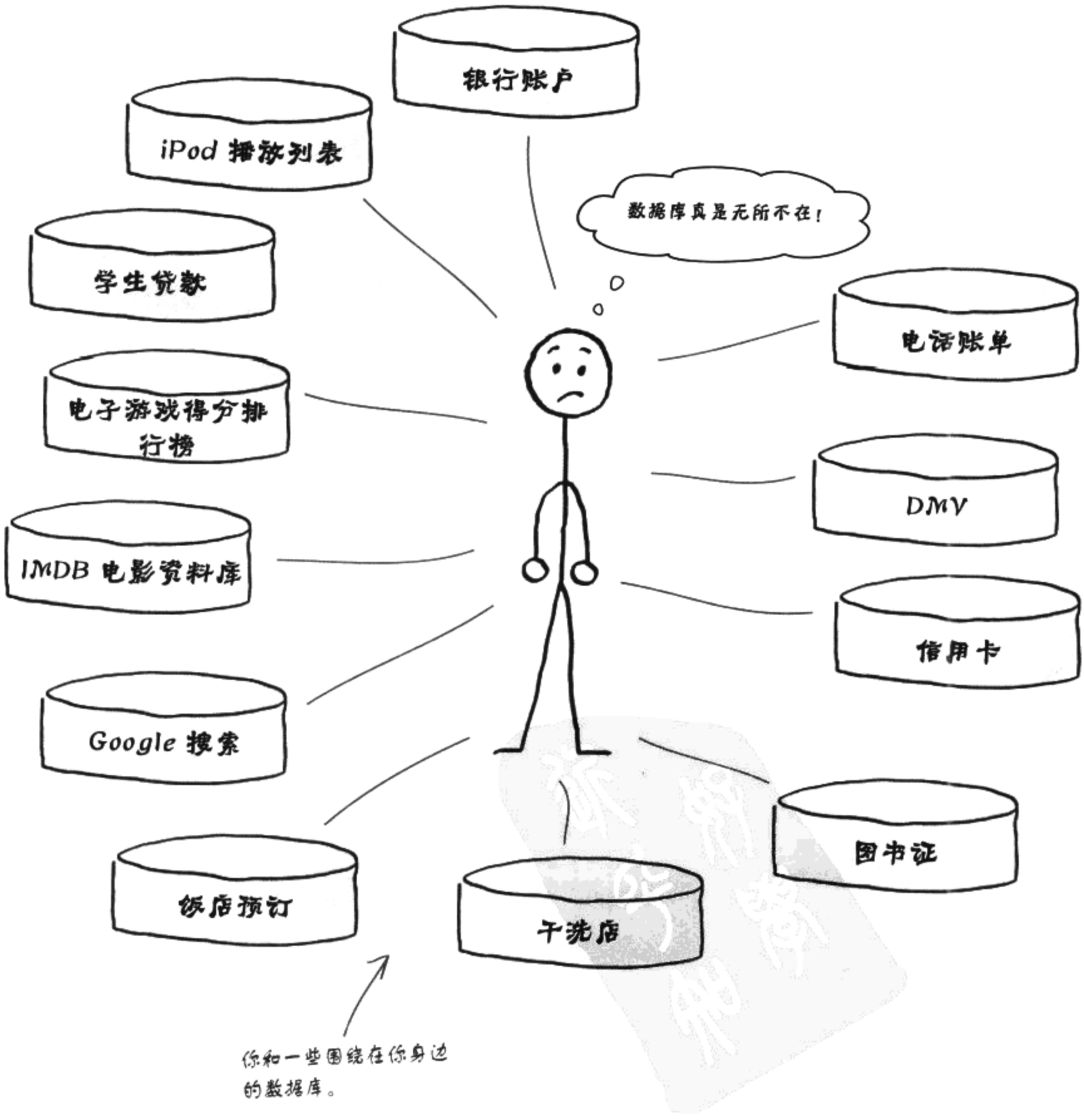
详谈什么是表、列、行的细节前，先向后退一步，看看大局。各位必须知道的第一个 SQL 结构其实是盛装所有表的**数据库**（database）。

数据库是保存表和其他相关SQL结构的容器。

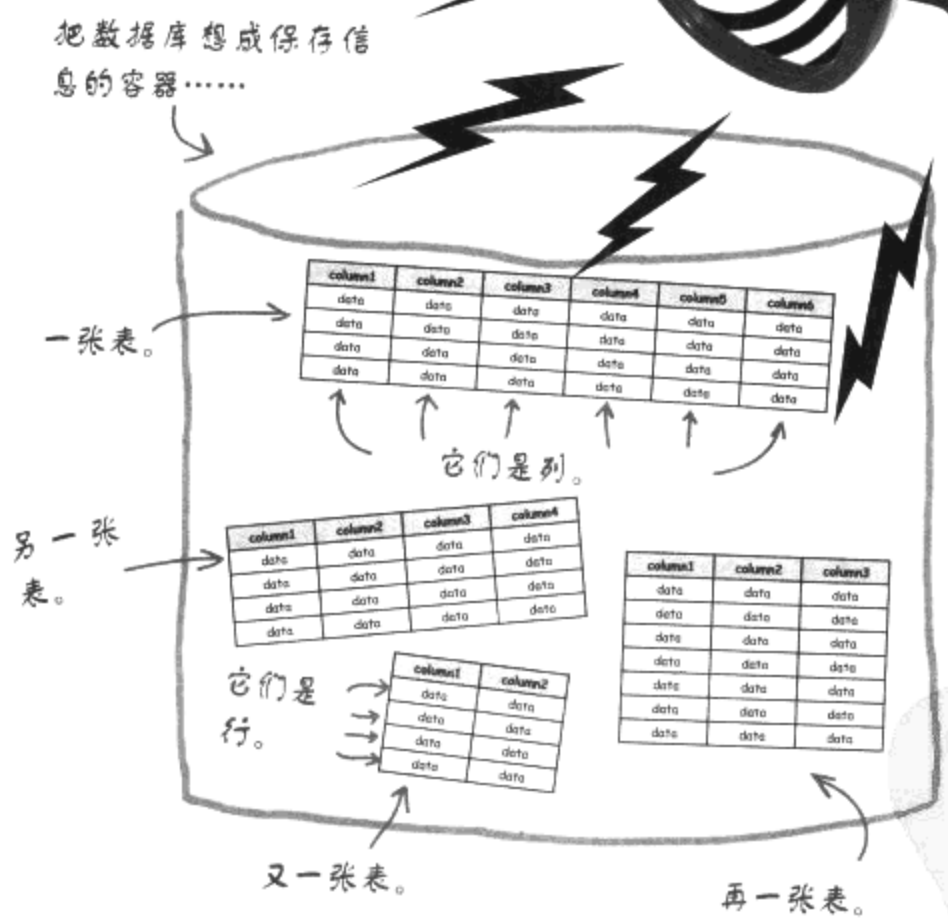
每次上网搜索、出门购物、询问资讯、使用 TiVo、预订房间或座位、收到超速罚单、购买杂货，你都会到某个数据库查找信息，这就是人称**查询**（query）的行为。



在流程图中，数据库都是以圆柱体表示。看到它，你就应该想到数据库。


银行账户
iPod 播放列表
数据库真是无所不在！
学生贷款
电话账单
电子游戏得分排行榜
DMV
IMDB 电影资料库
信用卡
Google 搜索
图书证
饭店预订
干洗店
你和一些围绕在你身边的数据库。

数据库解剖课



数据库内的信息组成了**表**。

数据库由表构成。

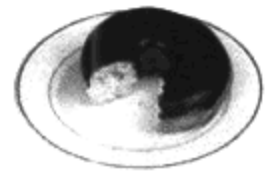
表是在数据库中包含数据的结构，由列和行组成。

还记得前面提到的分类吗？每个分类都变成表中的一**列**。Single、Married、Divorced等这些值可能都在相同列下。

表的**行**包含了表中某个对象的所有信息。在Greg的新表中，行是关于某个人的所有信息，例如：John、Jackson、single、writer、jj@boards-r-us.com。

与表天人合一

下面有几张便笺和一张表。你的工作就是把自己当成那份不完整的表，然后从便笺中取出信息填满表。做完功课后翻到下一页，看看你是否成功与表天人合一。



Duncan's Donuts

7

4/24

not enough jelly

10:35 pm

Jelly-filled

Starbuzz Coffee

4/23

jelly filled

9

7:43 am

almost perfect

Duncan's Donuts

5

4/25

Jelly-filled

8:56 am

greasy

Jelly-filled

stale, but tasty

6

Krispy King

4/26

9:39 pm

以某一列作为标题，给表一个有意义的名称。

shop				
			9	
		4/25	5	
				not enough jelly

与表天人合一的解答

你的工作就是把自己当成那份不完整的表，然后从便笺中取出信息填满表。

根据便笺的内容，各位应该能够判断出表的标题。

jelly_doughnuts

shop	time	date	rating	comments
Starbuzz Coffee	7:43 am	4/23	9	almost perfect
Duncan's Donuts	8:56 am	4/25	5	greasy
Krispy King	9:39 pm	4/26	6	stale, but tasty
Duncan's Donuts	10:35 pm	4/24	7	not enough jelly

如果你回答的列名与我们的答案有点不同也没关系。

数据库包含关联数据

数据库中所有的表应该能以某种方式相互关联。例如，一个关于甜甜圈的数据库可能包含了如下表：

这个数据库里有三张表。我把数据库命名为"my_snacks"。

数据库和表的名称不一定要大写。

my_snacks

记录果酱甜甜圈信息的表。

jelly_doughnuts

shop	time	date	rating	comments
Starbuzz Coffee	7:43 am	4/23	9	almost perfect
Duncan's Donuts	8:56 am	4/25	5	greasy
Krispy King	9:39 pm	4/26	6	stale, but tasty
Duncan's Donuts	10:35 pm	4/24	7	not enough jelly

记录糖衣甜甜圈信息的表。

glazed_doughnuts

shop	time	date	rating	comments
Krispy King	9:39 pm	4/26	8	warm, but not hot
Starbuzz Coffee	7:43 am	4/23	4	not enough glaze
Duncan's Donuts	8:56 am	4/25	6	greasy
Duncan's Donuts	10:35 pm	4/24	7	stale

记录非甜甜圈的甜点信息的表。

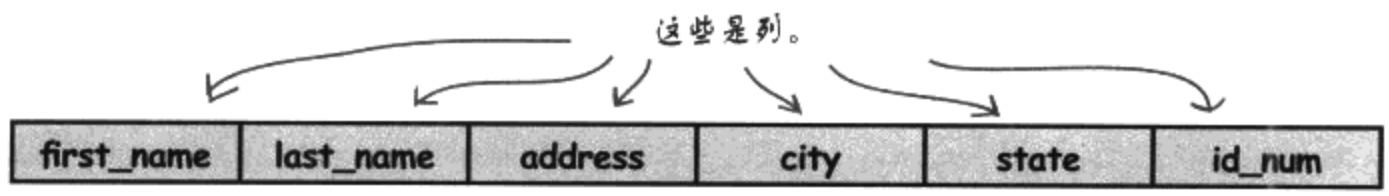
other_snacks

shop	time	date	cake	rating	comments
Starbuzz Coffee	10:35 pm	4/24	cinnamon cake	6	too much spice
Starbuzz Coffee	7:43 am	4/23	rocky road	8	marshmallows!
Krispy King	9:39 pm	4/26	trail bar	4	not enough fruit
Duncan's Donuts	8:56 am	4/25	plain cookie	9	warm, crumbly

放大表

列是存储在表中的一块数据。行是一组能够描述某个事物的列的集合。列和行构成了表。

下面是一张通讯录表，你可能看过类似的个人信息收集表格。另外，**字段**（field）也常用来代称**列**，**记录**（record）与**行**也常交替使用。



把列和行组合起来就是你要的表。

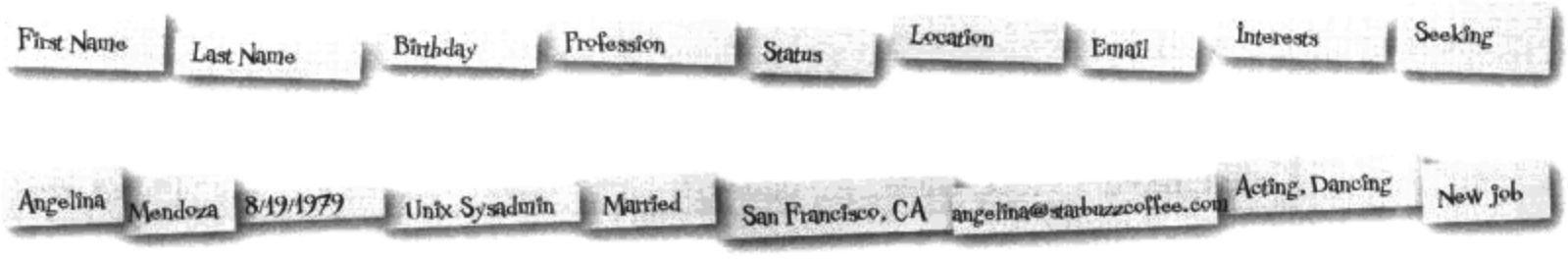
first_name	last_name	address	city	state	id_num
Joe	Epps	data	data	data	data
Al	Jones	data	data	data	data
Mary	Morris	data	data	data	data
Lou	Green	data	data	data	data

这么说，便笺上的数据已经可以形成一张表了？

没错！我们已经从收集到的个人信息中找出分类了。

每个种类变成列，每一张便笺则是一条记录。现在你可以把每张便笺的信息转录入表中了。

来自第7页的分类。



把一张便笺里的数据水平摊开形成一行。

现在各位应该知道这些分类被称为“列”了。

last_name	first_name	email	birthday	profession	location	status	interests	seeking
Branson	Ann	annie@boards-r-us.com	7-1-1962	Aeronautical Engineer	San Antonio, TX	Single, but involved	RPG, Programming	New Job
Hamilton	Jamie	dontbother@yahoo.com	9-10-1966	System Administrator	Sunnyvale, CA	Single	Hiking, Writing	Friends, Women to date
Soukup	Alan	fprose@yahoo.com	12-2-1975	Aeronautical Engineer	San Antonio, TX	Married	RPG, Programming	Nothing
Mendoza	Angelina	angel79@gmail.com	8-19-1979	Unix System Administrator	San Francisco, CA	Married	Acting, Dancing	New Job

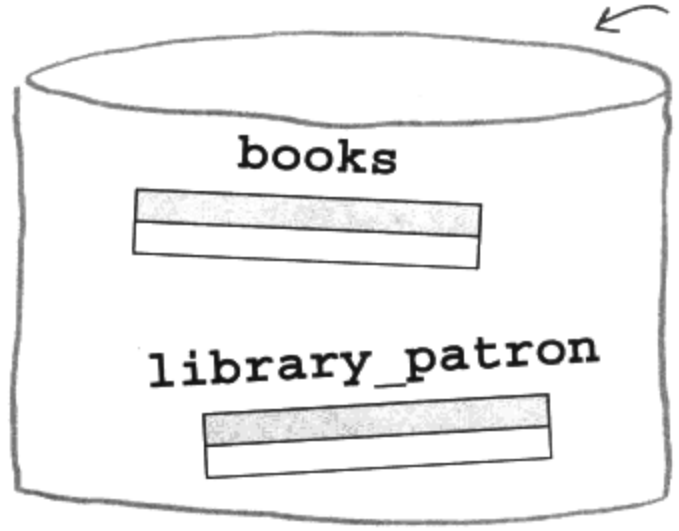
……每张便笺则可形成一行，也称为“记录”。

理论终于讲完了。究竟该如何创建表呢？

观察下列数据库和表。思考一下你能从中找出哪些数据分类，并为每张表定出可用的列。

library_db

图书馆的数据库。

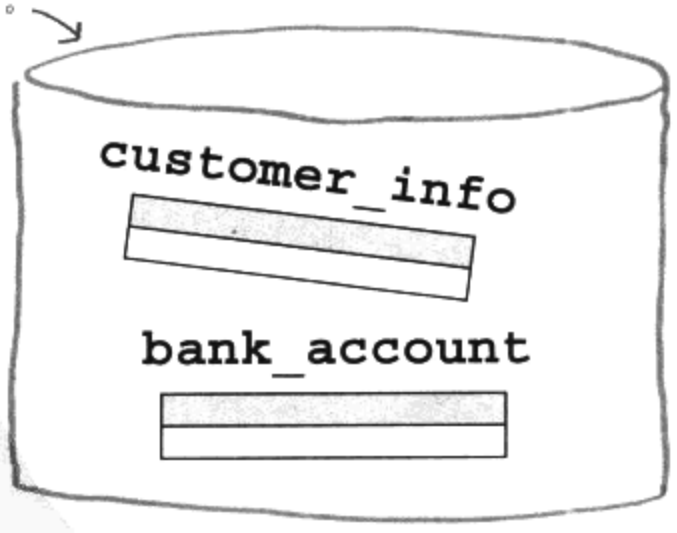


books: .....

library_patron: .....

bank_db

银行的数据库。



customer_info: .....

bank_account: .....

onlinestore_db

在线商店的数据库。

product_info: .....

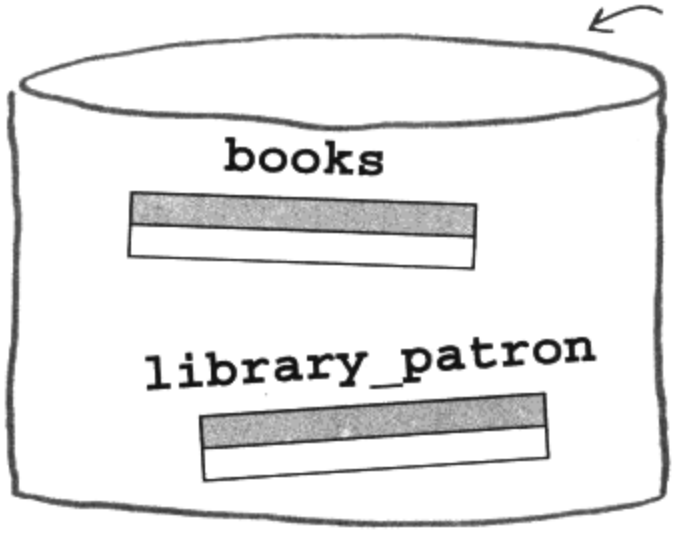
shopping_cart: .....

观察下列数据库和表。思考一下你能从中找出哪些数据分类，并为每张表定出可用的列。

library_db

图书馆的数据库。

你想到的列名与我们的答案有点不一样是没有关系的。

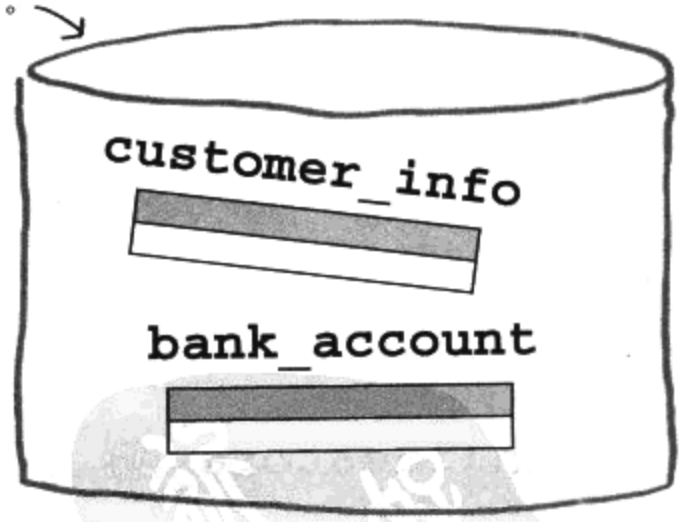


books: *title, author, cost, scan_code*

library_patron: *first_name, last_name, address*

bank_db

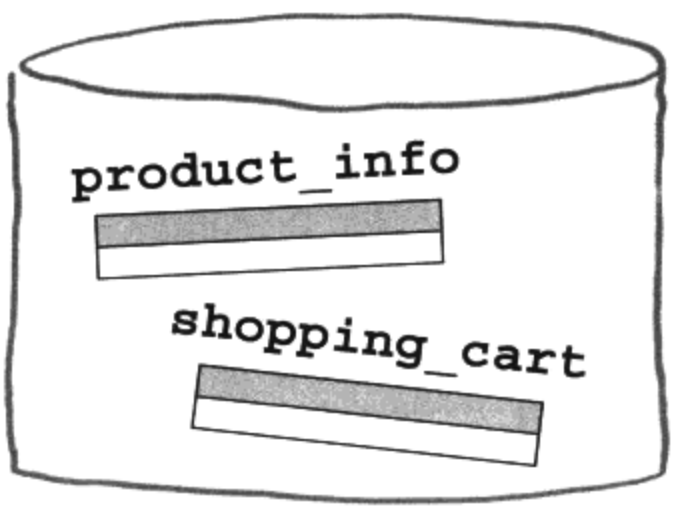
银行的数据库。



customer_info: *first_name, last_name, address, account_number, ssn*

bank_account: *balance, deposits, withdrawals*

onlinestore_db



在线商店的数据库。

product_info: *name, size, cost*

shopping_cart: *total_charge, customer_id*

接受命令！

启动你的 SQL 关系型数据库管理系统（relational database management system，RDBMS），打开命令行窗口或图形环境来让你与 RDBMS 沟通。下图是开启 MySQL 的终端窗口。

```
File Edit Window Help CommandMeBaby
Welcome to the SQL monitor. Commands end with ; or \g.

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

>
```

尖括号（>）是命令行提示符。在它后面输入 SQL 命令。

第一步，你需要创建用来装表的数据库。

SQL中的数据库和表的名称里不可出现空格，请以下划线取代空格。

❶ 输入如下代码，创建一个名为 **gregs _ list** 的数据库。

```
CREATE DATABASE gregs _ list;
```

CREATE DATABASE 是命令。

数据库的名称为 gregs_list。

命令必须以分号结束。

```
File Edit Window Help CommandMeBaby
> CREATE DATABASE gregs_list;
Query OK, 1 row affected (0.01 sec)
```

这一行是 RDBMS 的响应信息，让我们知道查询成功地执行了。

注意！

看过序了吗？

本书使用 MySQL 示范数据库命令，但你的数据库管理系统（DBMS）可能会用其他命令。关于 MySQL 的安装说明，请参考附录2。

❷ 现在则要告诉 RDBMS使用刚刚创建的数据库：

接下来我们做的每件事都是在 gregs_list 数据库中进行！

```
USE gregs _ list;
```

```
File Edit Window Help USEful
> USE gregs _ list;
Database changed
```

问： **如果我只有一张表，为什么我还要创建数据库？**

答： SQL 语言要求所有表都需放在数据库里。这项设计当然有它好的理由。SQL 能控制多位用户同时访问表的行为，能够授予或撤销对整个数据库的访问权，这有时比控制每张表的权限要简单很多。

问： **我发现CREATE DATABASE命令的字母全是大写，一定要这样吗？**

答： 有些系统确实要求某些关键字采用大写形式，但 SQL 本身不区分大小写。也就是说，命令不大写也可以，但命令大写是良好的 SQL 编程惯例。请看我们刚才键入的命令：

```
CREATE DATABASE
gregs _ list;
```

大写让我们很容易分辨命令（CREATE DATABASE）与数据库名称（gregs _ list）。

问： **给数据库、表和列命名时有什么注意事项吗？**

答： 创建具有描述性的名称通常有不错的效果。有时候要多用几个单词来命名。所有名称都不能包含空格，所以使用下划线能够让你创建更具描述性的名称。以下都是可以选用的名称：

```
gregs _ list
gregslist
Gregslist
gregsList
```

命名时最好避免首字母大写，因为 SQL 不区分大小写，极可能会搞错数据库。

问： **我就是想用“gregsList”，不加下划线，会发生什么事吗？**

答： 随便你。统一才是重点。如果用 gregsList 作为数据库的名称，最好在命名其表时也采用“不加下划线，第二个单词首字母大写”的惯例以求一致，例如 myContacts。

问： **数据库不是应该叫做greg's_list吗？为什么省略了代表所有格的撇号（'）呢？**

答： 在 SQL 中，撇号被保留起来另有特殊用途。当然也有很多把撇号加入名称的方式，不过还是省略它比较简单。

问： **我还发现了整段 CREATE DATABASE 命令后有个分号（;），它有什么作用呢？**

答： 分号用于表示命令的结束。

大写和下划线有助于编写 SQL 程序（虽然 SQL 不需要它们！）

设定表：CREATE TABLE 语句

让我们利用甜甜圈的数据，看看这一切如何运作。从甜甜圈的名称有时不太容易判断它的类型，或许你会考虑**创建一个表**来存储甜甜圈及其类型，而不是把整个价目表背下来。下面是一个要在命令行窗口中输入的命令，输入后按下 RETURN（ENTER），让你的 SQL RDBMS 执行此命令。

doughnut_list

doughnut_name	doughnut_type
Blooberry	filled
Cinnamondo	ring
Rockstar	cruller
Carameller	cruller
Appleblush	filled

它就是创建表的SQL命令——注意大写的应用。

此处的表名应该都用小写字母，并于应该是空格的地方采用下划线分隔。

按下 return/enter 即可在窗口中换行，让命令更容易阅读。

开括号打开要创建的列的列表。

逗号（,）用于区分新增的列。

表的第一列的名称。

第二列的名称。

```
CREATE TABLE doughnut _ list
(
  doughnut _ name VARCHAR(10),
  doughnut _ type VARCHAR(6)
);
```

闭括号关闭列的列表。

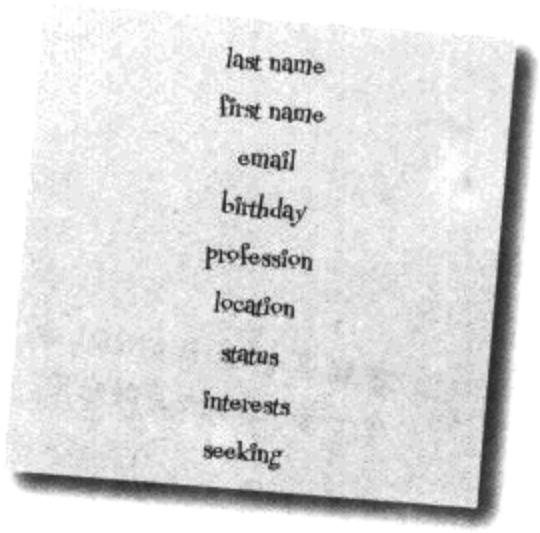
分号告诉 SQL RDBMS 这段命令已经结束了。

这就是**数据类型**（*data type*）。“VARCHAR”是可变动字符（VARiable CHARacter）的意思，用于保存以文本格式存储的信息。“(6)”是指这段文字的长度最多只能有6个字符。

喂，别忘了我啊！如何为我的 gregs_list 数据库 CREATE TABLE？

创建更复杂的表

还记得 Greg 的表的列名吗？写到另外一张便笺上，等一下我们会在 **CREATE TABLE** 时用到它们。



各位要使用 CREATE TABLE 命令，把这张纸……

……做成这样

last_name	first_name	email	birthday	profession	location	status	interests	seeking

动动脑

便笺与表格里的列名有哪两个地方不同？这有什么重要之处？

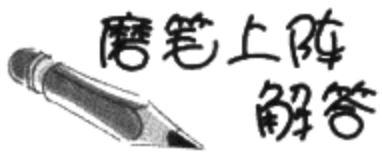
看，设计SQL是多么简单

各位已经知道如何从分类成列的数据中创建一张表。接下来需要确定每列的正确数据类型与长度。估算每列的长度后，写出 SQL 代码就很简单了。

磨笔上阵

下面的左侧是 Greg 的数据库所需的 CREATE TABLE 语句。猜猜看每一行命令的作用。记得为各列加上数据示例。

```
CREATE TABLE my_contacts
(
  last_name VARCHAR(30),
  first_name VARCHAR(20),
  email VARCHAR(50),
  birthday DATE,
  profession VARCHAR(50),
  location VARCHAR(50),
  status VARCHAR(20),
  interests VARCHAR(100),
  seeking VARCHAR(100)
);
```




```
CREATE TABLE my_contacts
(
  last_name VARCHAR(30),
  first_name VARCHAR(20),
  email VARCHAR(50),
  birthday DATE,
  profession VARCHAR(50),
  location VARCHAR(50),
  status VARCHAR(20),
  interests VARCHAR(100),
  seeking VARCHAR(100)
);
```

每一行 CREATE TABLE 命令的用途都写在下面，还有每列数据类型所需的示例。

创建名为 "my_contacts" 的表	
打开欲新增的列的列表	
加入名为 "last_name" 的列，最多可存储 30 个字符	'Anderson'
加入名为 "first_name" 的列，最多可存储 20 个字符	'Jillian'
加入名为 "email" 的列，最多可存储 50 个字符	'jill_anderson@breakneckpizza.com'
加入名为 "birthday" 的列，可存储日期数据	'1980-09-05'
加入名为 "profession" 的列，最多可存储 50 个字符	'Technical Writer'
加入名为 "location" 的列，最多可存储 50 个字符	'Palo Alto, CA'
加入名为 "status" 的列，最多可存储 20 个字符	'Single'
加入名为 "interests" 的列，最多可存储100 个字符	'Kayaking, Reptiles'
加入名为 "seeking" 的列，最多可存储 100 个字符	'Relationship, Friends'
结束列列表的新增，分号表示命令结束	

创建 my_contacts 表（终于！）

知道每一行的作用后，你就可以输入 **CREATE TABLE** 命令。你可以一次输入一行，就像上面的格式。

或写成非常长的一行：

```
CREATE TABLE my_contacts (last_name VARCHAR(30), first_name VARCHAR(20), email VARCHAR(50), birthday DATE, profession VARCHAR(50), location VARCHAR(50), status VARCHAR(20), interests VARCHAR(100), seeking VARCHAR(100));
```

相信我，这一行真的是命令，只有用很.)·很.)·很.)·的字排版才能塞入这一页！

无论各位喜欢哪种形式，在输入分号、按下 return/enter 前，请务必确认没有遗漏任何字符：

`last_name VARCHAR(3)` 与 `last_name VARCHAR(30)` 是两个完全不同的列！

您的表已经准备好了

```
File Edit Window Help AllDone
> CREATE TABLE my_contacts
    -> (
    ->  last_name VARCHAR(30),
    ->  first_name VARCHAR(20),
    ->  email VARCHAR(50),
    ->  birthday DATE,
    ->  profession VARCHAR(50),
    ->  location VARCHAR(50),
    ->  status VARCHAR(20),
    ->  interests VARCHAR(100),
    ->  seeking VARCHAR(100)
    -> );
Query OK, 0 rows affected (0.07 sec)
```

你有没有注意到，在输入分号后按下return/enter，就像告诉SQL RDBMS执行命令一样？

事实上，其他类的数据，例如数字，还需要其他数据类型。

假设我们在甜甜圈表里加入价格列。我们并不想用 VARCHAR 存储价格数据。以 VARCHAR 类型存储的数据会被解释为文字，当然也无法套用数学计算。不过，还有很多你没看到的数据类型呢……

动动脑

思考一下，还有哪些数据会需要VARCHAR和DATE以外的数据类型呢？

认识一下其他数据类型

这里有一些最有用的数据类型，它们的工作就是存储你的数据但不会破坏数据。各位已经见过VARCHAR小姐和DATE小姐了，也向其他人打声招呼吧！

这些数据类型的名称可能不适用于你的SQL RDBMS！

注意！ 很可惜，世界上没有统一的数据类型名称。你所用的 SQL RDBMS 可能会有某些与我们不同的数据类型名称。请参考你的文档，找出你的 RDBMS 所需的正确名称。

哪一种数据类型

为下列各列设计最合理的数据类型。同时填入其他缺漏的信息。

这里的两个数字表示数据库希望的浮点数格式，前者代表总位数，后者是小数点后的位数。

列名	说明	范例	最佳数据类型
price	某物品的售价	5678.39	DEC(6,2)
zip_code			
atomic_weight	原子量可能是小数点后超过6位的浮点数		
comments	一段文本，超过255个字符	Joe, I'm at the shareholder's meeting. They just gave a demo and there were rubber duckies flying around the screen. Was this your idea of a joke? You might want to spend some time on Monster.com.	
quantity	某项物品的库存量		
tax_rate		3.755	
book_title		Head First SQL	
gender	单个字符，F 或 M		CHAR(1)
phone_number	10位数，不加标点符号	2105552367	
state	两个字符，美国州名缩写	TX, CA	
anniversary		11/22/2006	DATE
games_won			INT
meeting_time		10:30 a.m. 4/12/2020	

没有蠢问题

问：为什么不能直接把 BLOB 当成所有文本值的类型？

答：因为这样很浪费空间。VARCHAR或CHAR只会占用特定空间，不会多于 256 个字符。但BLOB需要很大的存储空间。随着数据库的增长，占用存储空间就是冒着耗尽硬盘空间的风险。另外，有些重要的字符串运算无法操作BLOB类型的数据，只能用于VARCHAR 或CHAR（本章稍后另有说明）。

问：为什么需要 INT 和 DEC 这类数值类型呢？

答：答案还是跟节省数据库存储空间和效率有关。为表的每列选择最合适的数据类型可以为表瘦身，还可使数据操作更为快速。

问：这些就是所有的数据类型吗？

答：不是，但我们列出了最重要的几种类型。因为 RDBMS 的不同，数据类型也会稍有不同，详细的信息你最好翻一下说明文档。我们推荐《SQL 技术手册》，这本参考书列举了常用 RDBMS 的不同写法。

哪一种数据类型

为下列各列设计最合理的数据类型，同时填入其他缺漏的信息。

邮编不一定都是10个字符，所以采用VARCHAR来节省数据库的空间。各位也可以采用CHAR并指定所需长度。

列名	说明	范例	最佳数据类型
price	某物品的售价	5678.39	DEC(5,2)
zip_code	5至10个字符	90210-0010	VARCHAR(10)
atomic_weight	原子量可能是小数点后超过6位的浮点数	4.002602	DEC(10, 6)
comments	一段文本，超过255个字元	Joe, I'm at the shareholder's meeting. They just gave a demo and there were rubber duckies flying around the screen. Was this your idea of a joke? You might want to spend some time on Monster.com.	BLOB
quantity	某项物品的库存量	239	INT
tax_rate	百分比	3.755	DEC(5, 3)
book_title	文本字符串	Head First SQL	VARCHAR(50)
gender	单个字符，F或M	M	CHAR(1)
phone_number	10位数，不加标点符号	2105552367	CHAR(10)
state	两个字符，美国州名缩写	TX, CA	CHAR(2)
anniversary	月、日、年	11/22/2006	DATE
games_won	以数字表示获胜的比赛场数	15	INT
meeting_time	时间和日期	10:30 a.m. 4/12/2020	DATETIME

电话号码必为这个长度。而且，我们要把电话号码当成文本字符串，因为它虽然是数字，却不需要任何数学运算。

TIMESTAMP通常用于记录“当下”这个时刻。DATETIME更适合存储将来的时间。

复习要点

- 在创建表前先把数据分类。尤其要注意每列的数据类型。
- 使用 CREATE DATABASE 语句来创建存储所有表的数据库。
- 使用 USE DATABASE 语句进入数据库，然后创建表。
- 所有表都以 CREATE TABLE 语句创建，句中包含列名及其数据类型。
- 一些常用的数据类型有 CHAR、VARCHAR、BLOB、INT、DEC、DATE、DATETIME。每种数据类型的存储规则都不一样。

等一下，我刚刚在数据库 gregs_list 中创建的表呢？我想确认做得对不对。

啊，很好的建议。检查自己的工作也是很重要的一环。

想要检查刚才创建的my _ contacts表，可以使用DESC命令：

```
DESC my _ contacts;
```

DESC是DESCRIBE的缩写

试试看。

```
File Edit Window Help DescTidy
> DESC my _ contacts;
```

请看您的表

输入DESC命令与想要检查的表后，你应该会看到类似下图的结果。

现在别担心这些东西，等一下我们就会讲到。

```
File Edit Window Help DescTidy
>DESC my_contacts;
+------------+--------------+------+-----+---------+-------+
| Column     | Type         | Null | Key | Default | Extra |
+------------+--------------+------+-----+---------+-------+
| last_name  | varchar(30)  | YES  |     | NULL    |       |
| first_name | varchar(20)  | YES  |     | NULL    |       |
| email      | varchar(50)  | YES  |     | NULL    |       |
| birthday   | date         | YES  |     | NULL    |       |
| profession | varchar(50)  | YES  |     | NULL    |       |
| location   | varchar(50)  | YES  |     | NULL    |       |
| status     | varchar(20)  | YES  |     | NULL    |       |
| interests  | varchar(100) | YES  |     | NULL    |       |
| seeking    | varchar(100) | YES  |     | NULL    |       |
+------------+--------------+------+-----+---------+-------+
9 rows in set (0.07 sec)
```

动动脑

你觉得如何？添加新的列会带来多少问题呢？

SQL冰箱磁铁

创建一个带有性别列的表的 SQL 代码都在这里，只是很散乱地贴在我们的 SQL 冰箱上。你可以把 SQL 代码重组为正确的顺序吗？有些括号和分号因为太小而搞丢了，记得在需要的地方加上去哦！

email VARCHAR(50)

birthday DATE

USE gregs_list

first_name VARCHAR(20)
last_name VARCHAR(30)

interests VARCHAR(100)
seeking VARCHAR(100)

status VARCHAR(20)

CREATE DATABASE gregs_list

profession VARCHAR(50)
location VARCHAR(50)

CREATE TABLE my_contacts

gender CHAR(1)

完成练习后，请试着在你的 SQL 控制台里输入这些 SQL 代码，并且加入一个新的性别列！

SQL冰箱磁铁解答

你的任务是重新组合SQL代码片段，创建一个含有性别列的表。

gregs_list 已经在数据库中了。

这就是重组后的 SQL 代码，与你的答案比较一下，然后继续往下读……

```
CREATE DATABASE gregs_list;

USE gregs_list;

CREATE TABLE my_contacts
(
  last_name VARCHAR(20),
  first_name VARCHAR(30),
  email VARCHAR(50),
  birthday DATE,
  gender CHAR(1),
  profession VARCHAR(50),
  location VARCHAR(50),
  status VARCHAR(20),
  interests VARCHAR(100),
  seeking VARCHAR(100)
);
```

不可以重建已经存在的表或数据库！

你试过输入新的 CREATE TABLE 语句吗？如果试过了，你应该已经知道刚才的练习的解答无法帮助你添加新列至已经创建好的表中。

如果你真的不死心，可以输入上述的命令，你应该会看到类似下图的画面：

```
File Edit Window Help OhCrap!
> CREATE TABLE my_contacts
    -> (
    ->  last_name VARCHAR(30),
    ->  first_name VARCHAR(20),
    ->  email VARCHAR(50),
    ->  gender CHAR(1),
    ->  birthday DATE,
    ->  profession VARCHAR(50),
    ->  location VARCHAR(50),
    ->  status VARCHAR(20),
    ->  interests VARCHAR(100),
    ->  seeking VARCHAR(100)
    -> );
ERROR 1050 (42S01): Table 'my_contacts' already exists
```

这里是新添加的性别列。

哎呀，出现错误信息了。看来这个新表并未成功创建。

没有蠢问题

问： 刚才那个 SQL 磁铁的练习题为什么会得到错误信息？

答： 我们不能创建已经存在的表。一旦你创建了数据库，就不需要再次创建它。忘记加分号也是很可能发生的错误。还要记得检查一下有没有打错 SQL 的关键字。

问： 为什么在“seeking VARCHAR(100)”后面没有逗号，而其他列后面都有？

答： “seeking”已经是最后一列了，接下来是告诉 RDBMS 语句结束的括号，所以不需要加上逗号。

问： 快告诉我，到底能不能追加列，还是我非得重新开始啊？

答： 必须重新开始，不过在创建包含性别列的新表前，你还要摆脱旧表。因为当前表里没有内容，我们可以轻易地丢弃旧表，重新创建一份。

问： 如果表中已有内容，又需要添加新列，该怎么办？真的没有删除整张表以外的方式了吗？

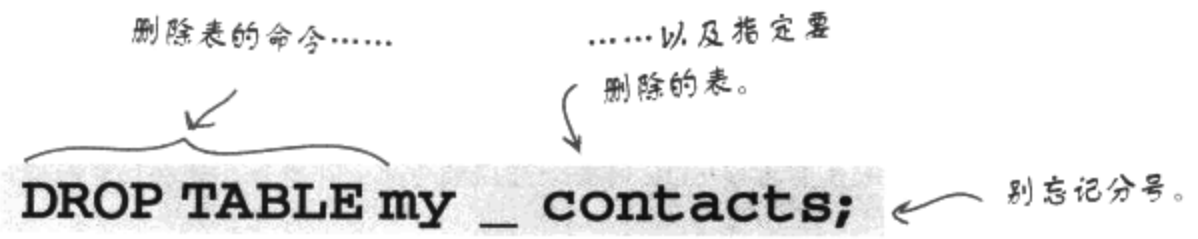
答： 好问题！的确有改变表且不需破坏其中数据的方式，这部分内容要过一阵子才会提到。但是就目前而言，表是空的，我们就删除这张表，重新创建一份吧！

这真是一个非常棒的想法！在后续章节里，你也会需要文本编辑软件。

如此一来，你就可以复制语句并粘贴到 SQL 控制台中，把我们从反复输入类似命令的深渊中解救出来。另外，复制、粘贴 SQL 代码也可帮助写出新的语句。

辞旧迎新

❶ 摆脱表比创建表简单很多。使用下面简单的指令：




```
File Edit Window Help ByeByeTable
> DROP TABLE my _ contacts;
Query OK, 0 rows affected (0.12 sec)
```

无论表里有无数据，DROP TABLE 都会执行，务必要非常小心谨慎地使用这个命令。一旦删除表后，它就随风而逝了，它里面的数据也会烟消云散。

DROP TABLE会删除你的表和表里面所有的数据！

❷ 现在可以输入新的CREATE TABLE语句了：

```
File Edit Window Help Success
> CREATE TABLE my_contacts
  -> (
  ->  last_name VARCHAR(30),
  ->  first_name VARCHAR(20),
  ->  email VARCHAR(50),
  ->  gender CHAR(1),
  ->  birthday DATE,
  ->  profession VARCHAR(50),
  ->  location VARCHAR(50),
  ->  status VARCHAR(20),
  ->  interests VARCHAR(100),
  ->  seeking VARCHAR(100)
  -> );
Query OK, 0 rows affected (0.05 sec)
```

这一次成功了。

有一群 SQL 关键字正在举行化装舞会，现在是“猜猜我是谁”的游戏时间。它们会给一些提示，让我们猜测它的身份。对了，所有关键字都会说实话。如果某些提示适用于多个关键字，请把所有符合的关键字都写下来。

今晚的贵宾：

CREATE TABLE、USE DATABASE、CREATE TABLE、DESC、DRAPTABLE、CHAR、VARCHAR、BLOB、DATE、DATETIEM、DEC、INT

我负责你的数字。

我可以扔掉你不想要的表。

T 或 F 是我的最爱。

我帮你记住你母亲的生日。

我手上拥有所有表。

我跟一般数字的感情不错，但我讨厌分数。

我喜欢长篇大论。

这里是存储所有东西的地方。

如果没有我，表根本不会存在。

我知道下星期的牙医门诊在什么时候。

会计师最爱我了。

我可以让你看到表的格式。

没有我们，你根本无法创建表。

贵宾名称

答案在第 51 页。

解剖SQL语句

为了把数据添加进表里，您需要 INSERT 语句

可以从字面上清楚地看出INSERT 的功能。请观察下面的语句，理解句中各部分有什么功用。第二组括号中的值**必须和列名的顺序相同。**

下述命令并非真正的命令，它只是展示 INSERT 语句格式的模板。

从关键字INSERT INTO开始语句。

表名。以 Greg 的数据库为例，此处应该替换为 my_contacts。

接下来的部分是列名，以逗号分隔。我们已经知道 Greg 的表中有一些像 first_name、last_name、email这样的列。

其他列名，最后一列后面不需要加逗号。

```
INSERT INTO your_table ( column_name1, column_name2, ··· )
       VALUES ( 'value1', 'value2', ··· );
```

另一个关键字。这表示接下来是各列的值。

接下来的部分是值的列表，以逗号分隔。以 Greg 的数据库为例，这些值应该来自他的便笺。

用单引号（'）是正确的。输入文本类型的数据时都要加上单引号，即使只是单一字符，如 'M' 或 'F'。

其他数据值，最后一个值后面不需要加逗号。

仍然要在语句结尾处加上分号。

注意：值的顺序必须与对应的列名的顺序一样！

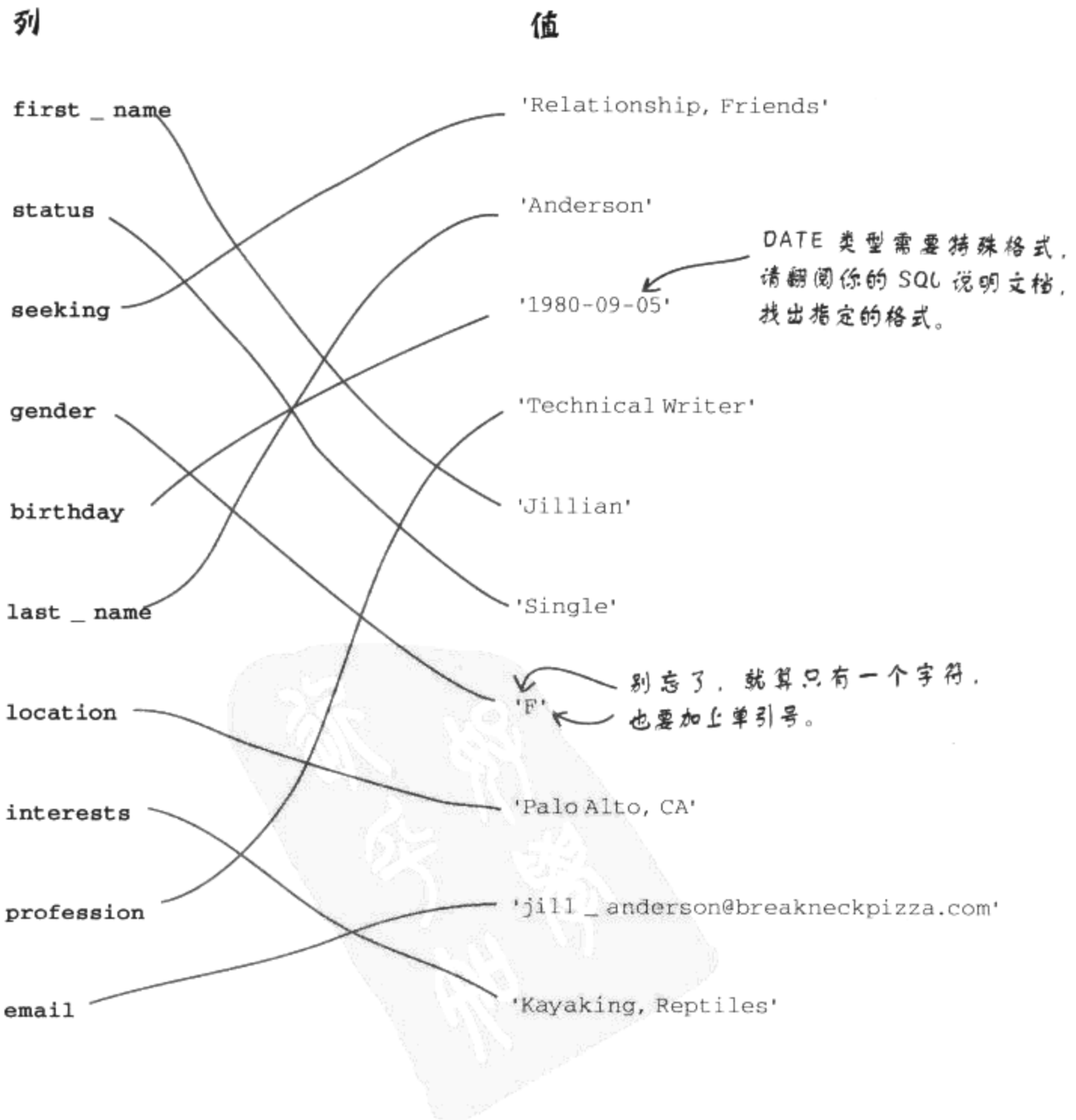
谁干的好事？

在动手写出 INSERT 语句前，我们需要把列名和数据值配成对。

列	值
first _ name	'Relationship, Friends'
status	'Anderson'
seeking	'1980-09-05'
gender	'Technical Writer'
birthday	'Jillian'
last _ name	'Single'
location	'F'
interests	'Palo Alto, CA'
profession	'jill _ anderson@breakneckpizza.com'
email	'Kayaking, Reptiles'

谁干的好事？

在动手写出 INSERT 语句前，我们需要把列名和数据值配成对。



创建INSERT语句

列名放在第一组括号里并以逗号分隔。

在输入开括号前，可以先按一下 return/enter，让 SQL 代码比在控制台窗口中更易于阅读。

```
INSERT INTO my_contacts
(last_name, first_name, email, gender, birthday,
profession, location, status, interests,
seeking)
VALUES
('Anderson', 'Jillian', 'jill_anderson@
breakneckpizza.com', 'F', '1980-09-05',
'Technical Writer', 'Palo Alto, CA', 'Single',
'Kayaking, Reptiles', 'Relationship, Friends');
```

在列名这组的结束括号后和关键字 VALUE 后都按一下 return/enter，一样能使 SQL 代码更容易阅读。

每列的值都由第二组括号内的内容提供，同样必须以逗号分隔。

任何属于 VARCHAR、CHAR、DATE、BLOB 列类型的值都需要加单引号。

注意！

顺序很重要！

数据值的顺序必须和**列名的顺序**完全一样。

在家试试看

上例是给表添加行的一种方式。试着自己输入。先在文本编辑器里试，这样如果打错字，至少不用重新输入。特别要注意单引号和逗号的存在。在此写下你从 SQL 控制台得到的响应：

你只是说 INSERT 语句中的 CHAR、VARCHAR、DATE、BLOB 的值需加上单引号，这表示 DEC、INT 等数值不需要加上单引号吗？

正是如此。

下例是可能用于甜甜圈订购表的INSERT语句。请注意，订购数量（以“一打”为单位）和价格列的值都没有加单引号。

dozens 列的类型是 INT，因为很少有人只买半打，所以此处不需要浮点数。

price列的类型是 DEC(4,2)，表示它有4位数，小数点后有2位数。

```
INSERT INTO doughnut_purchases
(donut_type, dozens, topping, price)
VALUES
('jelly', 3, 'sprinkles', 3.50);
```

插入 dozens 列和 prices 列的值，不需要单引号！

磨笔上阵

SQL RDBMS 会抱怨我们的语句写错了，但抱怨的内容可能有点不详细。请看下列 INSERT 语句。试着猜测每一组语句的问题，然后在 RDBMS 里输入，看看实际汇报的错误信息。

```
INSERT INTO my _ contacts
(last _ name, first _ name, email, gender, birthday, profession, location, status,
interests, seeking) VALUES ('Anderson', 'Jillian', 'jill _ anderson@breakneckpizza.com',
'F', '1980-09-05', 'Technical Writer', 'Single', 'Kayaking, Reptiles', 'Relationship,
Friends');
```

哪里出问题了？.....

RDBMS 的反应是：.....

```
INSERT INTO my _ contacts
(last _ name, first _ name, gender, birthday, profession, location, status, interests,
seeking) VALUES ('Anderson', 'Jillian', 'jill _ anderson@breakneckpizza.com', 'F',
'1980-09-05', 'Technical Writer', 'Palo Alto, CA', 'Single', 'Kayaking, Reptiles',
'Relationship, Friends');
```

哪里出现问题了？.....

RDBMS 的反应是：.....

```
INSERT INTO my _ contacts
(last _ name, first _ name, email, gender, birthday, profession, location, status,
interests, seeking) VALUES ('Anderson', 'Jillian', 'jill _ anderson@breakneckpizza.com',
'F', '1980-09-05', 'Technical Writer' 'Palo Alto, CA', 'Single', 'Kayaking, Reptiles',
'Relationship, Friends');
```

哪里出现问题了？.....

RDBMS 的反应是：.....

```
INSERT INTO my _ contacts
(last _ name, first _ name, email, gender, birthday, profession, location, status,
interests, seeking) VALUES ('Anderson', 'Jillian', 'jill _ anderson@breakneckpizza.com',
'F', '1980-09-05', 'Technical Writer', 'Palo Alto, CA', 'Single', 'Kayaking, Reptiles',
'Relationship, Friends);
```

哪里出现问题了？.....

RDBMS 的反应是：.....

.....

.....

如果这个练习题造成你的 RDBMS "挂" 在那边，试着在输入整段语句后多输入一个单引号并加上分号。

磨笔上阵解答

SQL RDBMS 会抱怨我们的语句写错了，但抱怨的内容可能有点不详细。请看下列 INSERT 语句。试着猜测每一组语句的问题，然后在 RDBMS 里输入，看看实际汇报的错误信息。

```
INSERT INTO my_contacts

(last_name, first_name, email, gender, birthday, profession, location, status,
 interests, seeking) VALUES ('Anderson', 'Jillian', 'jill_anderson@breakneckpizza.com',
'F', '1980-09-05', 'Technical Writer', 'Single', 'Kayaking, Reptiles', 'Relationship,
 Friends');
```

哪里出现问题了？ 缺少 location 值 ← 在列列表中有 location 列，但数据值列表中没有 location 值，少了一个值。

RDBMS 的反应是： ERROR 1136 (21S01): Column count doesn't match value count at row 1

注意这里，很多不同的问题却造成相同的错误信息。也请大家小心打错字的问题，这一点很难追踪。

```
INSERT INTO my_contacts

(last_name, first_name, gender, birthday, profession, location, status, interests,
 seeking) VALUES ('Anderson', 'Jillian', 'jill_anderson@breakneckpizza.com', 'F',
'1980-09-05', 'Technical Writer', 'Palo Alto, CA', 'Single', 'Kayaking, Reptiles', '
 Relationship, Friends');
```

哪里出现问题了？ 列列表中缺少 email列 ← 这次所有列都有对应的值，可是却忘了在列列表中列出 email 列。

RDBMS 的反应是： ERROR 1136 (21S01): Column count doesn't match value count at row 1

```
INSERT INTO my_contacts

(last_name, first_name, email, gender, birthday, profession, location, status,
 interests, seeking) VALUES ('Anderson', 'Jillian', 'jill_anderson@breakneckpizza.com',
'F', '1980-09-05', 'Technical Writer' 'Palo Alto, CA', 'Single', 'Kayaking, Reptiles',
'Relationship, Friends');
```

哪里出现问题了？ 两个值没有以逗号分隔 ← 在 'Technical Writer' 和 'Palo Alto, CA' 中间少了逗号

RDBMS 的反应是： ERROR 1136 (21S01): Column count doesn't match value count at row 1

```
INSERT INTO my_contacts

(last_name, first_name, email, gender, birthday, profession, location, status,
 interests, seeking) VALUES ('Anderson', 'Jillian', 'jill_anderson@breakneckpizza.com',
'F', '1980-09-05', 'Technical Writer', 'Palo Alto, CA', 'Single', 'Kayaking, Reptiles',
'Relationship, Friends);
```

哪里出现问题了？ 最后一个值后忘记加上单引号

RDBMS 的反应是： ERROR 1064 (42000): You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near '' at line 4

各种INSERT语句

有三种我们应该知道的 INSERT 语句形式。

❶ 改变列顺序

我们可以改变列名的顺序，只要记得数据值的顺序也要一起调整！

```
INSERT INTO my_contacts

(interests, first_name, last_name, gender, email, birthday,
profession, location, status, seeking)

VALUES

('Kayaking, Reptiles', 'Jillian', 'Anderson', 'F',
'jill_anderson@breakneckpizza.com', '1980-09-05', 'Technical
Writer', 'Palo Alto, CA', 'Single', 'Relationship, Friends');
```

注意到列名的顺序了吗？再看看数据值的顺序，它们的确以相同顺序排列。只要数据值和列名相互对应，INSERT 的顺序其实对你或对 SQL RDBMS 而言都不是问题！

❷ 省略列名

列名列表可以省略，但数据值必须**全部填入**，而且必须与当初创建表时的**列顺序完全相同**（不记得列顺序，请翻回第 37 页确认）。

```
INSERT INTO my_contacts

VALUES

('Anderson', 'Jillian', 'jill_anderson@breakneckpizza.com',
'F', '1980-09-05', 'Technical Writer', 'Palo Alto, CA',
'Single', 'Kayaking, Reptiles', 'Relationship, Friends');
```

我们省略了所有列名，但这么做时一定要填入所有数据值，而且要和表中的列顺序完全相同！

❸ 省略部分列

也可以只填入一部分列值就好了。

```
INSERT INTO my_contacts

(last_name, first_name, email)

VALUES

('Anderson', 'Jillian', 'jill_anderson@
breakneckpizza.com');
```

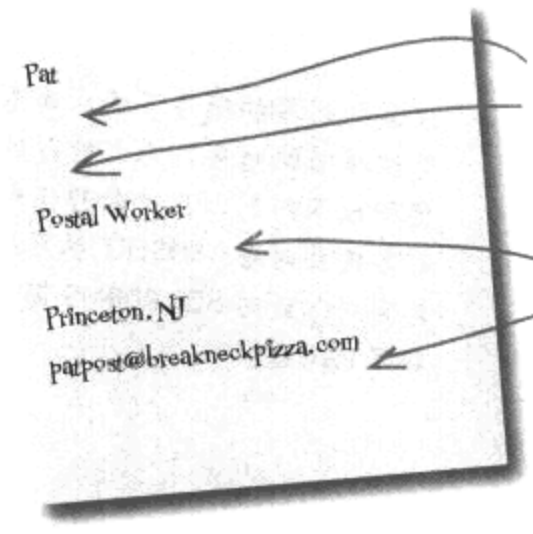
这一次，我们只输入部分数据。因为 SQL RDBMS 不知道输入的数据属于哪个列，所以我们需要明确指出数据值对应的列名。

动动脑

你觉得在同一张表中但没有赋值的列中会出现什么？

没有值的列

把这张信息不完整的便笺输入my _ contacts数据库。



缺少姓氏（last_name）和生日（birthday），而且也无法确定性别（gender）。

这里还少了婚姻状况（status）、兴趣（interests）和 seeking 列。

COLUMNS:	VALUES:
last _ name	?
first _ name	'Pat'
email	'patpost@breakneckpizza.com'
gender	?
birthday	?
profession	?
location	'Postal Worker'
status	'Princeton, NJ'
interests	?
seeking	?
	?

因为便笺缺少某些信息，Greg只好输入一条不完整的记录。没有关系，他可以日后再补上缺少的信息。

这里采用不需要提供所有列数据的 INSERT 形式，因为这样我们可以只提供已经知道值的列。

```
INSERT INTO my_contacts
(first_name, email, profession, location)
VALUES
('Pat', 'patpost@breakneckpizza.com', 'Postal
Worker', 'Princeton, NJ');
```

```
File Edit Window Help MoreDataPlease
> INSERT INTO my_contacts (first_name, email, profession,
location) VALUES ('Pat', 'patpost@breakneckpizza.com',
'Postal Worker', 'Princeton, NJ');

Query OK, 1 row affected (0.02 sec)
```

以SELECT语句窥探表

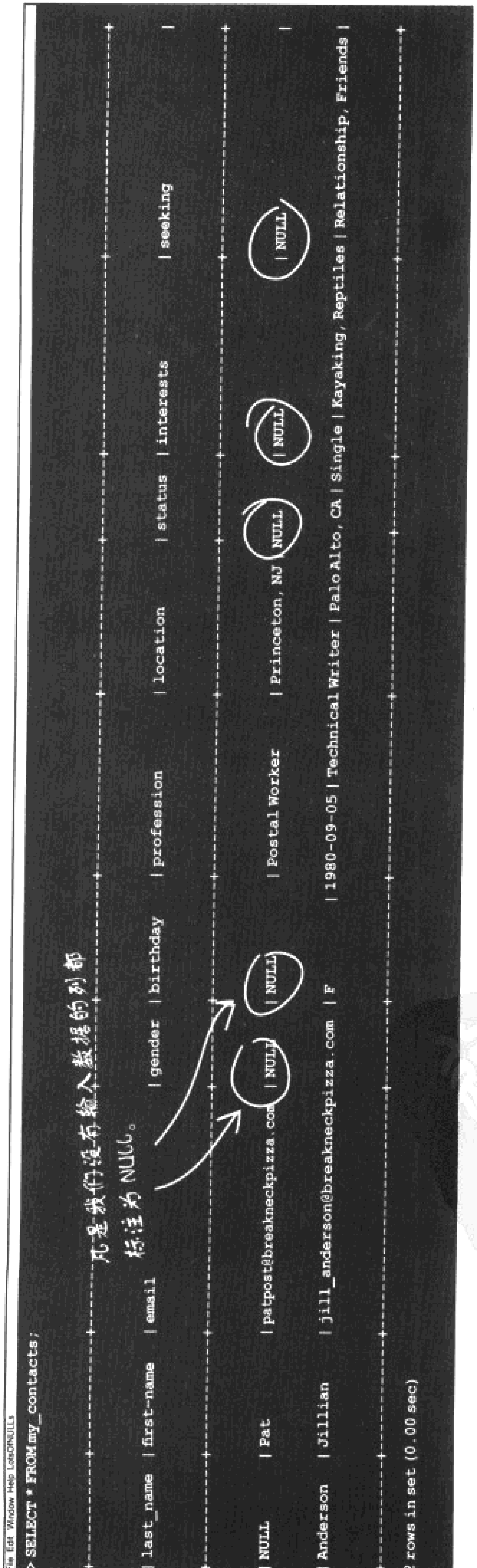
你现在很想看看输入数据后表的样子吧？嗯……不过DESC无法提供这方面的服务，因为它只负责表的结构，而非其中的数据内容。此时应该改用一个简单的SELECT语句才能看到表里的内容。

我想看看表中的所有数据……

……星号（*）代表选择所有内容。

这里是表名。

SELECT * FROM my_contacts;



放松

别担心SELECT做了哪些事。

第2章会详细讨论SELECT，现在，各位只要放松一点，享受由它所呈现出的表之美，这样就足够了。

现在动手试试看。你可能要把窗口拉长才能看到编排得如此整齐的画面。

动动脑

现在你知道NULL会出现在没有被赋值的列中。请问，你觉得NULL代表什么意思呢？

SQL真情指数

本周主题：
NULL的真情告白

Head First：欢迎你，NULL！我必须承认，没想到真的能见到你！我一直以为你根本不存在。大街小巷的传闻都说你其实和零差不多，甚至什么都没有。

NULL：主持大人，真不敢相信您也听信那些谣言！没错，我就在这里，货真价实地在这里！你还是觉得我什么都不是，还不如你脚下的灰尘吗？你说啊？

Head First：别激动、别激动。我也不是故意的，毕竟你都出现在某些没有值的地方嘛……

NULL：是，没错，可是我比零或空字符串强多了！

Head First：谁是空字符串啊？

NULL：例如只有两个单引号，中间什么都没有，而你把它当成数据值输入的情况。它还是个文本字符串，但长度为零。就像把 my_contacts 表的 first_name 值设为 ' '。

Head First：这么说来，你不是“什么都没有”的美称啰？

NULL：当然不是！！我从来就不等于零。而且我也不等于另一个 NULL，事实上，两个 NULL 根本不能放在一起比较。值可以**是** NULL，但它不会**等于** NULL，因为 NULL 代表未定义的值！理解了吗？

Head First：放轻松，别激动！所以说，你不等于零，你也不是空字符串变量。而且你甚至不等于你自己？这样真的说不通啊！

NULL：我知道这听起来有点混乱。这么说吧，我是未定义的。我就像身在一个未打开的盒子里。盒子里可能装有任何事物，所以无法比较两个未打开的盒子，因为我们根本不知道盒子里有什么。我甚至可能是个空盒子，但就是没人知道。

Head First：我也听说过有人不想用你，或许是因为你们 NULL 有时候会造成问题吧？

NULL：这点我承认。我曾经出现在自己也不想出现的地方。有些列一定要有值，例如 last_name，把 NULL 当成姓氏一点用也没有啊！

Head First：所以你不会随便跑到不想现身的地方啰？

NULL：当然！我是很好说话的！只要你在创建表时设置好不要我出现的列，我就不会出现。以 NULL 的信用保证，绝不食言！

Head First：你看起来不太像未打开的盒子！

NULL：……访问够了，我很忙，还要赶着去当数据值呢！

控制内心的NULL

在我们的表中，有些列应该一定要有数据值。还记得 Pat 的不完整便利贴吗？居然没有姓！当表中有12人的姓的记录都是NULL时，肯定很难找到正确的人。我们可以轻易地把列改为不接受 NULL。

```
CREATE TABLE my _ contacts
(
    last _ name VARCHAR (30) NOT NULL,
    first _ name VARCHAR (20) NOT NULL
);
```

只要在数据类型后加入 NOT NULL 就可以了。

如果你这么做了，在 INSERT 语句中一定要提供 NOT NULL 列的值，否则就会跳出错误信息。

磨笔上阵

```
CREATE TABLE my_contacts
(
    last_name VARCHAR(30) NOT NULL,
    first_name VARCHAR(20) NOT NULL,
    email VARCHAR(50),
    gender CHAR(1),
    birthday DATE,
    profession VARCHAR(50),
    location VARCHAR(50),
    status VARCHAR(20),
    interests VARCHAR(100),
    seeking VARCHAR(100)
);
```

请检查 my_contacts 的 CREATE TABLE 命令。哪些列应该加上 NOT NULL？请找出不应该填入 NULL 的列并把它们圈起来。

我们给出了两个不该为 NULL 的列，请继续我们的工作。在思考时，主要应该考虑列是否会用于后续搜索或者列是否具有唯一性。

磨笔上阵解答

```
CREATE TABLE my _ contacts
(
  last _ name VARCHAR(30) NOT NULL,
  first _ name VARCHAR(20) NOT NULL,
  email VARCHAR(50),
  gender CHAR(1),
  birthday DATE,
  profession VARCHAR(50),
  location VARCHAR(50),
  status VARCHAR(20),
  interests VARCHAR(100),
  seeking VARCHAR(100)
);
```

请检查 my_contacts 的 CREATE TABLE 命令。哪些列应该加上 NOT NULL？请找出不应该填入 NULL 的列并把它们圈起来。

我们给出了两个不该为 NULL 的列，请继续我们的工作。在思考时，主要应该考虑列是否会用于后继搜索或者列是否具有唯一性。

所有列都不该为 NULL。

因为我们会利用所有列进行搜索。所以，确保记录的完整性、让表存储良好的数据都是重要的课题……

……但是，如果你遇到以后才需要输入数据的列时，或许就会允许 NULL 的存在了。

NOT NULL 出现在 DESC 的结果中

下图是把my_contacts表的每一列都设置成NOT NULL的结果。

这里用来创建每列都 NOT NULL 的表。

```
File Edit Window Help NoMoreNULLs
CREATE TABLE my _ contacts
(
    last _ name VARCHAR(30) NOT NULL,
    first _ name VARCHAR(20) NOT NULL,
    email VARCHAR(50) NOT NULL,
    gender CHAR(1) NOT NULL,
    birthday DATE NOT NULL,
    profession VARCHAR(50) NOT NULL,
    location VARCHAR(50) NOT NULL,
    status VARCHAR(20) NOT NULL,
    interests VARCHAR(100) NOT NULL,
    seeking VARCHAR(100) NOT NULL
);
Query OK, 0 rows affected (0.01 sec)

> DESC my _ contacts;
+------------+--------------+------+-----+---------+-------+
| Column     | Type         | Null | Key | Default | Extra |
+------------+--------------+------+-----+---------+-------+
| last _ name | varchar(30) | NO   |     |         |       |
| first _ name | varchar(20) | NO   |     |         |       |
| email      | varchar(50)  | NO   |     |         |       |
| gender     | char(1)      | NO   |     |         |       |
| birthday   | date         | NO   |     |         |       |
| profession | varchar(50)  | NO   |     |         |       |
| location   | varchar(50)  | NO   |     |         |       |
| status     | varchar(20)  | NO   |     |         |       |
| interests  | varchar(100) | NO   |     |         |       |
| seeking    | varchar(100) | NO   |     |         |       |
+------------+--------------+------+-----+---------+-------+
10 rows in set (0.02 sec)
```

这里是表的构造说明。请注意 NULL 下都是 NO。

用DEFAULT填补空白

如果某些列通常有某个特定值，我们就可以把特定值指派为**DEFAULT默认值**。

跟在 DEFAULT 关键字后的值会在每次新增记录时自动插入表中 —— **只要没有另外指派其他值**。默认值的类型必须和列的类型相同。

```
CREATE TABLE doughnut_list
(
 doughnut_name VARCHAR(10) NOT NULL,
 doughnut_type VARCHAR(6) NOT NULL,
 doughnut_cost DEC(3,2) NOT NULL DEFAULT 1.00
);
```

我们希望能够确保这一列有值，但不只是 NOT NULL而已，也可以指派默认值(DEFAULT)为$1。

若是没有指派其他值，这就是插入表里的 doughnut_cost 列的值。

doughnut_list

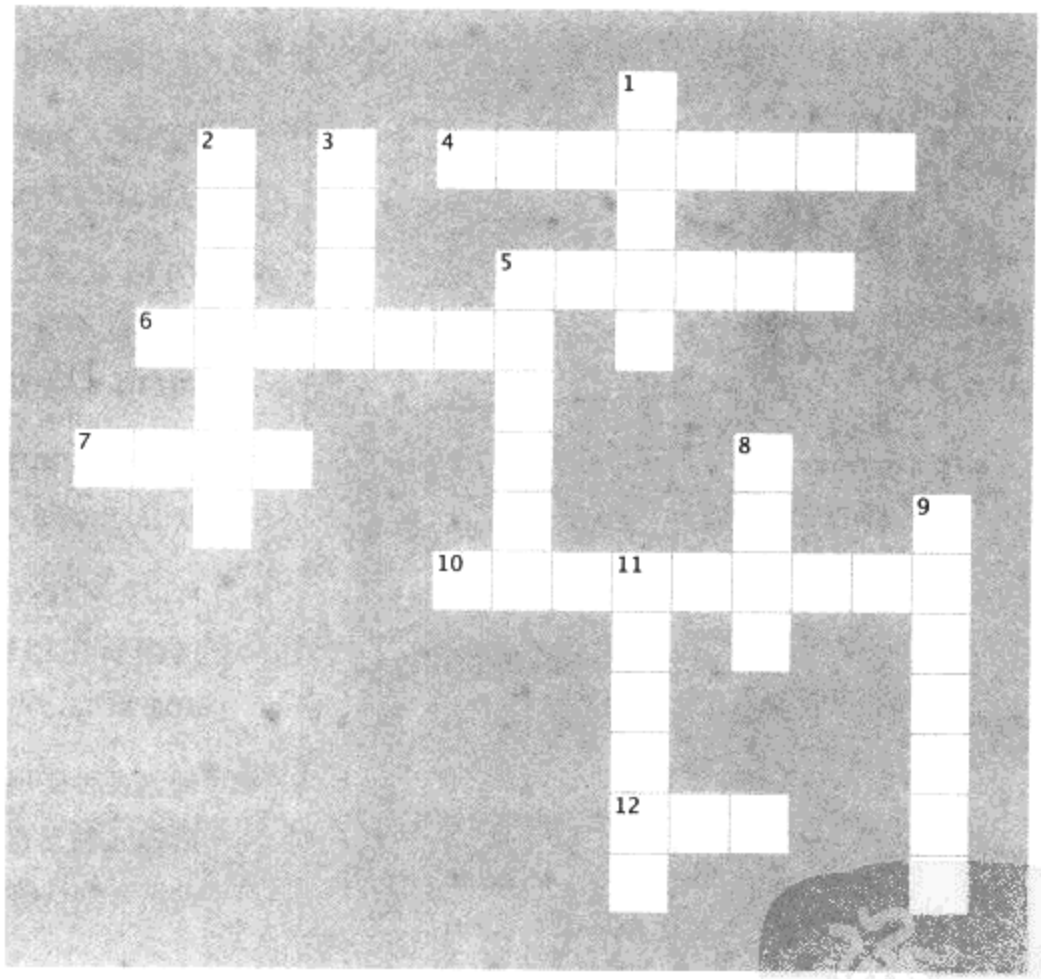
doughnut_name	doughnut_type	doughnut_cost
Blooberry	filled	2.00
Cinnamondo	ring	1.00
Rockstar	cruller	1.00
Carameller	cruller	1.00
Appleblush	filled	1.40

如果留下 doughnut_cost 不填的话，这就是它们在表中看起来的样子。本例中 Cinnamondo、Rockstar、Carameller 的列都没有填入 doughnut_cost 的值。

使用DEFAULT值填满空白列的值。

SQL填字游戏

让我们的左脑也运动一下吧！下面是个典型的填字游戏；所有解答的词条都曾在本章出现过。



横向

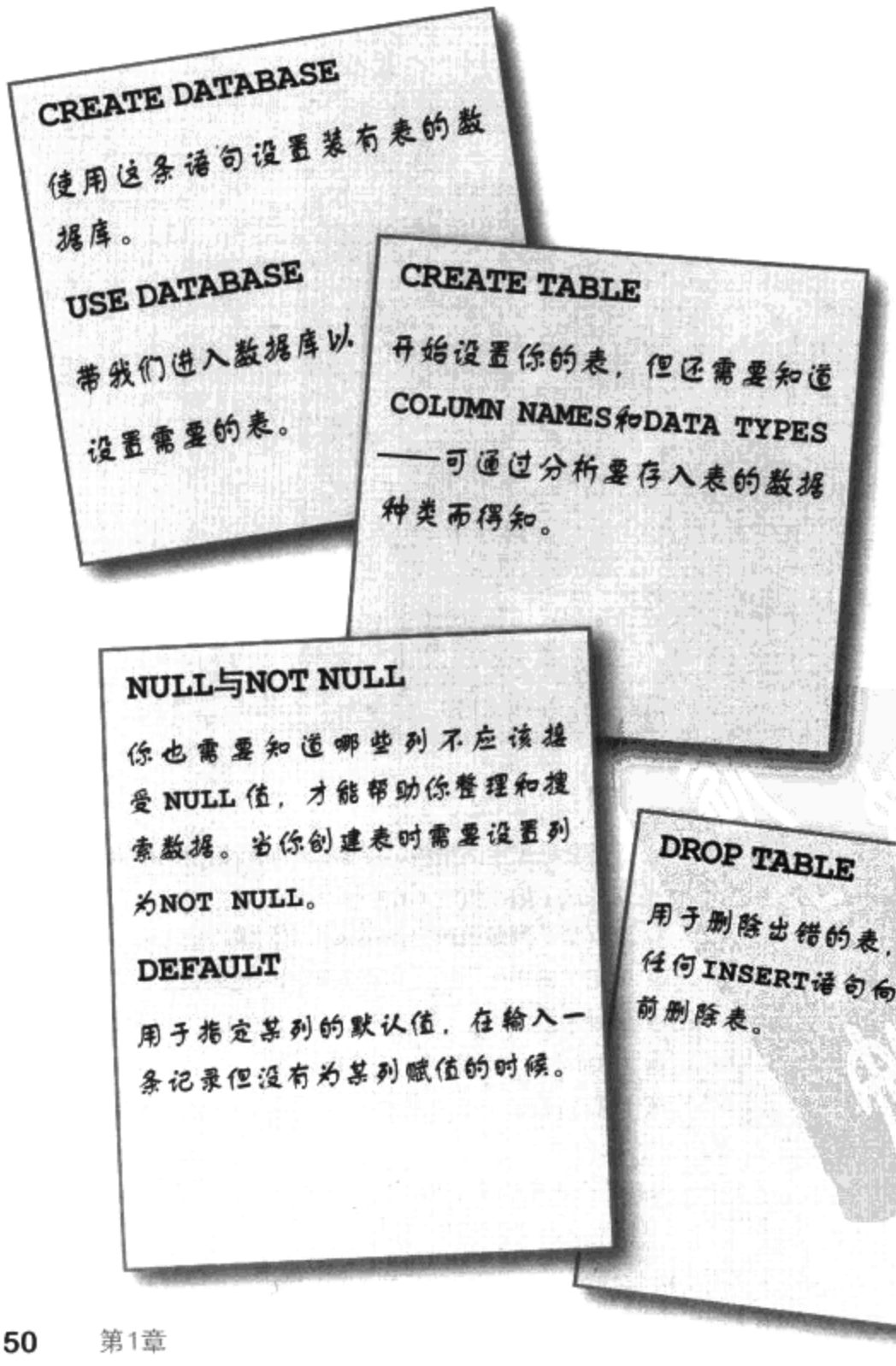
4. _____就像是个容器，里面装着表，还有其他与表相关的 SQL 结构。
5. _____是表存储的一格数据。
6. 它可以存储文本字符，最多可达 255 个字符。
7. 我们不可以比较两个_____。
10. 每条 SQL 语句都以它结尾。
12. 一组列，可以形容某物的属性。

纵向

1. 这是数据库中的结构，存储以列和行组成的数据。
2. 在你的 CREATE TABLE 语句中使用_____，可以在没有给某列赋值时自动填入指定的值。
3. 使用关键字_____可查看刚刚创建的表。
5. 关键字_____可以用在 TABLE 或 DATABASE 前。
8. 想删除表，就用_____ TABLE。
9. 这种数据类型认为数字应该完整，但又不畏惧面对负数。
11. 想在表中添加数据，应该使用_____语句。

你的SQL工具包

第1章已被收进你的工具包，而且你已经知道该如何创建数据库和表，也知道如何在表中插入最常见的数据类型的同时保证需要值的列有值。



复习要点

- 如果想查看表的结构，可以使用**DESC**语句。
- **DRAP TABLE**语句可用于丢弃表。谨慎使用！
- 为表插入数据时，可以使用任何一种**INSERT**语句。
- **NULL**是未定义的值。它不等于零，也不等于空值。值可以是**NULL**，但绝非等于**NULL**。
- 没有在**INSERT**语句中被赋值的列默认为**NULL**。
- 可以把列修改为不接受**NULL**值，这需要在创建表时使用关键字**NOT NULL**。
- 创建表时使用**DEFAULT**，可于日后输入缺乏部分数据的记录时自动填入默认值。

猜猜我是谁

有一群 SQL 关键字正在举行化装舞会，现在是“猜猜我是谁”的游戏时间。它们会给一些提示，让我们猜测它的身份。对了，所有关键字都会说实话。如果某些提示适用于多个关键字，请把所有符合的关键字都写下来。

今晚的来宾：

CREATE TABLE、USE DATABASE、CREATE TABLE、DESC、DRAPTABLE、CHAR、VARCHAR、BLOB、DATE、DATETIEM、DEC、INT

我负责你的数字。

我可以扔掉你不想要的表。

T 或 F 是我的最爱。

我帮你记住你母亲的生日。

我手上拥有所有表。

我跟一般数字的感情不错，但我讨厌分数。

我喜欢长篇大论。

这里是存储所有东西的地方。

如果没有我，表根本不会存在。

我知道下星期的牙医门诊在什么时候。

会计师最爱我了。

我可以让你看到表的格式。

没有我们，你根本无法创建表。

贵宾名称

DEC, INT

DROP TABLE

CHAR(1) ← 如果你加上 (1)，额外赠送一分哦！

DATE

CREATE DATABASE

INT

BLOB

CREATE TABLE

CREATE DATABASE

DATETIME

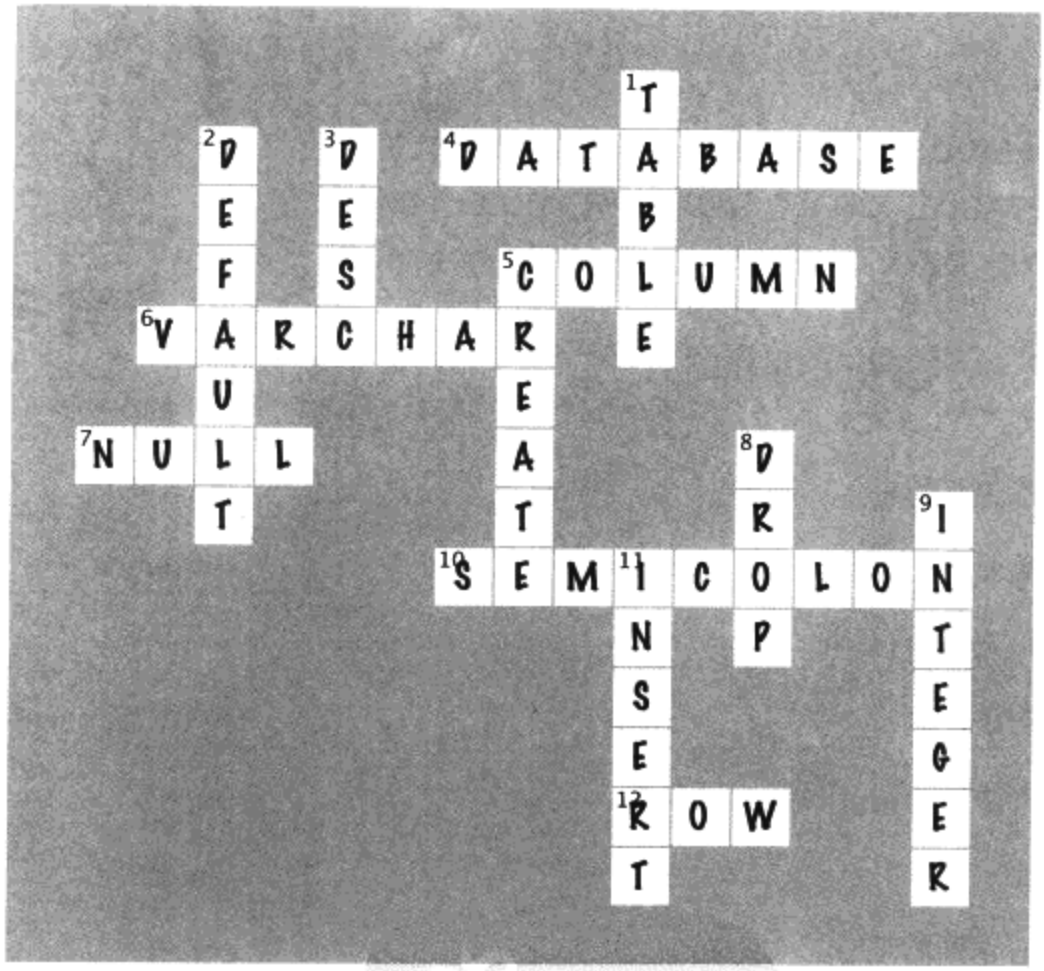
DEC

DESC

CREATE DATABASE, USE DATABASE

DROP TABLE

数据和表填字游戏解答


1 T
2 D
3 D
4 D A T A B A S E
E
E
B
F
S
5 C O L U M N
6 V A R C H A R
E
U
E
7 N U L L
A
8 D
T
T
R
9 I
10 S E M I C O L O N
N
P
T
S
E
E
G
12 R O W
E
T
R

2 SELECT 语句

取得精美包装里的数据

施予真的胜过取得吗？在数据库的世界里，**取得数据**的需求很可能与插入数据的需求一样多。这就是本章的目的：各位将见识到功能非常强大的 **SELECT** 语句，并且学习到**如何取得放在表里的重要信息**。我们还会学到如何利用 **WHERE**、**AND**、**OR** 选择数据，甚至可以过滤掉不想选择的数据。

要约会吗？

Greg刚把所有便笺上的数据输入my_contacts表中。他想好好地休息一下。Greg刚好有两张音乐会门票，他想邀请一位联络清单上的女孩，她住在San Francisco。

Greg需要找出她的电子邮件地址，所以他用第1章提过的**SELECT**语句来查看表。

```
SELECT * from my _ contacts;
```

她的详细信息在Greg的表里……的某个地方。

Greg的困境

现在假设你是Greg，请搜索右边那页上的my_contacts表的部分数据，寻找住在San Francisco的Anne。

my_contacts 表有好几列，这里只列出最前面的部分。

File Edit Window Help AnneWho

last_name	first_name	email	gender	location
Anderson	Jillian	jill_anderson@breakneckpizza.com	F	Palo Alto, CA
Joffe	Kevin	joffe@simuduck.com	M	San Jose, CA
Newsome	Amanda	aman2luv@breakneckpizza.com	F	San Fran, CA
Garcia	Ed	ed99@b0tt0msup.com	M	San Mateo, CA
Roundtree	Jo-Ann	jojoround@breakneckpizza.com	F	San Fran, CA
Briggs	Chris	cbriggs@boards-r-us.com	M	Austin, TX
Harte	Lloyd	hovercraft@breakneckpizza.com	M	San Jose, CA
Toth	Anne	Anne_Toth@leapinlimos.com	F	San Fran, CA
Wiley	Andrew	andrewwiley@objectville.net	M	NYC, NY
Palumbo	Tom	palofmine@mightygumball.net	M	Princeton, NJ
Ryan	Alanna	angrypirate@breakneckpizza.com	F	San Fran, CA
McKinney	Clay	clay@starbuzzcoffee.com	M	NYC, NY
Meeker	Ann	ann_meeker@chocoholic-inc.com	F	San Fran, CA
Powers	Brian	bp@honey-doit.com	M	Napa, CA
Manson	Anne	am86@objectville.net	M	Seattle, WA
Mandel	Debra	debmonster@breakneckpizza.com	F	Natchez, MS
Tedesco	Janis	janistedesco@starbuzzcoffee.com	F	Las Vegas, NV
Talwar	Vikram	vikt@starbuzzcoffee.com	M	Palo Alto, CA
Szwed	Joe	szwed_joe@objectville.net	M	NYC, NY
Sheridan	Diana	sheridi@mightygumball.net	F	Phoenix, AZ
Snow	Edward	snowman@tikibeanlounge.com	M	Fargo, ND
Otto	Glenn	glenn0098@objectville.net	M	Boulder, CO
Hardy	Anne	anneh@b0tt0msup.com	F	San Fran, CA
Deal	Mary	nobigdeal@starbuzzcoffee.com	F	Boston, MA
Jagel	Ann	dreamgirl@breakneckpizza.com	F	San Fran, CA
Melfi	James	drmelfi@b0tt0msup.com	M	Dallas, TX
Oliver	Lee	lee_oliver@weatherorama.com	M	St. Louis, MO
Parker	Anne	annep@starbuzzcoffee.com	F	San Fran, CA
Ricci	Peter	ricciman@tikibeanlounge.com	M	Reno, NV
Reno	Grace	grace23@objectville.net	F	Palo Alto, CA
Moss	Zelda	zelda@weatherorama.com	F	Sunnyvale, CA
Day	Clifford	cliffnight@breakneckpizza.com	M	Chester, NJ
Bolger	Joyce	joyce@chocoholic-inc.com	F	Austin, TX
Blunt	Anne	anneblunt@breakneckpizza.com	F	San Fran, CA
Bolling	Lindy	lindy@tikibeanlounge.com	F	San Diego, CA
Gares	Fred	fgares@objectville.net	M	San Jose, CA
Jacobs	Anne	anne99@objectville.net	F	San Jose, CA
Ingram	Dean	deaningram@breakneckpizza.com	M	Miami, FL

表还有没结束哦！Greg 有非常多的便笺。

Greg 的困境解答

现在假设你是 Greg，请搜索右边那页上的 my_contacts 表的部分数据，寻找住在 San Francisco 的 Anne。

你必须找出所有住在 San Francisco 的 Anne，并写下她们的姓名和电子邮件地址。

Toth, Anne: Anne_Toth@leapinlimos.com

Hardy, Anne: anneh@b0tt0msup.com

Parker, Anne: annep@starbuzzcoffee.com

Blunt, Anne: anneblunt@breakneckpizza.com

这些就是所有 Anne 和她们的电子邮件地址。

Greg要找Anne，不是Ann，所以凡是关于 Ann 的记录都要省略。

动手联络

这条寻人之路真是永无止境又极度无聊。而且 Greg 可能错过了其他条件相符的 Anne，很可能刚好错过他真正想找的那位 Anne。

总而言之，Greg 给每位他找到的 Anne发送电子邮件，然后收到下面这些回应……

To: Toth, Anne <Anne_Toth@leapinlimos.com>
From: Greg <greg@gregslist.com>
Subject: Did we meet at Starbuzz?

我已经有男朋友了，他叫 Tim。另外，我大概是在

To: Blunt, Anne <anneblunt@breakneckpizza.com>
From: Greg <greg@gregslist.com>
Subject: Did we meet at Starbuzz?

我要找的就是像你这样的牛仔帅哥！5点过来接我
咱们去找点好吃的！

To: Hardy, Anne <anneh@b0tt0msup.com>
From: Greg <greg@gregslist.com>
Subject: Did we meet at Starbuzz?

我不是你要找的 Anne，不过我相信她一定是个好
女孩。如

To: Parker, Anne <annep@starbuzzcoffee.com>
From: Greg <greg@gregslist.com>
Subject: Did we meet at Starbuzz?

我当然记得你！可是……真希望你早点联络我。我的前男友想要复合，所以那天晚上我已经有约了。

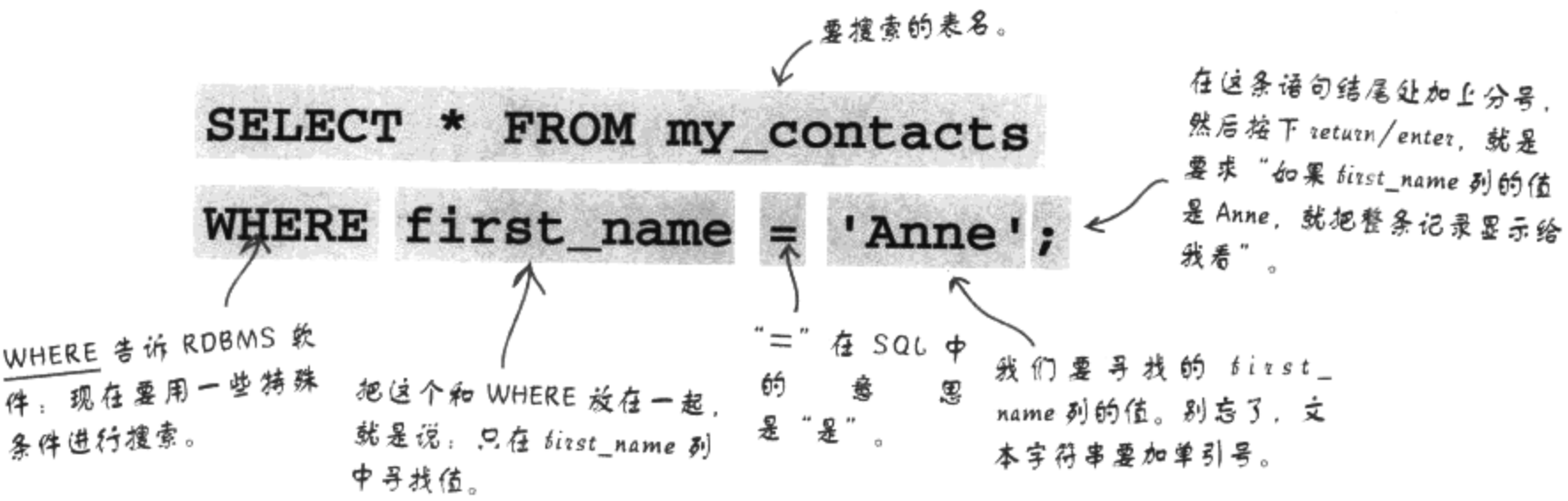
动动脑

你能想出只呈现名字是Anne的SQL语句的写法吗？

更好的SELECT

这里的SELECT语句能帮助 Greg 更快找到 Anne，而不用千辛万苦地翻找偌大一张表。在语句中，我们使用 **WHERE子句，它为 RDBMS 提供搜索的特定条件**，有助于缩小结果，而且**只会返回符合条件的行。**

WHERE子句中的等号用来检查first_name列中的每个值是否等于（符合）文本'Anne'。如果符合，即返回整行；如果不符合，则不返回该行。



下图的控制台窗口呈现了上述查询的返回结果：所有 first_name 列是 Anne 的行。

```
File Edit Window Help NoDate
> SELECT * FROM my_contacts WHERE first_name = 'Anne';
+------------+------------+--------------------------+--------+------------+--------------+
| last_name  | first_name | email                    | gender | birthday   | location     |
+------------+------------+--------------------------+--------+------------+--------------+
| Toth       | Anne       | Anne_Toth@leapinlimos.com | F      | NULL       | San Fran, CA |
| Manson     | Anne       | am86@objectville.net     | F      | NULL       | Seattle, WA  |
| Hardy      | Anne       | anneh@b0tt0msup.com      | F      | NULL       | San Fran, CA |
| Parker     | Anne       | annep@starbuzzcoffee.com | F      | NULL       | San Fran, CA |
| Blunt      | Anne       | anneblunt@breakneckpizza.com | F  | NULL       | San Fran, CA |
| Jacobs     | Anne       | anne99@objectville.net   | F      | NULL       | San Jose, CA |
+------------+------------+--------------------------+--------+------------+--------------+
6 rows in set (3.67 sec)
```

这些就是 SELECT 语句的搜索结果。

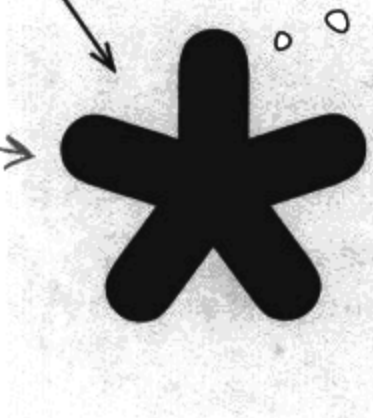
等一下，那个星号（*）别想蒙混过关。它到底有什么作用？

* 究竟是什么？

星号（*）告诉RDBMS返回表中的**所有**列。

```
SELECT * FROM my_contacts
WHERE first_name = 'Anne';
```

当你看到 SELECT *，就想到它好像是要求RDBMS选出所有列。



我是大明星！

以星号选出所有列。

没有蠢问题

问： 如果我不想选出所有列呢？可以用其他东西替换星号吗？

答： 当然可以。星号会选出所有内容，但再过几页，我们就会学到如何挑选部分列，让你的搜索结果更容易解读。

问： 星号（star）和星状物（asterisk）一样吗？

答： 没错，就是那个在你的键盘上跟“8”住在一起的家伙，只要同时按下“SHIFT”和“8”键，就会打出它。不管是Mac还是PC，使用方式都一样。

问： 还有其他符号像星号一样，具有特殊意义吗？

答： SQL还有其他特殊的或保留字符，我们稍后会提到。不过，星号是你现在唯一需要认识的，在SELECT这部分只会用到它。

Head First Lounge 新增了特调的果汁饮料。使用你在第1章学到的一切，根据本页的数据创建一张表并插入所示的数据。

下表是 **drinks** 数据库的一部分，其中包含 **easy_drinks** 表，记录着只用两种成分调成的饮料。

easy_drinks

drink_name	main	amount1	second	amount2	directions
Blackthorn	tonic water	1.5	pineapple juice	1	stir with ice, strain into cocktail glass with lemon twist
Blue Moon	soda	1.5	blueberry juice	.75	stir with ice, strain into cocktail glass with lemon twist
Oh My Gosh	peach nectar	1	pineapple juice	1	stir with ice, strain into shot glass
Lime Fizz	Sprite	1.5	lime juice	.75	stir with ice, strain into cocktail glass
Kiss on the Lips	cherry juice	2	apricot nectar	7	serve over ice with straw
Hot Gold	peach nectar	3	orange juice	6	pour hot orange juice in mug and add peach nectar
Lone Tree	soda	1.5	cherry juice	.75	stir with ice, strain into cocktail glass
Greyhound	soda	1.5	grapefruit juice	5	serve over ice, stir well
Indian Summer	apple juice	2	hot tea	6	add juice to mug and top off with hot tea
Bull Frog	iced tea	1.5	lemonade	5	serve over ice with lime slice

amount1 和 amount2 的单位是“盎司”（ounce）。

答案见第 117 页。

注意！

在你动手前先计划一下。

小心选择数据类型，别忘了 NULL 的存在。然后比较你的 SQL 代码和第 117 页上的解答。

磨笔上阵

别担心查询中有你没看过的字符。只管照样把命令输入控制台，然后观察运行结果。

找出饮料名称

使用刚才创建的 easy_drinks 表并尝试下列查询。写下作为每个查询结果返回的饮料。

```
SELECT * FROM easy _ drinks WHERE main = 'Sprite';
```

饮料名称：.....

```
SELECT * FROM easy _ drinks WHERE main = soda;
```

饮料名称：.....

```
SELECT * FROM easy _ drinks WHERE amount2 = 6;
```

饮料名称：.....

```
SELECT * FROM easy _ drinks WHERE second = "orange juice";
```

饮料名称：.....

```
SELECT * FROM easy _ drinks WHERE amount1 < 1.5;
```

饮料名称：.....

```
SELECT * FROM easy _ drinks WHERE amount2 < '1';
```

饮料名称：.....

```
SELECT * FROM easy _ drinks WHERE main > 'soda';
```

饮料名称：.....

```
SELECT * FROM easy _ drinks WHERE amount1 = '1.5';
```

饮料名称：.....

等一下……你说“只管照样把命令输入……”，一副这些命令都会成功运作的样子，我还笨笨地相信了。结果里面有一条命令造成错误信息，而且有些看起来明明不能运作的却会返回查询结果。

噢，被你发现了！

有一个查询不能运作。其他都可以，但是不一定会出现你希望的结果。

加分题：写下不能运作的查询……

……以及你认为无法运作的查询。

找出饮料名称

试着以下列语句查询 **easy _ drinks** 表并写下查询结果。

```
SELECT * FROM easy _ drinks WHERE main = 'Sprite';
```

饮料名称：Lime Fizz

请注意单引号。

```
SELECT * FROM easy _ drinks WHERE main = soda;
```

嗯……看来它就是不能运作的查询。

饮料名称：Error

```
SELECT * FROM easy_drinks WHERE amount2 = 6;
```

这个查询成功地执行了。它是个 DEC 变量，所以不需要引号。

饮料名称：Hot Gold, Indian Summer

```
SELECT * FROM easy _ drinks WHERE second = "orange juice";
```

饮料名称：Hot Gold

```
SELECT * FROM easy _ drinks WHERE amount1 < 1.5;
```

饮料名称：Oh My Gosh

```
SELECT * FROM easy _ drinks WHERE amount2 < '1';
```

饮料名称：Blue Moon, Lime Fizz, Lone Tree

```
SELECT * FROM easy _ drinks WHERE main > 'soda';
```

另一个格式正确的 WHERE 子句。

饮料名称：Blackthorn, Lime Fizz

```
SELECT * FROM easy _ drinks WHERE amount1 = '1.5';
```

饮料名称：Blackthorn, Blue Moon, Lime Fizz, Lone Tree, Greyhound, Bull Frog

加分题：写下不能运作的查询……

这就是不能运作的 WHERE 子句。VARCHAR 变量需以单引号围起。

```
WHERE main = soda;
```

……以及你认为无法运作的查询。

```
WHERE second = "orange juice";
```

这个查询成功地运作，也没有造成错误信息，即使你在插入值时用了单引号。

```
WHERE amount2 < '1';
```

不过这个子句可以运作，虽然 DEC 变量应该不需要引号。

这个子句的问题也一样！

```
WHERE amount1 = '1.5';
```

最后两个查询能运作其实是因为多数 SQL RDBMS 的宽宏大量。它们会省略多余的引号，并把 DEC 和 INT 值视为数字，而不会因为引号就把这些值视为文本值。这两个查询都出错了，但你的 RDBMS 原谅了它们。

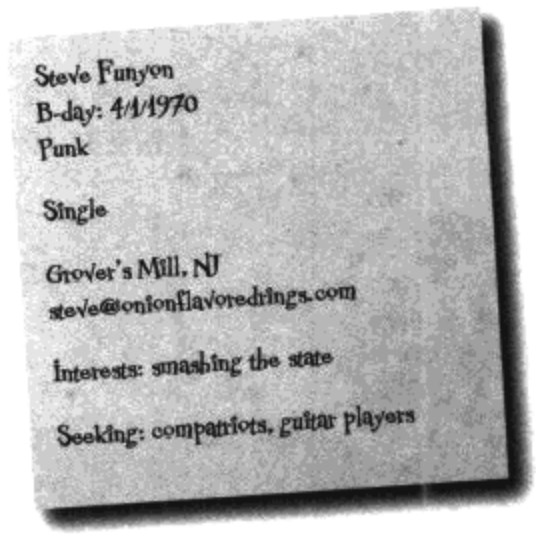
如何查询数据类型

为了写入合格的（valid） WHERE 子句，我们需确认所用数据类型的格式正确。下图说明了常用数据类型的格式惯例。

我们 ♥ 单引号	我们拒用引号
CHAR	DEC
VARCHAR	INT
DATE	
DATETIME，TIME， 或TIMESTAMP	
BLOB	

数据类型中，VARCHAR、CHAR、BLOB、DATA、TIME需要单引号。数字类的类型，DEC和INT则不需引号。

更多标点问题



某个夜晚，Greg 拿起一张联络人便笺，正想把这些数据加入表中：

```
INSERT INTO my_contacts
VALUES
('Funyon','Steve','steve@onionflavoredrings.com',
'M','1970-04-01','Punk','Grover's Mill, NJ',
'Single','smashing the state','compatriots, guitar
players');
```

不过，他的程序却没有响应。他多加了几次分号，想让查询结束。没有效果。

```
File Edit Window Help Aliens!
> INSERT INTO my_contacts VALUES ('Funyon', 'Steve', 'steve@
onionflavoredrings.com', 'M','1970-04-01', 'Punk', 'Grover's
Mill, NJ', 'Single', 'smashing the state', 'compatriots,
guitar players');
'>
'> ;
'> ;
'>
```

每次他按下 return/enter，都会看到提示符 "'>"。

动动脑

你觉得这里发么生了什么？

嗯，一个单引号一直出现在“>”前，想必是INSERT语句中的某个单引号出问题了……

不成对的单引号

没错！当Greg新增记录时，SQL程序期待收到成对的单引号，在VARCHAR、CHAR、DATE值的前后各有一个。但是镇名 **Grover's Mill**却把SQL搞糊涂了，它让单引号总数变成单数。SQL RDBMS还在等待能让程序结束的最后一个单引号。

你可以夺回控制台的掌控权。

输入单引号和分号即可结束语句。RDBMS 要一个多余的单引号，我们就给它单引号。

若是另外输入一组引号和分号，我们会得到错误信息，而且必须重新输入 **INSERT** 语句。

照着做，你会收到错误信息，但至少可以再试一次。

```
File Edit Window Help TakeTwo
> INSERT INTO my_contacts VALUES ('Funyon', 'Steve', 'steve@
onionflavoredrings.com', 'M','1970-04-01', 'Punk', 'Grover's
Mill, NJ', 'Single', 'smashing the state', 'compatriots,
guitar players');
'>
'> ;
'> ;
'>
'> ';
ERROR 1064 (42000): You have an error in your SQL syntax;
check the manual that corresponds to your SQL server version
for the right syntax to use near 's Mill, NJ', 'Single',
'smashing the state', 'compatriots, guitar players');
' at line 1
>
```

输入一个单引号和分号，终结破损的INSERT语句。

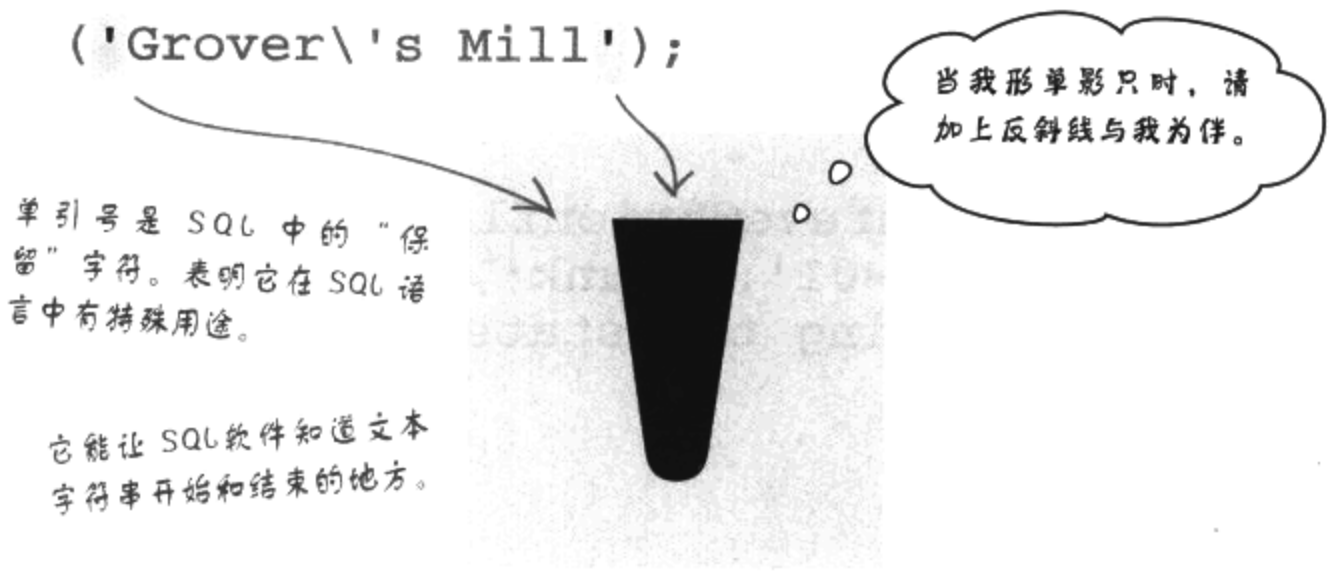
这条错误信息清楚地说出了什么地方出错了。它引述了部分查询，就从多出单引号的地方开始。

虽然这条记录并未成功插入，至少>提示符已再次出现，表示SQL又恢复响应了。

单引号是特殊字符

在试着插入包含单引号的 VARCHAR、CHAR、BLOB 数据时，必须对 RDBMS 说明：这些数据值中的单引号并非表示文本的结束，**而是文本的一部分**，应该保留在行中。在单引号字符的前面**加上反斜线**就能达到说明的效果。

```
INSERT INTO my_contacts
(location)
VALUES
('Grover\'s Mill');
```



没有蠢问题

问： 单引号跟撇号一样吗？

答： 是的。不过SQL 给它赋予了非常特殊的意义。单引号告诉 SQL 软件，在一对单引号之间的数据是文本字符串。

问： 哪些数据类型需要单引号？

答： 文本类的数据类型，也就是 VARCHAR、CHAR、BLOB、TIMEDATE 的列。只要不是数字，都算是文本类。

问： DEC 和 INT 列需要单引号吗？

答： 不需要。数字类的列中没有空格，所以很容易识别出数值结束和语句的下一个字开始的分界处。

问： 所以单引号只用在文本列中？

答： 是的。但有个问题，文本列有空格。所以在数据本身包含单引号时会造成问题，SQL 不知该如何判断单引号的位置，它不知道该把它放在列中，还是放在列的开始或结尾处。

问： 有没有区分这两种情况的简单方式呢？例如用双引号取代单引号？

答： 没有。不要使用双引号，因为你的 SQL语句日后会搭配其他编程语言（如 PHP）。在编程语言中使用“”表示“从这里开始是 SQL 语句”，这样单引号才会被视为SQL 语句的一部分，而不是其他编程语言的一部分。

INSERT包含单引号的数据

我们需要告诉 SQL 软件，这个单引号并非表示字符串的开始或结束，而是文本字符串的一部分。

用反斜线处理引号

在字符串中的单引号前加上反斜线就可以实现（同时还能修复 INSERT 语句）：

```
INSERT INTO my_contacts
VALUES
('Funyon','Steve','steve@onionflavoredrings.
com', 'M', '1970-04-01', 'Punk','Grover\'s Mill,
NJ','Single','smashing the state','compatriots,
guitar players');
```

在单引号前加上反斜线可以告诉 SQL 软件，这个单引号是文本字符串的一部分，这种行为称为“转义”。

用另一个单引号处理引号

另一种帮引号“转义”（escape）的方式则是在它前面再加一个单引号。

```
INSERT INTO my_contacts
VALUES
('Funyon','Steve','steve@onionflavoredrings.
com', 'M', '1970-04-01', 'Punk','Grover''s Mill,
NJ','Single','smashing the state','compatriots,
guitar players');
```

或在单引号前再加上另一个单引号，帮它“转义”。

动动脑

还有哪些字符也会造成类似问题呢？

如果你的表中存储了带有引号的数据，就表示你有可能在 WHERE子句 里查询有引号的数据。为了通过 WHERE 子句选择带有单引号的数据，你需要转换单引号的意义，就和插入时所做的准备一样。

请以两种字符转义方式重新编写下列 SQL 代码。

```
SELECT * FROM my_contacts
WHERE
location = 'Grover's Mill, NJ';
```

1

2

你更喜欢哪一种呢？

如果你的表中存储了带有引号的数据，就表示你有可能在 WHERE子句 里查询有引号的数据。为了通过 WHERE 子句选择带有单引号的数据，你需要转换单引号的意义，就和插入时所做的准备一样。

请以两种字符转义方式重新编写下列 SQL 代码。

```
SELECT * FROM my_contacts
WHERE
location = 'Grover's Mill, NJ';
```

1

```
SELECT * FROM my_contacts
WHERE
location = 'Grover\'s Mill, NJ';
```

方法一，使用反斜线。

2

```
SELECT * FROM my_contacts
WHERE
location = 'Grover''s Mill, NJ';
```

方法二，加上另一个单引号。

SELECT 特定数据

现在大家已经掌握了如何 SELECT 所有带引号的数据类型，以及如何 SELECT 包含引号的数据。

看起来我们需要只 SELECT 需要看到的列。

这样一来，我们需要更好的精确度来缩小结果。缩小结果表示输出结果中用到的列较少，只选出我们想看到的列。

在家试试看

在实际尝试下例 SELECT 查询前，先描述一下你心目中的查询结果。（如果需要查看 easy_drinks 表，可以翻到第59 页。）

我们用这些列名取代*。

```
SELECT drink_name, main, second
FROM easy_drinks
WHERE main = 'soda';
```

在家试试看

在实际尝试下例 SELECT 查询前，先描述一下你心目中的查询结果。

drink_name	main	second
Blue Moon	soda	blueberry juice
Lone Tree	soda	cherry juice
Greyhound	soda	grapefruit juice
Soda and It	soda	grape juice

老方式

```
SELECT * FROM easy_drinks;
```

用全选（*）的老方式会取得所有列，但我们的结果太长了，不适合在终端窗口中呈现。输出结果都需排成两行，看起来很混乱。

```
File Edit Window Help MessyDisplay
> SELECT * FROM easy_drinks;
+------------------+-------------+---------+------------------+---------+--------------
-----------------------------------------+
| drink_name       | main        | amount1 | second           | amount2 | directions
+------------------+-------------+---------+------------------+---------+--------------
-----------------------------------------+
| Kiss on the Lips | cherry juice |     2.0 | apricot nectar   |    7.00 | serve over ice
with straw                                |
| Hot Gold         | peach nectar |     3.0 | orange juice     |    6.00 | pour hot or-
ange
juice in mug and add peach nectar         |
| Lone Tree        | soda         |     1.5 | cherry juice     |    0.75 | stir with ice,
strain into cocktail glass                |
| Greyhound        | soda         |     1.5 | grapefruit juice |    5.00 | serve over
ice,
stir well                                 |
| Indian Summer    | apple juice  |     2.0 | hot tea          |    6.00 | add juice to
mug
and top off with hot tea                  |
| Bull Frog        | iced tea     |     1.5 | lemonade         |    5.00 | serve over ice
with lime slice                           |
| Soda and It      | soda         |     2.0 | grape juice      |    1.00 | shake in
cocktail glass, no ice                    |
| Blackthorn       | tonic water  |     1.5 | pineapple juice  |    1.00 | stir with ice,
strain into cocktail glass with lemon twist |
| Blue Moon        | soda         |     1.5 | blueberry juice  |    0.75 | stir with ice,
strain into cocktail glass with lemon twist |
| Oh My Gosh       | peach nectar |     1.0 | pineapple juice  |    1.00 | stir with ice,
strain into shot glass                    |
| Lime Fizz        | Sprite       |     1.5 | lime juice       |    0.75 | stir with ice,
strain into cocktail glass                |
+------------------+-------------+---------+------------------+---------+--------------
-----------------------------------------+
11 rows in set (0.00 sec)
```

SELECT 特定列来限制结果数量

通过指定想要查询返回的列，我们可以只选择需要的列值。就像使用 WHERE 子句限制列数一样，我们也可以选择列名来限制返回的列的数量。让 SQL 帮我们承担过滤数据的重担。

```
SELECT drink_name, main, second
FROM easy_drinks;
```

……但事实上，只选出想要的列就可以缩小可见的结果。

```
File Edit Window Help JustEnough
> SELECT drink_name, main, second FROM easy_drinks;
+------------------+--------------+------------------+
| drink_name       | main         | second           |
+------------------+--------------+------------------+
| Kiss on the Lips | cherry juice | apricot nectar   |
| Hot Gold         | peach nectar | orange juice     |
| Lone Tree        | soda         | cherry juice     |
| Greyhound        | soda         | grapefruit juice |
| Indian Summer    | apple juice  | hot tea          |
| Bull Frog        | iced tea     | lemonade         |
| Soda and It      | soda         | grape juice      |
| Blackthorn       | tonic water  | pineapple juice  |
| Blue Moon        | soda         | blueberry juice  |
| Oh My Gosh       | peach nectar | pineapple juice  |
| Lime Fizz        | Sprite       | lime juice       |
+------------------+--------------+------------------+
11 rows in set (0.00 sec)
```

SELECT 特定列以加快结果呈现

只选择需要的列是一个值得遵循的编程惯例，不过它还有其他好处。随着表的日益扩大，限定选择列还会加快检索结果的速度。当SQL 最终和其他编程语言（如 PHP）搭配使用时，这么做也会让运行速度更快。

磨笔上阵

有多种方式可以得到Kiss on the Lips

还记得这一章的 easy_drinks 表吗？下面的 SELECT 语句会找出 Kiss on the Lips：

```
SELECT drink _ name FROM easy _ drinks
WHERE
main = 'cherry juice';
```

请填写下一页的四条空白语句，从而用其他方式找出 Kiss on the Lips。

easy _ drinks

drink_name	main	amount1	second	amount2	directions
Blackthorn	tonic water	1.5	pineapple juice	1	stir with ice, strain into cocktail glass with lemon twist
Blue Moon	soda	1.5	blueberry juice	.75	stir with ice, strain into cocktail glass with lemon twist
Oh My Gosh	peach nectar	1	pineapple juice	1	stir with ice, strain into shot glass
Lime Fizz	Sprite	1.5	lime juice	.75	stir with ice, strain into cocktail glass
Kiss on the Lips	cherry juice	2	apricot nectar	7	serve over ice with straw
Hot Gold	peach nectar	3	orange juice	6	pour hot orange juice in mug and add peach nectar
Lone Tree	soda	1.5	cherry juice	.75	stir with ice, strain into cocktail glass
Greyhound	soda	1.5	grapefruit juice	5	serve over ice, stir well
Indian Summer	apple juice	2	hot tea	6	add juice to mug and top off with hot tea
Bull Frog	iced tea	1.5	lemonade	5	serve over ice with lime slice

SELECT .....

WHERE .....

SELECT .....

WHERE .....

SELECT .....

WHERE .....

SELECT .....

WHERE .....

现在写下三种可以找出 Bull Frog 的 SELECT 语句。

1. .....
   .....
2. .....
   .....
3. .....
   .....

磨笔上阵解答

填写下列四条空白语句，找出 Kiss on the Lips。

```
SELECT drink_name FROM easy_drinks
WHERE second = 'apricot nectar';

SELECT drink_name FROM easy_drinks
WHERE amount2 = 7;

SELECT drink_name FROM easy_drinks
WHERE directions = 'serve over ice with straw';

SELECT drink_name FROM easy_drinks
WHERE drink_name = 'Kiss on the Lips';
```

一般不太会用这个语句，但它确实可以找出想要的结果。采用这类语句时，多半是为了确认 drink_name 列里没有错字。

现在写下三种可以找出 Bull Frog 的 SELECT 语句。

1
```
SELECT drink_name FROM easy_drinks
WHERE main = 'iced tea';
```

2
```
SELECT drink_name FROM easy_drinks
WHERE second = 'lemonade';
```

3
```
SELECT drink_name FROM easy_drinks
WHERE directions = 'serve over ice with lime slice';
```

你也可以使用易于理解的查询：

```
SELECT drink_name FROM
easy_drinks
WHERE drink_name='Bull Frog';
```

复习要点

- 从文本字段中选择数据时要在WHERE子句中使用单引号。
- 从数字字段中选择数据时不要使用单引号。
- 想要选择所有列，可在SELECT中使用*。
- 如果输入查询后RDBMS没有完成处理，请检查单引号数量是否正确。
- 请尽量选择特定列来替代使用SELECT * 的全部选取。

问： **如果我真的需要取得所有列，还要在SELECT中指定所有列名吗？可否直接使用 * 呢？**

答： 如果需要所有列，一定要使用“*”。只有在不需要太多列时，才需要逐一指出要检索的列。

问： **我试着从网络上复制并粘贴查询，但在使用时却一直出现错误信息。我做错什么了吗？**

答：从Web浏览器剪切过来的查询有时包含了外观像空格但在SQL里有其他含义的隐形字符。你可以先把查询粘贴到文本编辑器中，如此一来，就可以仔细寻找并移除这些“小麻烦”。

问： **所以我应该把查询粘贴到 Microsoft Word 之类的软件中吗？**

答： 最好不要，Word不是很好的选择，它不会显示可能藏在文本中的隐形格式。请改用Notepad（PC）或TextEdit（Mac）的纯文本编辑模式。

问： **关于单引号的两种转义方法，哪一种比较好呢？**

答： 其实没有优劣之分。我们比较常用反斜线，因为这种方法在查询出错时可以轻易地发现多余单引号的位置。例如，下面第一行：

```
'Isn\'t that your sister\'s pencil?'
```

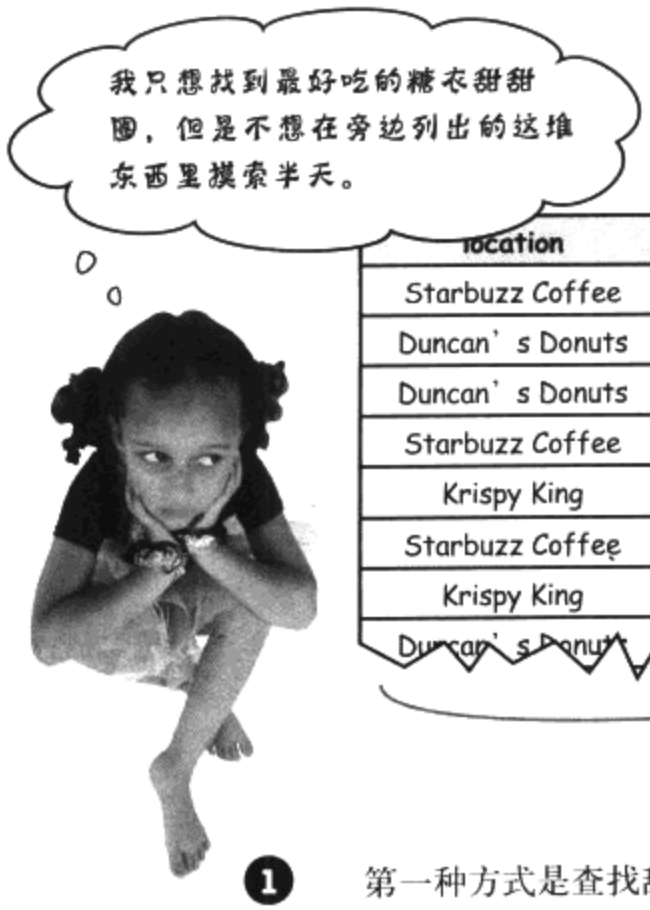
在视觉上的判断比第二行容易：

```
'Isn''t that your sister''s pencil?'
```

除此之外，没有特别偏好哪种方式的原因。两者都能在文本列中输入单引号。

寻求甜甜圈表为你服务前……

想找出最好吃的糖衣甜甜圈，至少需要对表 SELECT 两次。一次选出正确甜甜圈类型的行，另一次则选出评分为 10 的甜甜圈。



doughnut _ ratings

location	time	date	type	rating	comments
Starbuzz Coffee	7:43 am	4/23	cinnamon glazed	6	too much spice
Duncan' s Donuts	8:56 am	8/25	plain glazed	5	greasy
Duncan' s Donuts	7:58 pm	4/26	jelly	6	stale, but tasty
Starbuzz Coffee	10:35 pm	4/24	plain glazed	7	warm, but not hot
Krispy King	9:39 pm	9/26	jelly	6	not enough jelly
Starbuzz Coffee	7:48 am	4/23	rocky road	10	marshmallows!
Krispy King	8:56 am	11/25	plain glazed	8	maple syrup glaze
Duncan' s Donuts	11:43 pm	8/26	jelly		stale and dry

想象这是一张拥有10,000条记录的表。

❶ 第一种方式是查找甜甜圈类型：

你需要选择评分（rating）列，才能从中找出评分最高的甜甜圈；还需要锁定店名（location）列，这样你才知道该甜甜圈是哪家店的。

```
SELECT location, rating FROM doughnut_ratings
WHERE
type = 'plain glazed';
```

所有结果都是我想吃的甜甜圈的种类。

第一种方式的查询结果，不过请想象结果数量增加100倍后的样子。

location	rating
Duncan' s Donuts	5
Starbuzz Coffee	7
Krispy King	8
Starbuzz Coffee	10
Duncan' s Donuts	8

先问问你能为表提供什么

❷ 或改为查找评分最高的甜甜圈：

```
SELECT location, type FROM doughnut_ratings
WHERE
rating = 10;
```

你需要查看所有甜甜圈种类（type），还需要店家名称（location）的记录。

所有查询结果只限于具有最高评分的记录。

location	type
Starbuzz Coffee	rocky road
Krispy King	plain glazed
Starbuzz Coffee	plain glazed
Duncan's Donuts	rocky road

第二种方式的查询结果。同样，请把图示的结果数量增加100倍。

好像没什么帮助。我可以随便挑一份查询结果，然后开始查找我理想中的甜甜圈，但不管挑选哪一份，都要查询好几千条记录……我好饿，我只是想要个甜甜圈，现在就给我甜甜圈啦！

动动脑

如果用通俗易懂的话说，这些查询究竟试图回答什么问题？

结合查询

我们可以用 AND 同时处理两项查询，查找种类为“plain glazed”以及评分为“ 10” 的甜甜圈。这样取得的查询结果一定会满足两个条件。

现在只要选择 location。

```
SELECT location
FROM doughnut_ratings
WHERE type = 'plain glazed'
AND
rating = 10;
```

使用 AND 结合两个 WHERE 子句。

以下是加上 AND 的查询结果。即使我们找出的查询结果不只一列，但至少所有结果都是评分为 10 的糖衣甜甜圈，所以你随便选一家就可以了。

location	rating
Duncan' s Donuts	5
Starbuzz Coffee	7
Krispy King	8
Starbuzz Coffee	10
Duncan' s Donuts	8

AND

location	type
Starbuzz Coffee	rocky road
Krispy King	plain glazed
Starbuzz Coffee	plain glazed
Duncan' s Donuts	rocky road

这项查询结合了 'plain glazzed' 和 rating = 10，它会找出同时符合这两个条件的记录。

location
Starbuzz Coffee

请利用 my_contacts 表为 Greg设计查询，让他只需选择提供必要信息的列。请记得单引号的重要性。

写下一组查找所有电脑工程师的电子邮件地址的查询。

写下一组查找姓名和地点的查询，但只找出与你同一天生日的人。

写下一组查找姓名和电子邮件地址的查询，但只查找与你住在相同城市的单身人士。加分题：只过滤出你想约会的对象的性别。

写下 Greg 可以用来找出住在 San Francisco 的 Anne的查询。

习题解答

请利用 my_contacts 表为 Greg 设计查询，让他只需选择提供必要信息的列。请记得单引号的重要性。

写下一组查找所有电脑工程师的电子邮件地址的查询。

```
SELECT email FROM my_contacts
WHERE profession = 'computer programmer';
```

我们需要 email 列。

我们需要的 profession 是 "computer programmer"

写下一组查找姓名和地点的查询，但只找出与你同一天生日的人。

```
SELECT last_name, first_name, location
FROM my_contacts
WHERE birthday = '1975-09-05';
```

这里应该填入你的生日。

写下一组查找姓名和电子邮件地址的查询，但只查找和你住在相同城市的单身人士。加分题：只过滤出你想约会的对象的性别。

```
SELECT last_name, first_name, email
FROM my_contacts
WHERE location = 'San Antonio, TX'
AND gender = 'M'
AND Status ='single';
```

这里是你住的地方。

这里是查找的约会对象的性别。

写下 Greg 可以用来找出住在 San Francisco 的 Anne的查询。

```
SELECT last_name, first_name, email
FROM my_contacts
WHERE location = 'San Fran, CA'
AND first_name = 'Anne';
```

回头看看表，Greg 似乎缩写 "San Francisco"，希望他一直都这么做。

查找数值

假设你要用单一查询来找出easy_drinks表中包含一盎司以上 soda 的所有饮料。以下是找出结果的复杂方法。你可以使用两个查询：

我们只要饮料的名称。

```
SELECT drink_name FROM easy_drinks
WHERE
main = 'soda'
AND
amount1 = 1.5;
```

查找目标：包含 1.5 盎司 soda 的饮料。

```
File Edit Window Help MoreSoda
> SELECT drink_name FROM easy_drinks WHERE main = 'soda' AND
amount1 = 1.5;
+-------------+
| drink_name  |
+-------------+
| Blue Moon   |
| Lone Tree   |
| Greyhound   |
+-------------+
3 rows in set (0.00 sec)
```

查找目标：包含 2 盎司 soda 的饮料。

```
SELECT drink_name FROM easy_drinks
WHERE
main = 'soda'
AND
amount1 = 2;
```

```
File Edit Window Help EvenMoreSoda
> SELECT drink_name FROM easy_drinks WHERE main = 'soda' AND
amount1 = 2;
+--------------+
| drink_name   |
+--------------+
| Soda and It  |
+--------------+
1 row in set (0.00 sec)
```

easy _ drinks

drink_name	main	amount1	second	amount2	directions
Blackthorn	tonic water	1.5	pineapple juice	1	stir with ice, strain into cocktail glass with lemon twist
Blue Moon	soda	1.5	blueberry juice	.75	stir with ice, strain into cocktail glass with lemon twist
Oh My Gosh	peach nectar	1	pineapple juice	1	stir with ice, strain into shot glass
Lime Fizz	Sprite	1.5	lime juice	.75	stir with ice, strain into cocktail glass
Kiss on the Lips	cherry juice	2	apricot nectar	7	serve over ice with straw
Hot Gold	peach nectar	3	orange juice	6	pour hot orange juice in mug and add peach nectar
Lone Tree	soda	1.5	cherry juice	.75	stir with ice, strain into cocktail glass
Greyhound	soda	1.5	grapefruit juice	5	serve over ice, stir well
Indian Summer	apple juice	2	hot tea	6	add juice to mug and top off with hot tea
Bull Frog	iced tea	1.5	lemonade	5	serve over ice with lime slice
Soda and It	soda	2	grape juice	1	shake in cocktail glass, no ice

一次就够了

但是使用两组查询只是浪费时间，而且可能会忽略含量为 1.75 盎司或 3 盎司这类饮料。其实我们可以改用**大于**运算符：

```
SELECT drink_name FROM easy_drinks
WHERE
main = 'soda'
AND
amount1 > 1;
```

大于号把所有包含 1 盎司以上 soda 的饮料提供给我们。

```
File Edit Window Help DoItOnce
> SELECT drink_name FROM easy_drinks  WHERE main = 'soda'
AND
amount1 > 1;
+--------------+
| drink_name   |
+--------------+
| Blue Moon    |
| Lone Tree    |
| Greyhound    |
| Soda and It  |
+--------------+
4 rows in set (0.00 sec)
```

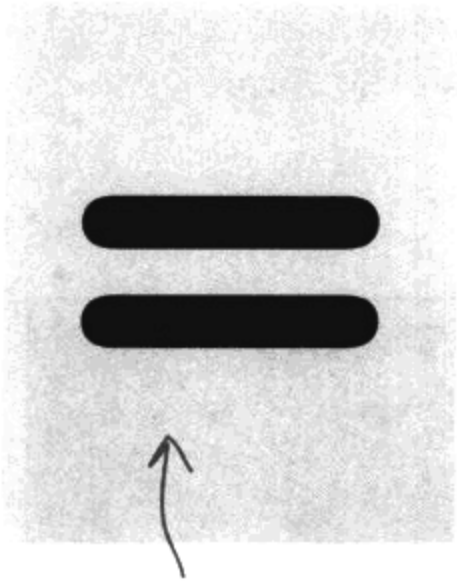
动动脑

为什么不用一个 AND 结合两组查询呢？

~~顺利~~运用比较运算符

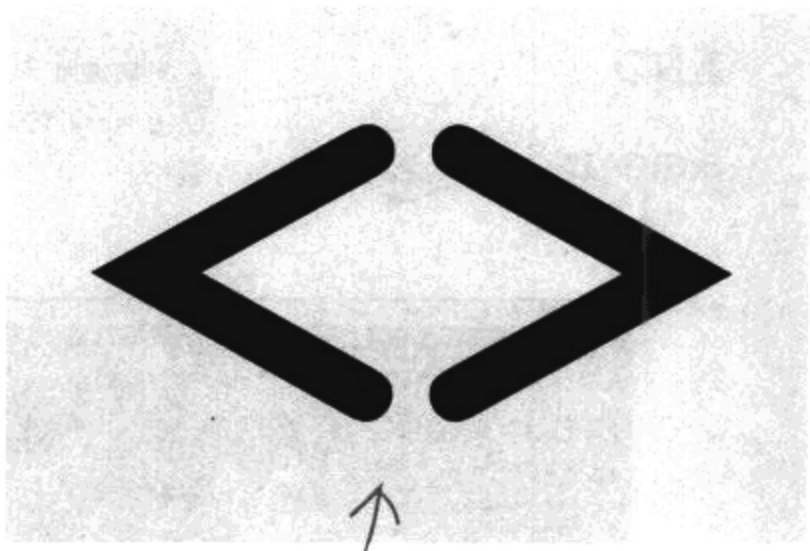
到目前为止，我们只在 WHERE 子句中用过**等号**。**大于**号（>）刚刚出现，它会比较两个值。接下来隆重介绍所有比较运算符：

等号查找精确相同的数据。当我们需要大于或小于的数据时，它就帮不上忙了。



大家都认识的好朋友：等号。

这位外表让人困惑的新朋友是**不等**运算符。它返回的结果和等号恰好相反。两个值只可能等于彼此，或不等于彼此。



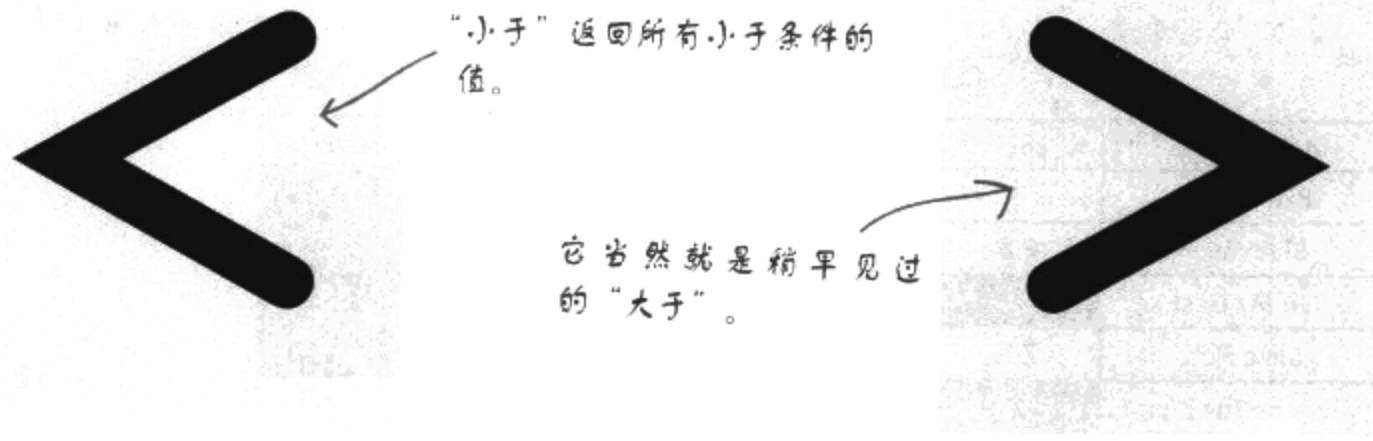
它的意思是“不等于”，返回所有不符合条件的记录。

脑力锻炼

你可曾注意到：到目前为止，每个WHERE子句都把列名放在运算符的左边。如果把列名放在右边会发生什么事？

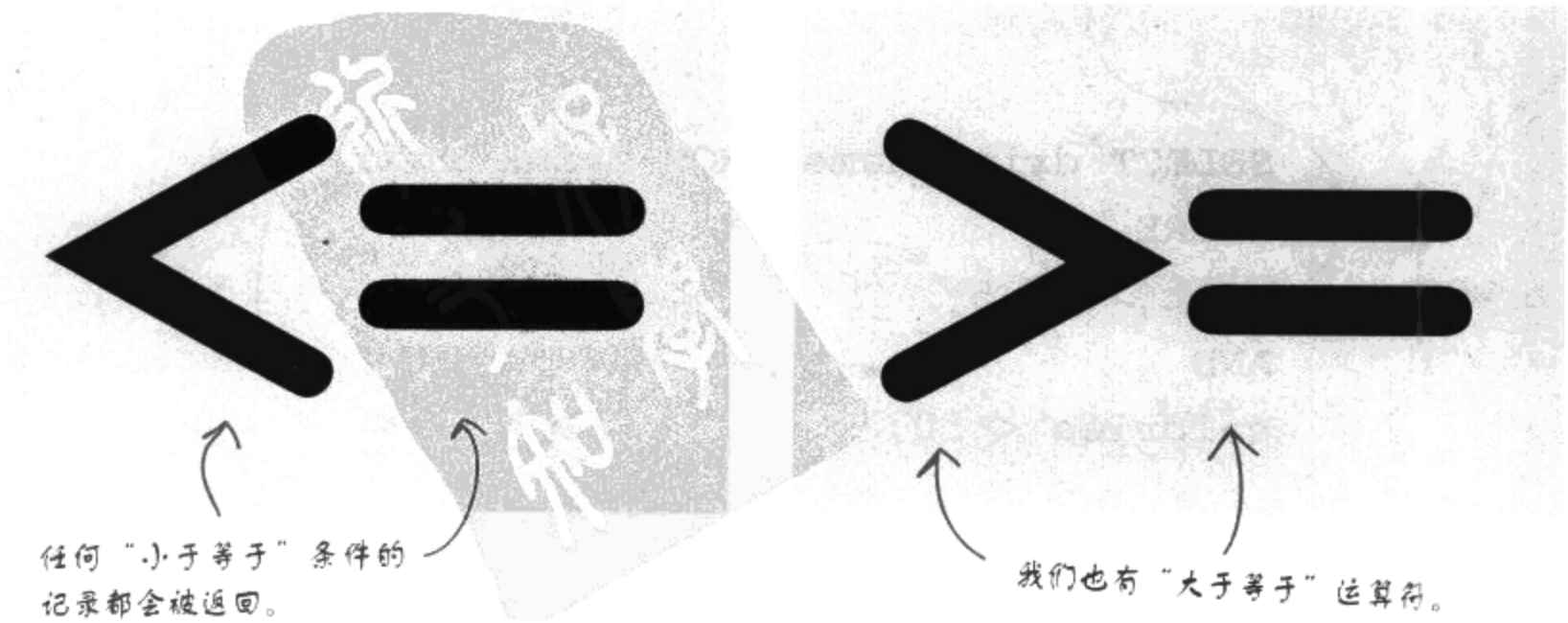
小于号会把它左侧的列值与它右侧的值进行比较。如果列值小于右侧值，则返回该行。

大于号则与小于号相反。它会把左侧的列值与右侧的值相比较，如果列值大于右侧值，则返回该行。



小于等于运算符的唯一差别是它也会返回左侧列值等于右侧值的记录。

大于等于运算符也有相同的差异。如果左侧列值等于或大于条件值，就会返回该行。



利用比较运算符取得数字数据

Head First Lounge 有一张记录价格和营养成分信息的饮料表。他们想突显高单价但低热量的特色来提高利润。

他们正利用比较运算符从 `drink_info` 表中找出定价至少是 3.5 美元、热量又少于 50 卡路里的饮料。

每种饮料的碳水化合物含量。

每种饮料的热量（卡路里）。

drink_info

drink_name	cost	carbs	color	ice	calories
Blackthorn	3	8.4	yellow	Y	33
Blue Moon	2.5	3.2	blue	Y	12
Oh My Gosh	3.5	8.6	orange	Y	35
Lime Fizz	2.5	5.4	green	Y	24
Kiss on the Lips	5.5	42.5	purple	Y	171
Hot Gold	3.2	32.1	orange	N	135
Lone Tree	3.6	4.2	red	Y	17
Greyhound	4	14	yellow	Y	50
Indian Summer	2.8	7.2	brown	N	30
Bull Frog	2.6	21.5	tan	Y	80
Soda and It	3.8	4.7	red	N	19

```
SELECT drink_name FROM drink_info
WHERE
cost >= 3.5
AND
calories < 50;
```

这句的意思是："找出价格在 3.5 美元以上的饮料。"这会包括售价 3.5 美元的饮料。

这句的意思是："找出热量少于 50 卡路里的饮料。"

这组查询只返回同时符合上述两项条件的饮料，因为 AND 合并过滤了两项条件的查询结果。被返回的饮料包括：Oh My Gosh、Lone Tree、Soda and It。

磨笔上阵

我们也来混合不同条件吧。写出能返回下列需求信息的查询，也要写出每项查询的返回结果：

每一种加冰、热量高于 33 卡路里的黄色饮料的价格。

查询结果：

每一种碳水化合物低于 4 克而且加冰的饮料的名称和颜色。

查询结果：

每一种热量大于或者等于 80 卡路里的饮料的价格。

查询结果：

只会返回 Greyhound 和 Kiss on the Lips 的查询，并附上它们的颜色及调制时是否加冰，不要在你的查询中使用饮料名称。

查询结果：

磨笔上阵解答

我们也来混合不同条件吧。写出能返回下列需求信息的查询，也要写出每项查询的返回结果：

每一种加冰、热量高于 33 卡路里的黄色饮料的价格。

```
SELECT cost FROM drink_info
WHERE ice = 'Y'
AND
color = 'yellow'
AND
calories > 33;
```

查询结果：$4.00

每一种碳水化合物低于 4 克而且加冰的饮料的名称和颜色。

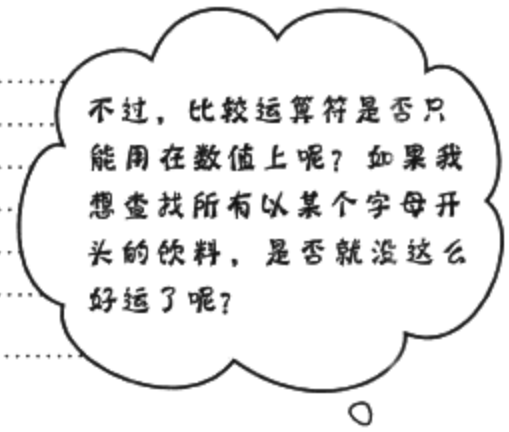
```
SELECT drink_name, color FROM drink_info
WHERE
carbs <= 4
AND
ice = 'Y';
```

查询结果：Blue Moon, blue

每一种热量大于或者等于 80 卡路里的饮料的价格。

```
SELECT cost FROM drink_info
WHERE
calories >= 80;
```

查询结果：$5.50, $3.20, $2.60



只会返回 Greyhound 和 Kiss on the Lips 的查询，并附上它们的颜色及调制时是否加冰，不要在你的查询中使用饮料名称。

```
SELECT drink_name, color, ice FROM drink_info
WHERE
cost > 3.8;
```

这一题比较奸诈，你需要研究表的内容，找出能够只选出这两种饮料的条件。

查询结果：Kiss on the Lips, purple, Y
Greyhound, yellow, Y

对文本数据套用比较运算符

比较像CHAR 和 VARCHAR 这样的文本数据时，运作方式和数字其实很相似。比较运算符会按字母顺序地评估所有事物的大小。假设你要选出所有名称以“L”开头的饮料，下面的查询就能满足我们的需求。

drink_info

drink_name	cost	carbs	color	ice	calories
Blackthorn	3	8.4	yellow	Y	33
Blue Moon	2.5	3.2	blue	Y	12
Oh My Gosh	3.5	8.6	orange	Y	35
Lime Fizz	2.5	5.4	green	Y	24
Kiss on the Lips	5.5	42.5	purple	Y	171
Hot Gold	3.2	32.1	orange	N	135
Lone Tree	3.6	4.2	red	Y	17
Greyhound	4	14	yellow	Y	50
Indian Summer	2.8	7.2	brown	N	30
Bull Frog	2.6	21.5	tan	Y	80
Soda and It	3.8	4.7	red	N	19

```
SELECT drink_name
FROM drink_info
WHERE
drink_name >= 'L'
AND
drink_name < 'M';
```

这个查询返回名称首字母为“L”或位于其后的但要早于“M”的饮料。

放松

现在先别担心你的查询结果排序。

在后面的章节中，我们会教你怎么按字母顺序给查询结果排序。

选出成分

一位吧台人员接到一杯要求包含樱桃汁的特调饮料订单。他可以用两组查询找出饮料：

一组查询只检查一个成分列。

```
File Edit Window Help..
> SELECT drink_name FROM easy_drinks WHERE main = 'cherry juice';
+--------------------+
| drink_name         |
+--------------------+
| Kiss on the Lips   |
+--------------------+
1 row in set (0.02 sec)

> SELECT drink_name FROM easy_drinks WHERE second = 'cherry juice';
+------------+
| drink_name |
+------------+
| Lone Tree  |
+------------+
1 row in set (0.01 sec)
```

这看起来真是很没效率。我相信一定有結合两组查询的方式。

drink_info

drink_name	cost	carbs	color	ice	calories
Blackthorn	3	8.4	yellow	Y	33
Blue Moon	2.5	3.2	blue	Y	12
Oh My Gosh	3.5	8.6	orange	Y	35
Lime Fizz	2.5	5.4	green	Y	24
Kiss on the Lips	5.5	42.5	purple	Y	171
Hot Gold	3.2	32.1	orange	N	135
Lone Tree	3.6	4.2	red	Y	17
Greyhound	4	14	yellow	Y	50
Indian Summer	2.8	7.2	brown	N	30
Bull Frog	2.6	21.5	tan	Y	80
Soda and It	3.8	4.7	red	N	19

OR，只要符合一项条件

这两组查询可以用 OR 结合。结合后的条件会返回**任何**符合条件之一的记录。
所以，不需要再像前一页那样采用两次查询，而是用 OR 结合如下：

```
File  Edit  Window  Help..SweetCherryPie
> SELECT drink_name from easy_drinks
WHERE main = 'cherry juice'
OR
second = 'cherry juice';
+--------------------+
| drink_name         |
+--------------------+
| Kiss on the Lips   |
| Lone Tree          |
+--------------------+
2 rows in set (0.02 sec)
```

磨笔上阵

划掉下面两组 SELECT 查询中多余的部分并加上一个“OR”，把它们结合成单一 SELECT 语句。

```
SELECT drink_name FROM easy_drinks WHERE
main = 'orange juice';


SELECT drink_name FROM easy_drinks WHERE
main = 'apple juice';
```

使用新学到的选择技巧重新设计 SELECT。

.....

.....

磨笔上阵解答

划掉下面两组 SELECT 查询中多余的部分并加上一个“OR”，把它们结合成单一 SELECT 语句。

```
SELECT drink_name FROM easy_drinks WHERE
main = 'orange juice';
OR
SELECT drink_name FROM easy_drinks WHERE
main = 'apple juice';
```

删除这个分号，语句才不会在这里结束。

加上这个 OR 就能取得主要成分是橙汁或苹果汁的 *drink_name*。

只要简单地划掉这行就可以了，用 OR 结合两组查询后，它的作用已经由前面出现的相同语句负责了。

使用新学到的选择技巧重新设计 SELECT。

```
SELECT drink_name FROM easy_drinks
WHERE
main = 'orange juice'
OR
main = 'apple juice';
```

最后完成的查询。

OR 看来是个很好用的运算符，不过，我不明白为什么我们不是只用 AND就够了。

别把 AND 和 OR 搞混了！

需要**所有**条件都成立时，请用 **AND**。

需要**任何**条件成立时，请用 **OR**。

还是搞不清楚？请翻到下一页。

AND OR

没有蠢问题

问： 在同一个 WHERE 子句中可以用多个 AND 或 OR 吗？

答： 当然可以。使用的 AND 和 OR 的数量根据我们的需要而定，我们还可以在一个子句中同时使用 AND 和 OR。

AND与OR的差异

在接下来的查询中，我们将从范例中了解用 AND 和 OR 组合两个条件的所有可能性。

doughnut_ratings

location	time	date	type	rating	comments
Krispy King	8:50 am	9/27	plain glazed	10	almost perfect
Duncan's Donuts	8:59 am	8/25	NULL	6	greasy
Starbuzz Coffee	7:35 pm	5/24	cinnamon cake	5	stale, but tasty
Duncan's Donuts	7:03 pm	4/26	jelly	7	not enough jelly

```
SELECT type FROM doughnut_ratings
WHERE location = 'Krispy King' AND rating = 10;
```

是的，有符合这项条件的行　符合

查询结果：plain glazed

```
WHERE location = 'Krispy King' OR rating = 10;
```

查询结果：plain glazed

不符合

```
WHERE location = 'Krispy King' AND rating = 3;
```

查询结果：没有结果

```
WHERE location = 'Krispy King' OR rating = 3;
```

查询结果：plain glazed

不符合

```
WHERE location = 'Snappy Bagel' AND rating = 10;
```

查询结果：没有结果

```
WHERE location = 'Snappy Bagel' OR rating = 10;
```

查询结果：plain glazed

```
WHERE location = 'Snappy Bagel' AND rating = 3;
```

查询结果：没有结果

```
WHERE location = 'Snappy Bagel' OR  rating = 3;
```

查询结果：没有结果

与条件天人合一

下面有几段包含 *AND* 和 *OR* 的 WHERE 子句。你要做的功课就是与这些子句天人合一，判断它们是否能产生查询结果。

```
SELECT type FROM doughnut_ratings
```

查询有结果吗?

```
WHERE location  = 'Krispy King' AND  rating <> 6;
```

.....

```
WHERE location  = 'Krispy King' AND  rating = 3;
```

.....

```
WHERE location  = 'Snappy Bagel' AND  rating >= 6;
```

.....

```
WHERE location  = 'Krispy King' OR  rating > 5;
```

.....

```
WHERE location  = 'Krispy King' OR  rating = 3;
```

.....

```
WHERE location  = 'Snappy Bagel' OR  rating = 6;
```

.....

若想进一步提高，请标注出其中两项查询结果与其他查询不同的原因。

与条件大人合一的解答

下面有几段包含AND和OR的WHERE子句。你要做的功课就是与这些子句大人合一，判断它们是否能产生查询结果。

```
SELECT type FROM doughnut_ratings
```

查询有结果吗?

```
WHERE location  = 'Krispy King' AND  rating <> 6;
```

plain glazed

```
WHERE location  = 'Krispy King' AND  rating = 3;
```

没有结果

```
WHERE location  = 'Snappy Bagel' AND  rating >= 6;
```

没有结果

```
WHERE location  = 'Krispy King' OR  rating > 5;
```

plain glazed, NULL, jelly

```
WHERE location  = 'Krispy King' OR  rating = 3;
```

plain glazed

```
WHERE location  = 'Snappy Bagel' OR  rating = 6;
```

NULL

若想进一步提高，请标注出其中两项查询结果与其他查询不同的原因。

有两组查询返回了NULL。

这些 NULL 值会给以后的查询带来麻烦。最好填入某些值，而不是放任 NULL 待在列中，**因为无法直接从表中选择 NULL**。

用IS NULL找到NULL

我试着直接选择 NULL，但就是没有办法。我该怎么找出表中的 NULL？

drink _ info

drink_name	cost	carbs	color	ice	calories
Holiday	NULL	14	NULL	Y	50
Dragon Breath	2.9	7.2	brown	N	NULL

不可以直接选择 NULL。

```
SELECT drink_name FROM drink_info
WHERE
calories = NULL ;
```

因为没有东西等于 NULL，所以不会成功。NULL 代表未定义的值。

```
SELECT drink_name FROM drink_info
WHERE
calories = 0;
```

因为零不等于 NULL，所以也不会成功。

```
SELECT drink_name FROM drink_info
WHERE
calories = 'NULL';
```

因为 NULL 不是文本字符串，所以还是不成功。

但可以利用关键字选择 NULL 。

```
SELECT drink_name
FROM drink_info
WHERE
calories IS NULL;
```

唯一直接选择 NULL 的方法就是利用关键字 IS NULL。

关键字不是文本字符串，所以不要加上单引号

没有蠢问题

问： 你说“不可以直接选择”NULL，除非使用 IS NULL。这么说来，我们可以间接选择 NULL？

答： 没错。如果想要取得某列中的 NULL，可在 WHERE子句中选择其他列。例如，下例查询的选取结果就是 NULL。

```
SELECT calories FROM drink_info
WHERE drink_name = 'Dragon Breath';
```

问： 上个查询结果到底是什么样子？

答： 就像下面这样：

```
+------------+
| calories   |
+------------+
| NULL       |
+------------+
```

回头看看 Greg 又遇到了什么问题……

Greg 正在试着从 my_contacts 表中找出每个住在California的人。以下只是他正在努力制作的查询的一部分：


```
SELECT * FROM my_contacts
WHERE
location = 'San Fran, CA'
OR
location = 'San Francisco, CA'
OR
location = 'San Jose, CA'
OR
location = 'San Mateo, CA'
OR
location = 'Sunnyvale, CA'
OR
location = 'Marin, CA'
OR
location = 'Oakland, CA'
OR
location = 'Palo Alto, CA'
OR
location = 'Sacramento, CA'
OR
location = 'Los Angeles, CA'
OR
And the list goes on and on...
```

Greg 至少用两种方式来表示旧金山，如果还要考虑打错字的可能性呢？

节省时间就用关键字：LIKE

城镇数量和输入时的变化实在太多了，而且还可能打错字。如果用OR囊括所有条件，Greg大概要花很长很长的时间才写得完。幸好，有个可以节省时间的关键字LIKE，若配合通配符（wildcard）一起使用，可查找部分文本字符串并返回所有符合匹配条件的行。

Greg 可以这样使用 LIKE：

```
SELECT * FROM my_contacts
WHERE location LIKE '%CA';
```

在单引号中放入百分比符号（%），就是告诉软件：我们要在 location 列中查找所有以 "CA" 结尾的值。

调用通配符

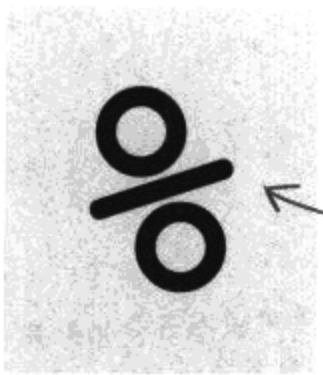
LIKE和两个通配符一起运作。通配符是实际存在于该处的字符的替身。不像扑克牌中的王牌，通配符等于字符中的任何字符。

脑力锻炼

在本章的稍早内容中你看过其他通配符吗？

再谈LIKE

LIKE喜欢和通配符在一起。第一个通配符是%，它是任意数量的未知字符的替身。

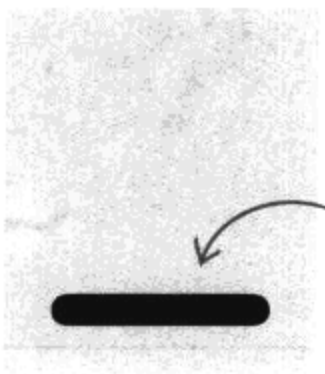


```
SELECT first_name FROM my_contacts
WHERE first_name LIKE '%im';
```

百分比号是任意数量的未知字符的替身。

在 first_name 列中，只要在 "im" 前有其他字符，例如 Ephraim、Slim、Tim，查询都会返回结果。

LIKE 喜欢的第二个通配符是下划线（_），它是一个未知字符的替身。



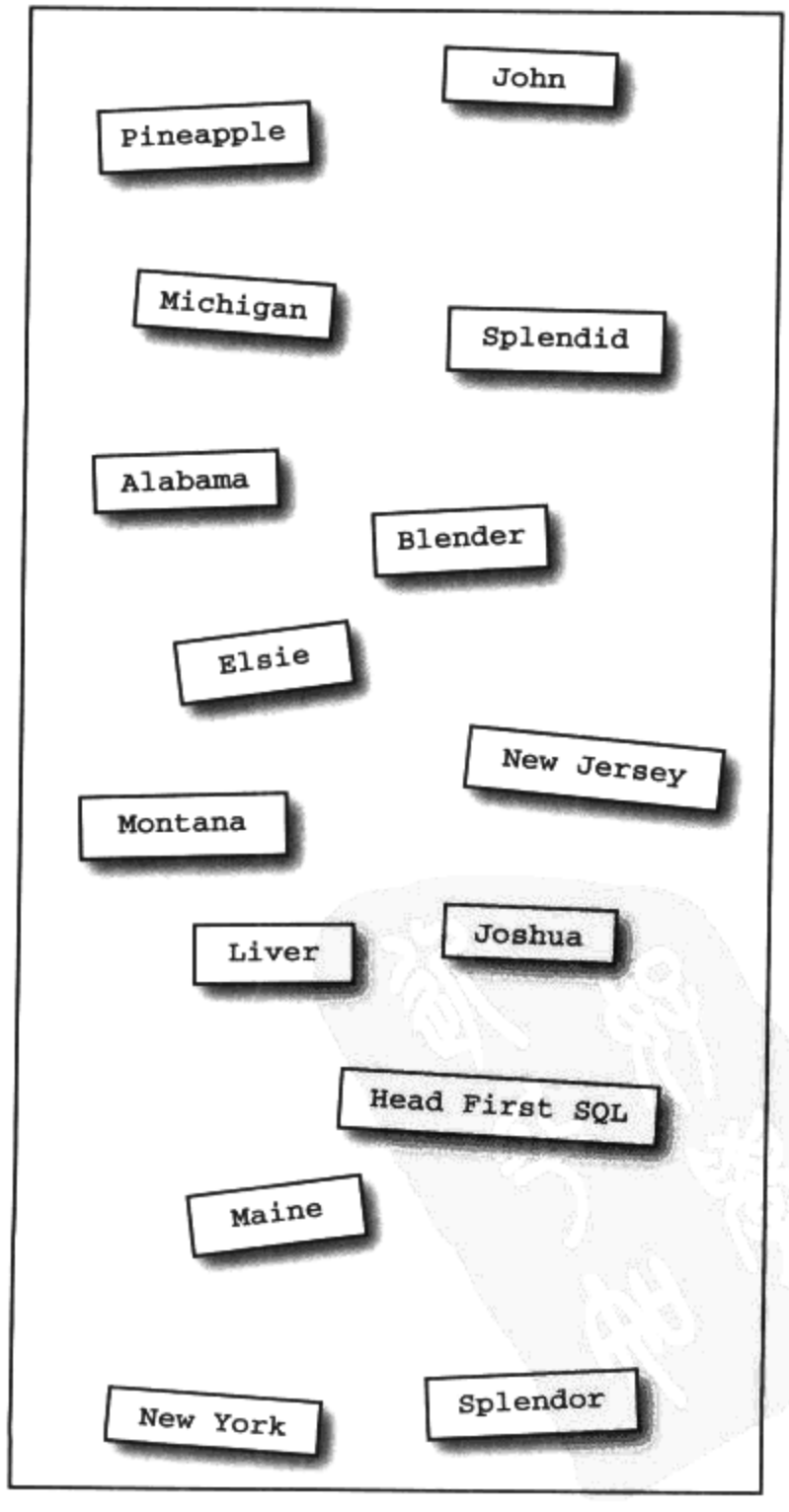
```
SELECT first_name FROM my_contacts
WHERE first_name LIKE '_im';
```

下划线只是一个未知字符的替身。

在 first_name 列中，只要在 "im" 前只有一个字符，例如 Jim、Kim、Tim，查询都会返回结果。

SQL冰箱磁铁

在冰箱上有很多包含LIKE的WHERE子句。你能正确找出各个子句以及它的查询结果吗？有些结果可能不只一枚磁铁。如果冰箱上还有剩下的磁铁，请写下新的 LIKE 语句和通配符来匹配出剩余的磁铁。




```
WHERE state LIKE 'New %';
```

```
WHERE cow_name LIKE '_lsie';
```

```
WHERE title LIKE 'HEAD FIRST%';
```

```

```

```
WHERE rhyme_word LIKE '%ender';
```

```

```

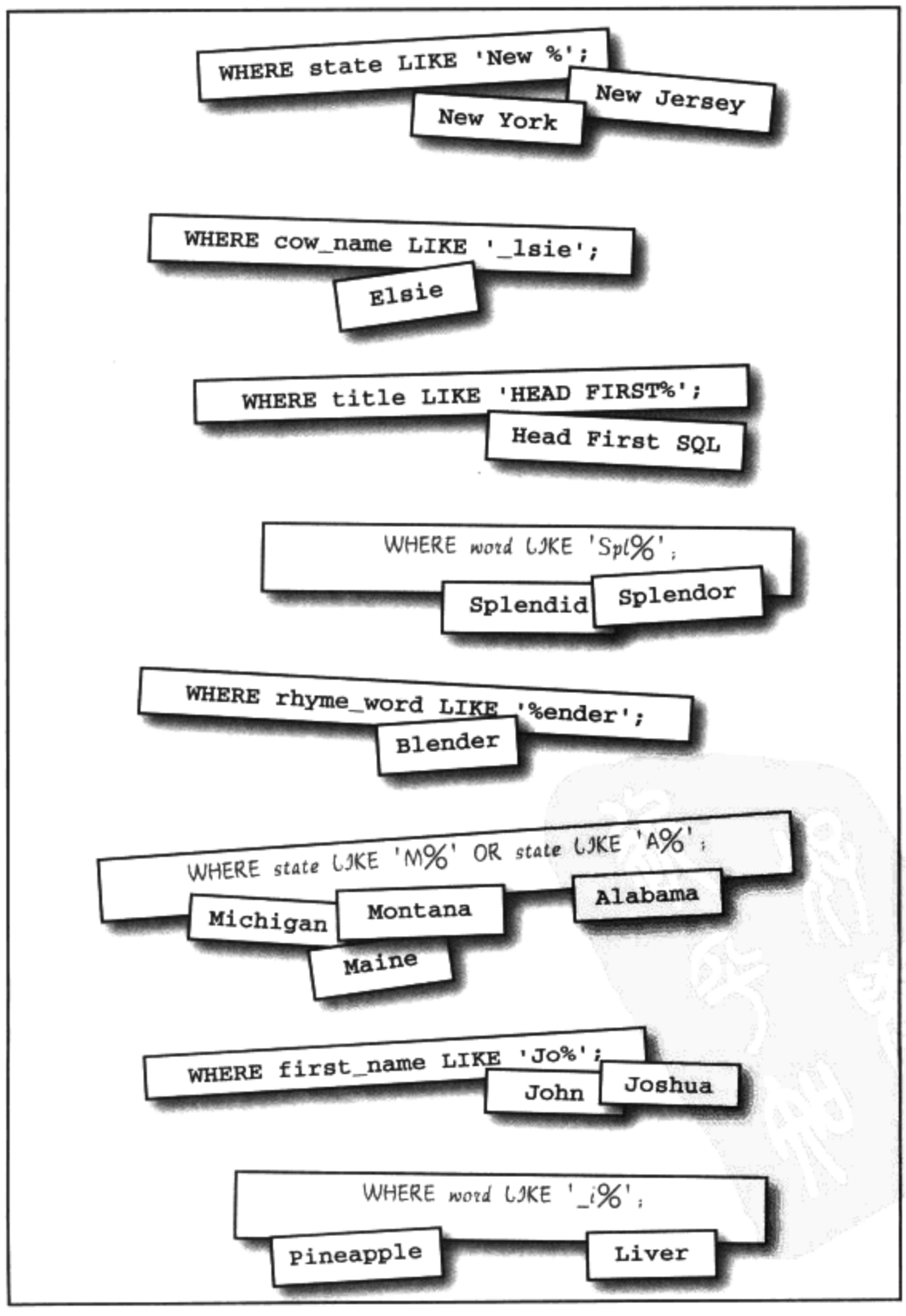
```
WHERE first_name LIKE 'Jo%';
```

```

```

SQL冰箱磁铁解答

在冰箱上有很多包含 LIKE 的 WHERE 子句。你能正确找出各个子句以及它的查询结果吗？有些结果可能不只一枚磁铁。如果冰箱上还有剩下的磁铁，请写下新的 LIKE 语句和通配符来匹配出剩余的磁铁。



利用AND和比较运算符选取一个范围

Head First Lounge的经营者现在想找出热量在某个范围内的饮料。应该如何查询，才能找出热量在 30 到 60 卡路里间（包括 30 和 60）的饮料的名称呢？

drink_info

drink_name	cost	carbs	color	ice	calories
Blackthorn	3	8.4	yellow	Y	33
Blue Moon	2.5	3.2	blue	Y	12
Oh My Gosh	3.5	8.6	orange	Y	35
Lime Fizz	2.5	5.4	green	Y	24
Kiss on the Lips	5.5	42.5	purple	Y	171
Hot Gold	3.2	32.1	orange	N	135
Lone Tree	3.6	4.2	red	Y	17
Greyhound	4	14	yellow	Y	50
Indian Summer	2.8	7.2	brown	N	30
Bull Frog	2.6	21.5	tan	Y	80
Soda and It	3.8	4.7	red	N	19

```
SELECT drink_name FROM drink_info
WHERE
calories >= 30
AND
calories <= 60;
```

查询结果会包括热量等于 30 卡路里、等于 60 卡路里以及在这个范围内的饮料。

偷偷告诉你……BETWEEN 更好

我们可以改用关键字BETWEEN。不仅查询的长度比较短，而且能返回相同结果。请注意：范围的起止值（30 和 60）也会包含在查找范围内。BETWEEN等于使用 <= 加 >=，但不等于 < 加 >。

```
SELECT drink_name FROM drink_info
WHERE
calories BETWEEN 30 AND 60;
```

这段查询的结果和前一页上的查询的完全相同。你看，省下了多少打字的时间啊！

选取范围包括 30 和 60卡路里。

```
File Edit Window Help MediumCalories
> SELECT drink_name FROM drink_info
WHERE
calories BETWEEN 30 AND 60;
+-------------------+
| drink_name        |
+-------------------+
| Blackthorn        |
| Oh My Gosh        |
| Greyhound         |
| Indian Summer     |
| Soda and It       |
+-------------------+
```

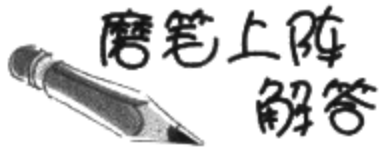
磨笔上阵

重新设计前一页的查询，改为 SELECT 所有热量高于 60 卡路里和低于 30 卡路里的饮料的名称。

试着在文本类型的列中运用 BETWEEN。写出SELECT以“G”到“O”为首字母的饮料的名称的查询。

你觉得这段查询会有什么结果？

```
SELECT drink_name FROM drink_info WHERE
calories BETWEEN 60 AND 30;
```



重新设计前一页的查询，改为 SELECT 所有热量高于 60 卡路里和低于 30 卡路里的饮料的名称。

```
SELECT drink_name FROM drink_info
WHERE
calories < 30 OR calories > 60;
```

查询热量高于 60 卡路里的饮料的名称。

查询热量低于 30 卡路里的饮料的名称。

试着在文本类型的列中运用 BETWEEN。写出SELECT以“G”到“O”为首字母的饮料的名称的查询。

```
SELECT drink_name FROM drink_info
WHERE
drink_name BETWEEN 'G' AND 'P';
```

这里是一个小陷阱。我们需要使用“O”之后的字母，以确保我们能得到“O”开头的饮料名称。自己尝试一下，看看结果如何。

你觉得这段查询会有什么结果？

```
SELECT drink_name FROM drink_info WHERE
calories BETWEEN 60 AND 30;
```

顺序很重要，这段查询不会有任何结果。

上例查找 60 到 30 间的值。但是60 到 30 间没有任何值，因为按数字排序时 60 比 30 更晚出现。较小的数值必须先交给 **BETWEEN**，解释的结果才会如同我们所期待的。

约会后，你的评价是IN……

Greg 的朋友Amanda通过 Greg 的联络名单认识了几位男士。约会的次数多了，她开发出自己的“黑皮书”来记录每次约会的印象。

Amanda 把自己的表命名为 `black _ book`。她现在想列出一份印象良好的约会对象列表，所以她用正面评价作为筛选条件。

```
SELECT date_name
FROM black_book
WHERE
rating = 'innovative'
OR
rating = 'fabulous'
OR
... ;
```

每一项正面评价都需要一行条件。

这些是正面评价。

black_book

date_name	rating
Alex	innovative
James	boring
Ian	fabulous
Boris	ho hum
Melvin	plebian
Eric	pathetic
Anthony	delightful
Sammy	pretty good
Ivan	dismal
Vic	ridiculous

除了使用这么多的 OR，我们也可以简单地利用关键字 IN，加上用括号围起的值的集合。只要列值匹配集合中的任何值，即返回该行或该列。

```
SELECT date_name
FROM black_book
WHERE
rating IN ('innovative',
'fabulous', 'delightful',
'pretty good');
```

用关键字 IN 来告诉 RDBMS，接下来是值的集合。

这一组是正面评价的集合。

```
File Edit Window Help GoodDates
> SELECT date_name FROM black_book
WHERE
rating IN ('innovative', 'fabulous',
'delightful', 'pretty good');

+------- ----+
| date_name  |
+------------+
| Alex       |
| Ian        |
| Anthony    |
| Sammy      |
+------------+
```

……不然就是NOT IN

当然，Amanda也想知道谁的评价很差，这样如果这些人打电话来，她就可以借口说自己正在洗头或是刚好很忙。

为了找出评价很差的人，我们将对目前的IN语句添加一个关键字：NOT。NOT能反转查询结果，找出值不在集合中的记录。


```
SELECT date_name
FROM black_book
WHERE
rating NOT IN ('innovative',
'fabulous', 'delightful',
'pretty good');
```

使用关键字 NOT IN，就是说查询结果不包含在值的集合中。

NOT IN 的查询结果是获得负面评价的人，他们不会有第二次机会了。

```
File Edit Window Help BadDates
> SELECT date_name FROM black_book
WHERE
rating NOT IN ('innovative', 'fabu-
lous', 'delightful', 'pretty good');

+------- ----+
| date_name  |
+------------+
| James      |
| Boris      |
| Melvin     |
| Eric       |
| Ivan       |
| Vic        |
+------------+
6 rows in set (2.43 sec)
```

动动脑

为什么有时会选用 NOT IN 而不是 IN 呢？

更多NOT

NOT 可以和BETWEEN或LIKE 一起使用。重点是记得 **NOT 一定要紧接在 WHERE 后面**。让我们看一些例子。

```
SELECT drink_name FROM drink_info
WHERE NOT carbs BETWEEN 3 AND 5;
```

```
SELECT date_name from black_book
WHERE NOT date_name LIKE 'A%'
AND NOT date_name LIKE 'B%';
```

当 NOT 和 AND或OR 一起使用时，则要直接接在 AND或OR 的后面。

问： 等一下，你刚才说 NOT 必须紧接在 WHERE 之后，如果是使用 NOT IN 呢？

答： NOT IN是个例外。而且即使把NOT移到WHERE后也可以运作。下面两组语句会返回相同结果：

```
SELECT * FROM easy_drinks
WHERE NOT main IN ('soda', 'iced tea');
```

```
SELECT * FROM easy_drinks
WHERE main NOT IN ('soda', 'iced tea');
```

问： 对 <>（不等）运算符而言，套用 NOT 的方式也一样吗？

答： 是可以这么做，但不就成了双重否定了吗？此时用等号更合理。以下两组语句会返回相同结果：

```
SELECT * FROM easy_drinks
WHERE NOT drink_name <> 'Blackthorn';
```

```
SELECT * FROM easy_drinks
WHERE drink_name = 'Blackthorn';
```

问： NOT 可以套用在 NULL 上吗？

答： 有人可能已猜到答案了：可以。要取得某列中所有不是 NULL 的值，可以这样查询：

```
SELECT * FROM easy_drinks
WHERE NOT main IS NULL;
```

不过这样查询也可以：

```
SELECT * FROM easy_drinks
WHERE main IS NOT NULL;
```

问： 若是搭配 AND 或 OR 呢？

答： 如果想在 AND 或 OR 子句中使用 NOT，请直接将它放在关键字后面，如下所示：

```
SELECT * FROM easy_drinks
WHERE NOT main = 'soda'
AND NOT main = 'iced tea';
```

重新设计这些 WHERE 子句，尽可能改写成最简单的形式。你可以向 AND、OR、NOT、BETWEEN、LIKE、IN、IS NULL以及比较运算符寻求帮助。需要时，请参考本章用到的表。

```
SELECT drink_name from easy_drinks
WHERE NOT amount1 < 1.50;
```

.....

.....

.....

```
SELECT drink_name FROM drink_info
WHERE NOT ice = 'Y';
```

.....

.....

.....

```
SELECT drink_name FROM drink_info
WHERE NOT calories < 20;
```

.....

.....

.....

```
SELECT drink_name FROM easy_drinks
WHERE main = IN ('peach nectar',
'soda');
```

.....

.....

.....

```
SELECT drink_name FROM drink_info
WHERE NOT calories = 0;
```

.....

.....

.....

```
SELECT drink_name FROM drink_info
WHERE NOT carbs BETWEEN 3 AND 5;
```

.....

.....

.....

```
SELECT date_name from black_book
WHERE NOT date_name LIKE 'A%'
AND NOT date_name LIKE 'B%';
```

.....

.....

.....

重新设计这些 WHERE 子句，尽可能改写成最简单的形式。你可以向 AND、OR、NOT、BETWEEN、LIKE、IN、IS NULL以及比较运算符寻求帮助。需要时，请参考本章用到的表。

```
SELECT drink_name from easy_drinks
WHERE NOT amount1 < 1.50;
```

```
SELECT drink_name FROM easy_drinks
WHERE amount1 >= 1.50;
```

```
SELECT drink_name FROM drink_info
WHERE NOT ice = 'Y';
```

```
SELECT drink_name FROM drink_info
WHERE ice = 'N';
```

```
SELECT drink_name FROM drink_info
WHERE NOT calories < 20;
```

```
SELECT drink_name FROM drink_info
WHERE calories >= 20;
```

```
SELECT drink_name FROM easy_drinks
WHERE main = IN ('peach nectar',
'soda');
```

```
SELECT drink_name FROM easy_drinks
WHERE main BETWEEN 'P' AND 'T';
```

你也可以使用这个WHERE子句：WHERE (BETWEEH 'P' AND 'T');。它也可以运作，因为我们并没有任何其它主要成分满足这个条件。但一般来说，当你有一个真实世界中的大表时，你并不知道里面是什么，这就是为什么你一开始就进行查询的原因。

```
SELECT drink_name FROM drink_info
WHERE NOT calories = 0;
```

```
SELECT drink_name FROM drink_info
WHERE calories > 0;
```

因为热量不可能是负值，所以可以放心地采用大于运算符。

```
SELECT drink_name FROM drink_info
WHERE NOT carbs BETWEEN 3 AND 5;
```

```
SELECT drink_name FROM drink_info
WHERE carbs < 3
OR
carbs >= 5;
```

```
SELECT date_name from black_book
WHERE NOT date_name LIKE 'A%'
AND NOT date_name LIKE 'B%';
```

```
SELECT date_name FROM black_book
WHERE date_name NOT BETWEEN 'A' AND 'C';
```

你的SQL工具包

第2章的内容已经收进你的工具包中，
运算符是我们学到的最新利器。

SELECT *
用于选择表中的所有列。

用 ' 与 \ 转义
字符串中的单引号前应该加上另一个单引号或反斜线来把它转换成直接量。

= <> < > <= >=
现在，这些相等和不相等运算符都在你的掌控中了。

IS NULL
可用于创建检查麻烦的 NULL 值的条件。

AND 与 OR
有了 AND 与 OR，就可以在 WHERE 子句中结合查询条件，让查询更精确。

NOT
NOT 反转查询结果，取得相反的值。

BETWEEN
选择一个范围内的值。

LIKE 搭配 % 或 _
使用 LIKE 搭配通配符，可搜索部分文本字符串。

你的新工具：运算符！

第59页的习题解答

Greg 想为“快速约会之夜”创建一张特调饮料表以供吧台人员查询调制方法。使用你在第1章学到的工具，创建第 59 页上所示的表并插入数据。

下表是 **drinks** 数据库的一部分，它包含的 **easy_drinks** 表里记录了只用两种成分调成的饮料。

```
CREATE DATABASE drinks;

USE drinks;

CREATE TABLE easy_drinks
(drink_name VARCHAR(16), main VARCHAR(20), amount1 DEC(3,1),
second VARCHAR(20), amount2 DEC(4,2), directions VARCHAR(250));
```

最好预留一些字符空间，以免日后增加的数据超出目前的限制。

```
INSERT INTO easy_drinks
VALUES
('Blackthorn', 'tonic water', 1.5, 'pineapple juice', 1, 'stir with ice, strain
into cocktail glass with lemon twist'), ('Blue Moon', 'soda', 1.5, 'blueberry
juice', .75, 'stir with ice, strain into cocktail glass with lemon twist'),
('Oh My Gosh', 'peach nectar', 1, 'pineapple juice', 1, 'stir with ice, strain
into shot glass'),
('Lime Fizz', 'Sprite', 1.5, 'lime juice', .75, 'stir with ice, strain into
cocktail glass'),
('Kiss on the Lips', 'cherry juice', 2, 'apricot nectar', 7, 'serve over ice
with straw'),
('Hot Gold', 'peach nectar', 3,' orange juice', 6, 'pour hot orange juice in mug
and add peach nectar'),
('Lone Tree', 'soda', 1.5, 'cherry juice', .75, 'stir with ice, strain into
cocktail glass'),
('Greyhound', 'soda', 1.5, 'grapefruit juice', 5, 'serve over ice, stir well'),
('Indian Summer', 'apple juice', 2, 'hot tea', 6, 'add juice to mug and top off
with hot tea'),
('Bull Frog', 'iced tea', 1.5, 'lemonade', 5, 'serve over ice with lime slice'),
('Soda and It', 'soda', 2, 'grape juice', 1, 'shake in cocktail glass, no ice');
```

别忘了：数字类型不需要单引号！

各种饮料的相关信息集合都要放在一对括号里。

每种饮料间则以逗号分隔。

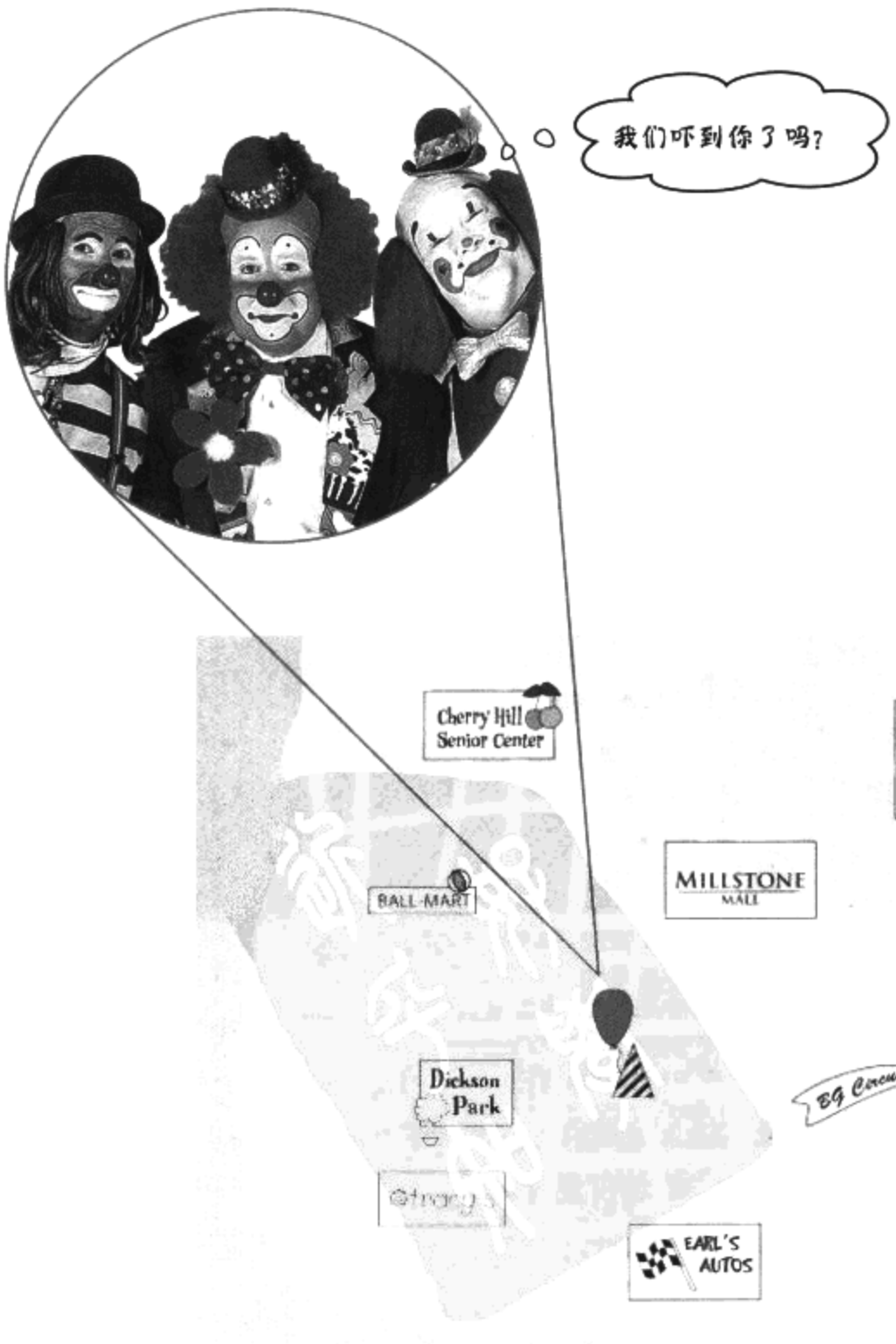
3 DELETE 和 UPDATE

改变是件好事

一直在改变你的心意吗？现在没有问题了！有了接下来会提到的命令 —— **DELETE** 和 **UPDATE**，我们不再受限于6个月前所做的决定，当时可能适合捕捞鳐鱼，但现在已经不是季节了。有了 UPDATE，我们可以**改变数据**，而 DELETE 则可**删除不需要的数据**。这一章不只是给你鱼竿，还会教你如何选择性地使用这些新能力，避免舍弃了需要的数据。

小丑真恐怖

假设我们要追踪出现在 Dataville 的小丑的行迹。我们可以先创建一份记录小丑资料的`clown_info`表，其中`last_seen`列用于记录小丑出现的地点。



追踪小丑

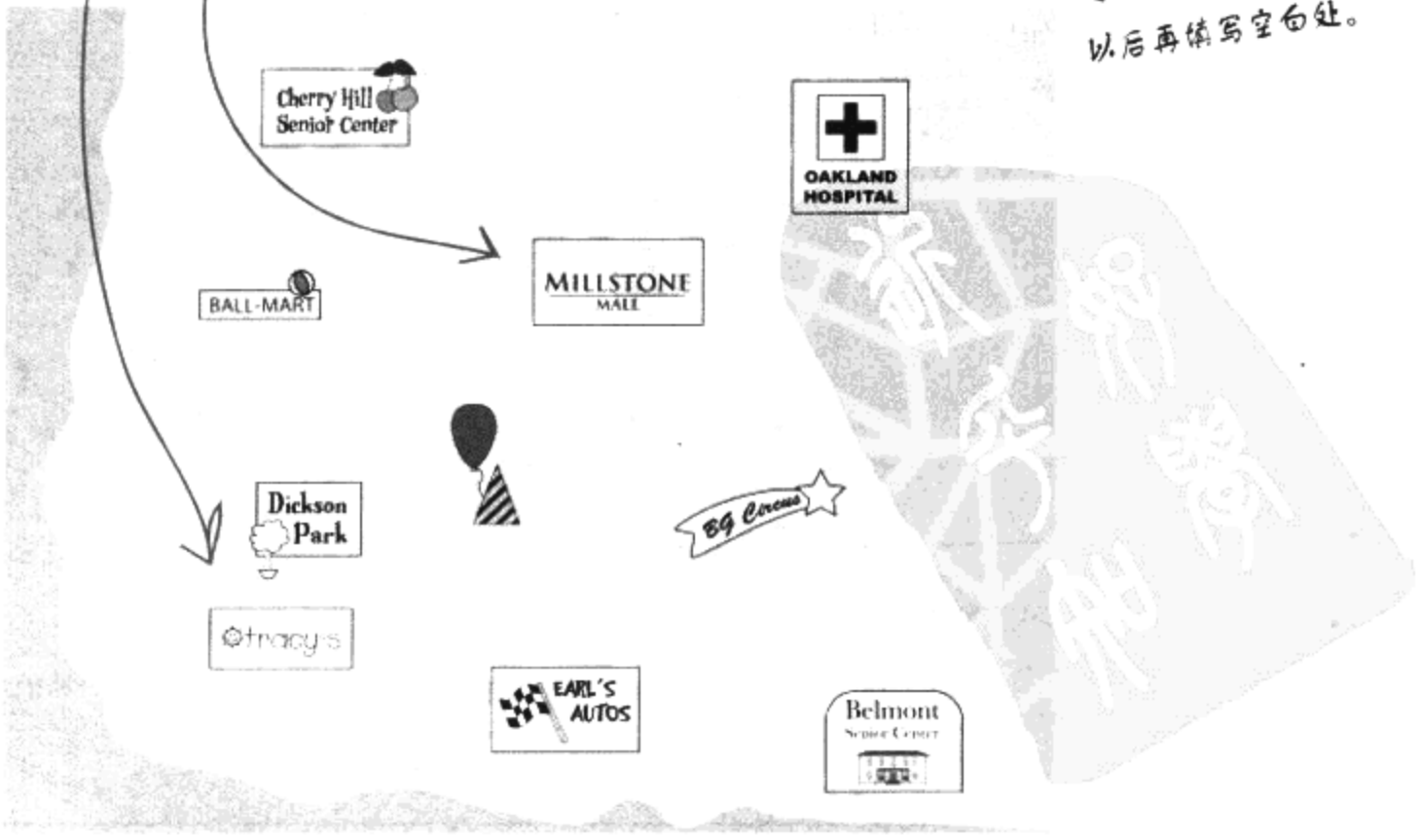
以下就是我们的表。我们可以先略过不知道的信息，以后再填入。每次有人看到小丑，我们就增加一条记录。所以需要频繁改变表，才能及时更新数据。

看到小丑的地点。

clown _ info

name	last_seen	appearance	activities
Elsie	Cherry Hill Senior Center	F, red hair, green dress, huge feet	balloons, little car
Pickles	Jack Green's party	M, orange hair, blue suit, huge feet	mime
Snuggles	Ball-Mart	F, yellow shirt, baggy red pants	horn, umbrella
Mr. Hobo	BG Circus	M, cigar, black hair, tiny hat	violin
Clarabelle	Belmont Senior Center	F, pink hair, huge flower, blue dress	yelling, dancing
Scooter	Oakland Hospital	M, blue hair, red suit, huge nose	balloons
Zippo	Millstone Mall	F, orange suit, baggy pants	dancing
Babe	Earl's Autos	F, all pink and sparkly	balancing, little car
Bonzo		M, in drag, polka dotted dress	singing, dancing
Sniffles	Tracy's	M, green and purple suit, pointy nose	

以后再填写空白处。



小丑的行踪飘忽不定

你的任务是编写 SQL 命令，以把每次目击报告输入 clown_info 表。请注意，有些小丑的信息不会每次都改变，所以请参考第 121 上页的表，以取得其他需要加入的信息。

Zippo spotted singing

```
INSERT INTO clown_info
VALUES
('Zippo', 'Millstone Mall', 'F, orange suit,
baggy pants', 'dancing, singing');
```

Snuggles now wearing baggy blue pants

```
INSERT INTO clown_info
VALUES
('Snuggles', 'Ball-Mart', 'F, yellow shirt, baggy
blue pants', 'horn, umbrella');
```

Bonzo sighted at Dickson Park

Sniffles seen climbing into tiny car

Mr. Hobo last seen at party for Eric Gray

接着给clown_info表输入数据，就像前两章那样使用INSERT 命令。

name	last_seen	appearance	activities
Elsie	Cherry Hill Senior Center	F, red hair, green dress, huge feet	balloons, little car
Pickles	Jack Green's party	M, orange hair, blue suit, huge feet	mime
Snuggles	Ball-Mart	F, yellow shirt, baggy red pants	horn, umbrella
Mr. Hobo	BG Circus	M, cigar, black hair, tiny hat	violin
Clarabelle	Belmont Senior Center	F, pink hair, huge flower, blue dress	yelling, dancing
Scooter	Oakland Hospital	M, blue hair, red suit, huge nose	balloons
Zippo	Millstone Mall	F, orange suit, baggy pants	dancing
Babe	Earl' s Autos	F, all pink and sparkly	balancing, little car
Bonzo		M, in drag, polka dotted dress	singing, dancing
Sniffles	Tracy's	M, green and purple suit, pointy nose	

磨笔上阵解答

小丑的行踪飘忽不定

你的任务是编写SQL命令，以把每次目击报告输入clown_info表，接着给clown_info表输入数据，就像前两章那样使用INSERT命令。

Zippo spotted singing

```
INSERT INTO clown_info
VALUES
('Zippo', 'Millstone Mall', 'F, orange suit,
baggy pants', 'dancing, singing');
```

Snuggles now wearing baggy blue pants

```
INSERT INTO clown_info
VALUES
('Snuggles', 'Ball-Mart', 'F, yellow shirt, baggy
blue pants', 'horn, umbrella');
```

Bonzo sighted at Dickson Park

```
INSERT INTO clown_info
VALUES
('Bonzo', 'Dickson Park', 'M, in drag, polka
dotted dress', 'singing, dancing');
```

Sniffles seen climbing into tiny car

```
INSERT INTO clown_info
VALUES
('Snibbles', 'Tracy\'s, 'M, green and purple suit,
pointy nose', 'climbing into tiny car');
```

别忘记 VARCHAR 值中的单引号要转义。

Mr. Hobo last seen at party for Eric Gray

```
INSERT INTO clown_info
VALUES
('Mr. Hobo', 'Party for Eric Gray', 'M, cigar,
black hair tiny hat', 'violin');
```

name	last_seen	appearance	activities
Elsie	Cherry Hill Senior Center	F, red hair, green dress, huge feet	balloons, little car
Pickles	Jack Green's party	M, orange hair, blue suit, huge feet	mime
Snuggles	Ball-Mart	F, yellow shirt, baggy red pants	horn, umbrella
Mr. Hobo	BG Circus	M, cigar, black hair, tiny hat	violin
Clarabelle	Belmont Senior Center	F, pink hair, huge flower, blue dress	yelling, dancing
Scooter	Oakland Hospital	M, blue hair, red suit, huge nose	balloons
Zippo	Millstone Mall	F, orange suit, baggy pants	dancing
Babe	Earl' s Autos	F, all pink and sparkly	balancing, little car
Bonzo		M, in drag, polka dotted dress	singing, dancing
Sniffles	Tracy's	M, green and purple suit, pointy nose	
Zippo	Millstone Mall	F, orange suit, baggy pants	dancing, singing
Snuggles	Ball-Mart	F, yellow shirt, baggy blue pants	horn, umbrella
Bonzo	Dickson Park	M, in drag, polka dotted dress	singing, dancing
Sniffles	Tracy's	M, green and purple suit, pointy nose	climbing into tiny car
Mr. Hobo	Party for Eric Gray	M, cigar, black hair, tiny hat	violin

动动脑

如何找出小丑现在的位置？

如何输入小丑数据

我们的小丑追踪机制都依赖于目击者的自愿汇报。有时候，小丑的行踪会搁置一到两个星期才被输入。有时候，可能**拆开了目击报告**，由两个人**同时输入数据**。

```
File Edit Window Help CatchTheClown
SELECT * FROM clown_info WHERE name = 'Zippo';
```

name	last_seen	appearance	activities
Zippo	Millstone Mall	F, orange suit, baggy pants	dancing
Zippo	Millstone Mall	F, orange suit, baggy pants	dancing, singing
Zippo	Oakland Hospital	F, orange suit, baggy pants	dancing, singing
Zippo	Tracy's	F, orange suit, baggy pants	dancing, singing
Zippo	Ball-Mart	F, orange suit, baggy pants	dancing, juggling
Zippo	Millstone Mall	F, orange suit, baggy pants	dancing, singing

这两条记录完全一样。

这两条记录也一样。

这条信息一直重复出现。

有没有办法只找出关于Zippo的最近一次的目击记录？你能找出她的位置吗？

很可惜，我们无法确定最后一条记录就是最新的目击报告。

同时有许多人在输入目击资料，而目击报告可能杂乱地放在收件箱中。就算最后一行真的是最新记录，**我们也不可以相信表中的记录真的按时间顺序排列。**

有几个数据库内部的因素可以改变行在表中存储的顺序。如采用的 RDBMS 软件，还有对列创建的索引（以后会讨论到索引）。

没人能够保证表的最后一行就是最新输入的记录。

Bonzo，我们出问题了

既然不能相信最后一条记录就是最新的记录，那我们的设计就出问题了。前面设计的小丑表列出了小丑曾在的地点。但是**表的主要目的应该是记录小丑最后出现的地点**。

不仅如此，注意到重复的记录了吗？有两条记录显示了Zippo在相同地方做相同的事情。重复的数据会占用空间，而且随着数据量的日益增加，总有一天会拖垮 RDBMS。**表中不应该存储重复的数据**。再过几章，我们会讨论数据重复的坏处，以及如何通过设计良好的表来避免重复情况的发生。

问： **为什么不能假设最后一条记录就是最新的记录？**

答： 因为表中记录的排序方式没有一定的规则，而且我们很快又要调整查询结果的顺序，所以实在无法保证表的最后一条记录是最后插入的记录。另外，单纯的人工操作错误也可能会搞乱顺序。假设我们为同一位小丑输入了两条 INSERT语句，除非我们记住哪份目击报告先进来，否则在数据输入后就没有分辨孰先孰后的方式了。

问： **假设我们记得顺序，为什么还是不能采用最后一条记录呢？**

答： 让我们扩展这个例子。如果我们追踪这些小丑的行踪已经好多年了，或许用了很多名助理负责追踪并输入小丑们的记录。有些小丑的目击记录可能有好几百条。当我们要选出这些记录时，就要在好几百行记录里找出最后一条，而且还要祈祷它就是最新的记录。

问： **我们平时真的会想把这类数据保存在表中吗？总是 INSERT 新记录，还把旧记录放在表中合理吗？**

答： 当然合理。仍然以小丑的行踪为例。表现在不仅能提供某个小丑最近被目击到的地点，还能提供小丑们的行踪历史。这是一种具备潜在作用的信息。问题出在目击到小丑的时间上，我们没有关于这一点的信息。如果加入一列记录日期与时间，小丑追踪的准确度立刻就可得到提升。

不过，目前要先把重复的记录从表中删除，才能简化处理过程。

问： **好吧，等我看完这本书，我就会知道该怎样设计没有重复数据的表。但如果我接下别人的任务，而他留给我一个设计很差的表，我该怎么办呢？**

答： 设计很差的表在现实生活中垂手可得，很多学习 SQL 的人都必须修正其他人设计的 SQL 烂账。

有很多技巧能清除重复的数据，如利用联接（join），稍后会在其他章节讨论。目前，我们还没学到修正糟糕数据的新工具，但耐心点，日后一你定会遇到。

用DELETE删除记录

看起来我们要先从删除某些记录开始了。为了让我们的表更为有用，每个小丑应该只可以占用一行。当我们在等待关于 Zippo 的新目击报告（想必是最新记录）传入时，可以先删除与 Zippo 相关的某些太旧、对我们没有用的信息。

DELETE 语句就是从表中删除一行数据所需的工具，它也使用与上一章相同的 WHERE 子句。请你试着设计删除数据的语法，再和我们的范例进行比较。

下面再度列出了 Zippo 的行踪记录。

name	last_seen	appearance	activities
Zippo	Millstone Mall	F, orange suit, baggy pants	dancing
Zippo	Millstone Mall	F, orange suit, baggy pants	dancing, singing
Zippo	Oakland Hospital	F, orange suit, baggy pants	dancing, singing
Zippo	Tracy's	F, orange suit, baggy pants	dancing, singing
Zippo	Ball-Mart	F, orange suit, baggy pants	dancing, juggling
Zippo	Millstone Mall	F, orange suit, baggy pants	dancing, singing
Zippo	Oakland Hospital	F, orange suit, baggy pants	dancing, singing

冰箱上的DELETE磁铁

我们写下了一个用于删除某条 Zippo 记录的简单命令，不过它的各个组成部分被随意地贴在冰箱上。请重组各个部分，并注明它们在新的删除命令中的用途。

dancing

WHERE

DELETE

clown_info

name

activities

singing

FROM

Zippo

单引号、逗号、相等运算符、分号，它们都小得捡不起来。请在需要时加上这些符号。

冰箱上的DELETE磁铁解答

我们写下了一个用于删除某条 Zippo 记录的简单命令，不过它的各个组成部分被随意地贴在冰箱上。请重组各个部分，并注明它们在新的删除命令中的用途。

和SELECT不一样，我们不需要指定删除的对象。它会删除整条记录。

```
DELETE
```

指定要从中移除记录的表。

```
FROM  clown_info
```

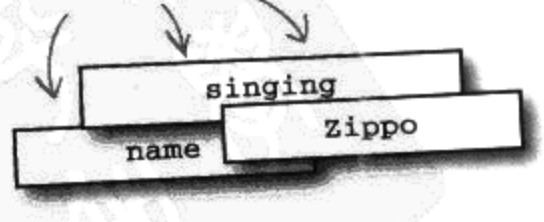
这里应该加上相等运算符，单引号则需要出现在 dancing 的两侧，最后还要加上分号来结束命令。

```
WHERE  activities  =  dancing  ;
```

确定自己没有忘记 WHERE 子句，否则将会删除所有行！

前一章学到的WHERE子句实在很方便。它也是对DELETE 指定删除对象的方式。

这段命令中不需要这几个磁铁。



DELETE语句可以和WHERE子句搭配使用，使用方式和SELECT与WHERE的搭配方式一样。

使用新学会的DELETE语句

让我们执行刚刚创建的 DELETE 语句，它的行为就像它的名字一样。所有符合 WHERE 条件的记录都会从表中被删除。

```
DELETE FROM clown_info
WHERE
activities = 'dancing';
```

name	last_seen	appearance	activities
Elsie	Cherry Hill Senior Center	F, red hair, green dress, huge feet	balloons, little car
Pickles	Jack Green's party	M, orange hair, blue suit, huge feet	mime
Snuggles	Ball-Mart	F, yellow shirt, baggy red pants	horn, umbrella
Mr. Hobo	BG Circus	M, cigar, black hair, tiny hat	violin
Clarabelle	Belmont Senior Center	F, pink hair, huge flower, blue dress	yelling, dancing
Scooter	Oakland Hospital	M, blue hair, red suit, huge nose	balloons
Zippo	Millstone Mall	F, orange suit, baggy pants	dancing
Babe	Earl' s Autos	F, all pink and sparkly	balancing, little car
Bonzo		M, in drag, polka dotted dress	singing, dancing
Sniffles	Tracy's	M, green and purple suit, pointy nose	
Zippo	Millstone Mall	F, orange suit, baggy pants	singing
Snuggles	Ball-Mart	F, yellow shirt, baggy blue pants	horn, umbrella
Bonzo	Dickson Park	M, in drag, polka dotted dress	singing, dancing
Sniffles	Tracy's	M, green and purple suit, pointy nose	climbing into tiny car
Mr. Hobo	Party for Eric Gray	M, cigar, black hair, tiny hat	violin

这是将被删除的记录。

动动脑

你觉得你可以使用 DELETE 从表中删除一条记录的某一列吗？

DELETE的规则

- DELETE不能删除单一列中的值或表中某一列的所有值。
- DELETE可用于删除一行或多行，根据WHERE子句而定。
- 我们已经知道如何从表中删除一行，也可以删除多行。为了实现这个目标，我们利用WHERE子句告诉DELETE该选择哪些行。WHERE子句和第2章中搭配SELECT时的使用方法完全相同，凡是第2章用于WHERE子句中的关键字，如LIKE、IN、BETWEEN，都可以在此处使用，而且所有条件都能更准确地要求RDBMS删除特定行。
- 还有，这一段语句可以删除表中的每一行：

```
DELETE FROM your_table
```

问： 用WHERE搭配DELETE与用WHERE搭配SELECT有什么不同吗？

答： 没有不同。WHERE都一样，只是SELECT和DELETE所做的不同。SELECT从符合WHERE条件的行中返回列的副本，但不会修改表。DELETE则移除所有符合WHERE条件的行，而且会移除整行。

与 DELETE 加 WHERE 子句天人合一

你要与下列搭配AND与OR的DELETE加WHERE子句天人合一，并判断他们是否会删除任何行。

划掉各个查询会删除的行：

```
DELETE FROM doughnut_ratings

WHERE location  = 'Krispy King' AND  rating <> 6;

WHERE location  = 'Krispy King' AND  rating = 3;

WHERE location  = 'Snappy Bagel' AND  rating >= 6;

WHERE location  = 'Krispy King' OR  rating > 5;

WHERE location  = 'Krispy King' OR  rating = 3;

WHERE location  = 'Snappy Bagel' OR  rating = 3;
```

doughnut_ratings

location	time	date	type	rating	comments
Krispy King	8:50 am	9/27	plain glazed	10	almost perfect
Duncan' s Donuts	8:59 am	8/25	NULL	6	greasy
Starbuzz Coffee	7:35 pm	5/24	cinnamon cake	5	stale, but tasty
Duncan' s Donuts	7:03 pm	4/26	jelly	7	not enough jelly

与DELETE加WHERE子句天人合一解答

你要与下列搭配AND与OR的DELETE加WHERE子句天人合一，并判断他们是否会删除任何行。

```
DELETE FROM doughnut_ratings
```

划掉各个查询会删除的行：

```
WHERE location  = 'Krispy King' AND  rating <> 6;
```

```
WHERE location  = 'Krispy King' AND  rating = 3;
```

无相符数据，不进行 DELETE

```
WHERE location  = 'Snappy Bagel' AND  rating >= 6;
```

```
WHERE location  = 'Krispy King' OR  rating > 5;
```

```
WHERE location  = 'Krispy King' OR  rating = 3;
```

```
WHERE location  = 'Snappy Bagel' OR  rating = 3;
```

无相符数据，不进行DELETE

doughnut_ratings

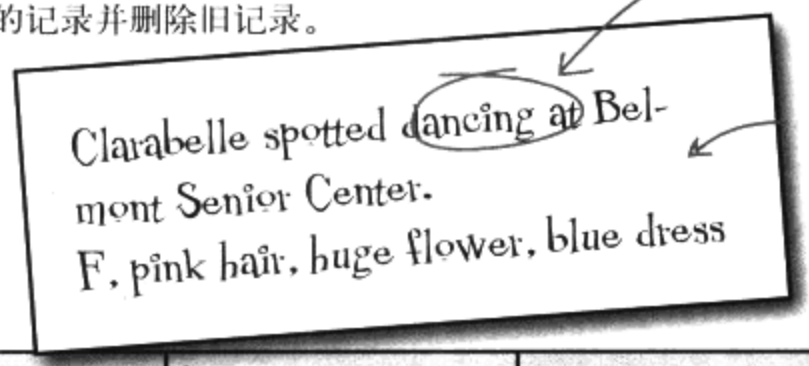
location	time	date	type	rating	comments
Krispy King	8:50 am	9/27	plain glazed	10	almost perfect
Duncan's Donuts	8:59 am	8/25	NULL	6	greasy
Starbuzz Coffee	7:35 pm	5/24	cinnamon cake	5	stale, but tasty
Duncan's Donuts	7:03 pm	4/26	jelly	7	not enough jelly

这些 NULL 值可能会在日后的查询中造成问题。最好在列中输入某些值，而不是放任它继续为 NULL，因为NULL无法用相等条件表达式找出来。

INSERT-DELETE双步运作

在整个表中，Clarabelle的记录只有一条。既然我们只希望为每个小丑保留一条最新目击地点的记录，现在就只需要创建新的记录并删除旧记录。

只有表演的活动与当前行不同。



我们的工作是把这些数据输入表内。为了节省页数，这里只列出第131页上的表中的一行。

name	last_seen	appearance	activities
Clarabelle	Belmont Senior Center	F, pink hair, huge flower, blue dress	yelling, dancing

❶ 首先，以 INSERT 添加新的信息（以及所有旧信息）。

```
INSERT INTO clown_info
VALUES
('Clarabelle', 'Belmont Senior Center', 'F, pink hair,
huge flower, blue dress', 'dancing');
```

INSERT 时使用所有原始数据，只修改需要修改的列。

name	last_seen	appearance	activities
Clarabelle	Belmont Senior Center	F, pink hair, huge flower, blue dress	yelling, dancing
Clarabelle	Belmont Senior Center	F, pink hair, huge flower, blue dress	dancing

INSERT

❷ 接下来，利用 DELETE 搭配 WHERE 子句删除旧记录。

```
DELETE FROM clown_info
WHERE
activities='yelling,dancing'
AND name = 'Clarabelle';
```

使用WHERE子句查找要删除的旧记录。

现在表中只剩下新记录了。

name	last_seen	appearance	activities
Clarabelle	Belmont Senior Center	F, pink hair, huge flower, blue dress	dancing

磨笔上阵

根据下列要求，使用 INSERT 和 DELETE 改变表 drink_info，然后于右页画出改变后的表。

drink_info

drink_name	cost	carbs	color	ice	calories
Blackthorn	3	8.4	yellow	Y	33
Blue Moon	2.5	3.2	blue	Y	12
Oh My Gosh	3.5	8.6	orange	Y	35
Lime Fizz	2.5	5.4	green	Y	24
Kiss on the Lips	5.5	42.5	purple	Y	171
Hot Gold	3.2	32.1	orange	N	135
Lone Tree	3.6	4.2	red	Y	17
Greyhound	4	14	yellow	Y	50
Indian Summer	2.8	7.2	brown	N	30
Bull Frog	2.6	21.5	tan	Y	80
Soda and It	3.8	4.7	red	N	19

把 Kiss on the Lips 的热量（calories）改为 170。

把所有 yellow 值改为 gold。

drink_info

drink_name	cost	carbs	color	ice	calories
Blackthorn					
Blue Moon					
Oh My Gosh					
Lime Fizz					
Kiss on the Lips					
Hot Gold					
Lone Tree					
Greyhound					
Indian Summer					
Bull Frog					
Soda and It					

把所有定价 2.5 美元的饮料改为 3.5美元，并把所有定价 3.5 美元的饮料改为 4.5 美元。

根据下列要求，使用 INSERT 和 DELETE 改变表drink_info，然后于右页画出改变后的表。

drink _ info

drink_name	cost	carbs	color	ice	calories
Blackthorn	3	8.4	yellow	Y	33
Blue Moon	2.5	3.2	blue	Y	12
Oh My Gosh	3.5	8.6	orange	Y	35
Lime Fizz	2.5	5.4	green	Y	24
Kiss on the Lips	5.5	42.5	purple	Y	171
Hot Gold	3.2	32.1	orange	N	135
Lone Tree	3.6	4.2	red	Y	17
Greyhound	4	14	yellow	Y	50
Indian Summer	2.8	7.2	brown	N	30
Bull Frog	2.6	21.5	tan	Y	80
Soda and It	3.8	4.7	red	N	19

把 Kiss on the Lips 的热量（calories）改为 170。

```
INSERT INTO drink_info VALUES ('Kiss on the Lips', 5.5, 42.5, 'purple', 'Y', 170);
DELETE FROM drink_info WHERE calories = 171;
```

把所有 yellow 值改为 gold。

```
INSERT INTO drink_info VALUES ('Blackthorn', 3, 8.4, 'gold', 'Y', 33),
('Greyhound', 4, 14, 'gold', 'Y', 50);
DELETE FROM drink_info WHERE color = 'yellow';
```

drink_info

drink_name	cost	carbs	color	ice	calories
Blackthorn	3	8.4	gold	Y	33
Blue Moon	3.5	3.2	blue	Y	12
Oh My Gosh	4.5	8.6	orange	Y	35
Lime Fizz	3.5	5.4	green	Y	24
Kiss on the Lips	5.5	42.5	purple	Y	170
Hot Gold	3.2	32.1	orange	N	135
Lone Tree	3.6	4.2	red	Y	17
Greyhound	4	14	gold	Y	50
Indian Summer	2.8	7.2	brown	N	30
Bull Frog	2.6	21.5	tan	Y	80
Soda and It	3.8	4.7	red	N	19

完成所有改变后，你的表应该像这样。顺序或许不太一样，不过请记得，呈现的顺序其实没有任何意义。

这题没有什么诡计，只不过需要稍微思考一下。如果先把定价 2.5 美元的饮料改为 3.5 美元，稍后其他饮料的价格从 3.5 美元变为 4.5 美元时，它们的定价也会跟着提高，Blue Moon的价格就会被提高 2 美元。所以，我们要先改变定价较高的值（把3.5 美元改为 4.5 美元），再把2.5美元的 Blue Moon 改为 3.5 美元。

这题是你的另一个诡计吗？

把所有定价 2.5 美元的饮料改为 3.5美元，并把所有定价 3.5 美元的饮料改为 4.5 美元。

```
INSERT INTO drink_info VALUES ('Oh My Gosh', 4.5, 8.6, 'orange', 'Y', 35);

DELETE FROM drink_info WHERE cost = 3.5;

INSERT INTO drink_info VALUES ('Blue Moon', 3.5, 3.2, 'blue', 'Y', 12),

('Lime Fizz', 3.5, 5.4, 'green', 'Y', 24);

DELETE FROM drink_info WHERE cost = 2.5;
```

如果你把插入两组值的 INSERT 写成一条语句，请为自己加分喔！

慎用DELETE

每次删除记录时，其实都有意外删除你不想删除的记录的风险。假设我们要为 Mr. Hobo 添加一条新记录。

谨慎使用 DELETE。确认自己加入了非常精确的WHERE子句，可以只选出你真正想要删除的行。

这是我们要插入的信息，以及用于执行的 INSERT 语句。

Mr. Hobo sighted at Tracy's

别忘了要在这种单引号前加入反斜线进行字符转义。

```
INSERT INTO clown_info
VALUES
('Mr. Hobo', 'Tracy\'s', 'M, cigar,
black hair, tiny hat', 'violin');
```

name	last_seen	appearance	activities
Elsie	Cherry Hill Senior Center	F, red hair, green dress, huge feet	balloons, little car
Pickles	Jack Green's party	M, orange hair, blue suit, huge feet	mime
Snuggles	Ball-Mart	F, yellow shirt, baggy red pants	horn, umbrella
Mr. Hobo	Oakland Hospital	M, cigar, black hair, tiny hat	violin
Clarabelle	Belmont Senior Center	F, pink hair, huge flower, blue dress	yelling, dancing
Scooter	Oakland Hospital	M, blue hair, red suit, huge nose	balloons
Zippo	Millstone Mall	F, orange suit, baggy pants	dancing, singing
Babe	Earl's Autos	F, all pink and sparkly	balancing, little car
Bonzo		M, in drag, polka dotted dress	singing, dancing
Sniffles	Tracy's	M, green and purple suit, pointy nose	
Zippo	Millstone Mall	F, orange suit, baggy pants	singing
Snuggles	Dickson Park	F, yellow shirt, baggy blue pants	horn, umbrella
Bonzo	Ball -Mart	M, in drag, polka dotted dress	singing, dancing
Sniffles	Tracy's	M, green and purple suit, pointy nose	climbing into tiny car
Mr. Hobo	Dickson Park	M, cigar, black hair, tiny hat	violin
Mr. Hobo	Tracy's	M, cigar, black hair, tiny hat	violin

删除重复的行

现在是 DELETE 的时候了。

与DELETE天人合一

下面是提供给DELETE使用的WHERE子句，用于清理旁边的*clown_info*表。请判断哪些子句能帮我们删除数据，哪些子句则会帮倒忙。

这些子句有帮助吗？如果没有，请说明原因。

```
DELETE FROM clown_info
```

```
WHERE last_seen = 'Oakland Hospital';
```

.....

.....

```
WHERE activities = 'violin';
```

.....

.....

```
WHERE last_seen = 'Dickson Park'
AND name = 'Mr. Hobo';
```

.....

.....

```
WHERE last_seen = 'Oakland Hospital' AND
last_seen = 'Dickson Park';
```

.....

.....

```
WHERE last_seen = 'Oakland Hospital' OR
last_seen = 'Dickson Park';
```

.....

.....

```
WHERE name = 'Mr. Hobo'
OR last_seen = 'Oakland Hospital';
```

.....

.....

请写下一条 **DELETE** 语句，让它清除所有旧的 **Mr. Hobo** 记录且不会动到其他记录。

.....

.....

.....

.....

与DELETE天人合一解答

下面是提供给 DELETE 使用的 WHERE 子句，用于清理旁边的 clown_info 表。请判断哪些子句能帮我们删除数据，哪些子句则会帮倒忙。

这些子句有帮助吗？如果没有，请说明原因。

```
DELETE FROM clown_info
WHERE last_seen = 'Oakland Hospital';
```

Scooter 也有一行相符的数据。

本句删除 Mr. Hobo 的一条记录。
也会删除 Scooter 的记录。

```
WHERE activities = 'violin';
```

我不想删除新的记录。

删除所有 Mr. Hobo 的记录，包括新的记录。

```
WHERE last_seen = 'Dickson Park'
AND name = 'Mr. Hobo';
```

只删除 Mr. Hobo 的一条旧记录。

```
WHERE last_seen = 'Oakland Hospital'
AND last_seen = 'Dickson Park';
```

AND 表示这两个条件都要成立。

不会删除任何内容。

```
WHERE last_seen = 'Oakland Hospital'
OR last_seen = 'Dickson Park';
```

删除Bonzo和Scooter的记录，同时也删除 Mr. Hobo 的旧记录。

```
WHERE name = 'Mr. Hobo'
OR last_seen = 'Oakland Hospital';
```

删除所有 Mr. Hobo 的记录，包括新记录，同时删除 Scooter 的记录。

请写下一条 DELETE 语句，让它清除所有旧的 Mr. Hobo 记录且不会动到其他记录。

```
DELETE FROM clown_info
WHERE name = 'Mr. Hobo'
AND last_seen <> 'Tracy\'s';
```

看起来好像有些你不想删除的数据被删除了。也许应该先SELECT一下，看看哪些数据会被某些WHERE子句删除。

说得太好了！除非你可以非常确定WHERE子句只会删除你打算删除的行，否则都应该用 SELECT 确认情况。

因为它们使用的WHERE子句都是一样的，所以 SELECT 返回的行会反映出 DELETE 加上相同 WHERE 子句后会删除的行。

这是一个确保不会意外删除所需数据的安全方式，而且也有助于选出所有要删除的记录。

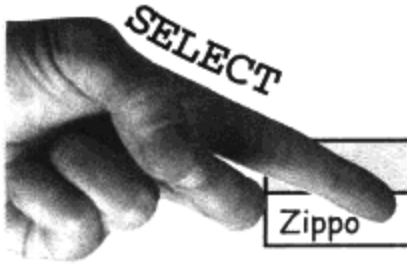
DELETE不精确的麻烦

DELETE 真的很棘手，稍不小心，它就会瞄错删除的对象。我们可以在INSERT-DELETE的两个步骤间增加一个步骤来避免删错数据。

先使用 SELECT 语句，确定只改变了你真正想要改变的记录。

❶ 首先，用 SELECT 挑出你必须移除的记录，确认记录无误而且没有误删其他记录。

```
SELECT * FROM clown_info
WHERE
activities = 'dancing';
```



	last_seen	appearance	activities
Zippo	Millstone Mall	F, orange suit, baggy pants	dancing

❷ 下一步，用 INSERT 插入新记录。

```
INSERT INTO clown_info
VALUES
('Zippo', 'Millstone Mall', 'F, orange suit,
baggy pants', 'dancing, singing');
```

INSERT 记录时需要使用所有原始数据，但只修改需要改变的列。

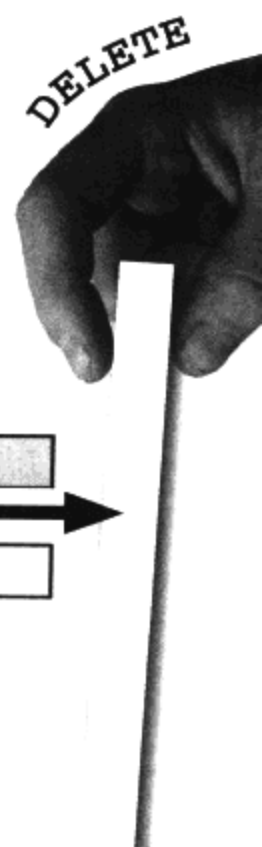
name	last_seen	appearance	activities
Zippo	Millstone Mall	F, orange suit, baggy pants	dancing

Zippo	Millstone Mall	F, orange suit, baggy pants	dancing, singing

❸ 最后，用 DELETE 删除旧记录，记得要用第一步的 SELECT 所用的 WHERE 子句。

```
DELETE FROM clown _ info
WHERE
activities = 'dancing';
```

使用新的第一步中SELECT记录时所用的WHERE子句来查找并DELETE旧记录。



name	last_seen	appearance	activities
Zippo	Millstone Mall	F, orange suit, baggy pants	dancing, singing

现在只剩下新记录了。

name	last_seen	appearance	activities
Zippo	Millstone Mall	F, orange suit, baggy pants	dancing, singing

如果可以只用一个步骤改变记录，而不用担心新旧记录一起被删除，那该有多好啊！不过，这可能只是我的白日梦吧……

以UPDATE改变数据

到现在为止，各位对使用INSERT和DELETE来更新表已经驾轻就熟了。我们也讨论了一起使用它们来间接调整某一行的方式。

但是与其插入新行后再删除旧行，其实可以重新使用已经存在的记录，真正做到只调整需要改变的列。

我们会用到的SQL语句是UPDATE。正如其名，它能更新一列或多列的值。就像SELECT和DELETE，UPDATE也能通过使用WHERE子句来精确地指定要更新的行。

这里是我们指定新值的地方。

```
UPDATE doughnut _ ratings
SET
type = 'glazed'
WHERE type = 'plain glazed';
```

这里是标准的WHERE子句，就跟以前在SELECT和DELETE中的使用方式一样。

关键字 SET 告诉 RDBMS，它要把 WHERE 子句提到的原始列值改为它这一句中的值。以上例而言，'plain glazed' 会被改为'glazed'。WHERE子句指出只改变type列的值是'plain glazed'的行。

doughnut _ ratings

location	time	date	type	rating	comments
Krispy King	8:50 am	9/27	plain glazed	10	almost perfect
Duncan' s Donuts	8:59 am	8/25	NULL	6	greasy
Starbuzz Coffee	7:35 pm	5/24	cinnamon cake	5	stale, but tasty
Duncan' s Donuts	7:03 pm	4/26	jelly	7	not enough jelly

doughnut_ratings

location	time	date	type	rating	comments
Krispy King	8:50 am	9/27	glazed	10	almost perfect
Duncan' s Donuts	8:59 am	8/25	NULL	6	greasy
Starbuzz Coffee	7:35 pm	5/24	cinnamon cake	5	stale, but tasty
Duncan' s Donuts	7:03 pm	4/26	jelly	7	not enough jelly

UPDATE的规则

- 使用 UPDATE，你可以改变单一列或所有列的值。在SET 子句中加入更多 column = value 组，其间以逗号分隔：

```
UPDATE your_table
SET first_column = 'newvalue',
second_column = 'another_value';
```

- UPDATE 可用于更新单一行或多行，一切都交给 WHERE 子句决定。

问： 如果不加上 WHERE 子句，会发生什么事？

答： SET 子句提到的表中的每行的每列都会被修改为新值。

问： 左页的SQL查询中有两个等号，但是它们的作用好像不太一样，是吗？

答： 是的。在 SET 子句中的等号表示“把这一列设定为这个值”，至于WHERE子句中的等号则是检查列值是否与等号右边的值相等。

问： 我能用下面这条语句完成相同的事吗？

```
UPDATE doughnut_ratings SET type =
'glazed' WHERE location = 'Krispy King';
```

答： 是的，可以。这条语句会以相同方式更新相同行。对于我们这个只有4行的表而言，效果还不错。但如果在有成千上百条记录的表中使用这种方式，你将会改变每个 Knispy King 行的 type 列的值。

问： 哦！那我该怎么确保只更新了需要的部分呢？

答： 就像刚才DELETE的前置动作一样，除非确保WHERE的子句瞄准了正确的行，否则就先用SELECT确认一下吧！

问： 语句中可以放入多个SET子句吗？

答： 不行，而且你也不需要这样。你可以把所有要更改的列和值都放在一个 SET 子句中，如前例所示。

UPDATE是我们的新INSERT-DELETE

使用UPDATE时，完全不会删除任何内容，只是把旧记录回收并替换为新记录。

以UPDATE开始……

……然后写下包含要更新记录的表的名称。

SET 用于指定要对记录做的改变。

```
UPDATE table_name
SET column_name = newvalue
WHERE column_name = somevalue;
```

可靠的 WHERE 子句能帮助我们精确地瞄准要改变的记录。

UPDATE语句可以取代DELETE与INSERT的组合

让我们看看这个命令实际套用在 clown_info 表上的行动。

更新clown_info表中的一条记录。

改变Tracy在last_seen列中的值。

别忘记用于转换单引号意义的反斜线。

```
UPDATE clown_info
SET last_seen = 'Tracy\'s'
WHERE name = 'Mr. Hobo'
AND last_seen = 'Dickson Park';
```

这里的 WHERE 子句用于精确指定要改变的记录——本例中，Mr. Hobo的记录的last_seen值是Dickson Park。

UPDATE 在行动

使用UPDATE，Mr. Hobo的记录的last_seen列值已从Party fro Eric Gray 改变为Tracy's。

Mr. Hobo sighted at Tracy's

这是需要添加的信息，我们会用 UPDATE 来实现。

```
UPDATE clown_info
SET last_seen = 'Tracy\'s'
WHERE name = 'Mr. Hobo'
AND last_seen = 'Party for Eric Gray';
```

name	last_seen	appearance	activities
Elsie	Cherry Hill Senior Center	F, red hair, green dress, huge feet	balloons, little car
Pickles	Jack Green's party	M, orange hair, blue suit, huge feet	mime
nuggles	Ball-Mart	F, yellow shirt, baggy red pants	horn, umbrella
Hobo	BG Circus	M, cigar, black hair, tiny hat	violin
abelle	Belmont Senior Center	F, pink hair, huge flower, blue dress	yelling, dancing
	Oakland Hospital	M, blue hair, red suit, huge nose	balloons
	illstone Mall	F, orange suit, baggy pants	dancing, singing
	rl' s Autos	F, all pink and sparkly	balancing, little car
		M, in drag, polka dotted dress	singing, dancing
	racy's	M, gr nd purple suit, pointy	
	Millstone Mall	F ay	singing
		lue pants	horn, umbrella
		dress	singing, dancing
		ointy nose	climbing into tiny car
Mr. Hobo	Tracy's		violin

UPDATE

使用 UPDATE即可直接编辑，而不用冒着删除不正确数据的风险（虽然会覆盖现有的数据）。

更新小丑的活动

这一次，让我们做对一切。为每次的目击报告填写UPDATE语句。我们先做一个范例来带领大家开始。然后再填写右边的 clown_info 表，让它看起来像是执行 UPDATE 语句后的样子。

Zippo spotted singing

```
UPDATE clown_info
SET activities = 'singing'
WHERE name = 'Zippo';
```

Snuggles now wearing baggy blue pants

Bonzo sighted at Dickson Park

Sniffles seen climbing into tiny car

Mr. Hobo last seen at party for Eric Gray

name	last_seen	appearance	activities
Elsie	Cherry Hill Senior Center	F, red hair, green dress, huge feet	balloons, little car
Pickles	Jack Green's party	M, orange hair, blue suit, huge feet	mime
Snuggles	Ball-Mart	F, yellow shirt, baggy red pants	horn, umbrella
Mr. Hobo	BG Circus	M, cigar, black hair, tiny hat	violin
Clarabelle	Belmont Senior Center	F, pink hair, huge flower, blue dress	yelling, dancing
Scooter	Oakland Hospital	M, blue hair, red suit, huge nose	balloons
Zippo	Millstone Mall	F, orange suit, baggy pants	dancing
Babe	Earl' s Autos	F, all pink and sparkly	balancing, little car
Bonzo		M, in drag, polka dotted dress	singing, dancing
Sniffles	Tracy's	M, green and purple suit, pointy nose	

name	last_seen	appearance	activities
Elsie	Cherry Hill Senior Center	F, red hair, green dress, huge feet	balloons, little car
Pickles	Jack Green's party	M, orange hair, blue suit, huge feet	mime
Snuggles			
Mr. Hobo			
Clarabelle	Belmont Senior Center	F, pink hair, huge flower, blue dress	yelling, dancing
Scooter	Oakland Hospital	M, blue hair, red suit, huge nose	balloons
Zippo			
Babe	Earl's Autos	F, all pink and sparkly	balancing, little car
Bonzo			
Snibbles			

更新小丑的活动

这一次，让我们做对一切。为每次的目击报告填写UPDATE语句。我们先做一个范例来带领大家开始。然后再填写右边的clown_info表，让它看起来像是执行 UPDATE 语句后的样子。

Zippo spotted singing

```
UPDATE clown_info
SET activities = 'singing'
WHERE name = 'Zippo';
```

我们并不想丢弃其他已经在appearance列中的信息。请确定所有信息都在这里。

Snuggles now wearing baggy blue pants

```
UPDATE clown_info
SET appearance = 'F, yellow shirt, baggy blue pants'
WHERE name = 'Snuggles';
```

Bonzo sighted at Dickson Park

```
UPDATE clown_info
SET last_seen = 'Dickson Park'
WHERE name = 'Bonzo';
```

Sniffles seen climbing into tiny car

```
UPDATE clown_info
SET activities = 'climbing into tiny car'
WHERE name = 'Snibbles';
```

Mr. Hobo last seen at party for Eric Gray

```
UPDATE clown_info
SET last_seen = 'Eric Gray\'s Party'
WHERE name = 'Mr. Hobo';
```

name	last_seen	appearance	activities
Elsie	Cherry Hill Senior Center	F, red hair, green dress, huge feet	balloons, little car
Pickles	Jack Green's party	M, orange hair, blue suit, huge feet	mime
Snuggles	Ball-Mart	F, yellow shirt, baggy red pants	horn, umbrella
Mr. Hobo	BG Circus	M, cigar, black hair, tiny hat	violin
Clarabelle	Belmont Senior Center	F, pink hair, huge flower, blue dress	yelling, dancing
Scooter	Oakland Hospital	M, blue hair, red suit, huge nose	balloons
Zippo	Millstone Mall	F, orange suit, baggy pants	dancing
Babe	Earl' s Autos	F, all pink and sparkly	balancing, little car
Bonzo		M, in drag, polka dotted dress	singing, dancing
Sniffles	Tracy's	M, green and purple suit, pointy nose	

灰色的记录没有改变，因为我们没有进行更新。

name	last_seen	appearance	activities
Elsie	Cherry Hill Senior Center	F, red hair, green dress, huge feet	balloons, little car
Pickles	Jack Green's party	M, orange hair, blue suit, huge feet	mime
Snuggles	Ball-Mart	F, yellow shirt, baggy blue pants	horn, umbrella
Mr. Hobo	Eric Gray's Party	M, cigar, black hair, tiny hat	violin
Clarabelle	Belmont Senior Center	F, pink hair, huge flower, blue dress	yelling, dancing
Scooter	Oakland Hospital	M, blue hair, red suit, huge nose	balloons
Zippo	Millstone Mall	F, orange suit, baggy pants	singing
Babe	Earl's Autos	F, all pink and sparkly	balancing, little car
Bonzo	Dickson Park	M, in drag, polka dotted dress	singing, dancing
Snibbles	Tracy's	M, green and purple suit, pointy nose	climbing into tiny car

只有这些在UPDATE语句中使用SET设定的记录部分才会改变。至此，我们终于填好了第121页上留下的空白。

UPDATE定价

还记得稍早曾试着修改一些drink_info表中的饮料定价吗？要把原本定价 2.5 美元的饮料改为 3.5 美元，原本定价 3.5 美元的饮料的改为 4.5美元。

drink_info

drink_name	cost	carbs	color	ice	calories
Blackthorn	3	8.4	yellow	Y	33
Blue Moon	2.5	3.2	blue	Y	12
Oh My Gosh	3.5	8.6	orange	Y	35
Lime Fizz	2.5	5.4	green	Y	24
Kiss on the Lips	5.5	42.5	purple	Y	171
Hot Gold	3.2	32.1	orange	N	135
Lone Tree	3.6	4.2	red	Y	17
Greyhound	4	14	yellow	Y	50
Indian Summer	2.8	7.2	brown	N	30
Bull Frog	2.6	21.5	tan	Y	80
Soda and It	3.8	4.7	red	N	19

让我们看看 UPDATE 语句会如何逐行解决这个问题，并写出一系列如下所示的 UPDATE 语句：

```
UPDATE drink_info
SET cost = 3.5
WHERE drink_name = 'Blue Moon';
```

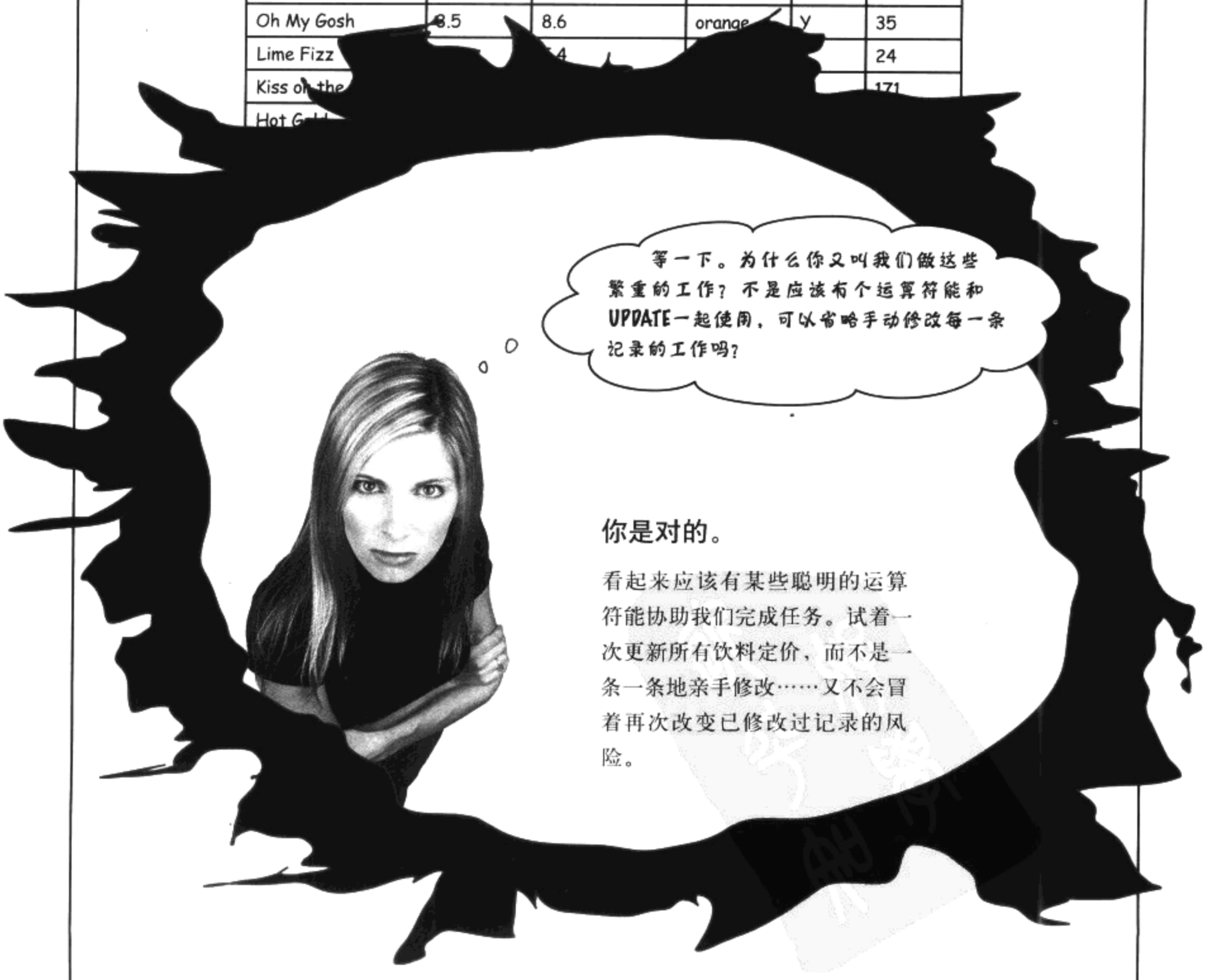
定价加1美元。

我们使用WHERE送出特定列，才知道该更新哪一条记录。

磨笔上阵

为drinks_info表中的每一条记录设计UPDATE语句，把每种饮料的定价都调高 1 美元。

drink_name	cost	carbs	color	ice	calories
Blackthorn	3	8.4	yellow	Y	33
Blue Moon	2.5	3.2	blue	Y	12
Oh My Gosh	8.5	8.6	orange	Y	35
Lime Fizz		.4			24
Kiss on the					171
Hot Gold					



你是对的。

看起来应该有某些聪明的运算符能协助我们完成任务。试着一次更新所有饮料定价，而不是一条一条地亲手修改……又不会冒着再次改变已修改过记录的风险。

只需要一次 UPDATE

我们的cost列存储数字。在SQL中，可以对数字列套用基础的数学运算。以cost列为例，只要+1就能更新表中所有需要更新的列。以下是运用方式：

```
UPDATE drink_info
SET cost = cost + 1;
WHERE
drink_name='Blue Moon'
OR
drink_name='Oh My Gosh'
OR
drink_name= 'Lime Fizz';
```

同时更新需要修改的记录（原价为2.5和3.5美元的饮料）。

没有蠢问题

问： 可以对数值套用减法吗？还有其他数学运算可以套用吗？

答： 乘法、除法、减法……都可以使用。而且运算时也可以采用其他数值，不只是1。

问： 能否告诉我什么情况下会想用乘法呢？

答： 没问题。假设表中列出许多物品的清单且附有它们的价格。利用UPDATE语句可以同时为每个物品的价格乘上固定数字来计算税后价格。

问： 除了简单的数学运算，还有其他运算能套用在数据上吗？

答： 确实还有。稍后，除了其他可对数值套用的操作，我们还会讨论可以套用在文本变量上的操作。

问： 真的吗？示范一下吧！

答： 好吧，例如，UPPER()函数能把表中的文本列改为大写，而各位可能也猜到了，LOWER()函数能把一切文本都改为小写。

UPDATE语句能运用在表的多条记录上。它可以和基础数学运算符一起使用，可以操作数值数据。

数据会改变，所以知道如何改变数据才会如此重要。

但表设计得越好，整体所需的更新操作就越少。良好的表设计能让我们从专心于表的内容中解放出来。

有兴趣吗？下一章，我们会讲到表的设计，完全没有痛苦，很简单，简单得有点……

你的SQL工具包

第3章很快就会成为记忆的一部分。但我们还是很快地回顾一下刚学到的SQL语句。如果需要本书工具的完整列表，请参考附录3。

DELETE

这是删除表中记录的工具。它和WHERE子句一起使用，可精确地瞄准你想删除的行。

UPDATE

这条语句以新值更新现有的一列或多列，它也可以使用WHERE子句。

SET

这个关键字属于UPDATE语句，可用于改变现有列的值。

4 聪明的表设计

为什么要规范化？

你已经创建了一些表，但都没有经过仔细考虑。没有关系，这些表都可以用。你可以从中SELECT、INSERT、DELETE、UPDATE列，但随着**取得的数据越来越多**，你一定希望以前能多考虑一点，好让现在的 WHERE 子句简单一点。我们需要让表更正常、更*规范*。

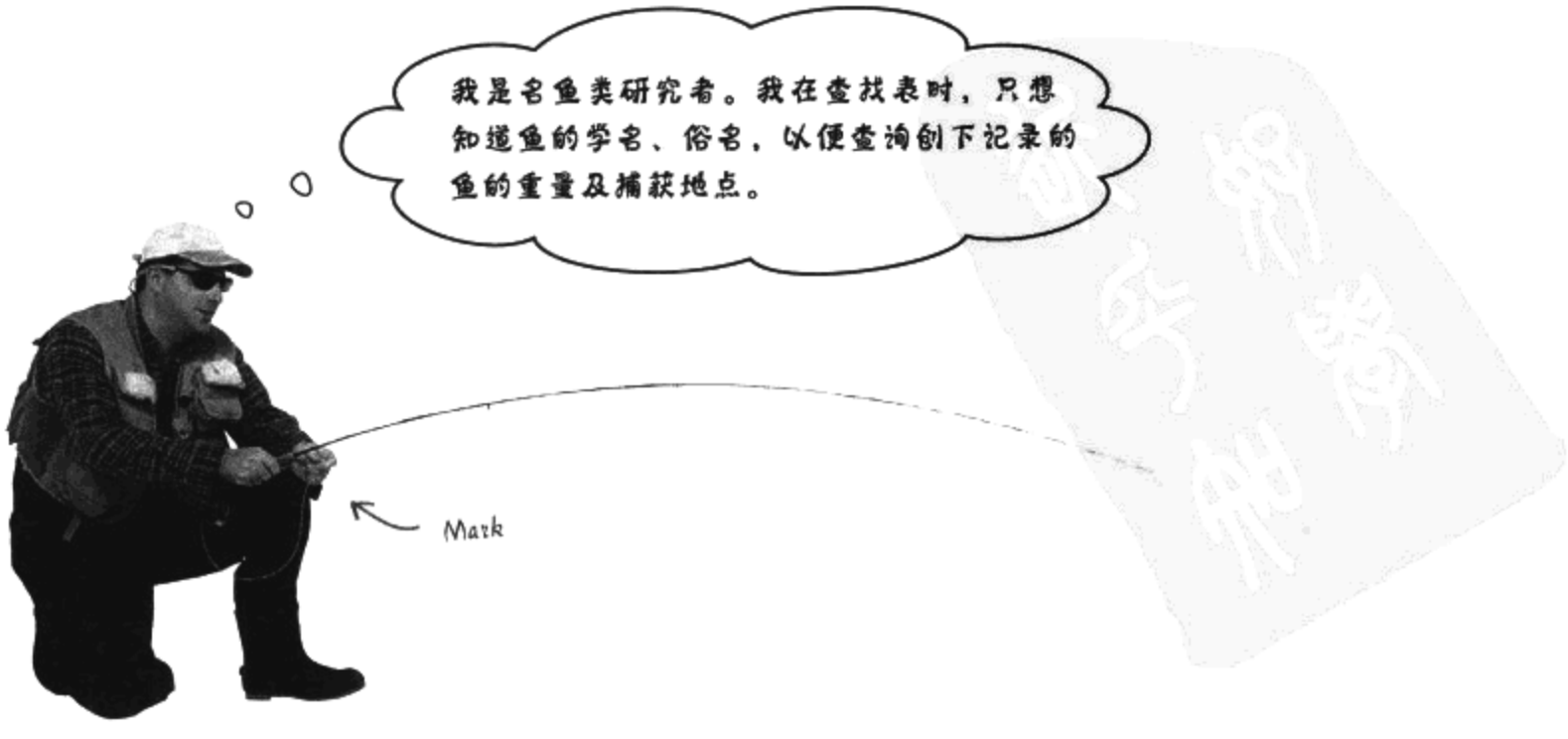
两张鱼的表

Jack和Mark各自创建了一张表，用来存储创下记录的鱼的信息。Mark的表中的列有鱼的学名、鱼的俗名、重量以及捕获地点。此表缺少捕获者的信息。

fish_info

这张表只有4列。请将此表和另一页的 fish_record 表进行比较。

common	species	location	weight
bass, largemouth	M. salmoides	Montgomery Lake, GA	22 lb 4 oz
walleye	S. vitreus	Old Hickory Lake, TN	25 lb 0 oz
trout, cutthroat	O. Clarki	Pyramid Lake, NV	41 lb 0 oz
perch, yellow	P. Flavescens	Bordentown, NJ	4 lb 3 oz
bluegill	L. Macrochirus	Ketona Lake, AL	4 lb 12 oz
gar, longnose	L. Osseus	Trinity River, TX	50 lb 5 oz
crappie, white	P. annularis	Enid Dam, MS	5 lb 3 oz
pickerel, grass	E. americanus	Dewart Lake, IN	1 lb 0 oz
goldfish	C. auratus	Lake Hodges, CA	6 lb 10 oz
salmon, chinook	O. Tshawytscha	Kenai River, AK	97 lb 4 oz



Jack 的表也有鱼的俗名、重量，但还包括了捕获者的姓名，而且他的地点列也被分成两列，分别记录捕获鱼的水域和州名。

这张表也是关于创下记录的鱼，但列的数量几乎是前页上的表格的两倍。

fish _ records

first_name	last_name	common	location	state	weight	date
George	Perry	bass, largemouth	Montgomery Lake	GA	22 lb 4 oz	6/2/1932
Mabry	Harper	walleye	Old Hickory Lake	TN	25 lb 0 oz	8/2/1960
John	Skimmerhorn	trout, cutthroat	Pyramid Lake	NV	41 lb 0 oz	12/1/1925
C.C.	Abbot	perch, yellow	Bordentown	NJ	4 lb 3 oz	5/1/1865
T.S.	Hudson	bluegill	Ketona Lake	AL	4 lb 12 oz	4/9/1950
Townsend	Miller	gar, longnose	Trinity River	TX	50 lb 5 oz	7/30/1954
Fred	Bright	crappie, white	Enid Dam	MS	5 lb 3 oz	7/31/1957
Mike	Berg	pickerel, grass	Dewart Lake	IN	1 lb 0 oz	6/9/1990
Florentino	Abena	goldfish	Lake Hodges	CA	6 lb 10 oz	4/17/1996
Les	Anderson	salmon, chinook	Kenai River	AK	97 lb 4 oz	5/17/1985

磨笔上阵

为上述两张表设计查询，找出在 New Jersey 创下记录的鱼。

.....

.....

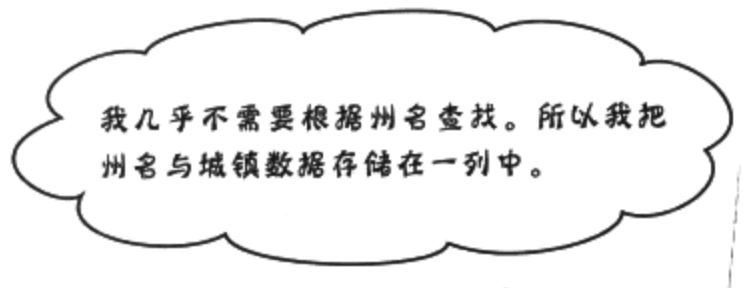
.....

.....

我是Reel and Creel杂志的撰稿人。我需要知道钓鱼者的姓名、钓到鱼的时间以及水域。

磨笔上阵解答

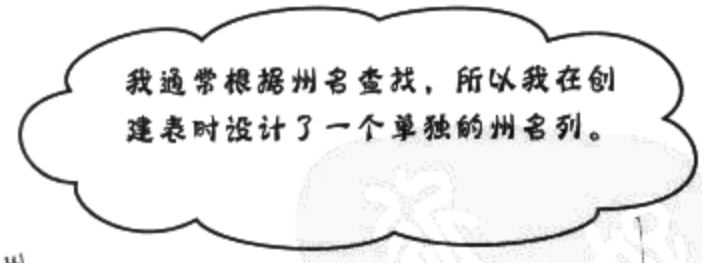
为上述两张表设计查询，找出在New Jersey创下记录的鱼。



我们必须使用LIKE来从结合城镇和州名的列中取得查询结果。

```
SELECT * FROM fish_info
WHERE location LIKE '%NJ';
```

common	species	location	weight
perch, yellow	P. Flavescens	Bordentown, NJ	4 lb 3 oz



这组查询可以直接查找州名列。

```
SELECT * FROM fish_records
WHERE state = 'NJ';
```

first_name	last_name	common	location	state	weight	date
C.C.	Abbot	perch, yellow	Bordentown	NJ	4 lb 3 oz	5/1/1865

问： Jack 的表是不是比Mark的好呢?

答： 不一定。他们的表能满足不同的需求。Mark几乎不会直接查找州名，他在乎的数据是创下记录的鱼的学名和俗名以及它们的重量。

另一方面，Jack在查询数据时经常要搜索州名。所以他的表会把州名独立成一列，这样才便于根据州名查询数据。

问： 查询表时是否应该避免使用 LIKE? LIKE 有问题吗?

答： LIKE没有问题，但可能很难运用到你的查询中，而且你会冒着找出你不需要的数据的风险。如果你的列包含复杂信息的话，LIKE 搜索精确数据的能力还不够。

问： 为什么简短的查询优于较长的查询?

答： 查询越简单越好。随着数据的增长，还有对新表的添加，你的查询会变得越来越复杂。如果现在就练习设计最简单的查询，以后你会感谢现在的及早训练。

问： 所以说，我应该只在表里存放少量数据?

答：不完全是。就像 Mark 和 Jack 的表对比，存放的数据量取决于数据的使用方式。

假设有张表为车厂技师列出了车辆清单，另外一张车辆表则要给销售人员使用。技师可能需要每辆车的精确信息，但销售人员或许只需要车辆的制造商、型号和 VIN 编号而已。

问： 假设我们手边有一组街道地址。为什么不能用一列来存储完整地址，然后再用其他列存储分开的版本呢?

答： 现在看来重复存储数据好像是个聪明的想法，不过还应该考虑一下，在数据库增长到极大容量后，它会吃掉多少硬盘空间。而且，每次重复存储数据，就代表每次修改数据时都要记得在 UPDATE 语句里多加一个子句。

让我们进一步讨论，如何根据你的用途，以最可行的好方式设计表。

使用数据的方式将影响设置表的方式。

SQL 是一种用于关系数据库的语言。你认为，在 SQL 数据库的世界里，“关系”表示什么意思?

表都是关于关系的

SQL 其实是因关系数据库管理系统（Relational Database Management System，RDBMS）而出名。别为要记得这个术语而烦恼。我们所关心的词只是“关系”（RELATIONAL）*。对于设计表的人而言，就是要设计一个杀手级的表，我们必须考虑列彼此之间如何产生关系、如何一起描述某项事物。

挑战之处在于使用列描述事物，并且让取得数据更为方便。设计方向取决于我们对表的需求，但在创建表时，有些非常广泛的步骤可供遵循。

1. 挑出事物，挑出你希望表描述的**某样事物**。

什么是你希望表说明的主要事物呢？

2. 列出一份关于那样事物的**信息列表**，这些信息都是**使用表**时的必要信息。

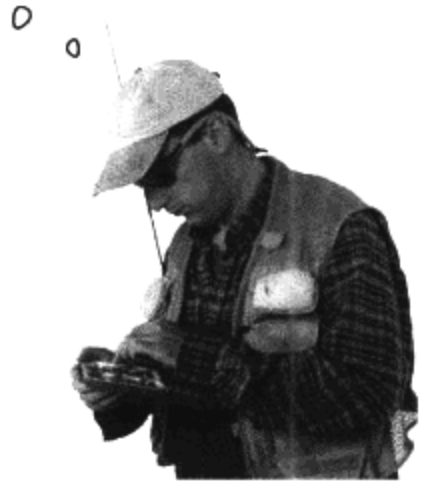
你将如何使用这张表？

3. 使用信息列表，把关于那样事物的综合信息**拆分成小块信息**，以便用于**组织表**。

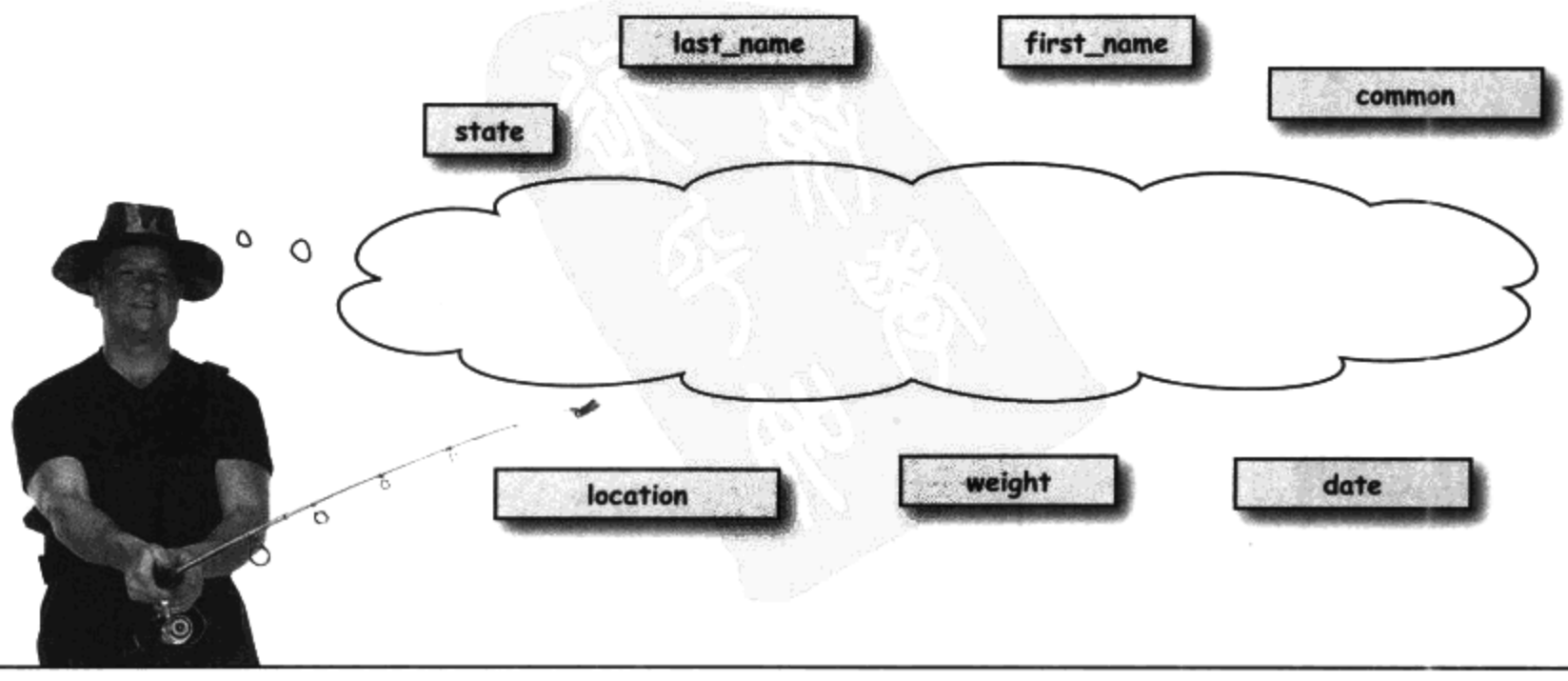
如何才能最轻松地查询这张表？

*有些人以为关系表示众多表彼此间互有关联。这并不正确。

你能从鱼类研究者 Mark 说明他对表的需求的句子中找出必要的列吗？

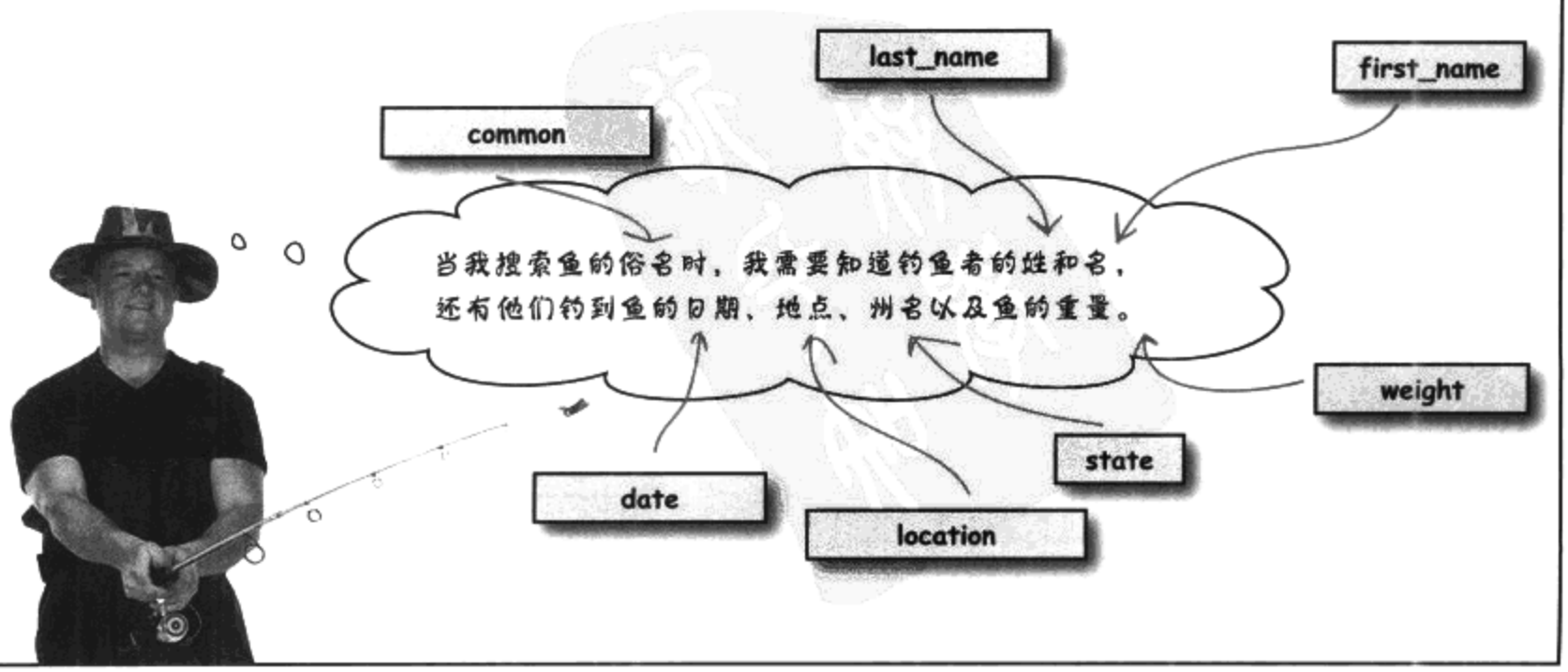


轮到你试试了。为 Reel and Creel 杂志的撰稿人 Jack 设计需求文句，他要从表中选出他的文章所需的详细信息。写下需求文句后，再把每列对应到文句中提到它们的地方。.



你能从鱼类研究者 Mark 说明他对表的需求的句子中找出必要的列吗？

轮到你试试了。为 Reel and Creel 杂志的撰稿人 Jack 设计需求文句，他要从表中选出他的文章所需的详细信息。写下需求文句后，再把每列对应到文句中提到它们的地方。



的确可以，但我们不需要存储这么细致的数据。

至少在这里的例子中还不需要。如果Jack撰写的文章内容是关于最佳渡假及钓鱼地点的，他或许会需要街道名称和门牌号码，以便读者寻找附近的住宿地点。

你觉得*原子性*在SQL数据里表示什么意思？

原子性数据

原子（atom）是什么？它们是一小块无法或不应该分割的信息。对数据而言也一样，当数据具有**原子性**（atomic），就表示它已经被分割至**最小块，已经不能或不应该再被分割**。

30分钟送到你家，否则免费

以比萨快递员为例，他需要知道送达的地址，此时就需要一列来存储街道名称和门牌号码。对他来说，这样就已经具有原子性了。快递员不需要单独查找门牌号码。

事实上，如果他的数据被拆分成街道名称和门牌号码两列，他的查询反而会更长、更复杂，比萨送达我们家的时间也就会变久。

以比萨快递员为例，用一列来存储完整的街道地址已经很具原子性了。

```
File Edit Window Help SimplePizzaFactory
+--------------+--------------------------+
| order_number | address                  |
+--------------+--------------------------+
| 246          | 59 N. Ajax Rapids        |
| 247          | 849 SQL Street           |
| 248          | 2348 E. PMP Plaza        |
| 249          | 1978 HTML Heights        |
| 250          | 24 S. Servlets Springs   |
| 251          | 807 Infinite Circle      |
| 252          | 32 Design Patterns Plaza |
| 253          | 9208 S. Java Ranch       |
| 254          | 4653 W. EJB Estate       |
| 255          | 8678 OOA&D Orchard       |
+--------------+--------------------------+
> SELECT address FROM pizza_deliveries WHERE order_num = 252;
+--------------------------+
| address                  |
+--------------------------+
| 32 Design Patterns Plaza |
+--------------------------+
1 row in set (0.04 sec)
```

地址、地址、地址

再看看房地产经纪人的情况。他们可能希望有独立的门牌号码列。或许他们有时候需要查询某个街道名称，看看那条街上所有待售的房屋的门牌号码。对房地产经纪人而言，街道名称和门牌号码都有原子性。

但对房地产经纪人而言，以不同列分开存储街道名称和门牌号码能让他轻松查询某条街道上的所有待售房屋。

```
File Edit Window Help IWantMyCommission
+---------------+-------------------------+---------------+---------+
| street_number | street_name             | property_type | price   |
+---------------+-------------------------+---------------+---------+
| 59            | N. Ajax Rapids          | condo         | 189000  |
| 849           | SQL Street              | apartment     | 109000  |
| 2348          | E. PMP Plaza            | house         | 355000  |
| 1978          | HTML Heights            | apartment     | 134000  |
| 24            | S. Servlets Springs     | house         | 355000  |
| 807           | Infinite Circle         | condo         | 143900  |
| 32            | Design Patterns Plaza   | house         | 465000  |
| 9208          | S. Java Ranch           | house         | 699000  |
| 4653          | SQL Street              | apartment     | 115000  |
| 8678          | OOA&D Orchard           | house         | 355000  |
+---------------+-------------------------+---------------+---------+
> SELECT price, property_type FROM real_estate WHERE street_name = 'SQL
Street' ;
+----------+---------------+
| price | property_type |
+----------+---------------+
| 109000.00 | apartment |
| 115000.00 | apartment |
+----------+---------------+
2 rows in set (0.01 sec)
```

原子性数据和你的表

下列问题有助于你理解表中需要的内容：

1. 你的表在描述什么事物？

是描述小丑、乳牛、甜甜圈或人？

2. 以何种方式使用表取得描述的事物呢？

设计表时要让查询容易一点！

3. 列是否包含原子性数据，可让查询既简短又直逼要害？

没有蠢问题

问： 原子不是很小吗？我不是应该把数据分割成非常、非常小的片段吗？

答： 不是哦。让数据具有原子性，表示把数据分割成创建有效率的表所需的最小片段。

别把数据切割得超出必要。如果不需要额外增加列，就别因为可以增加而增加。

问： 原子性对我有什么帮助？

答： 原子性有助于确保表内容的准确性。例如，你有一个门牌号码列，你可以确保有关门牌号码的数字只会出现在该列中。

原子性数据也能使查询更有效率，因为查询会因原子性而更容易设计，而且运行所需时间也更短，因此在面对大量数据时有加分效果。

磨笔上阵

下面列出了原子性数据的正式规则。请针对各项规则设计两张违反规则的表。

规则一：具有原子性数据的列中不会有多个类型相同的值。

Greg的my_contacts表就违反了这项规则。

规则二：具有原子性数据的表中不会有多个存储同类数据的列。

easy_drinks表违反了这项规则。

磨笔上阵 解答

下面列出了原子性数据的正式规则。请针对各项规则设计两张违反规则的表。

规则一：具有原子性数据的列中不会有多个类型相同的值。

各位的答案当然不会和我们相同，这里只是举例说明。

food_name	ingredients
bread	flour, milk, egg, yeast, oil
salad	lettuce, tomato, cucumber

还记得 Greg 的表吗？它有一列记录个人兴趣，但每个人常常有多种兴趣会让寻找这一列成为一场恶梦！

这张表也有相同的问题：请设想在成分列（ingredients）中该如何查找番茄。

规则二：具有原子性数据的表中不会有多个存储同类数据的列。

记录学生的列太多了！

teacher	student1	student2	student3
Ms. Martini	Joe	Ron	Kelly
Mr. Howard	Sanjaya	Tim	Julie

我们现在知道了原子性的正式规则以及让数据保持原子性的三个步骤，请回头查看以前创建的表并解释它是否具有原子性，如果没有原子性，为什么？

Greg 的表，第 47 页

甜甜圈的评分表，第 78 页

小丑目击表，第 121 页

饮料表，第 59 页

鱼的信息表，第 160 页

规范化的原因

当数据顾问去休假了，而你需要雇用更多SQL数据库设计师时，如果可以不需要浪费解释表的用途的时间，那该有多好啊！

让表规范化（normalization）表示表遵循某些标准规则，即使是刚接触它们的新设计师也能理解。好消息：刚才创建的具有原子性的表已经在规范化的半路上。

让数据具有原子性是创建一个规范化表的第一步。

我们现在知道了原子性的正式规则，以及让数据保持原子性的三个步骤，请回头查看以前创建的表并解释它是否具有原子性，如果没有原子性，为什么？

习题解答

Greg 的表，第 47 页

不具有原子性。*interests* 和 *seeking* 列都违反了第一项规则。

甜甜圈的评分表，第 78 页

具有原子性。每列都具有不同类型的信息，不像easy_drink表。另外，每列只存储了一块原子性信息，不像clown表的activities列。

小丑目击表，第 121 页

不具有原子性。某些记录的activities列中存储了多项活动，因此违反了第一项规则。

饮料表，第 59 页

不具有原子性。它有多个成分列，违反了第二项规则。

鱼的信息表，第 160 页

具有原子性。每列都有不同类型的信息，而且每列都只有一块原子性信息。

规范化表的优点

1. 规范化表中没有重复的数据，可以减小数据库的大小。

2. 因为查找的数据较少，你的查询会更为快速。

因为，即使你的表很小，但它还会增长。

而且表的确会变大。如果一开始就设计规范化表，那么在查询变得太慢时也不需要回头改变表。

小丑不太标准

还记得小丑的表吗？追踪小丑最近成为全国性的盛大活动，我们的旧表将不能再使用，因为appearance和activities列包含了非常多的数据。就我们的用途而言，这个表不具有原子性。

这两列非常难以查询，因为它们包含非常多的数据！

clown _ info

name	last_seen	appearance	activities
Elsie	Cherry Hill Senior Center	F, red hair, green dress, huge feet	balloons, little car
Pickles	Jack Green's party	M, orange hair, blue suit, huge feet	mime
Snuggles	Ball-Mart	F, yellow shirt, baggy blue pants	horn, umbrella
Mr. Hobo	Eric Gray's Party	M, cigar, black hair, tiny hat	violin
Clarabelle	Belmont Senior Center	F, pink hair, huge flower, blue dress	yelling, dancing
Scooter	Oakland Hospital	M, blue hair, red suit, huge nose	balloons
Zippo	Millstone Mall	F, orange suit, baggy pants	singing
Babe	Earl's Autos	F, all pink and sparkly	balancing, little car
Bonzo	Dickson Park	M, in drag, polka dotted dress	singing, dancing
Sniffles	Tracy's	M, green and purple suit, pointy nose	climbing into tiny car

磨笔上阵

试着让小丑的表更具有原子性。假设我们要从appearance、activities及last_seen列中查找数据，请试着为列设计更好的选择。

答案请见第195页。

达成1NF的半路上

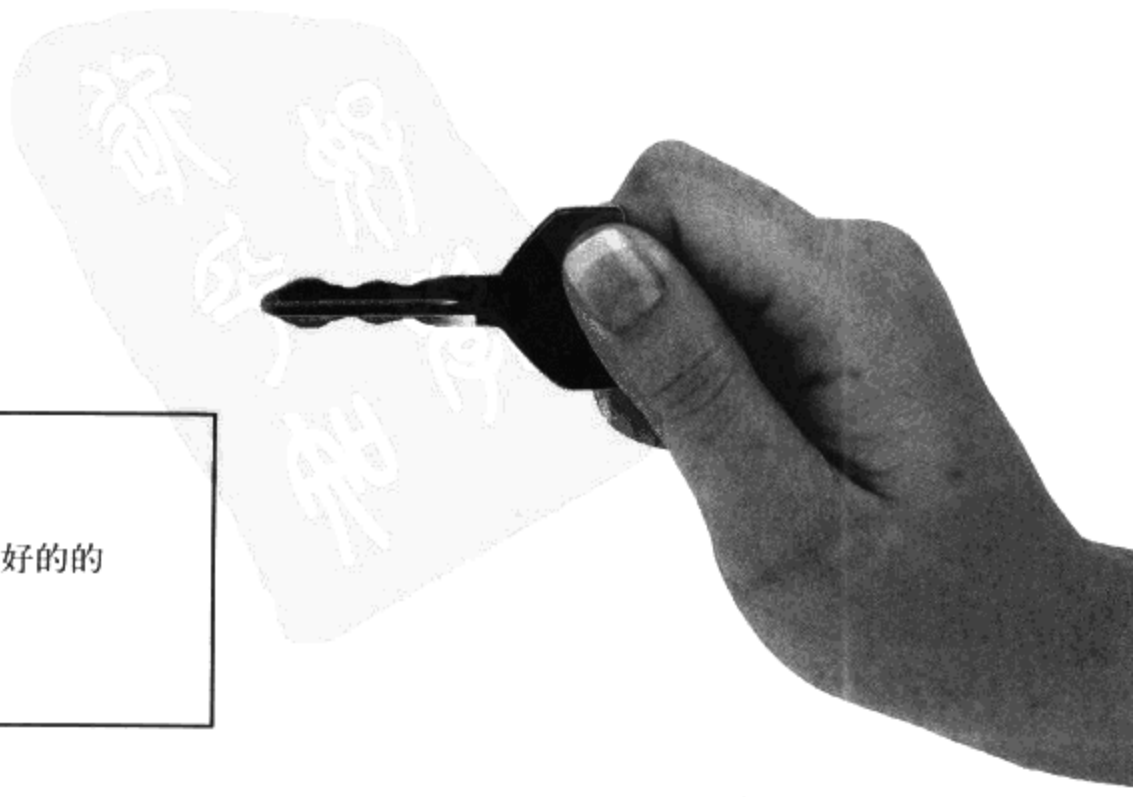
请记住，具有原子性的数据只让我们的表规范了一半。完全的规范化表示我们处于第一范式（First Normal Form）的状态，简称 1NF。

我们已经知道如何做这件事。

每个数据行必须包含具有原子性的值。

要让表完全地规范，需要为每条记录加上主键。

每个数据行必须有独一无二的识别项，人称主键（Primary Key）。



动动脑

你觉得什么类型的列能够成为好的的主键呢？

主键规则

即将成为主键的列必须在创建表时一并设置。再过几页，我们就会练习创建表并设计主键。但在实际动手前，我们先仔细了解一下主键是什么。

主键是表中的某个列，它可以让每一条记录成为唯一的。

主键用于独一无二地识别出每条记录。

这是说，主键列中的数据不能重复。假设有张表的结构如下，你认为哪一列会是优秀的主键？

SSN	last_name	first_name	phone_number

因为每个人的社会安全号都不相同，或许它可作为主键。

这三列都可能包含重复的数据值，例如，表中可能有很多人的名字都叫John，或有很多人住在一起、共用一部电话，所以这三列不能作为主键好选择。

注意！

使用社会安全号作为主键时千万要小心。

盗用身份的人只会增加不会减少，大家都不太愿意透露社会安全号，而且他们的理由绝对充分。社会安全号实在太宝贵了，不适合拿来冒险。你可以保证数据库绝对安全吗？如果不行，数据库中的所有社会安全号都可能被偷走，你的客户资料也会跟着一起曝光。

主键不可以为 NULL

如果主键是 NULL，它就不可能唯一，因为其他记录也可能包含 NULL！

插入新记录时必须指定主键值

插入新记录时，如果没有主键值，你就会冒着主键值是NULL的风险，而且可能会让表存储重复的记录，这违反了 1NF 的原则。

主键必须简洁

主键应该只包含需要的独一无二的数据，不该有其他内容。

主键值不可以被修改

如果可以改变主键值，那你就会冒着意外输入已使用值的风险。请记住，主键必须保持唯一性。

根据这些规则，你可以想出适合表的主键了吗？

回头看看本书所用的表，其中有哪些列包含了真正独一无二的值？

最佳主键可能是新的主键。

讲到创建主键时，最佳方式或许是另外创建一个包含唯一性主键的列。以存储个人信息但另有一个编号列的表为例，在下面的表格中，我们称之为 ID列 。

如果没有ID列，两条 John Brown 的记录似乎完全一样。但在本例中，他们应该是两个不同的人。ID 列让他们的记录具有唯一性。这张表也因此而处于第一范式的状态。

id	last_name	first_name	nick_name
1	Brown	John	John
2	Ellsworth	Kim	Kim
3	Brown	John	John
4	Petrillo	Maria	Maria
5	Franken	Esme	Em

一条关于John Brown的记录。

也是关于John Brown的记录，但ID列显示这是独一无二的记录，所以他是另一位John Brown。

技客新知

在 SQL 的世界里，关于该使用虚构、**人造**（synthetic）主键（如上例的 ID 列），还是该使用**自然**（natural）主键——表中现有的数据（例如车牌号或社会安全号），目前还有争议。我们不会站在某一边，但我们会在第7章进一步讨论主键。

问： 目前只说到“第一”范式，也就是说，以后还有第二范式、第三范式吗？

答： 是的，的确有第二范式和第三范式，每多一级规范化，就会增加一组更严格、更精确的规则。第7章会讨论第二范式和第三范式。

问： 本章的表已经被我们修改为具有原子性，它们已经处于 1NF 状态了吗？

答： 还没有。到目前为止，我们创建的表都没有主键，没有一个独一无二的值。

问： 我认为甜甜圈表的评语列（comments）一点都不具有原子性。我是说，实在没有什么合理的方式能够轻松查询这一列。

答： 完全正确！该字段的确不具原子性，但根据我们的表设计，它不需要原子化。如果我们想把评语限定在一组预定词汇中，该列就可被原子化。但这样它就无法收集真实、自然的意见了。

朝规范化前进

现在要回头把我们的表规范化。我们需要让数据具有原子性并且加上主键。主键的创建通常会在编写 CREATE TABLE 代码时进行。

大家还记得如何在现有表里添加新列吗？

修理Greg的表

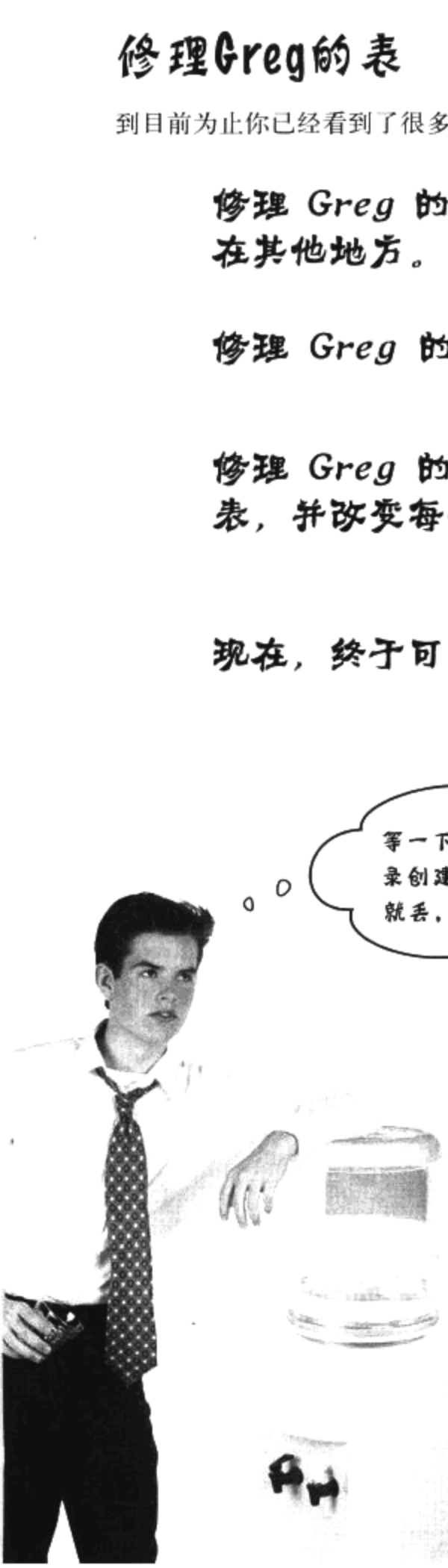
到目前为止你已经看到了很多技巧，以下就是修理 Greg 的表的方式：

修理 Greg 的表步骤 1：SELECT 所有数据并存储在其他地方。

修理 Greg 的表步骤 2：创建一个新的规范表。

修理 Greg 的表步骤 3：把所有旧数据 INSERT 到新表，并改变每一行以符合新表的结构。

现在，终于可以丢掉旧表了。



我们知道Greg的表并不完美。

它不具有原子性，也没有主键。Greg兄，幸好你**不用**忍受旧表，而且也**不需要丢弃现有的数据**。

我们会介绍一个新的命令，它可为Greg的表加上主键并使列更具有原子性。但先让我们回到过去……

我们设计的CREATE TABLE

Greg 需要一个主键，而且在讨论过原子性后，他也发现自己的表可以修理得更具有原子性。在了解修理现有表的方式前，让我们先试着重新设计更标准的表吧！

以下是我们在第1章创建的表。

```
CREATE TABLE my_contacts
(
    last_name VARCHAR(30),
    first_name VARCHAR(20),
    email VARCHAR(50),
    gender CHAR(1),
    birthday DATE,
    profession VARCHAR(50),
    location VARCHAR(50),
    status VARCHAR(20),
    interests VARCHAR(100),
    seeking VARCHAR(100)
);
```

这里没有主键

这些列能否设计得更具有原子性呢？

动动脑

假设当初没有把 CREATE TABLE 写下来怎么办？你知道该如何取得当初设计的 SQL 代码吗？

给我有内容的表

如果使用DESCRIBE my_contacts命令，能否看到设置表时所用的SQL代码？事实上，我们得到的结果如下所示：

```
File Edit Window Help GregsListAgain
+------------+--------------+------+-----+---------+----------------+
| Column     | Type         | Null | Key | Default | Extra          |
+------------+--------------+------+-----+---------+----------------+
| last_name  | varchar(30)  | YES  |     | NULL    |                |
| first_name | varchar(20)  | YES  |     | NULL    |                |
| email      | varchar(50)  | YES  |     | NULL    |                |
| gender     | char(1)      | YES  |     | NULL    |                |
| birthday   | date         | YES  |     | NULL    |                |
| profession | varchar(50)  | YES  |     | NULL    |                |
| location   | varchar(50)  | YES  |     | NULL    |                |
| status     | varchar(20)  | YES  |     | NULL    |                |
| interests  | varchar(100) | YES  |     | NULL    |                |
| seeking    | varchar(100) | YES  |     | NULL    |                |
+------------+--------------+------+-----+---------+----------------+
```

但我们想看的是**CREATE**代码，而不是表中的字段，这样才能知道一开始该如何设计，而不是事后努力地重新设计CREATE语句。

SHOW CREATE_TABLE语句将返回可以重建表但没有数据的CREATE TABLE语句。这样一来，你随时都能查看表的可能创建方式了。试试看：

```
SHOW CREATE TABLE my_contacts;
```

节省时间的命令

查看第183页上我们用于创建表的程序代码，还有SHOW CREATE_TABLE my_contacts 语句提供的下列 SQL 代码。两者并非完全相同，但如果把下面这段代码粘贴到CREATE TABLE命令中，最后的结果会是一样的。反撇号或数据设置不需要删除，但如果删除的话，看起来会更干净。

列名和表名前后的反撇号会在我们运行SHOW CREATE TABLE命令时出现。

除非我们另行通知SQL程序，否则它都会假设所有数据的默认值是 NULL。

最好在创建表时指定列是否可包含 NULL。

```
CREATE TABLE `my_contacts`
(
    `last_name` varchar(30) default NULL,
    `first_Name` varchar(20) default NULL,
    `email` varchar(50) default NULL,
    `gender` char(1) default NULL,
    `birthday` date default NULL,
    `profession` varchar(50) default NULL,
    `location` varchar(50) default NULL,
    `status` varchar(20) default NULL,
    `interests` varchar(100) default NULL,
    `seeking` varchar(100) default NULL,
) ENGINE=MyISAM DEFAULT CHARSET=latin1
```

你不需要担心结束括号后的文字，它说明数据如何存储以及使用的字符集。现在用默认值就够了。

虽然我们能清理 SQL 代码（删除最后一行文字和反撇号），但不能只靠复制和粘贴来创建表。

除非已经删除了原始表，否则你都要给表一个新的名称。

加上主键的CREATE TABLE

以下是SHOW CREATE _TABLE my_contacts提供的程序代码。我们删除了反撇号和最后一行。我们在列列表的最上方添加了 contact_id列并设定为 NOT NULL，在列列表的最下方则添加了一行PRIMARY KEY，把新的 contact_id 列设定为主键。

记住，主键值不可以是NULL，所以必须设为 NOT NULL！如果主键包含NULL值或根本没有值，就无法保证主键可以只识别出表中的某一行。

```
CREATE TABLE my_contacts
(
  contact_id INT NOT NULL,
  last_name varchar(30) default NULL,
  first_name varchar(20) default NULL,
  email varchar(50) default NULL,
  gender char(1) default NULL,
  birthday date default NULL,
  profession varchar(50) default NULL,
  location varchar(50) default NULL,
  status varchar(20) default NULL,
  interests varchar(100) default NULL,
  seeking varchar(100) default NULL,
  PRIMARY KEY (contact_id)
)
```

我们创建了名为 *contact_id* 的新列，它存储整数值来作为表的主键。这一列的每一个值都必须独一无二，而且能让我们的表具有原子性。

这里就是我们指定主键的地方。语法非常简单：只要说 PRIMARY KEY，并把主键列的名称放在括号中——本例是以 *contact_id* 列为主键。

没有蠢问题

问： 你说主键不能是 NULL。还有其他防止主键重复的机制吗？

答： 基本上是有的。每当INSERT 值至表中时，我们都要给 contact _ id 列插入新值。例如第一组 INSERT 语句把 contact _ id 设定为 1，第二组则把 contact _ id 设定为 2……依此类推。

问： 这样很伤脑筋吧？每次插入新记录时都要给主键列赋新值。没有更简单的方法吗？

答： 有两种方法。一种是直接利用原本就有唯一性的数据作为主键。稍早提过，这一点可能不太容易实现（例如想以社会安全号作为主键时的顾虑）。

比较简单的方法是创建一个全新的主键列，用它来存储独一无二的值，例如 contact _ id。每当使用主键时，我们可以要求 SQL 软件自动为主键填入新的值。下一页就会详细说明。

问： 除了CREATE 命令，SHOW还能用在其他命令上吗？

答： SHOW 也可以显示表中的某一列：

```
SHOW COLUMNS FROM tablename;
```

上述命令将显示表的所有列及其数据类型，还有其他关于各列的细节信息。

```
SHOW CREATE DATABASE databasename;
```

就像SHOW CREATE TABLE，上述命令将提供重建表所需的语句。

```
SHOW INDEX FROM tablename;
```

前述命令会显示任何编了索引的列以及索引类型。到目前为止，我们唯一看到的索引就是主键，但这个命令会随着你对 SQL 越来越熟悉而变得更有用。

还有一个非常有用的命令：

```
SHOW WARNINGS;
```

如果你从控制台收到 SQL 命令造成的错误信息，键入这个命令就可取得确切的警告内容。

还有一些相关命令，但我们只列出现在会用到的几个命令。

问： 所以，SHOW CREATE TABLE 中的反撇号究竟是做什么用的？我真的不需要用到反撇号吗？

答： 反撇号的存在是因为 RDBMS 有时无法分辨出列名。如果在列名前后加上反撇号就能以 SQL 保留字作为列名（尽管这不是个好主意）。

例如，由于某些奇怪的原因，你想把某列命名为 select。下列命令无法实现：

```
select varchar(50)
```

但这个命令可以：

```
`select` varchar(50)
```

问： 用保留字作为列名会有什么大问题啊？

答： 虽然你可以这么做，但这真的是很糟的主意。想象一下你的查询看起来会有多么混乱，还有每次都要键入反撇号的麻烦，还不如一开始就别把列取名为某个保留字。除此之外，select 也不是一个好列名，它无法说明该列包含的数据。

1、2、3……自动递增

为contact_id列加上关键字AUTO_INCREMENT，可以让SQL软件自动为该列填入数字，第一行填入1，后面的依序递增。

```
CREATE TABLE my_contacts
(
    contact_id INT NOT NULL AUTO_INCREMENT,
    last_name varchar(30) default NULL,
    first_name varchar(20) default NULL,
    email varchar(50) default NULL,
    gender char(1) default NULL,
    birthday date default NULL,
    profession varchar(50) default NULL,
    location varchar(50) default NULL,
    status varchar(20) default NULL,
    interests varchar(100) default NULL,
    seeking varchar(100) default NULL,
    PRIMARY KEY (contact_id)
)
```

就是它。加入关键字AUTO_INCREMENT的方式和使用其他SQL风味关键字的方式一样。（我要提醒MS SQL的用户，你们需要的关键字是IDNEX，还要记得加上起始值和递增值。详细信息请参考MS SQL的说明手册。）

这个关键字的作用就像字面上看到的意义一样：从1开始，每次插入新记录时逐次递增1。

好吧，这看起来很简单。不过，既然它可自动填入列值，我该如何设计INSERT呢？我会不会意外覆盖了自动增加的值？

你觉得会生什么事呢？

坐而言不如起而行，动手尝试一下，看看系统有什么反应吧！

1 设计一段存储姓和名的 CREATE TABLE 语句。你的表应该有个会自动递增（AUTO_INCREMENT）的主键列，另外还有两个具有原子性数据的列。

2 打开你的 SQL 控制台或 GUI 并运行刚刚设计的 CREATE TABLE 语句。

3 请尝试下列各条 INSERT 语句。圈出可以运作的语句。

```
INSERT INTO your_table (id, first_name, last_name)
VALUES (NULL, 'Marcia', 'Brady');

INSERT INTO your_table (id, first_name, last_name)
VALUES (1, 'Jan', 'Brady');

INSERT INTO your_table
VALUES (2, 'Bobby', 'Brady');

INSERT INTO your_table (first_name, last_name)
VALUES ('Cindy', 'Brady');

INSERT INTO your_table (id, first_name, last_name)
VALUES (99, 'Peter', 'Brady');
```

4 每位 Brady 都会出现在表中吗？利用下面的表描绘出运行 INSERT 语句后的数据内容。

your_table

id	*first_name*	*last_name*

1 设计一段存储姓和名的 CREATE TABLE 语句。你的表应该有个会自动递增（AUTO_INCREMENT）的主键列，另外还有两个具有原子性数据的列。

```
CREATE TABLE your_table
(
id INT NOT NULL AUTO_INCREMENT,
first_name VARCHAR(20),
last_name VARCHAR(30),
PRIMARY KEY (id)
);
```

2 打开你的 SQL 控制台或 GUI 并运行刚刚设计的 CREATE TABLE 语句。

3 请下列各条 INSERT 语句。圈出可以运作的语句。

```
INSERT INTO your_table (id, first_name, last_name)
VALUES (NULL, 'Marcia', 'Brady');

INSERT INTO your_table (id, first_name, last_name)
VALUES (1, 'Jan', 'Brady');

INSERT INTO your_table
VALUES (2, 'Bobby', 'Brady');

INSERT INTO your_table (first_name, last_name)
VALUES ('Cindy', 'Brady');

INSERT INTO your_table (id, first_name, last_name)
VALUES (99, 'Peter', 'Brady');
```

最后一条语句"可以运作"，但它会覆盖 AUTO_INCREMENT 列的值。

4 每位 Brady 都会出现在表中吗？利用下面的表，描绘出运行 INSERT 语句后的数据内容。

your_table

id	*first_name*	*last_name*
1	Marcia	Brady
2	Bobby	Brady
3	Cindy	Brady
99	Peter	Brady

看起来 Jan 的数据丢失了，因为我们试着把 Marcia 用掉的索引值再用到 Jan 的数据上。Marcia! Marcia! Marcia!

问： 为什么第一个查询，就是把 `id` 列设为 NULL 的那个，可以在要求 `id` 为 NOT NULL 的情况下插入行呢？

答： 虽然第一个查询看似不会成功，但 AUTO_INCREMENT 会忽略 NULL。然而在没有AUTO_INCREMENT 的情况下，我们就会收到错误信息，而且也不会插入行。试试看！

我说……你的解释一点保证都没有。当然，我可以贴上 SHOW CREATE TABLE 提供的 SQL 代码，但我还是感觉我又要删除表、重新输入所有数据，这一切只为了再次添加主键列。

你不用重新开始，事实上，我们可以改用 ALTER 语句。

带有数据的表不应该经历被丢弃、卸除、重建的步骤。事实上，我们可以改变现有的表。但为了这样做，我们要向第5章借用ALTER语句和关键字。

为现有的表添加主键

下面是在Greg的my_contacts表里添加一个AUTO_INCREMENT主键的SQL代码。（命令很长，把书转个方向看。）

这是一个新的 SQL 命令，ALTER。

这是给表添加新列的代码。看起来是不是很熟悉？

FIRST要求软件把新列放在最前面。它是一个可选关键字，但把主键列放在最前面是个不错的习惯。

```
ALTER TABLE my_contacts
ADD COLUMN contact_id INT NOT NULL AUTO_INCREMENT FIRST,
ADD PRIMARY KEY (contact_id);
```

ADD COLUMN 的作用正如其字面所示，它会添加一个新列并命名为 contact_id。

各位应该已经熟悉添加主键的语法了。

动动脑

你觉得这会为已经存在于表中的记录的新 contact _ id 列添加值吗？还是只适用于新插入的记录呢？应该怎么检查？

ALTER TABLE并添加PRIMARY KEY

自己试一下。请打开SQL终端，记得先USE gregs_list以切换到这个数据库，然后输入下列命令：

```
File Edit Window Help Alterations
> ALTER TABLE my_contacts
    -> ADD COLUMN contact_id INT NOT NULL AUTO_INCREMENT FIRST,
    -> ADD PRIMARY KEY (contact_id);

Query OK, 50 rows affected (0.04 sec)
Records: 50  Duplicates: 0  Warnings: 0
```

这一行告诉我们，命令为表中现有的50条记录添加了新列。不过各位的应该没有这么多的数据。

这一招真聪明！我的主键全都填上值了。ALTER TABLE 也可以帮我添加电话号码列吗？

为了查看修改后的表的样子，请

```
SELECT * from my_contacts;
```

添加的contact_id列被放在表中所有其他列的前面。

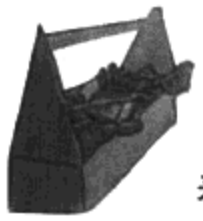
```
File Edit Window Help Alterations
+------------+------------+------------+---------------------
| contact_id | last_name  | first_name | email
+------------+------------+------------+---------------------
|     1      | Anderson   | Jillian    | jill_anderson@yahoo.
|     2      | Joffe      | Kevin      | kj@simuduck.com
|     3      | Newsome    | Amanda     | aman2luv@yahoo.com
|     4      | Garcia     | Ed         | ed99@mysoftware.com
|     5      | Roundtree  | Jo-Ann     | jojo@yahoo.com
|     6      | Br...ggs   | Chri...    | cbri...s@...mail...com
```

因为我们使用AUTO_INCREMENT，所以这一列会同时填入递增值，就如同更新了表中的每条记录一般。

下次我们INSERT新记录时，contact_id列会被自动赋予递增值（目前表中最大的contact_id值加1）。如果最新一条记录的contact_id值是23，下次添加新记录时的contact_id值即为24。

请记住，这里并未列出完整的表，Greg有很多联络人。

Greg 能得到他想要的电话号码列吗？翻到第5章便知分晓。



你的SQL工具包

现在，第4章已经收入各位的工具包了，看看你多了哪些新工具吧！若想一览本书所有工具提示，请参考附录 3。

ATOMIC DATA

数据原子性。列中的数据若已拆解成查询所需的最小单位，就是具有原子性。

ATOMIC DATA 规则一：

具原子性表示在同一列中不会存储多个类型相同的数据。

ATOMIC DATA 规则二：

具原子性表示不会用多个列来存储类型相同的数据。

SHOW CREATE TABLE

使用这个命令来呈现创建现有表的正确语法。

FIRST NORMAL FORM (1NF)

第一范式。每个数据行均需包含原子性数据值，而且每个数据行均需具有唯一的识别方法。

PRIMARY KEY

主键。一个或一组能识别出唯一数据行的列。

AUTO_INCREMENT

若在列的声明中使用这个关键字，则每次执行INSERT命令来插入数据时，它都会自动给列赋予唯一的递增整数值。

试着让小丑的表更具原子性。假设我们要从appearance、activities和last_seen列中查找数据，请试着为列设计更好的选择。

这个问题没有绝对的答案。

你可以做的就是另外存储事物，像性别、衣服颜色、裤子颜色、帽子外形、乐器、交通工具、气球（存储为Y/N值）、唱歌表演（存储为Y/N值）、舞蹈表演（存储为Y/N值）。

为了让表具有原子性，必须把各种表演分成不同列，还要把各种外观特性分成不同列。

加分点：如果你还打算把出现的地点分成街道、城镇和州，请为自己鼓掌！

5 ALTER

改写历史

你可曾希望更正以前年少无知犯下的错误？现在，你的机会来了。使用ALTER**命令**，你能对几天前、几个月前，甚至是几年前设计的表套用新学到的设计方法。更好的是，套用时不会影响现有数据。随着熟悉本章的过程，各位还会学到**规范化**的真正意义，并且能让你的所有表都符合规范化，无论它是过去的还是未来的产物。

我们需要一些改变

Greg 想对他的表做些修改，但又不想丢失任何数据。

```
File Edit Window Help KeyedUp
+-------------+------------+-------------+--------------------
| contact_id  | last_name  | first_name  | email
+-------------+------------+-------------+--------------------
|     1       | Anderson   | Jillian     | jill_anderson@yaho
|     2       | Joffe      | Kevin       | kj@simuduck.com
|     3       | Newsome    | Amanda      | aman2luv@yahoo.com
|     4       | Garcia     | Ed          | ed99@mysoftware.c
|     5       | Roundtree  | Jo-Ann      | jojo@yahoo.com
|     6       | Briggs     | Chris       | cbriggs@somemail.co
```

没错，你可以使用 ALTER TABLE 轻松地添加它。

事实上，因为前一章已经稍稍提过 ALTER 命令，我们觉得大家可以自己试试看。请做下一页的练习来得到你需要的SQL代码。

磨笔上阵

请仔细观察下列用来添加主键列的 ALTER TABLE 命令（出现于第4章），然后试着考虑添加可以存储10位数的电话号码列的命令。注意，你不需要在新命令中使用所有关键字。

```
ALTER TABLE my_contacts
ADD COLUMN contact_id INT NOT NULL AUTO_INCREMENT FIRST,
ADD PRIMARY KEY (contact_id);
```

写下你的 ALTER TABLE 命令：

.....

.....

.....

你甚至可以用关键字AFTER告诉软件电话号码列的安放位置。猜猜看，应该如何安排AFTER，才能让新列放在 first_name 列后呢？

写下你的 ALTER TABLE 命令：

.....

.....

.....

磨笔上阵解答

请请仔细观察下列用来添加主键列的 ALTER TABLE 命令（出现于第4章），然后试着考虑添加可以存储10位数的电话号码列的命令。注意，你不需要在新命令中使用所有关键字。

```
ALTER TABLE my_contacts
ADD COLUMN contact_id INT NOT NULL AUTO_INCREMENT FIRST,
ADD PRIMARY KEY (contact_id);
```

上一例用到的关键字中，NOT NULL、AUTO_INCREMENT、FIRST对本次练习没有用处。

写下你的 ALTER TABLE 命令：

我们修改的表，仍然名为 my_contacts。

```
ALTER TABLE my_contacts
ADD COLUMN phone VARCHAR(10)
```

我们假设所有电话号码的长度。Cies 不考虑其他国家的电话号码长度。

这里明确告诉 ALTER 指令我们想如何修改表。

新的列的名称是 phone。

你甚至可以用关键字AFTER告诉软件电话号码列的安放位置。猜猜看，应该如何安排AFTER，才能让新列放在 first_name 列后呢？

写下你的 ALTER TABLE 命令：

```
ALTER TABLE my_contacts
ADD COLUMN phone VARCHAR(10)
AFTER first_name;
```

关键字AFTER 紧跟在新添加的列的名称后面。本处的语法会把 phone 列放在 first_name列后。

AFTER 是可选关键字。如果不使用它，新列则会添加至表的最后。

各位已经看到关键字 *FIRST* 和 *AFTER your_column* 的使用方法了，不过你还可以使用 *BEFORE your_column* 和 *LAST*。另外还有 *SECOND*、*THIRD* 可供选用，我想大家应该能依此类推。

SQL关键字冰箱磁铁

请使用下列关键字磁铁来改变安放 phone 列的位置。请尽量创建任何你想到的命令组合，运行命令后请画出你得到的列。

phone	contact_id	last_name	first_name	email

```
ALTER TABLE my _ contacts
ADD COLUMN phone VARCHAR(10)
```

contact_id	last_name	first_name	email	phone

```
ALTER TABLE my _ contacts
ADD COLUMN phone VARCHAR(10)
```

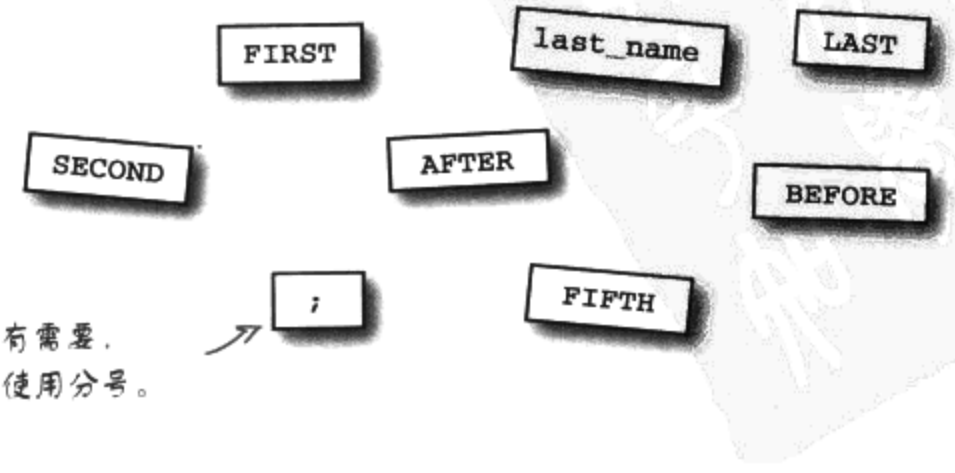
contact_id	phone	last_name	first_name	email

```
ALTER TABLE my _ contacts
ADD COLUMN phone VARCHAR(10)
```

contact_id	last_name	phone	first_name	email

```
ALTER TABLE my _ contacts
ADD COLUMN phone VARCHAR(10)
```

在语句末尾加上关键字磁铁。



只要有需要，尽量使用分号。

SQL关键字冰箱磁铁解答

请使用下列关键字磁铁来改变安放 phone 列的位置。请尽量创建任何你想到的命令组合，运行命令后请画出你得到的列。

```
ALTER TABLE my_contacts
ADD COLUMN phone VARCHAR(10) FIRST ;
```

FIRST可把phone列安置于所有其他列的前面。

phone	contact_id	last_name	first_name	email

```
ALTER TABLE my_contacts
ADD COLUMN phone VARCHAR(10) LAST ;

ALTER TABLE my_contacts
ADD COLUMN phone VARCHAR(10) FIFTH ;

ALTER TABLE my_contacts
ADD COLUMN phone VARCHAR(10) ;
```

LAST可把phone列安置于所有其他列的后面，FIFTH在本习题中也有相同效果。当然，完全不加这些关键字也可以。

contact_id	last_name	first_name	email	phone

```
ALTER TABLE my_contacts
ADD COLUMN phone VARCHAR(10) SECOND ;

ALTER TABLE my_contacts
ADD COLUMN phone VARCHAR(10)
BEFORE last_name ;
```

SECOND可把phone列安置为第二个列。在本习题中，若用 BEFORE *last_name*，也有相同效果。

contact_id	phone	last_name	first_name	email

```
ALTER TABLE my_contacts
ADD COLUMN phone VARCHAR(10)
AFTER last_name ;
```

AFTER *last_name*可把phone列安置为第三个列。如果你采用THIRD，也会有相同效果。

contact_id	last_name	phone	first_name	email

修改表

ALTER命令几乎能让你改变表里的一切，而且不需重新插入数据。但也要小心，如果改变列的类型，你可能就会有遗失数据的风险。

Dataville美容医学中心

针对现有表的特定服务项目：

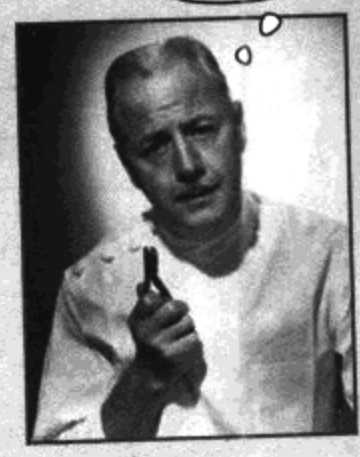
CHANGE 可同时改变现有列的名称和数据类型 *

MODIFY 修改现有列的数据类型或位置 *

ADD 在当前表中添加一列——可自选类型

DROP 从表中删除某列 *

*注：可能导致数据遗失，本中心不做任何担保。



附加服务

重新整理你的列

（只有 ADD 才有这项优惠哦）

动动脑

这张表为什么要改变？

projekts

number	descriptionofproj	contractoronjob
1	outside house painting	Murphy
2	kitchen remodel	Valdez
3	wood floor installation	Keller
4	roofing	Jackson

终极表美容沙龙

我们先从需要极大整容的表开始。

欢迎光临终极表美容沙龙！在接下来的几页中，我们会把一个乱七八糟的表转变为任何数据库都会高兴接受的表。

表名并未解释它的内容。

这个列名完全无法说明其存储的内容。

也许加上下划线会让这个名称更容易理解。

projekts

number	descriptionofproj	contractoronjob
1	outside house painting	Murphy
2	kitchen remodel	Valdez
3	wood floor installation	Keller
4	roofing	Jackson

虽然表和列的名称很糟糕，但其中的数据却很规范，我们想保留下来。

用 DESCRIBE 查看这个表的构成。它会说明此列是否为主键，并且告诉我们每列存储的数据的类型。

File Edit Window Help BadTableDesign

```
--> DESCRIBE projekts;
+--------------------+-------------+------+-----+---------+-------+
| Field              | Type        | Null | Key | Default | Extra |
+--------------------+-------------+------+-----+---------+-------+
| number             | int(11)     | YES  |     | NULL    |       |
| descriptionofproj  | varchar(50) | YES  |     | NULL    |       |
| contractoronjob    | varchar(10) | YES  |     | NULL    |       |
+--------------------+-------------+------+-----+---------+-------+
3 rows in set (0.01 sec)
```

表的改名换姓

在当前状态下，这张表有些问题，但幸好有 ALTER 可用，我们可以修改这张表，让它适合存储一系列陈年旧屋专用的修缮计划。第一步是以 ALTER TABLE 为表重新取一个有意义的名字。

"projekts" 是表的旧名。

```
ALTER TABLE projekts
RENAME TO project_list;
```

这里的语法就跟英语语法一样！我们想要 RENAME 表。

"project_list" 是我们给表取的新名字。

接下来的描述会帮助我们找出表中其他需要 ALTER 的地方。请从以下描述表用途的短文中找出所需的列名，然后把列名填入周围的空格中。

proj_id

为了让表规范，我们也会在表中添加一个具唯一性的编号列作为主键。然后我们需要描述修缮计划、动工日期、估价、承包公司名称及其联络电话的列。

接下来的描述会帮助我们找出表中其他需要ALTER的地方。请从以下描述表用途的短文中找出所需的列名，然后把列名填入周围的空格中。

如果你的列名和我们的不太一样，没关系。选用的缩写稍微不一样并不要紧，只要你选的名称能表示存储内容就好。

请确定你和其他使用数据库的人可以理解缩写名称（如proj_id）的意义。

proj_id

proj_desc

start_desc

为了让表规范，我们也会在表中添加一个具唯一性的编号列作为主键。然后我们需要描述修缮计划、动工日期、估价、承包公司名称及其联络电话的列。

est_date

con_name

con_phone

需要好好地计划一下

project_list

number	descriptionofproj	contractoronjob
1	outside house painting	Murphy
2	kitchen remodel	Valdez
3	wood floor installation	Keller
4	roofing	Jackson

显然，表中新的三列中的数据已经就定位了。此时与其创建新列，不如利用命令RENAME修改现有列的名称。既然为包含合格内容的列重新命名，也就不需把数据插入新列中了。

动动脑

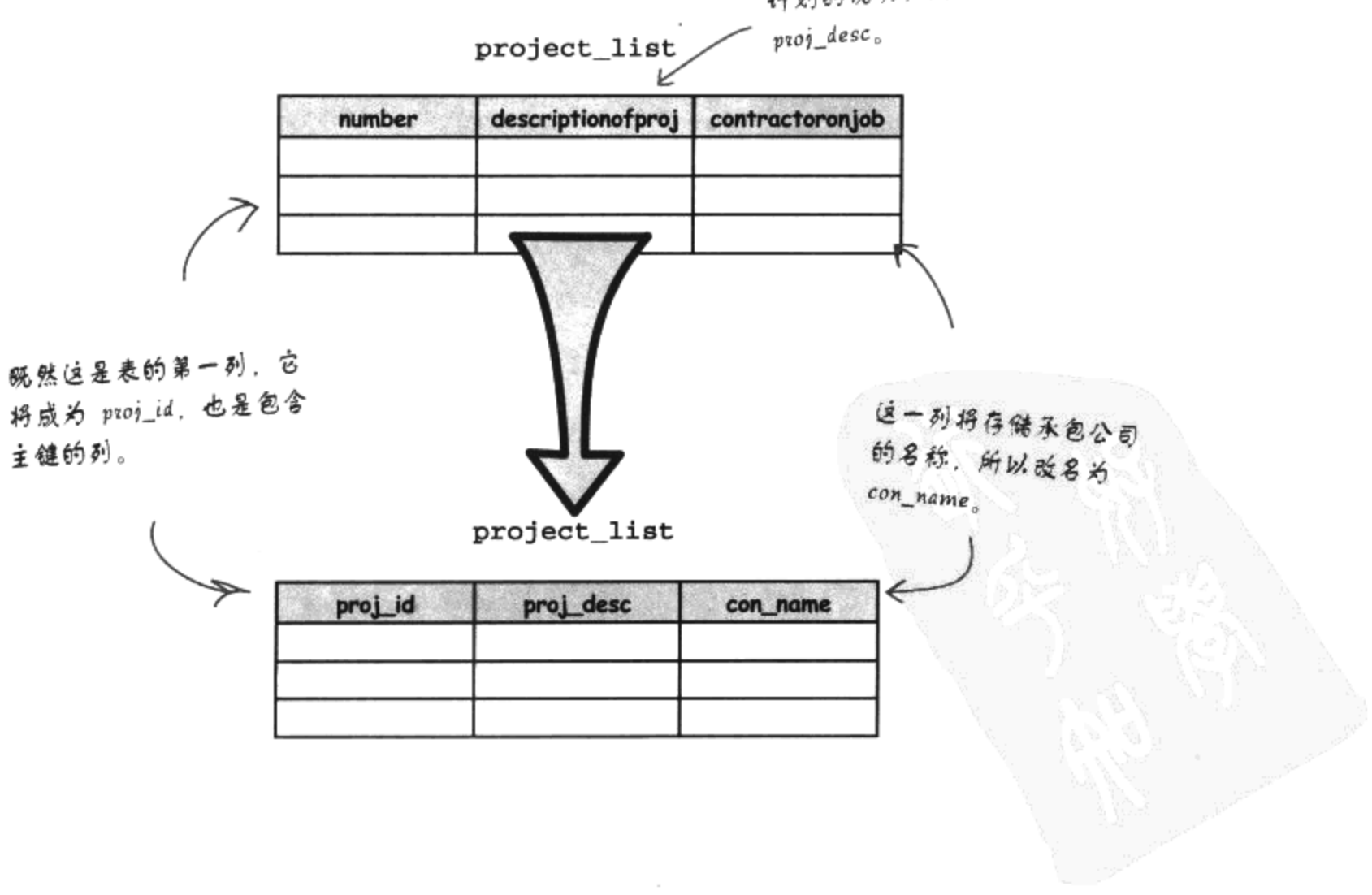
现有的哪一列适合作为主键候补呢？

重新装备列

有了着手修改的计划，我们就可以用 ALTER调整表中现有的三列的名称：

- number 是主键，改为**proj_id**
- descriptionofproj 是每项修缮计划的说明，调整为**proj_desc**
- contractoronjob 是承包公司的名称，调整为**con_name**

修改后，还有est_cost、con_phone、start_date 这三列需要补上。

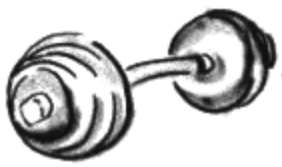


结构上的修改

刚才已经把现有的三列的名称修改为我们需要的名称。但除了修改名称外，我们还应该仔细研究每列存储的数据的类型。

以下是稍早出现过的列说明。

```
File Edit Window Help BadTableDesign
--> DESCRIBE projekts;
+--------------------+-------------+------+-----+---------+-------+
| Field              | Type        | Null | Key | Default | Extra |
+--------------------+-------------+------+-----+---------+-------+
| number             | int(11)     | YES  |     | NULL    |       |
| descriptionofproj  | varchar(50) | YES  |     | NULL    |       |
| contractoronjob    | varchar(10) | YES  |     | NULL    |       |
+--------------------+-------------+------+-----+---------+-------+
3 rows in set (0.01 sec)
```



脑力锻炼

请观察每列的类型（Type），并判断该类型是否符合日后存储数据的需求。

ALTER和CHANGE

接下来，我们要把number列改名为proj_id，并把它设置为AUTO_INCREMENT，然后将它标注为主键。听起来很复杂，其实不然。事实上，只要一行命令就可做到：

这一次使用CHANGE COLUMN，因为我们要同时改变原名为"number"的列的名称和类型。

我们仍然使用同一个表，但请记住，我们给了它新的名称了。

"proj_id"是我们希望给列的新名称，而且……

……我们希望它会自动填入递增的整数，而非NULL值。

```
ALTER TABLE project_list
CHANGE COLUMN number proj_id INT NOT NULL AUTO_INCREMENT,
ADD PRIMARY KEY (proj_id);
```

这部分要求SQL软件使用新命名的proj_id列作为主键。

磨笔上阵

画出运行完上述命令后的表。

答案请见第233页。

以一条SQL语句改变两个列

接下来我们不只是修改单一列，而是用一条语句改变两个列。我们要修改descriptionofproj和contractoronjob列的名称，同时更改它们的数据类型。我们所要做的只是在一条ALTER TABLE语句中放入两个CHANGE COLUMN，并在中间加上分隔用的逗号。

"descriptionobproj" 是我们要改变的旧名称。

"proj-desc" 是列的新名称。

这里增加可存储的字符容量，让描述能更详细一点。

```
ALTER TABLE project_list
CHANGE COLUMN descriptionofproj proj_desc VARCHAR(100),
CHANGE COLUMN contractoronjob con_name VARCHAR(30);
```

另一个旧名称 "contractoronjob"，它也要修改……

……修改为 "con_name"，另外还修改了它的数据类型。

注意！

如果把数据改成另一种类型，你可能会丢失数据。

如果你想改变的数据类型和原始类型不兼容，命令则不会执行，SQL 软件也会抱怨语句有问题。

但真正的惨剧可能发生在类型兼容的情况下，你的数据可能被**截断**。

例如：从 varchar(10) 改为 char(1)，数据“Bonzo”将被砍成“B”。

相同的惨剧也会发生在数字类型上。我们可以在各种数字类型间切换，但数据会被转换为新的类型，这时就可能丢失部分数据！

如果我只想改变列的数据类型，例如让它多存储几个字符，但又希望列名维持原状，我可以重复填入列名对不对？就像这样：

```
ALTER TABLE myTable
CHANGE COLUMN myColumn myColumn NEWTYPE;
```

当然可以这么做，不过还有更简单的方法。

其实你可以使用关键字MODIFY，它只会修改列的类型而不会干涉它的名称。

假设要把proj_desc列的字符长度修改为VARCHAR(120)以容纳更长的说明文字。只要这么做：

```
ALTER TABLE project_list
MODIFY COLUMN proj_desc VARCHAR(120);
```

要修改的列名。

新的数据类型。

当然，你必须确定新类型不会造成旧数据被截断！

没有蠢问题

问： 如果我想改变列的顺序呢？像ALTER TABLE MODIFY COLUMN proj_desc AFTER con_name;这样做可以吗？

答： 创建表后你就无法真正地改变列的顺序了。最多只能在指定位置添加新列，然后删除旧列，但这样会丢失旧列中的所有数据。

问： 但是列的顺序如果不对会造成另一个问题吗？

答： 不会，因为在SELECT查询中可以指定查询结果的列顺序。硬盘里存储数据的顺序并不重要，我们可以这样：

```
SELECT column3, column1 FROM your_table;
```

或

```
SELECT column1, column3 FROM your_table;
```

或改为其他你喜欢的顺序。

project_list

proj_id	proj_desc	con_name
1		
2		
3		

现在还需要再添加三个列：电话号码、动工日期、估价。

请你只用一条 ALTER TABLE 语句完成任务，特别要注意数据类型。然后把修改后的表填入下面的表格中。

project_list

习题解答

喂，我正在跟经纪人通话。大家继续完成剩下的列吧！

project_list

proj_id	proj_desc	con_name
1		
2		
3		

现在还需要再添加三个列：电话号码、动工日期、估价。

请你只用一条 ALTER TABLE 语句完成任务，特别要注意数据类型。然后把修改后的表填入下面的表格中。

最多可存储10个字符的 VARCHAR，即可存储区号。

```
ALTER TABLE project_list
  ADD COLUMN con_phone VARCHAR(10),
  ADD COLUMN start_date DATE,
  ADD COLUMN est_cost DECIMAL(7,2);
```

因为正在添加新列，所以用 ADD。

还记得DEC类型吗？这里把浮点数的总长设定为7位数，小数点后有2位。

project_list

proj_id	proj_desc	con_name	con_phone	start_date	est_cost
1					
2					
3					

快！卸除那一列！

停止所有事情！

我们刚刚发现修缮计划被搁置。因此，我们可以先卸除`start_date`。我们不需要保留一个不确定的列来浪费数据库的空间。

只在表中保留必要列是一个很好的编程习惯。如果用不到某列，请把它卸除（drop）。如果以后有需要，`ALTER`让我们可以轻松地把它添加回表中。

你的列越多，RDBMS的工作就越累，数据库所占用的空间也就越大。当表还小时，这种情况并不明显，但随着数据的增长，你会发现查询跑得越来越慢，而计算机的处理器也会运作得越来越辛苦。

磨笔上阵

请大家写出卸除 start_date 列的 SQL 语句。虽然还没提到它的语法，但试一下又何妨？

磨笔上阵 解答

请大家写出卸除 start_date 列的 SQL 语句。虽然还没提到它的语法，但试一下又何妨？

这是我们的表名。

```
ALTER TABLE project_list
DROP COLUMN start_date;
```

如果想卸除 start_date 列，可以使用 DROP 命令。很简单吧！

要从表中移除的列。

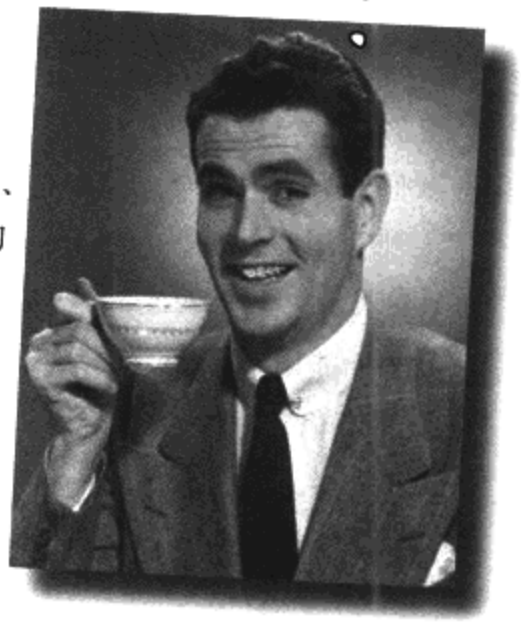
注意！

一旦你卸除列，原本存储在该列中的一切内容都会跟着被卸除。

使用 DROP COLUMN 时务必要小心。或许应该先以 SELECT 选取出列，确定那就是你想卸除的列。比起缺少必要性的数据，表中存储多余的数据总会更好些。

Pimp My Table

你现在该把那个二手拼装的破烂老爷车表换成崭新的数据万人迷，并且把表调校到前所未有的境界。



我们的习题很简单，请把可悲的二手旧表修改（ALTER）为焕然一新、美观实用的新表。各位面对的部分难题是如何在修改时不弄乱表中的任何数据，只是对它们进行操作。准备好接受挑战了吗？

如果你能用一条ALTER TABLE语句完成任务，就给自己加分吧！

Before

hooptie

color	year	make	mo	howmuch
silver	1998	Porsche	Boxter	17992.540
NULL	2000	Jaguar	XJ	15995
red	2002	Cadillac	Escalade	40215.9

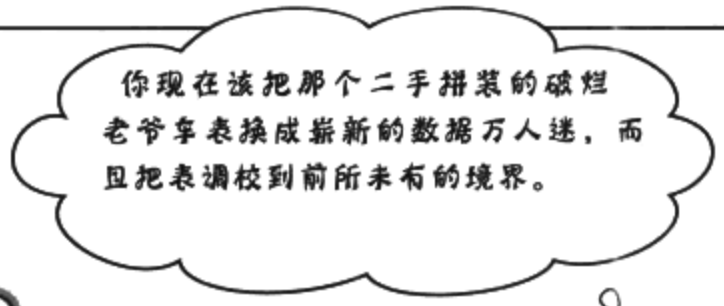
After

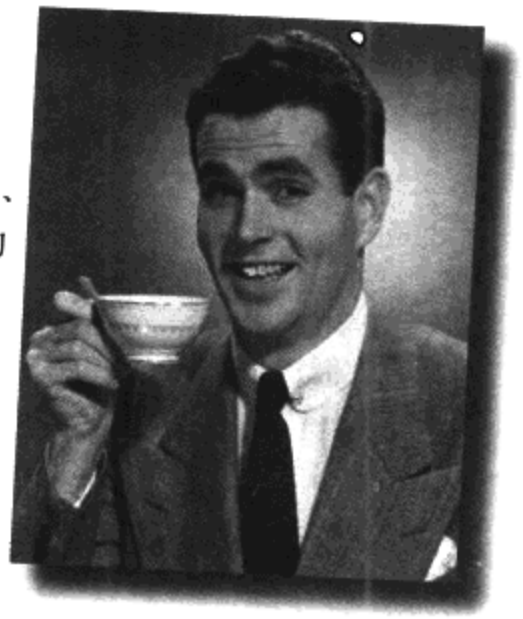
car_table

car_id	VIN	make	model	color	year	price
1	RNKLK66N33G213481	Porsche	Boxter	silver	1998	17992.54
2	SAEDA44B175B04113	Jaguar	XJ	NULL	2000	15995.00
3	3GYEK63NT2G280668	Cadillac	Escalade	red	2002	40215.90

习题解答

Pimp My Table





我们的习题很简单，请把可悲的二手旧表修改（ALTER）为焕然一新、美观实用的新表。各位面对的部分难题是如何在修改时不弄乱表中的任何数据，只是对它们进行操作。准备好接受挑战了吗？

如果你能用一条 ALTER TABLE 语句完成任务，就给自己加分吧！

Before

hooptie

color	year	make	mo	howmuch
silver	1998	Porsche	Boxter	17992.540
NULL	2000	Jaguar	XJ	15995
red	2002	Cadillac	Escalade	40215.9

After

car_table

car_id	VIN	make	model	color	year	price
1	RNKLK66N33G213481	Porsche	Boxter	silver	1998	17992.54
2	SAEDA44B175B04113	Jaguar	XJ	NULL	2000	15995.00
3	3GYEK63NT2G280668	Cadillac	Escalade	red	2002	40215.90

可以先用 DESCRIBE观察每列原本的数据类型，以避免修改时意外截断数据。

```
ALTER TABLE hooptie

RENAME TO car_table,

ADD COLUMN car_id INT NOT NULL AUTO_INCREMENT FIRST,

ADD PRIMARY KEY (car_id),

ADD COLUMN VIN VARCHAR(16) AFTER car_id,

CHANGE COLUMN mo model VARCHAR(20),

MODIFY COLUMN color AFTER model,

MODIFY COLUMN year SIXTH,

CHANGE COLUMN howmuch price DECIMAL(7,2);
```

这里需要把列名“mo”改为“model”，然后再把 color 和 year 列移到它的后面。

必须提供重新命名的列的数据类型。

* 编注：对本页的解答很有疑问吗？请快翻到下一页。

问： 之前你说过，MODIFY 无法重新排列列的顺序。但是我的 SQL 软件工具却能让我重新排列它们。它是怎么办到的？

答： 其实你的软件在背后执行了非常多的命令。它把我们要移除的列的内容暂时复制到临时表中，然后卸除你要移除的列，再用 ALTER 创建与旧列同名的新列，并放在我们指定的位置，而后把临时表的内容复制到新列中，最后再删除临时表。

一般而言，如果列中已经有内容，而且你使用的软件无法完成上述操作的话，最好不要对列的位置动手动脚。你可在SELECT 时用任何顺序排列列。

问： 所以说，只有添加列时才是调整顺序的好时机吗？

答： 没错。最好在设计表时就已构思好各个列的最佳顺序。

问： 如果我已经创建了主键，然后又意外地想改用另一列呢？可以只移除主键的设置而不改变其中的数据吗？

答： 可以，而且很简单：

```
ALTER TABLE your_table DROP PRIMARY KEY;
```

问： AUTO_INCREMENT 又该如何处理？

答： 你可以把它添加到没有自动递增功能的列中，如下所示：

```
ALTER TABLE your_table CHANGE your_id
your_id INT(11) NOT NULL AUTO_INCREMENT;
```

而且这样就可以将它删除：

```
ALTER TABLE your_table CHANGE your_id
your_id INT(11) NOT NULL;
```

有一点要记住：每个表中只有一列可以加上AUTO_INCREMENT，该列必须为整数类型而且不能包含 NULL。

复习要点

- 想同时改变列的名称和类型时请用 CHANGE。
- 只想改变数据类型时请用 MODIFY。
- DROP COLUMN的功能是从表中卸除指名的列。
- 使用 RENAME 改变表的名称。
- 使用FIRST、LAST、BEFORE column_name、AFTER column_name、SECOND、THIRD、FOURTH等关键字，可以调整列的顺序。
- 有些 RDBMS 只有在添加新列时才允许改变列的顺序。

ALTER TABLE有助于改善表设计。

使用SELECT、UPDATE时搭配ALTER TABLE，我们就可以把使用不便、没有原子性的列调整为具有精准原子性的列。这一切都是关于正确结合我们学过的 SQL 语句。

先观察一下用于my_contacts表的CREATE TABLE语句。

```
CREATE TABLE my_contacts
(
    contact_id INT NOT NULL AUTO_INCREMENT
    last_name VARCHAR(30) default NULL,
    first_name VARCHAR(20) default NULL,
    email VARCHAR(50) default NULL,
    gender CHAR(1) default NULL,
    birthday DATE default NULL,
    profession VARCHAR(50) default NULL,
    location VARCHAR(50) default NULL,
    status VARCHAR(20) default NULL,
    interests VARCHAR(100) default NULL,
    seeking VARCHAR(100) default NULL,
    PRIMARY KEY (contact_id)
)
```

我们增加了这两行来创建和设置主键。

这四列的单元性还不够，可用ALTER TABLE做些调整。

仔细研究不具原子性的location列

有时候，Greg只想知道朋友在哪个州或哪个城镇，显然location列刚好可以拆分为两部分。先看一下这一列中存储了什么数据：

```
File Edit Window Help LocationLocationLocation
--> SELECT location FROM my_contacts;
+-----------------+
|  location       |
+-----------------+
| Seattle, WA     |
| Natchez, MS     |
| Las Vegas, NV   |
| Palo Alto, CA   |
| NYC, NY         |
| Phoenix, AZ     |
```

出自my_contacts表中location列的部分数据示意。

Seattle, WA ← 州名缩写，只有两个字符。

Natchez, MS

Las Vegas, NV

Palo Alto, CA ← 逗号。

NYC, NY

城镇名称。

这些数据的格式相当一致。先列出城镇名，以逗号分隔，再列出州名缩写。因为数据输入时的一致性，我们可以把城镇名和州名分开。

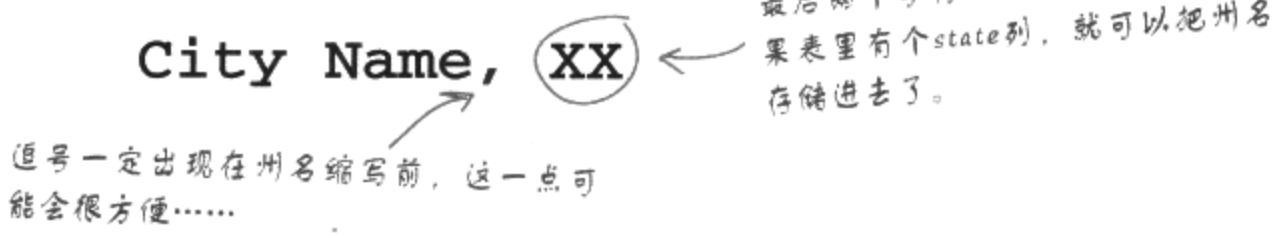
脑力锻炼

为什么会想把城镇名与州名分开呢？

你觉得接下来要怎么做？

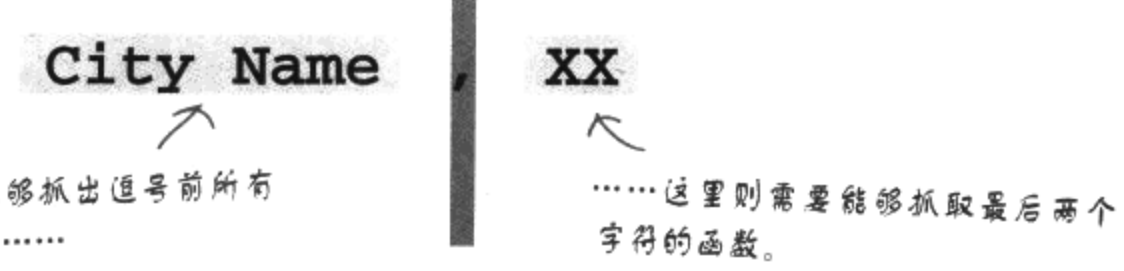
寻找模式

my_contacts表中存储的每个位置列都有相同模式：城镇名（City Name）、逗号、州名缩写（两个字符，表示为XX）。这种一致性和模式将有助于我们将其分割为更具原子性的数据。



我们可以抓出逗号前的所有内容，并放入专门存储 city names 的新列里。

然后把 location 列的最后两个字符放入名为 state 的新列里。



磨笔上阵

请写出在 my_contacts 表里添加城镇名和州名列的 ALTER TABLE 语句。

.....

.....

.....

ALTER TABLE my_contacts
ADD COLUMN city
VARCHAR(50),
ADD COLUMN state CHAR(2);

一些便利的字符串函数

我们已经看出两种模式。现在要提取州名缩写并存储到新增的**state**列里。接下来则要把逗号前的所有内容都存储到新增的**city**列里。创建了新列后，以下就是提取部分内容所需的步骤：

文本值以及有CHAR或VARCHAR类型的列中存储的值都被称为字符串(string)。

SELECT 最后两个字符

RIGHT()和LEFT()可从列中选出**指定数量的字符**。

```
SELECT RIGHT(location, 2) FROM my_contacts;
```

从列的右侧开始（LEFT()的使用语法也一样）。

这是使用的列。

这是要从列的右侧开始选取的字符数量。

字符串函数能选出文本列的部分内容。

SELECT 逗号前的所有内容

SUBSTRING_INDEX()则可撷取部分列值，也称为子字符串(substring)。这个函数会找出指定字符或字符串前的所有内容。所以我们只要把逗号用引号括起来，SUBSTRING_INDEX()就会为我们取出逗号前的所有内容。

```
SELECT SUBSTRING_INDEX(location, ',', 1) FROM my_contacts;
```

这个函数撷取部分列值（或称子字符串）。它寻找单引号里的字符串（本例为逗号），然后取出它前面的所有内容。

又看到列名了。

这里就是命令要寻找的逗号。

这里是比较棘手的部分。“1”表示命令要寻找第一个逗号。如果是“2”，函数就会寻找第二个逗号，然后才撷取它前面的所有内容。

回家试试看

SQL 有许多能在表里操纵字符串值的函数。字符串存储于文本类型的列中，通常是 VARCHAR 或 CHAR类型。

下面列出较常用的字符串辅助函数。请把它们放在 SELECT 语句中试试看。

SUBSTRING(your_string, start_position, length) 能撷取一部分 your_string 字符串，撷取的起始位置为 start_position，撷取长度当然由 length 指定。

```
SELECT SUBSTRING('San Antonio, TX', 5, 3);
```

UPPER(your_string) 和 **LOWER(your_string)** 分别可把整组字符串改为大写或小写。

```
SELECT UPPER('uSa');

SELECT LOWER('spaGHEtti');
```

REVERSE(your_string) 的作用正如其名：反转字符串里的字符排序。

```
SELECT REVERSE('spaGHEtti');
```

LTRIM(your_string) 与 **RTRIM(your_string)** 会返回清除多余空格后的字符串，它们分别清除字符左侧（前面）和右侧（后面）的多余空格。

```
SELECT LTRIM(' dogfood ');

SELECT RTRIM(' catfood ');
```

LENGTH(your_string) 返回字符串中的字符数量。

```
SELECT LENGTH('San Antonio, TX ');
```

重要：字符串函数不会改变存储在表中的内容；它们只是把字符串修改后的模样当成查询结果返回。

连连看

我们要试着把location列中的信息取出，然后分开存储到两个新列中（city和state）。

以下是我们的操作步骤。请把完成每个步骤的需求与旁边的SQL关键字（可能不只一个）连起来。

SUBSTRING_INDEX()

SELECT

1. 检查特定列中的数据以寻找模式。

LEFT

ADD COLUMN

2. 在表中添加新的空白列。

ADJUST

RIGHT

3. 从文本列中撷取部分数据。

ALTER TABLE

4. 把第三步中撷取的数据存入其中一个空白列里。

DELETE

INSERT

UPDATE

答案请见第228页。

我们知道如何使用各种正确的工具，但还是不知道该如何有效率地组合它们。也许可以试着使用UPDATE语句搭配字符串函数……

根据至目前所学的一切，我们只能先以SELECT找出正确数据，再用UPDATE一次更新一条记录。

但在SQL中，我们可以结合这些语句。翻到下一页，我们将讨论把数据值存储到新列的方式。

连连看

我们要试着把location列中的信息取出，然后分开存储到两个新列中（city和state）。

以下是我们的操作步骤。请把完成每个步骤的需求与旁边的SQL关键字（可能不只一个）连起来。

1. 检查特定列的数据以寻找模式。
2. 在表中添加新的空白列。
3. 从文本列中撷取部分数据。
4. 把第三步撷取的数据存入其中一个空白列里。

SUBSTRING_INDEX()

SELECT

ADD COLUMN

LEFT

ADJUST

RIGHT

ALTER TABLE

DELETE

INSERT

UPDATE

以现有列的内容填入新列

还记得UPDATE的语法吗？我们可以用它为表中的每一行都填上相同的新值。下列语句即为**改变每一行中的同一列的值**的语法。在newvalue处，可以指定新值或另一个列名。

```
UPDATE table_name
SET column_name = newvalue;
```

表中的每一行都会被设定为新值，一次一行。

为了给city、state列添加新值，我们可以在UPDATE语句中使用字符串函数RIGHT()。字符串函数会撷取location列的最后两个字符并放入新的state列中。

```
UPDATE my_contacts
SET state = RIGHT(location, 2);
```

这里是存储州名数据的新列。

这个字符串函数会撷取location列的最后两个字符。

为什么这个语法可行呢？并没有指示更新列的 WHERE 子句啊？

在这里，没有 WHERE 也行得通。请看下一页的说明。

UPDATE和SET搭档的成功之道

你的SQL软件把前页的语句解释为每次只对表的某一行进行操作，然后它会重头执行，直到所有州名缩写都存入新的state列为止。

my_contacts

contact_id	location	city	state
1	Chester, NJ		
2	Katy, TX		
3	San Mateo, CA		

简化版的范例表示意。

这里是我们要用的SQL语句。

```
UPDATE my_contacts
SET state = RIGHT(location, 2);
```

看看上述语句如何改变范例表。第一次遍历完整表时，这条语句抓出第一条记录的 location 列并套用函数。

然后这条语句再运行一次，这次抓到第二行的location列，对它套用函数，依此类推，直到所有州名缩写都已存储在新的state列中，而且没有任何记录符合语句条件为止。

字符串函数可以和 SELECT、UPDATE、DELETE 搭配使用。

第一次遍历表。

```
UPDATE my_contacts
SET state = RIGHT('Chester,NJ',2)
```

取出第一条记录的location值并套用函数进行运算。

第二次遍历表。

```
UPDATE my_contacts
SET state = RIGHT('Katy, TX',2)
```

现在处理第二条记录。

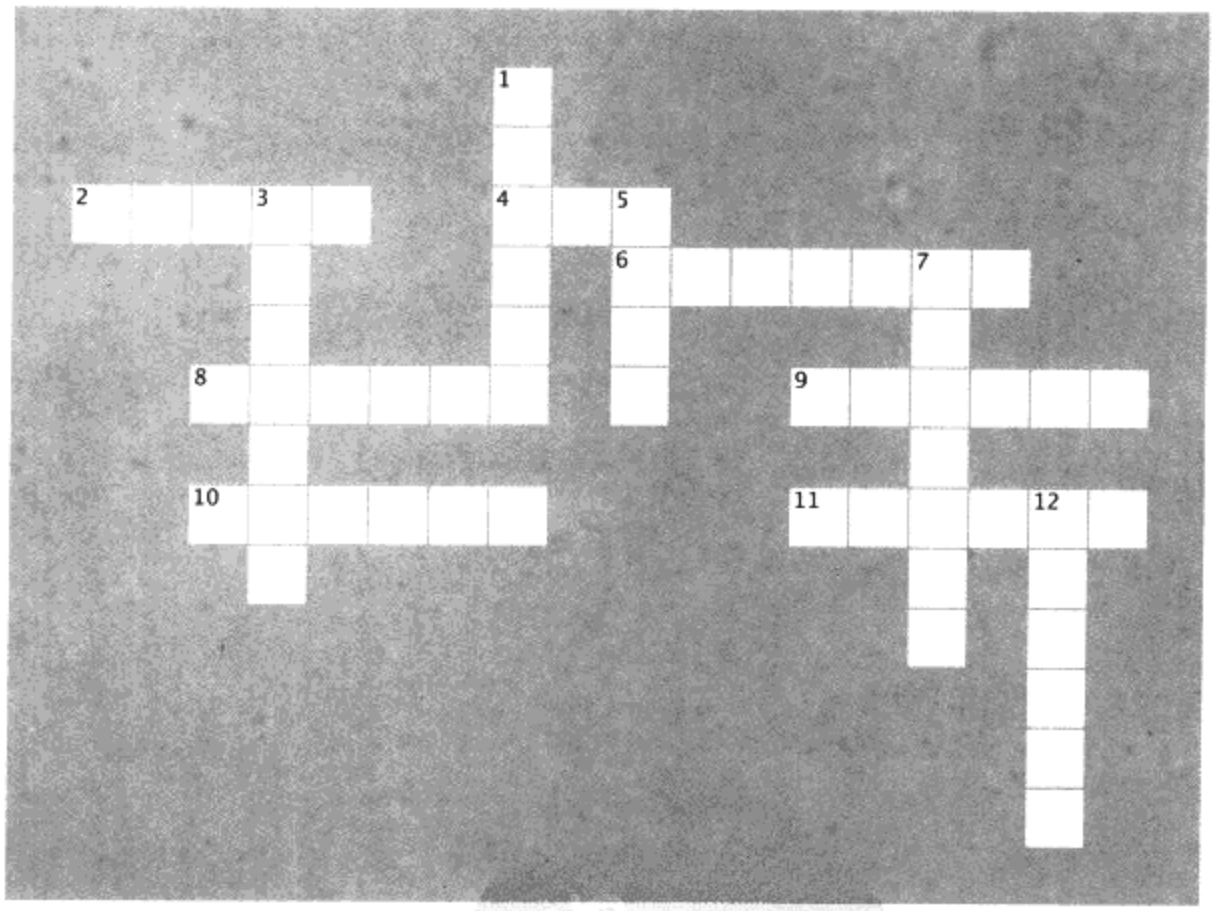
第三次也是最后一次遍历表（因为我们的简表只有三条记录）。

```
UPDATE my_contacts
SET state = RIGHT('San Mateo, CA',2)
```

最后处理第三条记录。

ALTER填字游戏

填字游戏怎么会对 SQL 学习有帮助？填填看就知道了。它能帮我们从另一个角度出发，仔细思考本章提到的命令和关键字。



横向

2.____(your_string) 将返回清除过前面多余空格后的字符串（“前面”是指从最左边开始算）。

4. ALTER 语句和 ____ COLUMN 子句能为表添加新列。

6. ____(your_string) 正如其名，反转字符串里的字符顺序。

8. ALTER TABLE projects ____ TO project_list;

9. ____ 函数可与 SELECT、UPDATE、DELETE 搭配使用。

10. SUBSTRING(your_string, start_position, length)能取出部分 your_string 的内容，撷取的起始位置为 start_position，____ 指定取出的字符串长度。

11. 使用 ____ 改变表名。

纵向

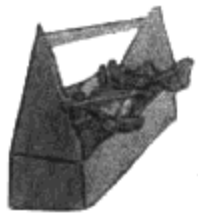
1. 以关键字 ____ 修改列中存储的数据的类型。

3. 每个表只能有一个 AUTO_INCREMENT 字段，而且类型必须是 ____。

5. 不再需要某列时，请用 ____COLUMN 搭配 ALTER。

7. 存储在 VARCHAR 或 CHAR 类型的列里的值通常称为 ____。

12. 只想修改数据类型时，请用 ____ 子句搭配 ALTER。



你的SQL工具包

为自己鼓掌吧！你已经熟悉第5章的内容，而且把 ALTER 收入 SQL 工具包了。若想一览本书所有工具提示，请参考附录 3。

ALTER TABLE

以保留表中现有数据为前提，修改表的名称及整体结构。

ALTER 搭配 ADD

以你指定的顺序把列添加到表中。

ALTER 搭配 DROP

从表中卸除列。

ALTER 搭配 CHANGE

同时修改现有列的名称和类型。

ALTER 搭配 MODIFY

只修改现有列的类型。

String Functions

字符串函数。这些函数可修改字符串列的内容副本并以查询结果的形式返回。同时，原始数据不会改变。

磨笔上阵解答

第210页上的习题。

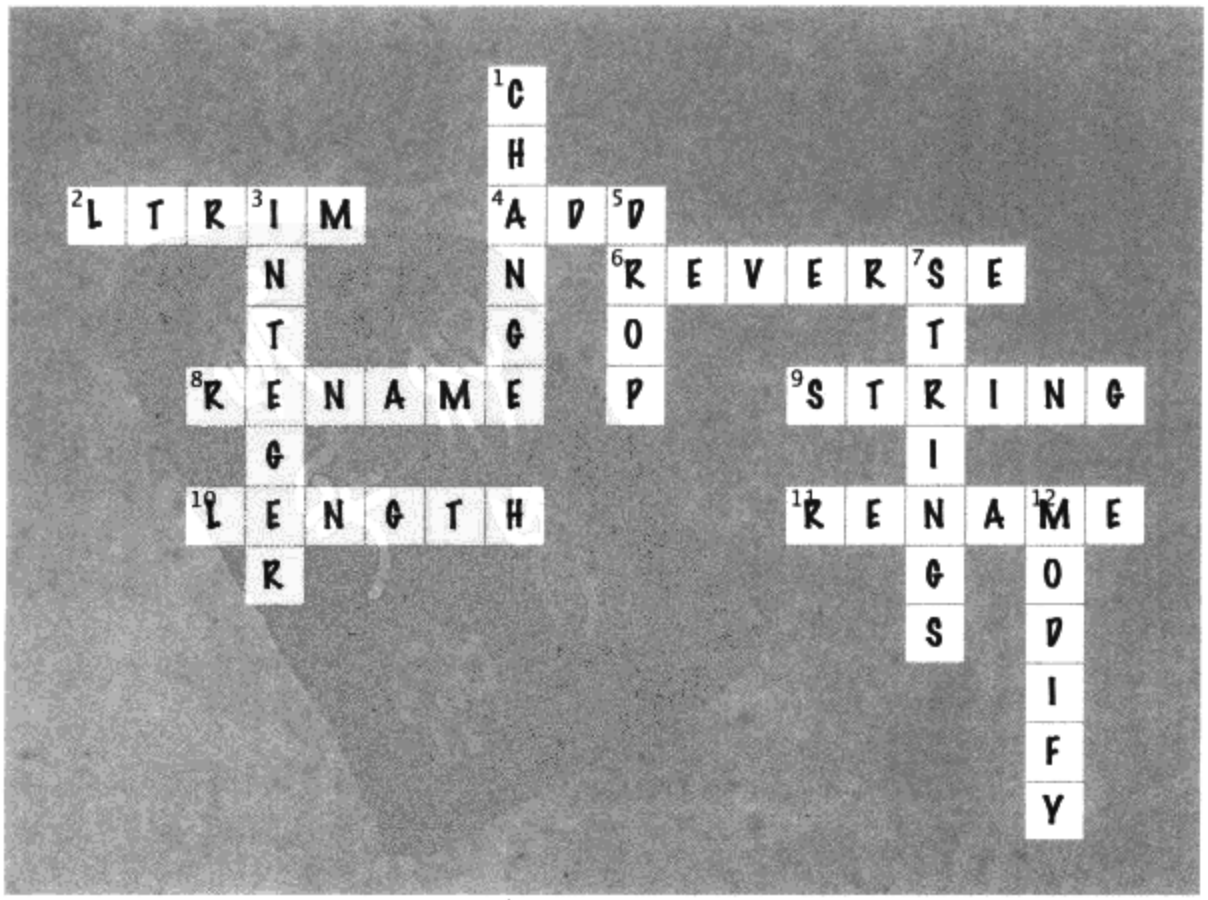
画出运行完第210页上的命令后的表。

project_list

原本的“number”列变成proj_id，而且新列还具有自动递增的主键值。

proj_id	descriptionofproj	contractoronjob
1	outside house painting	Murphy
2	kitchen remodel	Valdez
3	wood floor installation	Keller
4	roofing	Jackson

ALTER填字游戏解答



以新视角看你的数据

现在该为SQL工具包添加一些功能了。我们已经知道如何用SELECT和WHERE子句选出数据，但有时候我们需要比SELECT加上WHERE子句更**精确**的选取工具。在本章中，我们将学习如何给数据**排序**和**归组**，还会学习如何对查询结果套用**数学运算**。

Dataville Video影片出租店要改装升级

Dataville Video的老板实在不太懂得整理的他的商品。在他目前的系统中，可能因为上架的员工不一样，使得影片出现在不同架子上。老板最近刚订购了新的储藏架，而且打算利用这个机会好好地制定每部影片的分类。

To: Dataville Video 全体员工
From: 老板
Subject: 新储藏架意味着要做新分类啦！！

大家好：
新储藏架快送来了，所以我希望大家能一起让影片库存更有组织。我们可以使用下列分类：

动作冒险类（Action & Adventure）
剧情类（Drama）
喜剧类（Comedy）
家庭类（Family）
恐怖惊悚类（Horror）
科幻奇趣类（SciFi & Fantasy）
其他类（Misc）

至于该怎样让现有的标签方式符合上述分类，就交给各位了。

大家好好研究一下吧！

老板

现有系统是以 True 或 False 为影片分类的，对于如何判断影片的新分类没有太大帮助。例如一部同时在SciFi和Comedy类中标为“T”的影片，它究竟应该被归为哪一类？

“T”和“F”分别是True和False的缩写。

店里购入影片拷贝的日期。

movie_table

movie_id	title	rating	drama	comedy	action	gore	scifi	for_kids	cartoon	purchased
1	Monsters, Inc.	G	F	T	F	F	F	T	T	3-6-2002
2	The Godfather	R	F	F	T	T	F	F	F	2-5-2001
3	Gone with the Wind	G	T	F	F	F	F	F	F	11-20-1999
4	American Pie	R	F	T	F	F	F	F	F	4-19-2003
5	Nightmare on Elm Street	R	F	F	T	T	F	F	F	4-19-2003
6	Casablanca	PG	T	F	F	F	F	F	F	2-5-2001

这些列都是为了我们回答顾客关于电影内容的提问而设计的。

当前表的问题

这些是 Dataville Video 的当前表的缺点。

顾客归还影片时我们不知道影片原本的位置。

如果影片在表中多个列中的记录都为 T，那真的不太容易判断影片的上架分类。所以应该为影片指定**单一分类**。

大家都不太清楚影片的内容。

顾客有时会在喜剧区发现很恐怖的影片封面。目前T/F值的地位都一样，归类时无法判断哪个列应该优先考虑。

添加T/F数据很花时间，而且常常出错。

每次进新片时都必须插入所有列的 T/F 值，列越多，越容易填错。分类列应该有助于我们检查这些 T/F 列，而最后我们可以摆脱旧的分类方法。

我们需要一个分类列来加快分类上架的速度、帮助顾客了解他们选择的影片的类型，并减少店里的资料的错误。

现有的列该如何对应到新的**分类**呢？有任何电影符合一种或多种分类吗？

比对现有数据

我们已经知道如何使用ALTER添加新的影片分类列，但添加实际分类可能有点麻烦。幸好，表中的现有内容可以帮助我们识别出每部影片的分类，我们不用看过每部影片再决定。

用很简单的句子重新写出每部影片的关系吧：

如果影片的 **drama** 列是' T ' 则把它的 category 列设为 'drama'

如果影片的 **comedy** 列是' T ' 则把它的 category 列设为 'comedy'

如果影片的 **action** 列是' T ' 则把它的 category 列设为' action '

如果影片的 **gore** 列是' T ' 则把它的 category 列设为' horror '

如果影片的 **scifi** 列是' T ' 则把它的 category 列设为' scifi '

如果影片的 **for_kids** 列是' T ' 则把它的 category 列设为' family '

如果影片的 **cartoon** 列是' T ' 而且 **rating** 列是' G ' 则把它的 category 列设为' family '

如果影片的 **cartoon** 列是' T ' 但 **rating** 列不是' G ' 则把它的 category 列设为' misc '

有些卡通片不一定是给小孩看的，rating 列有助于判断影片是否适合全家观赏。如果它的值是G，就应该可以归类为 "family"；但如果不是，则归类为 "misc"。

产生新列

现在可以把前页的短句变成 SQL 的 UPDATE 语句：

```
UPDATE movie_table SET category = 'drama' where drama = 'T';
UPDATE movie_table SET category = 'comedy' where comedy = 'T';
UPDATE movie_table SET category = 'action' where action = 'T';
UPDATE movie_table SET category = 'horror' where gore = 'T';
UPDATE movie_table SET category = 'scifi' where scifi = 'T';
UPDATE movie_table SET category = 'family' where for_kids = 'T';
UPDATE movie_table SET category = 'family' where cartoon = 'T' AND rating = 'G';
UPDATE movie_table SET category = 'misc' where cartoon = 'T' AND rating <> 'G';
```

rating 的值不为 'G'。

磨笔上阵

请填入下列影片的分类值。

movie _ table

title	rating	drama	comedy	action	gore	scifi	for_ kids	cartoon	category
Big Adventure	G	F	F	F	F	F	T	F	
Greg: The Untold Story	PG	F	F	T	F	F	F	F	
Mad Clowns	R	F	F	F	T	F	F	F	
Paraskavedekatriaphobia	R	T	T	T	F	T	F	F	
Rat named Darcy, A	G	F	F	F	F	F	T	F	
End of the Line	R	T	F	F	T	T	F	T	
Shiny Things, The	PG	T	F	F	F	F	F	F	
Take it Back	R	F	T	F	F	F	F	F	
Shark Bait	G	F	F	F	F	F	T	F	
Angry Pirate	PG	F	T	F	F	F	F	T	

我们判断每个T/F列的顺序是否重要？ .....

请填入下列影片的分类值。

movie_table

title	rating	drama	comedy	action	gore	scifi	for_kids	cartoon	category
Big Adventure	G	F	F	F	F	F	T	F	family
Greg: The Untold Story	PG	F	F	T	F	F	F	F	action
Mad Clowns	R	F	F	F	T	F	F	F	horror
Paraskavedekatriaphobia	R	T	T	T	F	T	F	F	?
Rat named Darcy, A	G	F	F	F	F	F	T	F	family
End of the Line	R	T	F	F	T	T	F	T	misc
Shiny Things, The	PG	T	F	F	F	F	F	F	drama
Take it Back	R	F	T	F	F	F	F	F	comedy
Shark Bait	G	F	F	F	F	F	T	F	?
Angry Pirate	PG	F	T	F	F	F	F	T	misc
Potentially Habitable Planet	PG	F	T	F	F	T	F	F	?

问号表示这部影片会被多个 UPDATE 改变，它的分类值将根据 UPDATE 执行的顺序而改变。

我们判断每个 T/F 列的顺序是否重要？ 是的，确实重要。

顺序确实重要

以“Paraskavedekatriaphobia”为例（译注1），它最终将被归为 scifi类，尽管影片内容或许更接近喜剧。我们不确定它究竟该属于comedy、action、drama、cartoon、scifi 中的哪一类。既然不清楚放为哪一类，或许放在 misc 类中最为保险。

顺序很重要。
两个UPDATE语
句可能修改了
相同列的值。

▶ 译注1：paraskavedekatriaphobia 或 paraskevidekatriaphobia，13号星期五恐惧症。

如果数据较少的话，这似乎还不会构成问题，但如果表中有几百列呢？有没有结合成一个超大型 UPDATE 语句的方式呢？

嗯，是可以合成超大型UPDATE，但我们有更好的工具。

利用CASE检查现有列的值和条件，就可以结合所有UPDATE语句。如果现有列的值符合条件，我们才会在新列中填入指定的值。

CASE甚至能告诉RDBMS，**如果没有记录符合条件时该如何处理。**

以下是基本语法：

这里指定的列值将根据下列条件做适当的更新。

```
UPDATE my_table
SET new_column =
CASE
  WHEN column1 = somevalue1
    THEN newvalue1
  WHEN column2 = somevalue2
    THEN newvalue2
  ELSE newvalue3
END;
```

CASE 表达式由此开始。

WHEN：当符合这个条件时……

THEN：然后把 new_column 值设为此处指定的值。

当符合另一个条件时……

然后把 new_column 值设为此处指定的另一个值。

CASE 表达式和整段 UPDATE 语句在此结束（因为它后面有分号）。

任何不符合上述条件的记录都会被改为这个值。

缩排对运算没有影响，只是让我们更容易理解程序代码。

使用CASE表达式来UPDATE

让我们把CASE表达式套用到movie_table上。

```
UPDATE movie_table
SET category =
CASE
  WHEN drama = 'T' THEN 'drama'
  WHEN comedy = 'T' THEN 'comedy'
  WHEN action = 'T' THEN 'action'
  WHEN gore = 'T' THEN 'horror'
  WHEN scifi = 'T' THEN 'scifi'
  WHEN for_kids = 'T' THEN 'family'
  WHEN cartoon = 'T' THEN 'family'
  ELSE 'misc'
END;
```

这部分和 "UPDATE movie_table SET category ='drama' WHERE drama ='T'" 相同，但是要打的字可就少多了！

凡是不符合上述条件的数据都会被归类为 "misc"。

原本对新列套用 UPDATE 时还未知的数据现在都有分类值了。

不过，请注意我们如何为 "Angry Pirate" 和 "End of the Line" 的分类列填入新值的。

movie_table

title	rating	drama	comedy	action	gore	scifi	for_kids	cartoon	category
Big Adventure	PG	F	F	F	F	F	F	T	family
Greg: The Untold Story	PG	F	F	T	F	F	F	F	action
Mad Clowns	R	F	F	F	T	F	F	F	horror
Paraskavedekatriaphobia	R	T	T	T	F	T	F	F	drama
Rat named Darcy, A	G	F	F	F	F	F	T	F	family
End of the Line	R	T	F	F	T	T	F	T	drama
Shiny Things, The	PG	T	F	F	F	F	F	F	drama
Take it Back	R	F	T	F	F	F	F	F	comedy
Shark Bait	G	F	F	F	F	F	T	F	family
Angry Pirate	PG	F	T	F	F	F	F	T	comedy
Potentially Habitable Planet	PG	F	T	F	F	T	F	F	comedy

在CASE检查每部影片的T/F值时，RDBMS会寻找第一个出现的' T '，
以此设定每部影片的分类。

以下是 SQL 代码检查 “Big Adventure” 列的方式：

```
UPDATE movie_table
SET category =
CASE
  WHEN drama = 'T' THEN 'drama'
  WHEN comedy = 'T' THEN 'comedy'
  WHEN action = 'T' THEN 'action'
  WHEN gore = 'T' THEN 'horror'
  WHEN scifi = 'T' THEN 'scifi'
  WHEN for_kids = 'T' THEN 'family'
  WHEN cartoon = 'T' THEN 'family'
  ELSE 'misc'
END;
```

FALSE：还不能分类
FALSE：还不能分类
FALSE：还不能分类
FALSE：还不能分类
FALSE：还不能分类
FALSE：还不能分类
TRUE：分类值设为 'family'，直接跳到 END 的地方并退出代码。

再看看影片符合多种分类的情况。此时，RDBMS软件还是会找出第一个出现的' T '并据以设定分类。

以下是 SQL 代码检查“Paraskavedekatriaphobia”列的方式：

```
UPDATE movie_table
SET category =
CASE
  WHEN drama = 'T' THEN 'drama'
  WHEN comedy = 'T' THEN 'comedy'
  WHEN action = 'T' THEN 'action'
  WHEN gore = 'T' THEN 'horror'
  WHEN scifi = 'T' THEN 'scifi'
  WHEN for_kids = 'T' THEN 'family'
  WHEN cartoon = 'T' THEN 'family'
  ELSE 'misc'
END;
```

TRUE：分类值设为 'drama'，直接跳到 END并退出代码。其他 T 值都被忽略了。

看来我们遇到问题了

我们可能遇到问题了。《Great Adventure》是部R级（限制级）的卡通片，结果它却出现在family类中。

留言板

日期 今天　　时间 13：41

给 老板

当你外出时

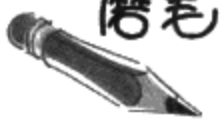
非常生气的顾客

来电	✓	请回电	✓
Called to see you		Will call again	
Wants to see you		Returned your call	

留言 有位小姐来电抱怨：她的小孩Nathan租到一部充斥着脏话的卡通片，结果他现在老是追着他昧昧讲 %#!@

记录者 我　　紧急事项 ✓

磨笔上阵



请修改CASE表达式，让卡通片归到misc类中，而不是family类。如果一部卡通片属于G级，就将它归到family类。

动动脑

我们该如何利用“rating值为R”这一点来防止同样的抱怨再度出现呢？

磨笔上阵解答

请修改CASE表达式，让卡通片归到misc类，而不是family类。如果一部卡通片属于G级，就将它归到family类。

```
UPDATE movie_table
SET category =
CASE
  WHEN drama = 'T' THEN 'drama'
  WHEN comedy = 'T' THEN 'comedy'
  WHEN action = 'T' THEN 'action'
  WHEN gore = 'T' THEN 'horror'
  WHEN scifi = 'T' THEN 'scifi'
  WHEN for_kids = 'T' THEN 'family'
  WHEN cartoon = 'T' AND rating = 'G' THEN 'family'
  ELSE 'misc'
END;
```

一个条件表达式可以包含许多部分：在WHEN子句中加上AND，检查影片是否既为cartoon又为'G'级。如果两项都符合就归为'family'类。

没有蠢问题

问： 需要用到 ELSE 吗？

答： 看个人选择。如果不需要，也可以不加ELSE，不过如果有 ELSE 子句，就算完全不符合其他条件，也会更新列。有分类值总比没有值或有 NULL 好吧。

问： 如果没有 ELSE而且列也不符合任何一个 WHEN 条件，会发生什么事？

答： 在你想更新的列里不会发生任何改变。

问： 如果我只想对部分列套用 CASE 表达式，应该怎么做呢？例如，只想对部分符合 category = ' misc '的套用 CASE，可以加上 WHERE 吗？

答： 是的，可以在关键字END后加上 WHERE 子句。这样，CASE 就只会套用在符合 WHERE 条件的列上。

问： CASE 表达式可以搭配 UPDATE 以外的语句吗？

答： 可以。CASE 表达式可以搭配 SELECT、INSERT、DELET，当然还包括这里提到的 UPDATE。

打造CASE

你的老板总是三心二意，他决定要稍微改变一下。请研究他的 E-mail，并设计一条能够达成老板要求的 SQL语句。

To: Dataville Video 全体员工
From: 老板
Subject: 新储藏架要做新分类啦!

大家好：

我决定添加几个新的储藏架。我想，限制级（R）影片不应该和普遍级（G）或辅导级（PG）的放在同一个架子上。所以让我们增加5个新分类：

horror-r
action-r
drama-r
comedy-r
scifi-r

另外，misc类中如果有G级的影片，请移到family类中。

谢谢大家。
老板

结果，新的分类方式反而让顾客很难寻找到影片。请写下删除刚才创建的那些 R 级影片分类的语句。

最后，删除所有只记录 T/F 的列，我们已经不再需要它们了。

打造CASE

你的老板总是三心二意，他决定要稍微改变一下。请研究他的 E-mail，并设计一条达成老板要求的 SQL 语句。

```
UPDATE movie_table
SET category =
CASE
  WHEN drama = 'T' AND rating = 'R' THEN 'drama-r'
  WHEN comedy = 'T' AND rating = 'R' THEN 'comedy-r'
  WHEN action = 'T' AND rating = 'R' THEN 'action-r'
  WHEN gore = 'T' AND rating = 'R' THEN 'horror-r'
  WHEN scifi = 'T' AND rating = 'R' THEN 'scifi-r'
  WHEN category = 'misc' AND rating = 'G' THEN 'family'
END;
```

结果，新的分类方式反而让顾客很难寻找影片。请写下删除刚才创建的那些 R 级影片分类的语句。

```
UPDATE movie_table
SET category =
CASE
  WHEN category = 'drama-r' THEN 'drama'
  WHEN category = 'comedy-r' THEN 'comedy'
  WHEN category = 'action-r' THEN 'action'
  WHEN category = 'horror-r' THEN 'horror'
  WHEN category = 'scifi-r' THEN 'scifi'
END;
```

最后，删除所有只记录 T/F 的列，我们已经不再需要它们了。

```
ALTER TABLE movie_table
DROP COLUMN drama,
DROP COLUMN comedy,
DROP COLUMN action,
DROP COLUMN gore,
DROP COLUMN scifi,
DROP COLUMN for_kids,
DROP COLUMN cartoon;
```

To: Dataville Video 全体员工
From: 老板
Subject: 新储藏架要做新分类啦！

大家好：

我决定添加几个新的储藏架。我想，限制级（R）影片不应该和普遍级（G）或辅导级（PG）的放在同一个架子上。所以让我们增加5个新分类：

horror-r
action-r
drama-r
comedy-r
scifi-r

另外，misc类中如果有G级的影片，请移到family类中。

谢谢大家。
老板

表可能会变得乱七八糟

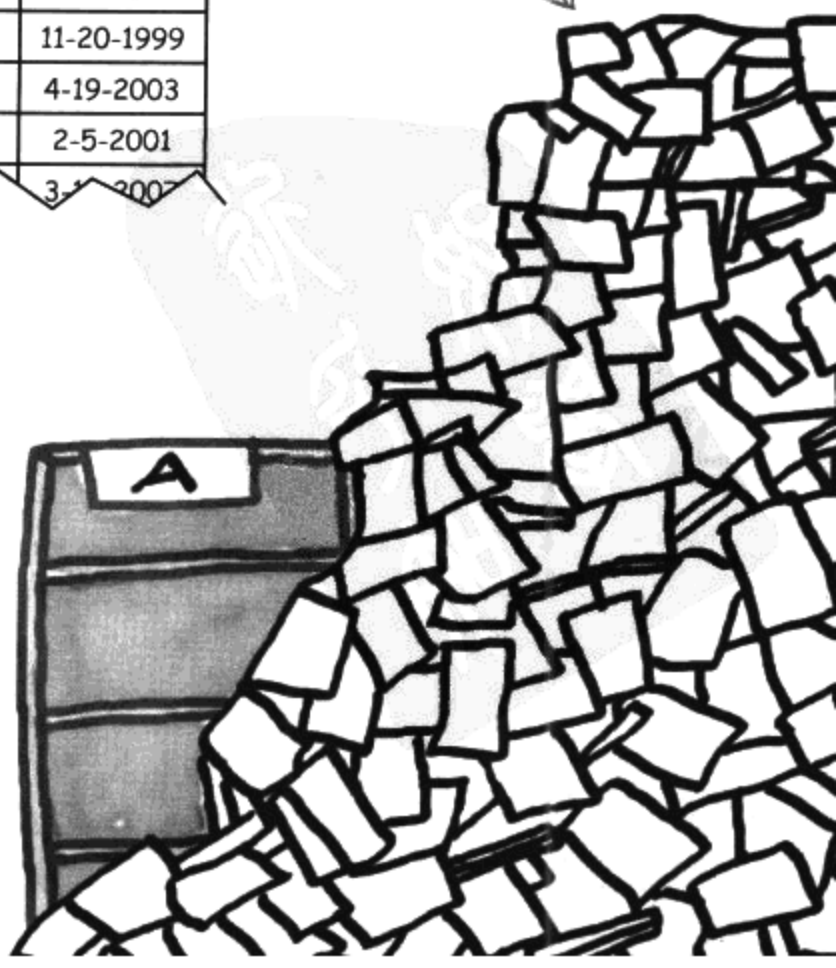
当影片到达店里时，我们把它的资料插入表中，它就成为表中最新的一条记录。表中的影片毫无次序可言。而现在要重新排列影片，我们也就面临了一些问题。我们知道新储藏架的每一层可放20部影片，而店中的3,000多部影片全都要贴上分类标签。**我们需要根据分类选出影片，并按字母顺序排列分类中的影片。**

我们知道如何查询数据库以找出某个分类中的所有影片，但我们还需要把各分类中的影片按字母顺序排列。

movie_table

movie_id	title	rating	category	purchased
83	Big Adventure	G	family	3-6-2002
84	Greg: The Untold Story	PG	action	2-5-2001
85	Mad Clowns	R	horror	11-20-1999
86	Paraskavedekatriaphobia	R	action	4-19-2003
87	Rat named Darcy, A	G	family	4-19-2003
88	End of the Line	R	misc	2-5-2001
89	Shiny Things, The	PG	drama	3-6-2002
90	Take it Back	R	comedy	2-5-2001
91	Shark Bait	G	misc	11-20-1999
92	Angry Pirate	PG	misc	4-19-2003
93	Potentially Habitable Planet	PG	scifi	2-5-2001
9	Cows Gone Wild	R	[illegible]	3-[illegible]-2002

这里只是Dataville Video店中3,000多部影片的冰山一角。



脑力锻炼

如何只用一条 SQL 语句就让这些数据按字母顺序组织？

我们需要一种方式来组织我们SELECT出的数据

Dataville Video店中的3,000多部影片的每一部都必须贴上影片所属分类的标签，接下来还要依字母顺序上架。

我们需要一份按影片分类排序、分类下再按影片标题字母排序的总清单。目前，我们已经知道如何 SELECT。我们可以轻松地按分类选出影片，甚至可以选出某个分类下以某个字母开头的影片。

但是若想组织这份庞大的清单，我们至少要写182次SELECT语句。下面只是部分这样的语句：

```
SELECT title, category FROM movie_table WHERE title LIKE 'A%' AND category = 'family';
SELECT title, category FROM movie_table WHERE title LIKE 'B%' AND category = 'family';
SELECT title, category FROM movie_table WHERE title LIKE 'C%' AND category = 'family';
SELECT title, category FROM movie_table WHERE title LIKE 'D%' AND category = 'family';
SELECT title, category FROM movie_table WHERE title LIKE 'E%' AND category = 'family';
SELECT title, category FROM movie_table WHERE title LIKE 'F%' AND category = 'family';
SELECT title, category FROM movie_table WHERE title LIKE 'G%' AND category = 'family';
```

我们需要知道影片标题（title），才能从仓库中找出影片；我们还需要知道影片分类（category），才能帮影片贴好标签并放上储藏架。

这是影片标题的首字母。

而这是我们寻找的影片分类。

SELECT语句要写182次的原因是有7个影片分类和26个字母。其中尚未包括以数字开头的影片标题（例如“101 Dalmatians”或“2001: A Space Odyssey”）。

动动脑

你觉得以数字或非字母字符（例如感叹号）开头的影片标题会出现在清单的什么地方？

磨笔上阵

虽然选出了某个分类中标题以“A”开头的影片，但我们还是要进一步按照字母顺序手动排序。

请仔细研究下表中的影片，它只是182次选取操作中某一次的查询结果。请大家依照字母顺序排列影片清单。

```
SELECT title, category FROM movie_table WHERE title LIKE 'A%' AND category = 'family';
```

一点点查询结果

title	category
Airplanes and Helicopters	family
Are You Paying Attention?	family
Acting Up	family
Are You My Mother?	family
Andy Sighs	family
After the Clowns Leave	family
Art for Kids	family
Animal Adventure	family
Animal Crackerz	family
Another March of the Penguins	family
Anyone Can Grow Up	family
Aaargh!	family
Aardvarks Gone Wild	family
Alaska: Land of Salmon	family
Angels	family
Ann Eats Worms	family
Awesome Adventure	family
Annoying Adults	family
Alex Needs a Bath	family
Aaargh! 2	family

虽然选出了某个分类中标题以“A”开头的影片，但我们还是要进一步按照字母顺序手动排序。

请仔细研究下表中的影片，它只是182次选取操作中某一次的查询结果。请大家依照字母顺序排列影片清单。

```
SELECT title, category FROM movie_table WHERE title LIKE 'A%' AND category = 'family';
```

title	category
Aaargh!	family
Aaargh! 2	family
Aardvarks Gone Wild	family
Acting Up	family
After the Clowns Leave	family
Airplanes and Helicopters	family
Alaska: Land of Salmon	family
Alex Needs a Bath	family
Andy Sighs	family
Angels	family
Animal Adventure	family
Animal Crackerz	family
Ann Eats Worms	family
Annoying Adults	family
Another March of the Penguins	family
Anyone Can Grow Up	family
Are You My Mother?	family
Are You Paying Attention?	family
Art for Kids	family
Awesome Adventure	family

你花了多少时间来为这20部影片完成排序？

你可以想象3,000多部影片要花多少时间才能完成手动排序吗？

以“Are You...”开头的两部影片标题出现在清单的后半段，因为 r 接在 A 后面。不过，接下来要比对标题中的第7个字母才能判断出两部影片的上架顺序。

有点秩序吧：ORDER BY

想要让查询结果有点秩序？刚好，我们可以要求SQL在SELECT时，根据某个列的排序（ORDER）所返回的查询结果。

这部分已经很熟悉了，和前面提过的 SELECT 查询一模一样。

这部分是新东西。就像它的字面意义，ORDER BY要求程序依照title的字母顺序返回数据。

```
SELECT title, category
FROM movie_table
WHERE
title LIKE 'A%'
AND
category = 'family'
ORDER BY title;
```

说真的，你该不会告诉我，这就是按字母排序的唯一方式吧？难道没办法一口气排列片名以26个字母开头的影片的顺序吗？

磨笔上阵

你的怀疑很有道理。上述查询应该如何修改，才能让查询和排序的威力更强大呢？

.....

.....

.....

.....

不要翻页！在翻页前务必要完成这道习题。

按单列排序

如果查询中使用了ORDER BY title，那自然不需另外选取某个特定的开头字母，查询会自动依照标题的字母顺序来排列查询结果。

所以只需拿掉title LIKE这部分，ORDER BY title自然会为我们完成剩余的工作。

ORDER BY能按任何列的字母顺序来排列查询结果。

磨笔上阵解答

应该如何修改，才能让查询和排序的威力更强大呢？

```
SELECT title, category
FROM movie_table
WHERE
title LIKE 'A%'
AND
category = 'family'
ORDER BY title;
```

```
SELECT title, category
FROM movie_table
WHERE
category = 'family'
ORDER BY title;
```

这次会选出所有family类中的影片清单。

更好的是，这份清单会包括以数字开头的影片标题，它们会出现在清单的最前面。

这里还不是查询结果的尾端，我们没有列出全部影片的空间。查询结果会从数字开始，一路排到标题以字母 Z 开头的影片。

请注意，最前面的几个标题是以数字开头的。

title	category
1 Crazy Alien	family
10 Big Bugs	family
101 Alsatians	family
13th Birthday Magic	family
2 + 2 is 5	family
3001 Ways to Fall	family
5th Grade Girls are Evil	family
7 Year Twitch	family
8 Arms are Better than 2	family
Aaargh!	family
Aaargh! 2	family
Aardvarks Gone Wild	family
Acting Up	family
After the Clowns Leave	family
Airplanes and Helicopters	family
Alaska: Land of Salmon	family
Alex Needs a Bath	family
Andy Sighs	family
Angels	family
Animal Adventure	family
Animal Crackerz	family
Ann Eats Worms	family
Annoying Adults	family
Another March of the Penguins	family
Anyone Can Grow Up	family
Are You My Mother?	family
Are You Paying Attention	family
Art for Kids	family
Awesome Adventure	family

创建一张简单的表，表内只有一列，名为“test_chars”，存储类型为CHAR(1)。

为表插入如下所示的数字、字母（包括大小写）及非字母字符，每个字符一行。有一行要输入空白字符，另外有一行保持 NULL。

接着请试用 ORDER BY 查询这一列，并在“SQL 的排序规则”中填空。

```
0123ABCDabcd!@#$%^&*()-_
+=[]{};:'"\|`~,.<>/?
```

SQL 的排序规则

运行 ORDER BY 查询后，参考查询结果的排序，填写下列空格。

非字母字符出现在数字的 _____。

数字出现在字母的 _____。

NULL 出现在数字的 _____。

NULL 出现在字母的 _____。

大写字母出现在小写字母的 _____。

“A 1”出现在“A1”的 _____。

SQL 的排序规则

运行 ORDER BY 查询后，请把下列字符按出现顺序排列。

```
+ = ! ( & ~ "
* @ ? '
```

还记得如何插入单引号吗？它们真是难缠的小东西。

创建一张简单的表，表内只有一列，名为“test_chars”，存储类型为CHAR(1)。

为表插入如下所示的数字、字母（包括大小写）及非字母字符，每个字符一行。有一行要输入空白字符，另外有一行保持 NULL。

接着请试用 ORDER BY 查询这一列，并在“SQL 的排序规则”中填空。

```
 !"#$%&'()*+,-./0123:;<=>
?@ABCD[\]^_`abcd{|}~
```

查询结果中可能呈现的字符顺序。请注意查询结果起始处有个空白字符。每个人的查询顺序，可能会因为RDBMS软件不同而有差异。这个练习的重点是让你知道确实有顺序，以及各种RDBMS采用的顺序。

SQL 的排序规则

运行 ORDER BY 查询后，参考查询结果的排序，填写下列空格。

非字母字符出现在数字的前面或后面。

数字出现在字母的前面。

NULL 出现在数字的前面。

NULL 出现在字母的前面。

大写字母出现在小写字母的前面。

“A 1”出现在“A1”的前面。

SQL 的排序规则

运行 ORDER BY 查询后，请把下列字符按出现顺序排列。

+ = ! (& ~ "
* @ ? '

! " & ' (* + = ? @ ~

按两列排序

看起来一切都回到掌控中了。我们可按字母顺序排列影片，也可以为每个分类创建按字母排序的清单。

很不幸地，老板又发出了一封电子邮件，说……

To: Dataville Video 全体员工
From: 老板
Subject: 旧片出局！

大家好：

我认为有些上架时间很长的老片应该淘汰了。大家能够利用这个周末，按日期列出各分类的影片清单吗？

希望大家能完成这么棒的工作，

老板

幸运的是，我们可以在一条语句内利用多个列进行排序。

我们希望影片的购买日期一起列入查询结果中。

```
SELECT title, category, purchased
FROM movie_table
ORDER BY category, purchased;
```

这是排序用的第一列。我们要取得店中的每部影片并按分类排序。

这则是排序用的第二列，在依据category列排序完后才会再依第二列排序。

脑力锻炼

每个分类中最老旧的影片会出现在该分类的查询结果的最前面还是最后面呢？如果某个分类中有两部影片的购买日期相同，又会发生什么事呢？哪部影片会先出现呢？

按多列排序

查询结果的排序依据不局限于使用一列或两列。我们可以利用所有需要的列来排序所有结果。

请观察下列使用三个列排序的ORDER BY子句。以下是发生的事情以及表的排序方式。

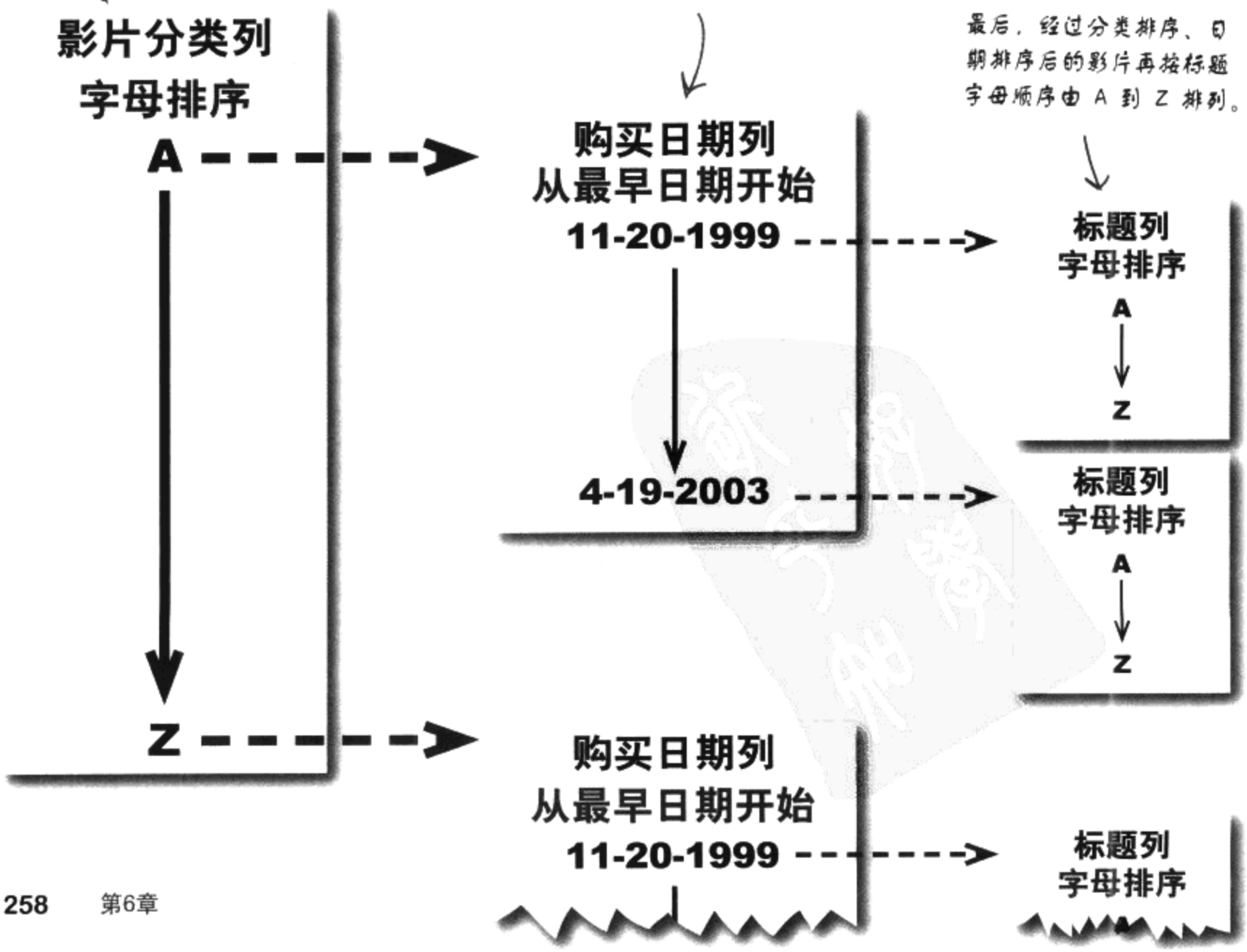
对于作为排序依据的列，你可以依需求尽量多的使用。

```
SELECT * FROM movie_table
ORDER BY category, purchased, title;
```

首先，查询结果按分类排序，因为ORDER BY后指定的第一列是category。查询结果由A到Z依序排列。

接下来，各分类中的查询结果再按日期排序（前一个步骤中已经按字母顺序，让分类由A到Z排列）。这个步骤进一步把各分类中的影片按年、月、日的顺序，由最早购买日期开始排列。

最后，经过分类排序、日期排序后的影片再按标题字母顺序由 A 到 Z 排列。



有秩序的movie_table

让我们观察这个SELECT语句作用于电影表后返回的结果。

我们的原始movie_table表

表中没有真正的顺序，影片按被插入表的顺序排列。

movie_id	title	rating	category	purchased
83	Bobby' s Adventure	G	family	3-6-2002
84	Greg: The Untold Story	PG	action	2-5-2001
85	Mad Clowns	R	horror	11-20-1999
86	Paraskavedekatriaphobia	R	action	4-19-2003
87	Rat named Darcy, A	G	family	4-19-2003
88	End of the Line	R	misc	2-5-2001
89	Shiny Things, The	PG	drama	3-6-2002
90	Take it Back	R	comedy	2-5-2001
91	Shark Bait	G	misc	11-20-1999
92	Angry Pirate	PG	misc	4-19-2003
93	Potentially Habitable Planet	PG	scifi	2-5-2001
9	C...s Go... Wil	R	ho...	2...-20...

查询后的有序结果：

第三个用于排序的列

第一个用于排序的列

第二个用于排序的列

movie_id	title	rating	category	purchased
84	Greg: The Untold Story	PG	action	2-5-2001
86	Paraskavedekatriaphobia	R	action	4-19-2003
90	Take it Back	R	comedy	2-5-2001
89	Shiny Things, The	PG	drama	3-6-2002
83	Bobby' s Adventure	G	family	3-6-2002
87	Rat named Darcy, A	G	family	4-19-2003
85	Mad Clowns	R	horror	11-20-1999
91	Shark Bait	G	misc	11-20-1999
88	End of the Line	R	misc	2-5-2001
93	Potentially Habitable Planet	PG	scifi	2-5-2001

SQL 有个反转顺序的关键字。

默认情况下，SQL根据ORDER BY指定的列中的升序排列查询结果。也就是从A到Z，从1到99,999。如果你希望排列顺序相反，希望数据以降序排列，可以在列名后加上关键字 DESC。

问： 我记得DESC是“表说明”（DESCRIPTION）的意思。你确定它能用在ORDER子句中？

答： 我确定。一切都跟上下文有关。当DESC用在表名前，例如`DESC movie_table;`，查询结果就是表的说明，此时DESC是DESCRIBE的缩写。

但DESC出现在ORDER子句中时，它则代表DESCENDING（降序），是一种排序方式。

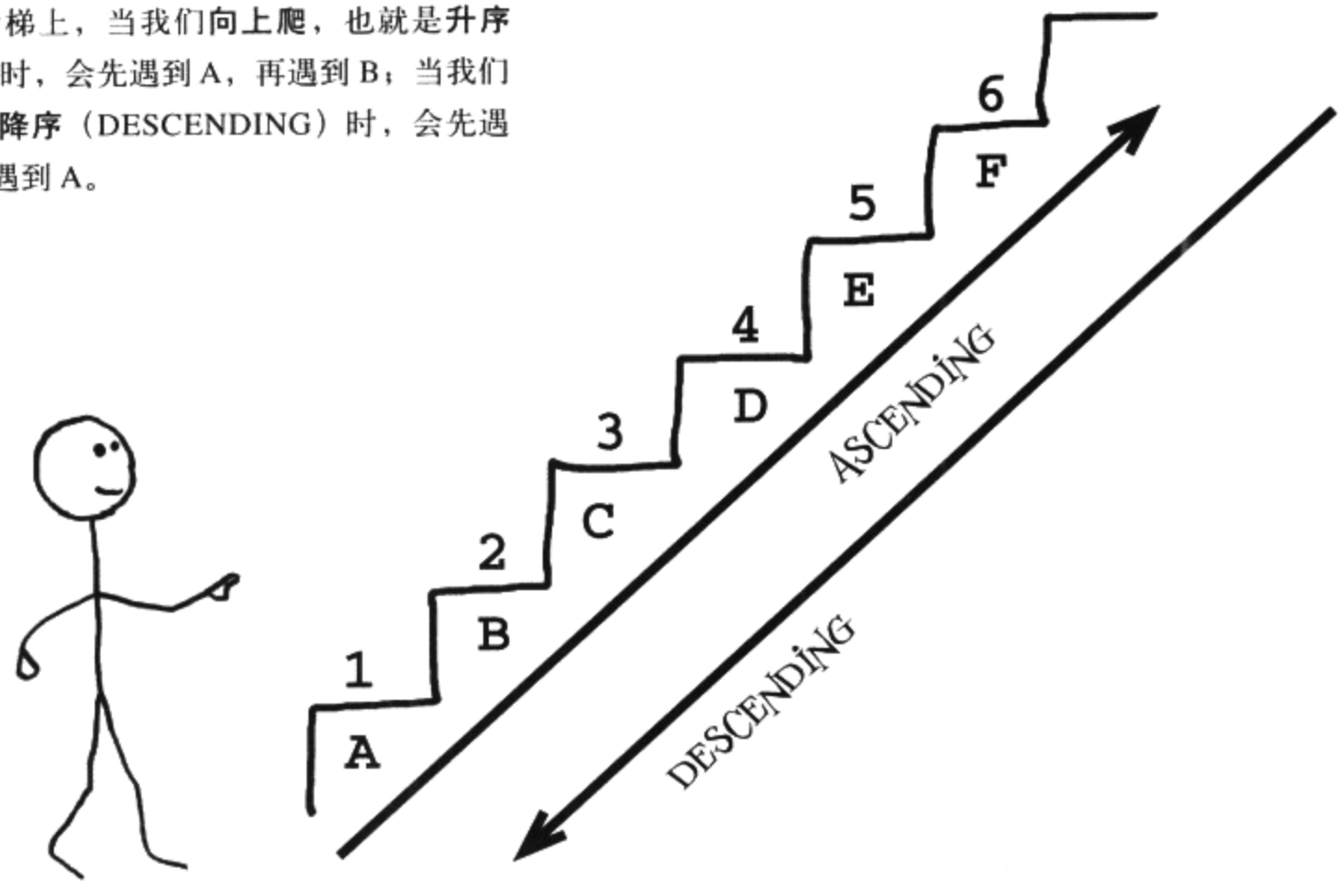
问： 我可以使用完整的DESCRIBE或DESCENDING，以免混淆吗？

答： 可以使用DESCRIBE，但没有DESCENDING这个关键字。

关键字DESC应位于ORDER BY子句中的列名后，用来反转查询结果的顺序。

以DESC反转排序

数据就像排在阶梯上，当我们**向上爬**，也就是**升序**（ASCENDING）时，会先遇到A，再遇到B；当我们**向下走**，也就是**降序**（DESCENDING）时，会先遇到Z，然后才会遇到A。



下列的查询会选出一份影片清单，以**最新的**购买日期开始排序。至于同一天买入的影片，则按字母顺序排列。

```
SELECT title, purchased
FROM movie_table
ORDER BY title ASC, purchased DESC;
```

这里可以加上关键字ASC，但并非必要。ASC已是默认的排序方式。

如果需要从Z到A或从9到1的排序方式，则必须使用关键字DESC。

To: Dataville Video 全体员工
From: 老板
Subject:免费招待!

大家好：

店面焕然一新，看起来真是太棒了。感谢大家同心协力把影片放到正确的地方，还有，幸好 SQL 查询中有神奇的 ORDER BY 子句，现在大家都能找到自己想找的影片了。

为了感谢大家辛苦地工作，我举办了一个小小的比萨庆功宴，今晚6点开始，在我家举行。

别忘了把报告带来!

老板

P.S.别穿得太不方便运动，我一直想把我家的书柜整理一下……

Girl Sprout®的饼干销售冠军问题

镇上的 Girl Sprout 的领队想找出哪个女孩卖出了最多的饼干。她有一份女孩们每天的销售量表。

销售饼干的 Girl Sprout 女孩姓名

cookie_sales

销售收入

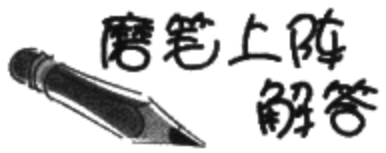
销售日期

ID	first_name	sales	sale_date
1	Lindsay	32.02	3-6-2007
2	Paris	26.53	3-6-2007
3	Britney	11.25	3-6-2007
4	Nicole	18.96	3-6-2007
5	Lindsay	9.16	3-7-2007
6	Paris	1.52	3-7-2007
7	Britney	43.21	3-7-2007
8	Nicole	8.05	3-7-2007
9	Lindsay	17.62	3-8-2007
10	Paris	24.19	3-8-2007
11	Britney	3.40	3-8-2007
12	Nicole	15.21	3-8-2007
13	Lindsay	0	3-9-2007
14	Paris	31.99	3-9-2007
15	Britney	2.58	3-9-2007
16	Nicole	0	3-9-2007
17	Lindsay	2.34	3-10-2007
18	Paris	13.44	3-10-2007
19	Britney	8.78	3-10-2007
20	Nicole	26.82	3-10-2007
21	Lindsay	3.71	3-11-2007
22	Paris	.56	3-11-2007
23	Britney	34.19	3-11-2007
24	Nicole	7.77	3-11-2007
25	Lindsay	16.23	3-12-2007
26	Paris	0	3-12-2007
27	Britney	4.50	3-12-2007
28	Nicole	19.22	3-12-2007

磨笔上阵

销售量最高的 Girl Sprout 女孩将获得一套免费的马术课程。每个女孩都希望赢得这项奖励，所以Edwina一定要找出正确的得奖者，以免奖励出现反效果。

请使用你的 ORDER BY 子句设计一个可以协助 Edwina 找出得奖者的查询。



销售量最高的 Girl Sprout 女孩将获得一套免费马术课程。每个女孩都希望赢得这项奖励，所以 Edwina 一定要找出正确的得奖者，以免奖励出现反效果。

请使用你的 ORDER BY 子句设计一个可以协助 Edwina 找出得奖者的查询。

```
SELECT first_name, sales
FROM cookie_sales
ORDER BY first_name;
```

这是我们的查询……

first_name	sales
Nicole	19.22
Nicole	0.00
Nicole	8.05
Nicole	26.82
Nicole	7.77
Nicole	15.21
Nicole	18.96
Britney	3.40
Britney	2.58
Britney	4.50
Britney	11.25
Britney	8.78
Britney	43.21
Britney	34.19
Lindsay	17.62
Lindsay	9.16
Lindsay	0.00
Lindsay	32.02
Lindsay	2.34
Lindsay	3.71
Lindsay	16.23
Paris	26.53
Paris	0.00
Paris	0.56
Paris	1.52
Paris	13.44
Paris	24.19
Paris	31.99

……这是查询结果。

96.03

107.91

81.08

98.23

每个女孩的销售量还需要手动加总后才能找出正确的赢家。

SUM能为我们加总

数据还真不少，我们千万不能出错，不然小女孩的怒火可是非常可怕的。与其冒着手动计算出错的危险，不如把这项工作交给 SQL。

SQL 语言中有些特殊关键字，称为函数（function）。函数是一段代码，可对数据值执行操作。我们要示范的第一个函数就是对整列执行数学运算。SUM可把括号里指定的列值全部加总。现在就来观察它的实际运作情况。

SUM函数会把sales列的值加总。

SUM是个函数。也就是说，它会对括号中出现的列采取一些行动。

```
SELECT SUM(sales)
FROM cookie_sales
WHERE first_name = 'Nicole';
```

WHERE子句限定了加总的范围，本例只限于加总Nicole的销售量。若不限定，你将得到整个列的总和。

```
File Edit Window Help TheWinnerIs
> SELECT SUM(sales) FROM cookie_sales
-> WHERE first_name = 'Nicole';
+-------------+
| SUM(sales)  |
+-------------+
|       96.03 |
+-------------+
1 row in set (0.00 sec)
```

现在还需要其他三个人的销售总和，一切就完成了。但如果能用一个查询就完成，不是更完美吗……

回家试试看

自己试着创建一个像 cookie_sales 的表并插入一些浮点数值，然后用它尝试后续几页的查询。

利用GROUP BY完成分组加总

有个同时加总（SUM）每个女孩的销售量的方式：在 SUM 语句中加上 GROUP BY。在本例中，它会根据女孩的姓名分组，再对各个组进行加总。

```
SELECT first_name, SUM(sales)
FROM cookie_sales
GROUP BY first_name
ORDER BY SUM(sales) DESC;
```

加总 sales 列的所有数据。

根据 first_name 值分组。

我们希望查询结果能由高至低排序，这样我才能轻易地看出优胜者。

必须根据加总所用的列排序。

上述语句根据名字分组，然后分别加总每个人的销售量。

first_name	sales
Nicole	19.22
Nicole	0.00
Nicole	8.05
Nicole	26.82
Nicole	7.77
Nicole	15.21
Nicole	18.96

first_name	sales
Paris	26.53
Paris	0.00
Paris	0.56
Paris	1.52
Paris	13.44
Paris	24.19
Paris	31.99

first_name	sales
Lindsay	17.62
Lindsay	9.16
Lindsay	0.00
Lindsay	32.02
Lindsay	2.34
Lindsay	3.71
Lindsay	16.23

first_name	sales
Britney	3.40
Britney	2.58
Britney	4.50
Britney	11.25
Britney	8.78
Britney	43.21
Britney	34.19

```
File Edit Window Help TheWinnerReallyIs
> SELECT first_name, SUM(sales)
-> FROM cookie_sales GROUP BY first_name
-> ORDER BY SUM(sales);
+------------+------------+
| first_name | sum(sales) |
+------------+------------+
| Britney    |     107.91 |
| Paris      |      98.23 |
| Nicole     |      96.03 |
| Lindsay    |      81.08 |
+------------+------------+
4 rows in set (0.00 sec)
```

优胜者是 Britney！

AVG搭配GROUP BY

没办法去骑马的女孩都很失望，所以 Edwina 决定为平均每日销量最高的女孩增加一个奖项。她使用了 AVG 函数。

每个女孩都有7天的销售时间，所以AVG函数会分别把销售量加总后再除以 7。

同样地，我们再次根据 first_name 分组……

……但这次改为计算平均值。

```
SELECT first _ name, AVG(sales)
FROM cookie _ sales
GROUP BY first _ name;
```

AVG先把组里的所有值加总，再以值的数量的分总值来求得平均值。

first_name	sales
Nicole	19.22
Nicole	0.00
Nicole	8.05
Nicole	26.82
Nicole	7.77
Nicole	15.21
Nicole	18.96

first_name	sales
Paris	26.53
Paris	0.00
Paris	0.56
Paris	1.52
Paris	13.44
Paris	24.19
Paris	31.99

first_name	sales
Lindsay	17.62
Lindsay	9.16
Lindsay	0.00
Lindsay	32.02
Lindsay	2.34
Lindsay	3.71
Lindsay	16.23

first_name	sales
Britney	3.40
Britney	2.58
Britney	4.50
Britney	11.25
Britney	8.78
Britney	43.21
Britney	34.19

糟糕，又是 Britney 获奖了。我们得想办法找出另一位赢家。

```
File Edit Window Help TheWinnerReallyIs
> SELECT  first _ name, AVG(sales)
-> FROM cookie _ sales GROUP BY first _ name;
+------------+------------+
| first _ name | AVG(sales) |
+------------+------------+
| Nicole     |  13.718571 |
| Britney    |  15.415714 |
| Lindsay    |  11.582857 |
| Paris      |  14.032857 |
+------------+------------+
4 rows in set (0.00 sec)
```

MIN和MAX

为了抓住每个机会，Edwina 很快地检查了表中的最大（MAX）值和最小（MIN）值，确认其他女孩是否曾有某天的销售量超越了 Britney，或者 Britney 曾有某天的销售量低于其他人……

我们可以使用MAX函数来寻找列里的最大值，用MIN函数则可找出列中的最小值。

MAX返回每个女孩的最高销售量。

```
SELECT first_name, MAX(sales)
FROM cookie_sales
GROUP BY first_name;
```

哎呀，Britney 还是拿到了单日销售量的冠军。

first_name	sales
Nicole	26.82
Britney	43.21
Lindsay	32.02
Paris	31.99

MIN返回每个女孩的最低销售量。

```
SELECT first_name, MIN(sales)
FROM cookie_sales
GROUP BY first_name;
```

你看，其他女孩都曾空手回家，但 Britney 表现最差的一天也还是卖出饼干了。

first_name	sales
Nicole	0.00
Britney	2.58
Lindsay	0.00
Paris	0.00

这下子我该认真起来了。或许我可以找出销售天数比其他人都多的女孩？

COUNT，计算天数

为了找出销售天数比其他人都多的女孩，Edwina 采用了 COUNT 函数来计算大家销售饼干的天数。COUNT 将返回**指定列中的行数**。

```
SELECT COUNT(sale _ date)
FROM cookie _ sales;
```

COUNT返回sales_date列中的行数。如果数据值是NULL，则不纳入计算。

磨笔上阵

cookie _ sales

ID	first_name	sales	sale_date
1	Lindsay	32.02	3-6-2007
2	Paris	26.53	3-6-2007
3	Britney	11.25	3-6-2007
4	Nicole	18.96	3-6-2007
5	Lindsay	9.16	3-7-2007
6	Paris	1.52	3-7-2007
7	Britney	43.21	3-7-2007
8	Nicole	8.05	3-7-2007
9	Lindsay	17.62	3-8-2007
10	Paris	24.19	3-8-2007
11	Britney	3.40	3-8-2007
12	Nicole	15.21	3-8-2007
13	Lindsay	0	3-9-2007
14	Paris	31.99	3-9-2007
15	Britney	2.58	3-9-2007
16	Nicole	0	3-9-2007
17	Lindsay	2.34	3-10-2007
18	Paris	13.44	3-10-2007
19	Britney	8.78	3-10-2007
20	Nicole	26.82	3-10-2007
21	Lindsay	3.71	3-11-2007
22	Paris	.56	3-11-2007
23	Britney	34.19	3-11-2007
24	Nicole	7.77	3-11-2007
25	Lindsay	16.23	3-12-2007
26	Paris	0	3-12-2007
27	Britney	4.50	3-12-2007
28	Nicole	19.22	3-12-2007

左边是原始表。你觉得查询会返回什么结果？

返回结果代表了实际卖出饼干的天数吗？

设计一个查询以返回每个女孩卖出饼干的天数。

磨笔上阵解答

左边是原始表。你觉得查询会返回什么结果？

答：算出 28 个销售日。

返回结果代表了实际卖出饼干的天数吗？

答：无法代表。返回结果只是*sale_date*中有记录的天数。

设计一个查询以返回每个女孩卖出饼干的天数。

```
SELECT first_name, COUNT(sale_date)
FROM cookie_sales
GROUP BY first_name;
```

嗯……不是这样哦。在最早和最晚日期间，我们无法确认其间每一天都销售了饼干。

不过的确有找出销售出饼干的确切天数的简便方式，那就是使用关键字DISTINCT。我们不仅可以利用DISTINCT计算出所需的 COUNT数量，还能取得没有重复的日期列表。

选出与众不同的值

让我们先单独观察关键字 DISTINCT 的作用，不要和 COUNT 函数搭配。

因为DISTINCT是个关键字，而非函数，所以不需要为sale_date加上括号。

```
SELECT DISTINCT sale_date
FROM cookie_sales
ORDER BY sale_date;
```

这里加上的 ORDER BY 可以让我们看出最早和最晚日期。

```
File Edit Window Help NoDupes
> SELECT DISTINCT sale_date
FROM cookie_sales
 -> ORDER BY sale_date;
+-------------+
| sale_date   |
+-------------+
| 2007-03-06  |
| 2007-03-07  |
| 2007-03-08  |
| 2007-03-09  |
| 2007-03-10  |
| 2007-03-11  |
| 2007-03-12  |
+-------------+
7 rows in set (0.00 sec)
```

请看，每个列出的日期都没有重复！

现在，让DISTINCT 和 COUNT 函数搭配使用：

请注意，DISTINCT和sale_date一同出现在括号中。

```
SELECT COUNT(DISTINCT sale_date)
FROM cookie_sales;
```

我们不需要 ORDER BY，因为 COUNT 只会返回一个值，不需要排序。

脑力锻炼

尝试这个查询，然后用它找出卖出饼干的天数最多的女孩。

答：Britney

有一群 SQL 函数和关键字经过精心打扮，一同参加“猜猜我是谁”化装舞会。它们会提示自己的身份，我们则要根据提示内容猜出它们的身份。所有关键字都会说实话。请在下面左边的空格中填上关键字或函数的身份，在右边的空格中注明它是关键字还是函数。

今晚的贵宾：

COUNT、DISTINCT、AVG、MIN、GROUP BY、SUM、MAX

贵宾名称	关键字或函数
使用我而得到的结果或许不会很高贵。	....
我交出来的结果会比我收入的大。	....
我会给你独一无二的结果。	....
我会说出数据的数量。	....
如果想求总和，你会需要我。	....
我只对大的数字有兴趣。	....
你问我好不好？普普通通，中等啦！	....

答案请见第 279 页。

问： **既然利用 AVG、MAX、MIN 等函数查找最大值，为什么不直接加上 ORDER BY 子句？**

答： 确实可以，而且这也是非常好的做法。不过本章是故意忽略了 ORDER BY。我不希望混淆查询的原始结果，而且也让你比较容易学习新函数。回头审视本章的函数，并想象加入ORDER BY后的查询会如何改变查询结果。

问： **关键字DISTINCT好像非常有用。我可以套用在任何列上吗？**

答： 可以。尤其在许多记录的某一列均为相同值时特别有用，还有，如果你只是想看看存储了哪些值，而不需要一长串重复的记录时，DISTINCT也很好用。

问： **使用MIN()的查询似乎没办法帮Edwina找出第二个赢家，不是吗？**

答： 的确没办法，但可以帮她找出表现最差的女孩。明年再办这个活动时，她可以特别鼓励成绩较落后的孩子。

问： **讲到 MIN，如果查询的列中有 NULL，这会有什么影响吗？**

答： 好问题。NULL 其实不会有影响。本章讲到的函数都不会返回 NULL，因为 NULL 代表此处无值，而不是此值为零。

动动脑

假设我们不只面对4位女孩，而是40位Girl Sprouts，我们应该如何用 SUM 找出第二名？

LIMIT查询结果的数量

Edwina决定用SUM判断第二名。让我们再度回头审视原始查询及结果，以便想出寻找第二名的方法。

```
SELECT first _ name, SUM(sales)
FROM cookie _ sales
GROUP BY first _ name
ORDER BY SUM(sales)DESC;
```

这里使用的 ORDER BY 非常重要，否则我们的查询结果不会有任何顺序。

first_name	sales
Britney	107.91
Paris	98.23
Nicole	96.03
Lindsay	81.08

其实我们只想知道最前面两条结果。

Paris 是第二名！Nicole 已经拒绝跟她说话了。

因为本例查询只会得到4条结果，当然很容易判断第二名。但如果需要更高的精确度，我们可用LIMIT限制查询结果的数量，像本例即可只呈现最前面两名女孩。套用LIMIT可以精确指定要从结果集返回的记录数量。

```
SELECT first _ name, SUM(sales)
FROM cookie _ sales
GROUP BY first _ name
ORDER BY SUM(sales)DESC
LIMIT 2;
```

这里的 LIMIT 指定呈现的结果为两行。

写了这么长的查询，最后只取得两条记录。

first_name	sales
Britney	107.91
Paris	98.23

虽然本例只有 4 名女孩参与竞赛，从中选出两名实在不算很难，但我们要把眼光放到大局上。假设电台里有一份 1,000 首热门歌曲的播放清单，但你还想进一步过滤到只剩下百首超级热门点播金曲。LIMIT就能只列出最常被点播的 100 首，而我们不会看到剩下的 900 首。

LIMIT，只限第二名出现

LIMIT甚至能直接点出第二名，连第一名都不用出现。只需在使用LIMIT时加上两个参数：

你正在揣测这两个数字的意义吗？正在猜它们会带来什么结果吗？你现在心中的答案可能是错误的。凡是出现两个参数的地方，其使用方式绝对和一个参数时大不相同。

```
LIMIT 0,4
```

这是查询结果的起始处。SQL 从 0 开始计算。

这是返回查询结果的数量。

first_name	sales
Britney	107.91
Paris	98.23
Nicole	96.03
Lindsay	81.08

Britney是0，Paris是1，Nicole是2，Lindsay是3。

还记得百首点播金曲榜吗？以查询第20到第29名的歌曲为例，对LIMIT 加上适当的参数的确有助于产生所需的结果。先按受欢迎程度排列歌曲，然后加上 LIMIT 19, 10。19表示从第20首歌开始列出数据（因为SQL从0开始为数据编号），10 则代表返回 10 条记录。

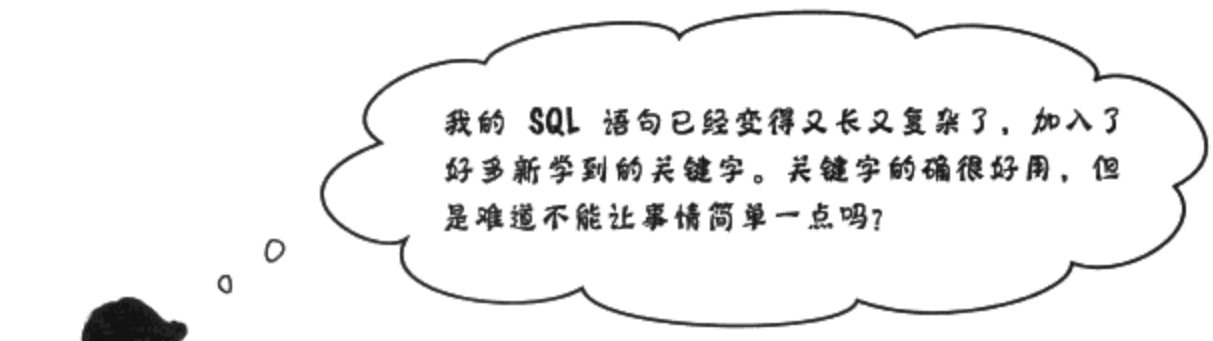
磨笔上阵

请使用 LIMIT 和它的两个参数，写出只返回第二条记录的查询。

请使用 LIMIT 和它的两个参数，写出只返回第二条记录的查询。

```
SELECT first_name, SUM(sales)
FROM cookie_sales
GROUP BY first_name
ORDER BY SUM(sales) DESC
LIMIT 1, 1;
```

请记住，SQL从0开始计数。所以 1 其实代表第二条记录。

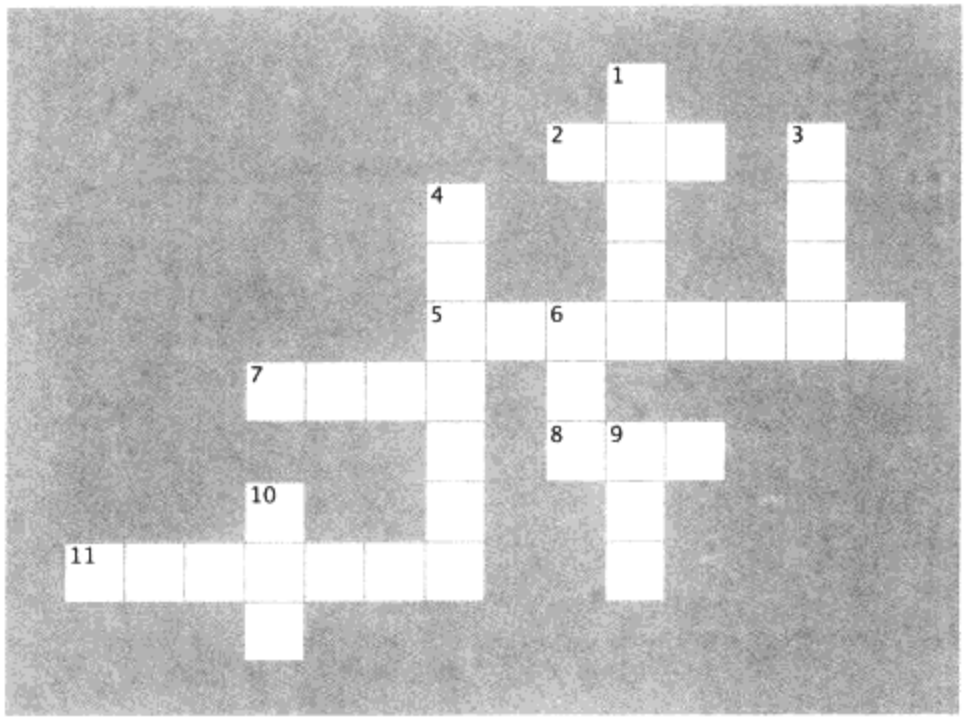


你的查询越变越长是因为你的数据越来越复杂了。

让我们仔细观察你的表，看看它是否过度增长了。接下来将是第 7 章的主题……

SELECT填字游戏

让我们的左脑也运动一下吧！下面是一个典型的填字游戏，所有解答的词汇都曾在本章出现过。



横向

2. 可用这个函数找出某一列中最小的值。

5. 这个函数只返回不重复的值。

7. CASE 中的关键字 ____，可于没有记录符合条件时规定 SQL 的行为。

8. 可用这个函数找出某一列中最大的值。

11. 利用这两个词，可以根据某列的值把相同记录联合成一组。

纵向

1. ____ 可以指定返回记录的数量以及开始返回的位置。

3. 如果利用这个关键字对列排序（ORDER BY），该列中的“9”会比“8”先出现。

4. 利用这两个词，能以指定列为基准，按字母顺序排列查询结果。

6. 这个函数会加总数值列的结果。

9. 如果利用这个关键字进行排序，该列中的“8”会比“9”先出现。

10. 在 SELECT 中使用它，可返回查询结果的总数，而非结果本身。

你的SQL工具包

第6章已经收进你的工具包，有了这些进阶SELECT的函数、关键字和查询，我们也终于进入最省力的状态。如果需要本书工具的完整列表，请参考附录3。

ORDER BY

根据指定的列，按字母顺序排列查询结果。

GROUP BY

根据共用列，把记录分成多个组。

COUNT

我们不需看到记录，就能知道有多少条记录符合 SELECT 查询。COUNT 只返回一个整数值。

DISTINCT

不同的值只会返回一次，返回的结果中没有重复值。

AVG

返回数值列的平均值。

MAX 和 MIN

MAX返回列中的最大值，MIN返回列中的最小值。

SUM

把数值列中的数据加总。

LIMIT

可以明确指定返回记录的数量，以及从哪一条记录开始返回。

你的新工具：进阶SELECT的函数、关键字和查询。

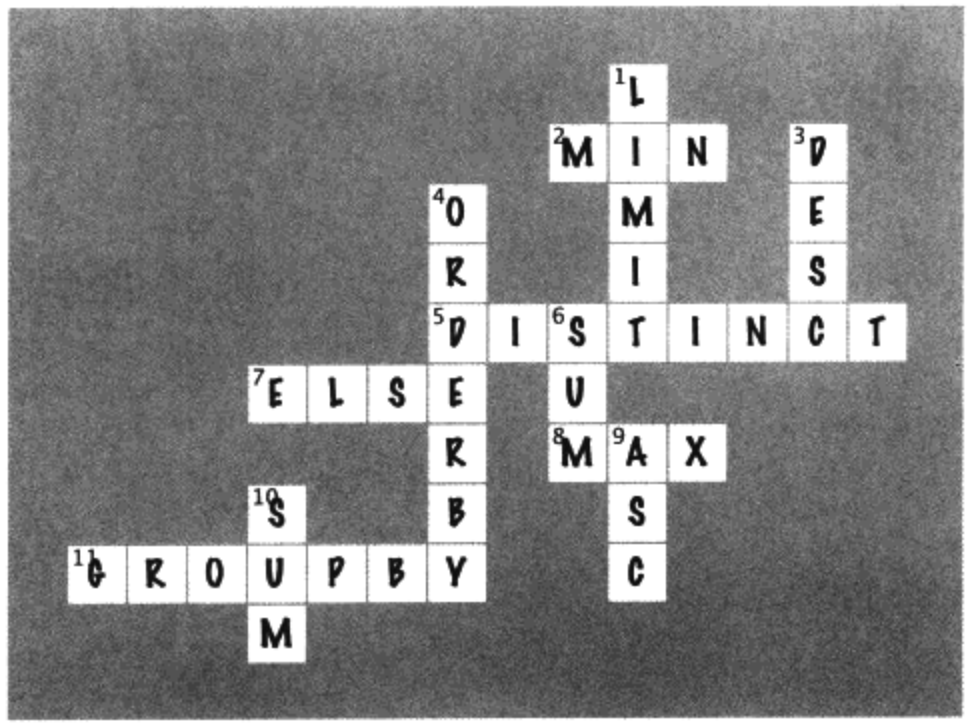
有一群 SQL 函数和关键字经过精心打扮，一同参加“猜猜我是谁”化装舞会。它们会提示自己的身份，我们则要根据提示内容猜出它们的身份。所有关键字都会说实话。请在下面左边的空格中填上关键字或函数的身份，在右边的空格中注明它是关键字还是函数。

今晚的贵宾：

COUNT、DISTINCT、AVG、MIN、GROUP BY、SUM、MAX

贵宾名称	关键字或函数
使用我而得到的结果或许不会很高贵。	MIN	函数
我交出来的结果会比我收入的更大。	SUM	函数
我会给你独一无二的结果。	DISTINCT	关键字
我会说出数据的数量。	COUNT	函数
如果想求总和，你会需要我。	GROUP BY	关键字
我只对大的数字有兴趣。	MAX	函数
你问我好不好啊？普普通通，中等啦！	AVG	函数

SELECT填字游戏解答



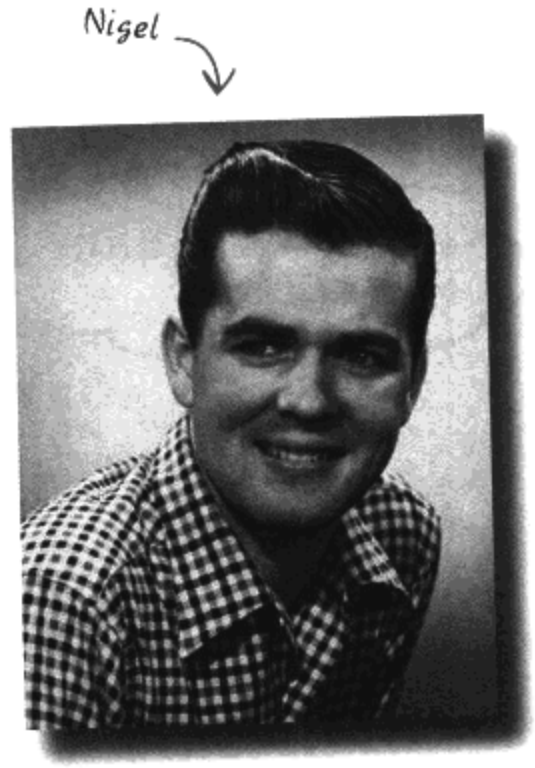
7 多张表的数据库设计

拓展你的表

到了某个时候，只有一张表就不够了。数据变得越来越复杂，你所使用的**唯一一张表**实在装不下了。表里充满了多余的数据，既浪费存储空间，又会拖慢查询的速度。一张表的负荷已经接近极限了，但是外面的世界还很宽广。我们将用**不只一张表**来记录数据、控制数据，最后它将成为你的数据库的主人。

Nigel需要一点爱

Greg有个很寂寞的朋友Nigel，他希望Greg能帮他找到有相同兴趣的女性并安排一次约会。于是Greg调出Nigel的记录。



Nigel 的信息如下：

```
contact _ id: 341
last _ name: Moore
first _ name: Nigel
phone: 5552311111
email: nigelmoore@ranchersrule.com
gender: M
birthday: 1975-08-28
profession: Rancher
city: Austin
state: TX
status: single
interests: animals, horseback riding,
movies
seeking: single F
```

interests栏并不具有单原子性，它包括了多种不同类型的相同信息。Greg担心查询兴趣将是一项大工程。

Greg 把 Nigel 的请求加入待办事项中（TO DO）：

TO DO

为Nigel设计一段查询：设计一段搜寻兴趣的查询。这应该会是一项痛苦的工程，我必须使用LIKE，但这种事情只会发生一次……

为何要改变？

Greg 决定不对 interests 栏做任何改变。他想设计一个比较复杂的查询，因为这种需求或许不会经常出现。

Greg 使用生日（DATE）字段找出与 Nigel 年龄相近的对象（比 Nigel 大或小 5 岁以内）。

磨笔上阵

请完成 Greg 的特制查询，帮 Nigel 找出能够与他共享所有兴趣的约会对象。请注明每一行 SQL 代码的功能。

```
SELECT * FROM my_contacts
WHERE gender = 'F'
AND status = 'single'
AND state='TX'
AND seeking LIKE '%single M%'
AND birthday > '1970-08-28'
AND birthday < '1980-08-28'
AND interests LIKE .....
AND .....
AND .....
```

磨笔上阵解答

请完成 Greg 的特制查询，帮 Nigel 找出能够与他共享所有兴趣的约会对象。请注明每一行 SQL 代码的功能。

从my_contact表中选出所有符合下列条件的记录。

```
SELECT * FROM my_contacts
WHERE gender = 'F'
AND status = 'single'
AND state='TX'
AND seeking LIKE '%single M%'
AND birthday > '1970-08-28'
AND birthday < '1980-08-28'
AND interests LIKE '%animals%'
AND interests LIKE '%horse%'
AND interests LIKE '%movies%';
```

Nigel 希望找到真命天女，所以查找女性……

……而且希望这位小姐仍是单身。

……至少与 Nigel 住在同一州。

还要确定她也打算寻找单身男性。

Nigel 希望见面的对象比他年长不超过 5 岁，比他年轻也不超过 5 岁。

这三项条件会找出兴趣（interests）与 Nigel 相同的对象。其实也可以使用 OR，不过我们希望能完全符合Nigel的兴趣。

查询运作良好

Greg 帮 Nigel 找到完美的对象了：

```
contact_id: 1854
last_name: Fiore
first_name: Carla
phone: 5557894855
email: cfiore@fioreanimalclinic.com
gender: F
birthday: 1974-01-07
profession: Veterinarian
city: Round Rock
state: TX
status: single
interests: horseback riding, movies, animals,
mystery novels, hiking
seeking: single M
```

年龄合适

职业也不错

住的地方也很近

两人的兴趣完全相符！

Carla 与爱马 Trigger

它运作得太好了

Nigel 与 Carla 一拍即合。但是 Greg 却因为这次成功的查询而受害：他的每一位单身朋友都想让他这样查询，而Greg又有非常多的单身朋友。

表应该是为了节省精力而设计。别为了克服设计不良的表而执意改善查询。

一直设计查询实在太浪费时间了。
Greg 在待办事项中加了新的说明。

TO DO

为 Nigel 设计一段查询，设计一段搜寻兴趣的查询。这应该会是一项痛苦的工程，我必须使用LIKE，但这种事情只会发生一次……

以后，忽略 interests 列，查询才会又快又简单。

忽视问题并非解决之道

Regis 也是 Greg 的朋友，也在寻找约会的对象。他希望女方的年龄不比他年长超过 5 岁，也不比他年轻超过 5 岁。他住在马萨诸塞州的剑桥市，他的兴趣跟 Nigel 不太一样。

Greg 决定忽略 `interests` 列，让查询既短又简单。

为 Regis 设计专属查询，但不使用 interests 列。

```
contact_id: 873
last_name: Sullivan
first_name: Regis
phone: 5552311122
email: me@kathieleeisaflake.com
gender: M
birthday: 1955-03-20
profession: Comedian
city: Cambridge
state: MA
status: single
interests: animals, trading cards, geocaching
seeking: single F
```

答案请见第342页。

配对失败率太高

Greg提供了一串很长的名单给 Regis。过了几周，Regis 打电话向Greg抱怨，说那张名单完全没有用处，上面的女性跟他一点共识都没有。

我不能完全忽略interests 列。一定会有更好的解决办法……

TO DO

为Nigel设计一段查询：设计一段搜寻兴趣的查询。这应该是项痛苦的工程，我必须使用LIKE，但这种事情只会发生一次……

以后，忽略interests列，查询才会又快又简单。

只查询第一项兴趣，忽略其他兴趣信息。

所有兴趣都很重要，不该忽略任何一项，所有兴趣都是很有价值的信息。

只采用第一项兴趣

Greg现在知道兴趣不能被忽略了。他假设大家提到兴趣时都是先提到最感兴趣的项目，因此决定只查询第一个兴趣。他的查询仍然十分冗长，但至少比为 interests列的每个项目使用LIKE查询要好点。

磨笔上阵

使用SUBSTRING_INDEX函数来取得interests列的第一个兴趣。

使用SUBSTRING_INDEX函数来取得interests列的第一个兴趣。

SUBSTRING_INDEX(*interests*, ',', 1)

"1"表示寻找的对象为第一个逗号。如果改为"2"，程序将继续往后寻找，直到遇见第二个逗号，然后抓取它前面所有的内容，也就是前两项兴趣。

这部分会从 *interests* 列抓出逗号前的一切内容，也称为子字符串。

它就是命令要寻找的逗号。

因此，Greg设计了新的查询，利用SUBSTRING_INDEX指定第一项兴趣必须为“animals”，想帮 Regis 找出最适合他的真命天女。

```
SELECT * FROM my _ contacts
WHERE gender = 'F'
AND status = 'single'
AND state='MA'
AND seeking LIKE '%single M%'
AND birthday > '1950-08-28'
AND birthday < '1960-08-28'
AND SUBSTRING _ INDEX(interests,',',1) = 'animals';
```

只有第一项兴趣为 "animal" 的女性才会出现在查询结果中。

可能成功的对象

终于，Greg 终于为 Regis 找到匹配的对象了：

```
contact_id: 459
last_name: Ferguson
first_name: Alexis
phone: 5550983476
email: alexangel@yahoo.com
gender: F
birthday: 1956-09-19
profession: Artist
city: Pflugerville
state: MA
status: single
interests: animals
seeking: single M
```

年龄相近

住得离 Regis 颇近

相符的兴趣

错配鸳鸯

Regis 邀请 Alexis 见面小叙，Greg 非常紧张地想知道事情的经过。他已经开始幻想他的my_contacts表将变成一个非常好的社交网络平台。

隔天，Regis 直接找到 Greg 家，而且他非常生气。

“她确实对动物很有兴趣，可是你没告诉我，她还有一项兴趣是制作标本。她家到处都是死掉的动物！”

TO DO

为 Nigel 设计一组查询：设计一段搜寻兴趣的查询。这应该会是一项痛苦的工程，我必须使用 LIKE，但这种事情只会发生一次……

以后，忽略 interests 列。查询才会又快又简单。

只查询第一项兴趣，忽略其他兴趣信息。

创建多个列来存储各项兴趣。因为把所有兴趣都塞在一列中实在很不方便查询。

Regis 的完美对象就在表中，但因为她的兴趣顺序不一样，所以完全找不到。

Greg 决定重新设计他的表。

动动脑

添加多个兴趣列后，Greg 的查询会变成什么样呢？

添加更多兴趣列

Greg发现：只有一个兴趣列会使得查询非常不精确。他必须使用 LIKE 比对各人的兴趣，有时还会出现错配鸳鸯的情况。

因为Greg刚学会用ALTER修改表，还学会了如何拆开文本字符串，他决定创建多个兴趣列并把各项兴趣存入不同的列中。他认为4个兴趣列应该足够了。

使用 ALTER 和 SUBSTRING_INDEX 函数把表修改成有如下列。请把需要的查询都写出来。

```
contact_id
last_name
first_name
phone
email
gender
birthday
profession
city
state
status
interest1
interest2
interest3
interest4
seeking
```

答案请见第 341 页。

一切从头开始

一想到 Regis 与 Alexis 的约会经验，Greg 就觉得很内疚，他想再次尝试。Greg 再次调出 Regis 的记录：

```
contact_id: 872
last_name: Sullivan
first_name: Regis
phone: 5554531122
email: regis@kathieleeisaflake.com
gender: M
birthday: 1955-03-20
profession: Comedian
city: Cambridge
state: MA
status: single
interest1: animals
interest2: trading cards
interest3: geocaching
interest4: NULL
seeking: single F
```

新的表格式，包含了4个兴趣列。

接下来，Greg特别为Regis设计了寻找合适约会的查询。他把一切想到的条件都加进来，希望能比对出最完美的对象。他从简单的列开始，例如gender、status、state、seeking、birthday，最后才挑战新设的4个兴趣列。

请写下他的查询。

接下来，Greg特别为Regis设计了寻找合适约会的查询。他把一切想到的条件都加进来，希望能比对出最完美的对象。他从简单的列开始，例如gender、status、state、seeking、birthday，最后才挑战新设的4个兴趣列。

请写下他的查询。

```
SELECT * FROM my_contacts

WHERE gender = 'F'
AND status = 'single'
AND state= 'MA'
AND seeking LIKE '%single M%'
AND birthday > '1950-03-20'
AND birthday < '1960-03-20'
AND
(
interest1 = 'animals'
OR interest2 = 'animals'
OR interest3 = 'animals'
OR interest4 = 'animals'
)
AND
(
interest1 = 'trading cards'
OR interest2 = 'trading cards'
OR interest3 = 'trading cards'
OR interest4 = 'trading cards'
)
AND
(
interest1 = 'geocaching'
OR interest2 = 'geocaching'
OR interest3 = 'geocaching'
OR interest4 = 'geocaching'
);
```

Regis 的理想对象应该是：单身女性，于1950年至1960年间出生，居住于马萨诸塞，而且在寻找与单身男性见面的机会。

Greg 必须逐一查找每个兴趣列，才能确认女方的兴趣与 Regis 相符，因为相同的兴趣可能出现在两方的任何一个兴趣列中。

Regis 的 interes4 列为 NULL，所以我们只要检查3个兴趣列。

一切都失败了……

添加了4个兴趣列对于解决最初的问题并无帮助，新的表设计并未让查询更为简单。每次修改表都还是无法满足数据原子性的需求。

TO DO

为 Nigel 设计一段查询：设计一段搜寻兴趣的查询。这应该会是一项痛苦的工程，我必须使用 LIKE，但这种事情只会发生一次……

以后，忽略 interests 列。查询才会又快又简单。

只查询单一项兴趣。忽略其他兴趣信息。

创建多个来存储各项兴趣。因为把所有兴趣都塞在一列中实在很不方便查询。

看起来似乎是个不错的解决方案，结果却只让查询更为复杂。

……等一下

我们可以创建一个专门用来存储兴趣的表吗？这样是否会有帮助？

动动脑

添加一个新表会有帮助吗？我们该如何让新表的内容与旧表相连？

跳出一张表的思考框框

显然，局限在目前的这张表内不会有什么好的方案。我们尝试过多种修正数据的方式，甚至还调整了表的结构，却没有任何效果。

现在需要跳出单一表的思考框框了。我们真正需要的是**更多能与现有内容合作的表，能让每个人和他们的兴趣产生关联**。这样，我们的数据就能保持精简整洁。

我们需要把不符合原子性的列移入新的表。

File Edit Window Help MessyTable

```
> DESCRIBE my_contacts;
+-------------+--------------+------+-----+---------+----------------+
| Field       | Type         | Null | Key | Default | Extra          |
+-------------+--------------+------+-----+---------+----------------+
| contact_id  | int(11)      | NO   | PRI | NULL    | auto_increment |
| last_name   | varchar(30)  | YES  |     | NULL    |                |
| first_name  | varchar(20)  | YES  |     | NULL    |                |
| phone       | varchar(10)  | YES  |     | NULL    |                |
| email       | varchar(50)  | YES  |     | NULL    |                |
| gender      | char(1)      | YES  |     | NULL    |                |
| birthday    | date         | YES  |     | NULL    |                |
| profession  | varchar(50)  | YES  |     | NULL    |                |
| city        | varchar(50)  | YES  |     | NULL    |                |
| state       | varchar(2)   | YES  |     | NULL    |                |
| status      | varchar(20)  | YES  |     | NULL    |                |
| interests   | varchar(100) | YES  |     | NULL    |                |
| seeking     | varchar(100) | YES  |     | NULL    |                |
+-------------+--------------+------+-----+---------+----------------+
13 rows in set (0.01 sec) >
```

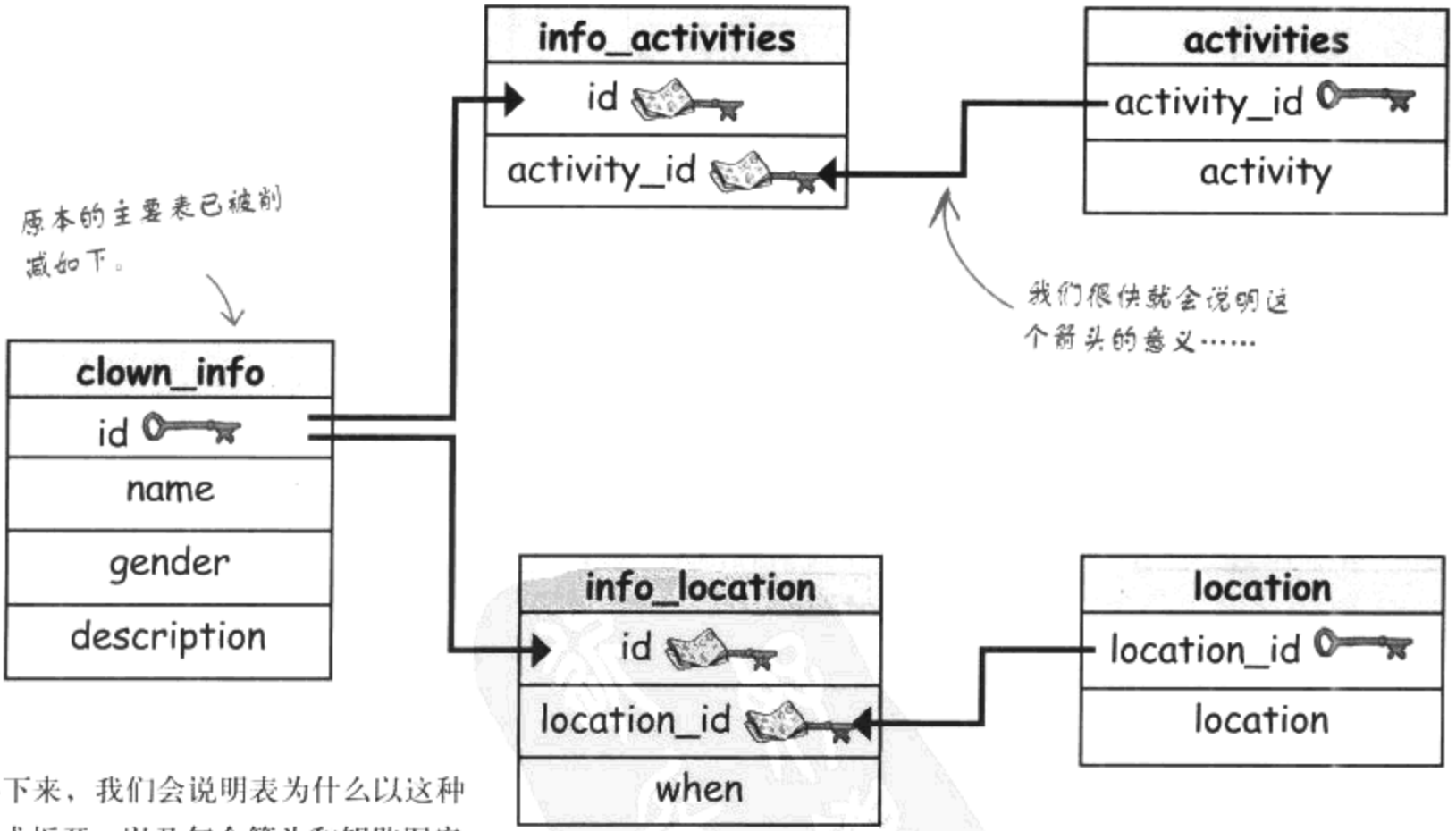
小丑追踪数据库中的多张表

还记得第3章中追踪小丑的表吗？Dataville 的小丑越来越多，所以我们把原本的一张表扩大成更有效果的一组相关的表。

旧的 clown_tracking 表。

clown_tracking

clown_info	name	last_seen	activities
Elsie	Cherry Hill Senior Center	F, red hair, green dress, huge feet	balloons, little car
Pickles	Jack Green's party	M, orange hair, blue suit, huge feet	mime
Snuggles	Ball-Mart	M, yellow shirt, baggy blue pants	horn, umbrella
Mr. Hobo	Fox Gray's Party	M, cigar, black hair, tiny hat	violin



接下来，我们会说明表为什么以这种方式拆开，以及每个箭头和钥匙图案的意义。讲解完后，我们就可以把原则套用在 gregs_list 上。

动动脑

你认为箭头代表什么？钥匙图案的意义呢？

clown_tracking 数据库模式

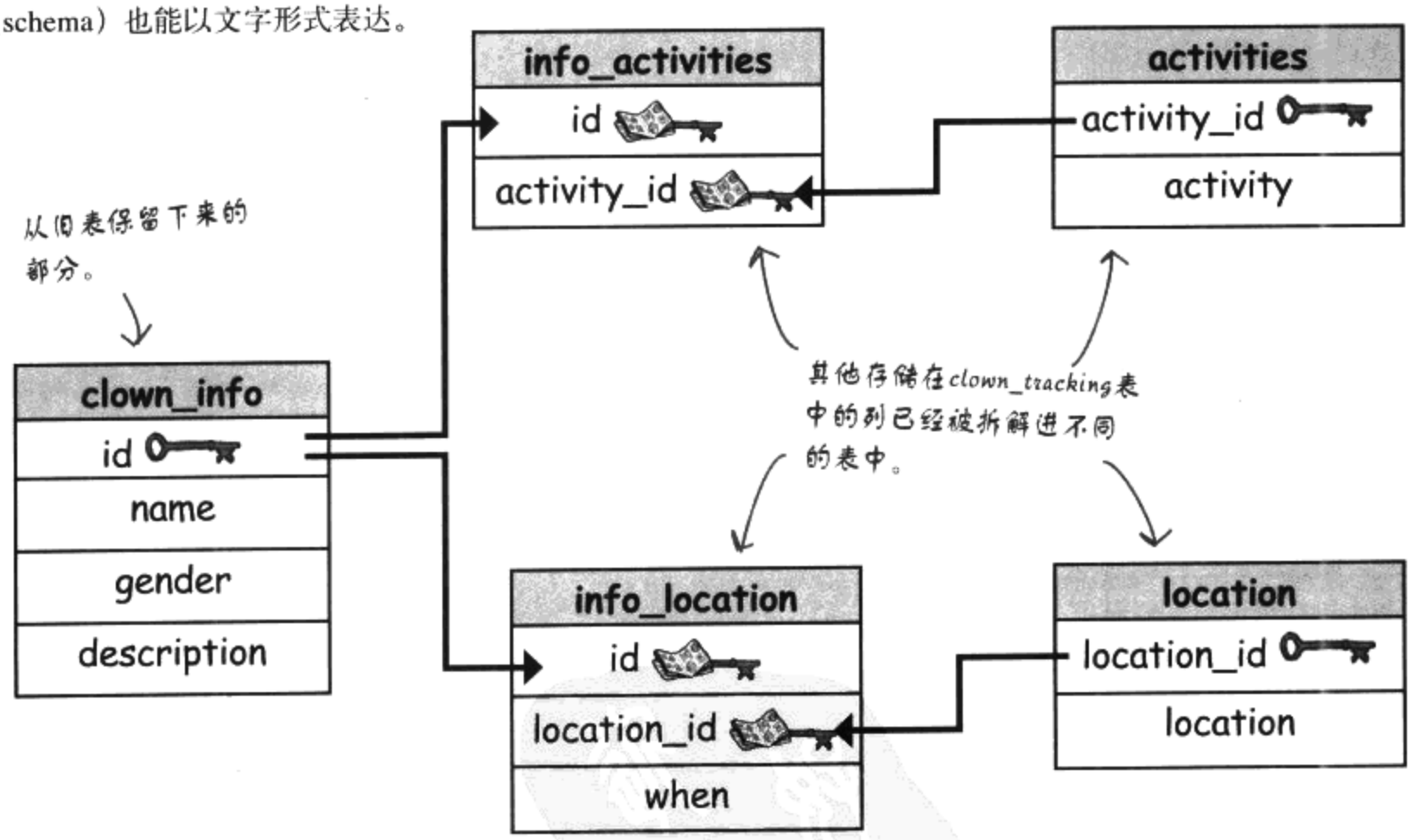
模式（schema）用于表达数据库内的结构，包括表和列，还有各种他们之间相互连接的方式。

创建数据库的视觉解析图，在设计查询时有助于理解数据相连的方式，但模式（schema）也能以文字形式表达。

旧表。

clown _ tracking

clown_info	name	last_seen	activities
Elsie	Cherry Hill Senior Center	F, red hair, green dress, huge feet	balloons, little car
Pickles	Jack Green's party	M, orange hair, blue suit, huge feet	mime
Snuggles	Ball-Mart	F, [illegible] shirt, baggy blue pants	horn, umbrella
Mr. Hobo	Eric Gray's Party	[illegible] hair, tiny hat	violin



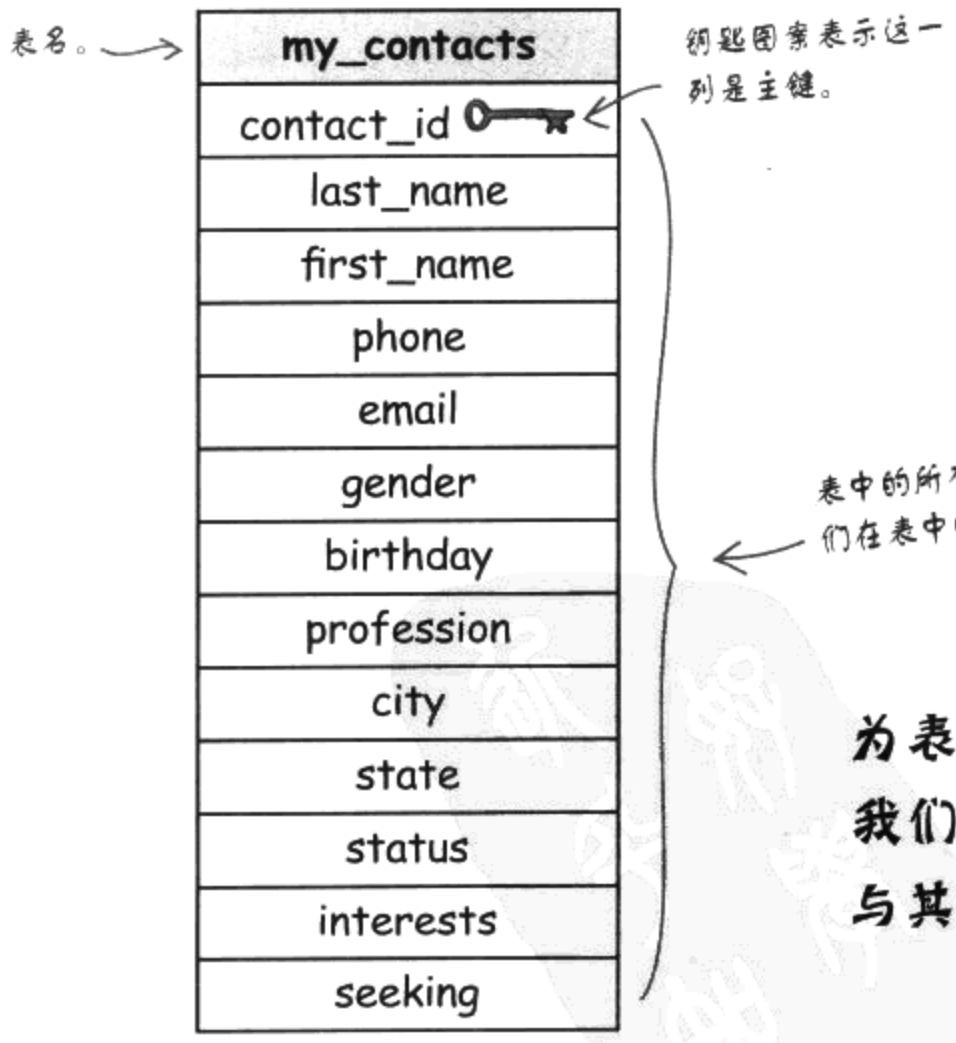
对数据库内的数据描述（列和表），以及任何相关对象和各种连接方式的描述就称为SCHEMA，模式。

表变图表的简单方式

我们已经看到小丑追踪表的转换结果，接着示范如何以相同方式修改 `my_contacts` 表。

到目前为止，每次查看表时，我们都是以最上面的列名和下面的数据来描述，或者在终端窗口中使用 `DESCRIBE` 语句。在只有一张表的情况下，这两种方式就足够了，但在想要创建多张表的图示时，它们就不太实用了。

以下是 `my_contacts` 表的速记方式：

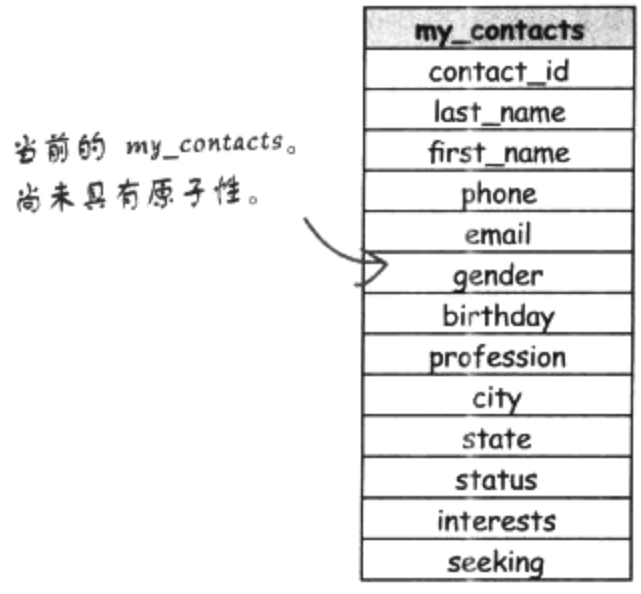


为表创建图表可以协助我们分别看待表的设计与其中的数据。

如何从一张表变成两张

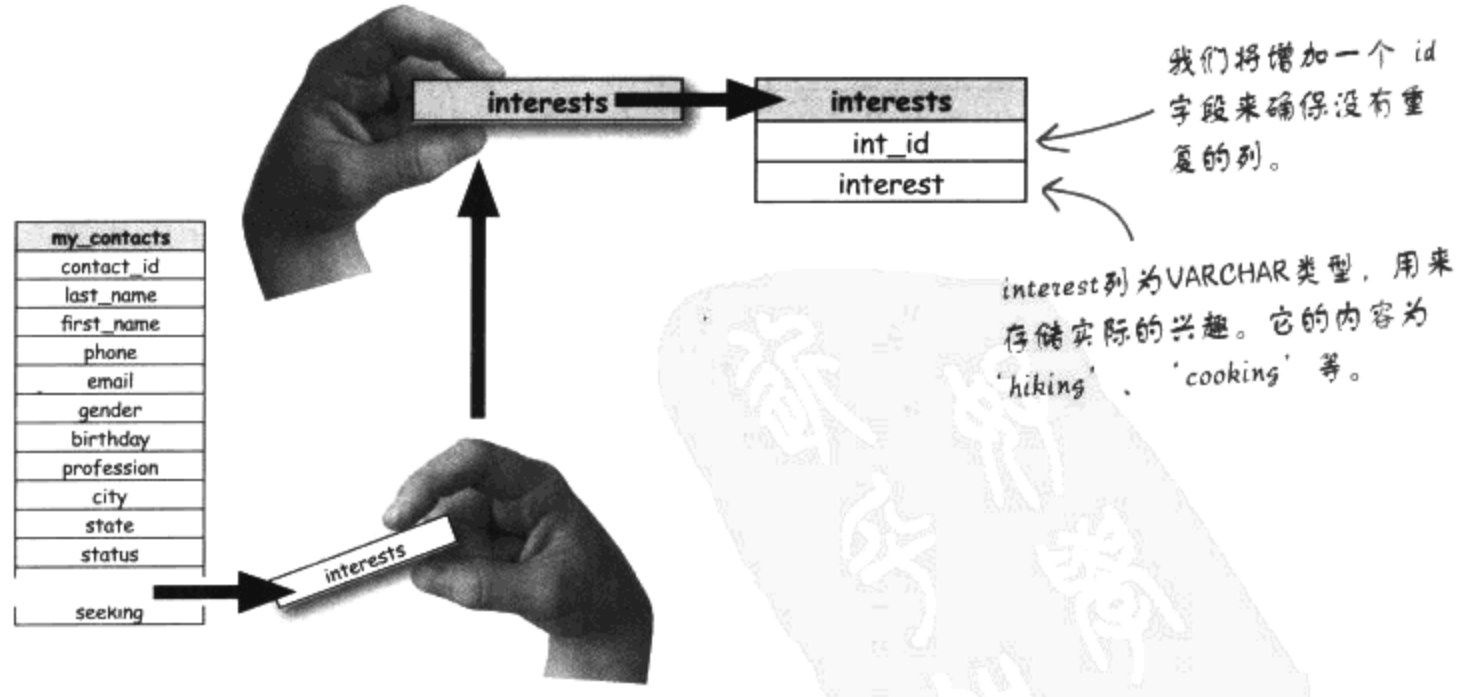
以兴趣列的现有状态而言，真的很难查询。同一列中包含多个值，如果试着把兴趣分开存储在多个列中，则会使查询非常难写。

看看当前的my_contacts表，它的兴趣列并不具有原子性。只有一个好办法能解决原子性的问题：我们需要另外一张用来存储兴趣的表。



1 **移出兴趣列并把它存储至专属表。**

第一步是把 interests 列移至新表中。

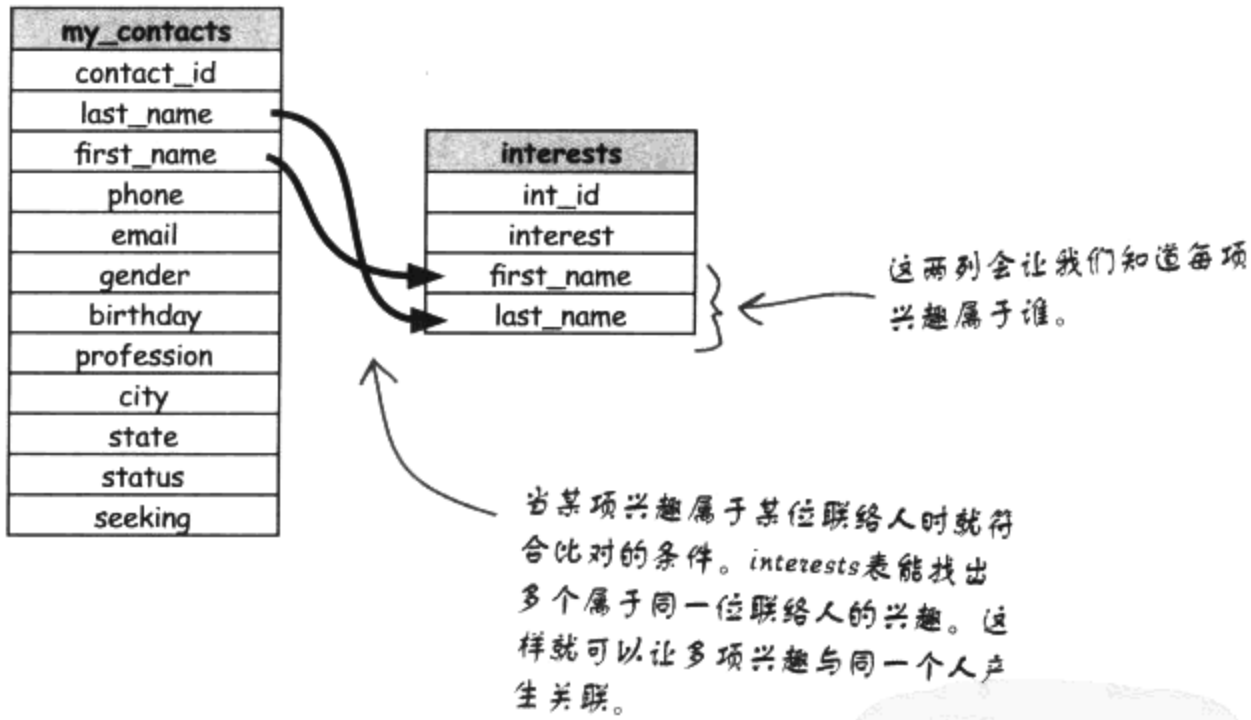


新设立的兴趣表将存储 my _ contacts 表中的所有兴趣数据，每项兴趣为一行。

❷ 添加足以识别my_contacts表中每个人的兴趣的列。

我们已经把兴趣从my_contacts表中移出，但无法识别各项兴趣分别属于谁。我们需要把my_contacts表中的某些列加入新表来建立连接。

其中一种方式是把first_name与last_name列加入兴趣表。



动动脑

我们的想法没错，但 first_name 与 last_name 列并非是连接这两张表的最佳选择。

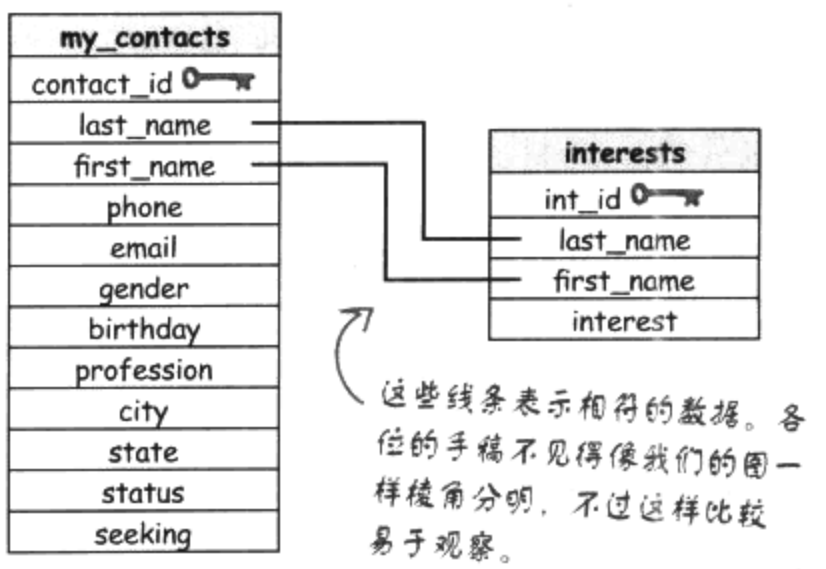
为什么呢？

在图表中加入链接

让我们一起仔细地看一下 my_contacts 表。

这部分是设计草稿：

这部分是新的模式（schema）：



请注意，连接表中相符列的线段都采取先向右弯的格式。因为使用了标准的模式（schema）图示符号，任何 SQL 开发者均可理解我们的草图。

下例是使用图中两张表中的数据的 SELECT 语句示例。

```
❶ SELECT first _ name, last _ name
  FROM my _ contacts
  WHERE (a bunch of conditions);
```

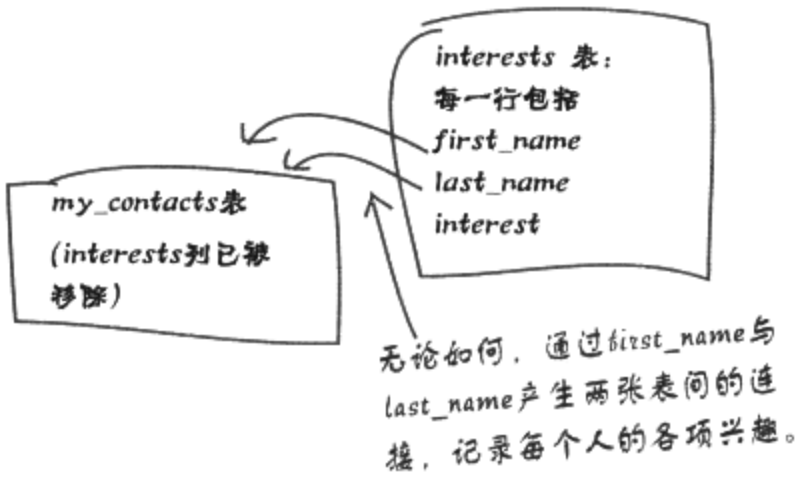
```
❷ SELECT interest FROM interests
  WHERE first _ name = 'Somename'
  AND last _ name = 'Lastname';
```

别担心查询看起来好像没效率的样子。本例只是呈现如何以一张表的内容来提取另一张表的内容。（我们很快就会讨论到更好的方法。）

磨笔上阵

请用本页的空白设计更多可以添加进greg_list数据库的新表，以协助我们追踪更多兴趣。

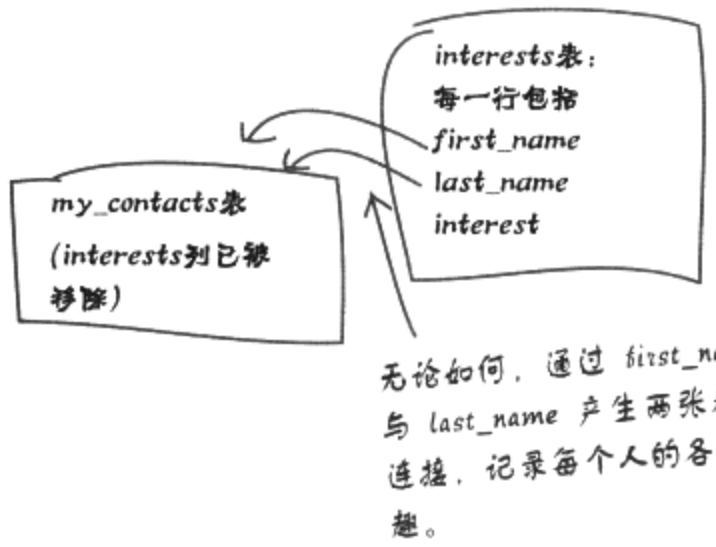
别把心思放在把图画得整洁清楚上，现在的重点是要想出新主意。我们已经画出了示范，但它还有缺陷。



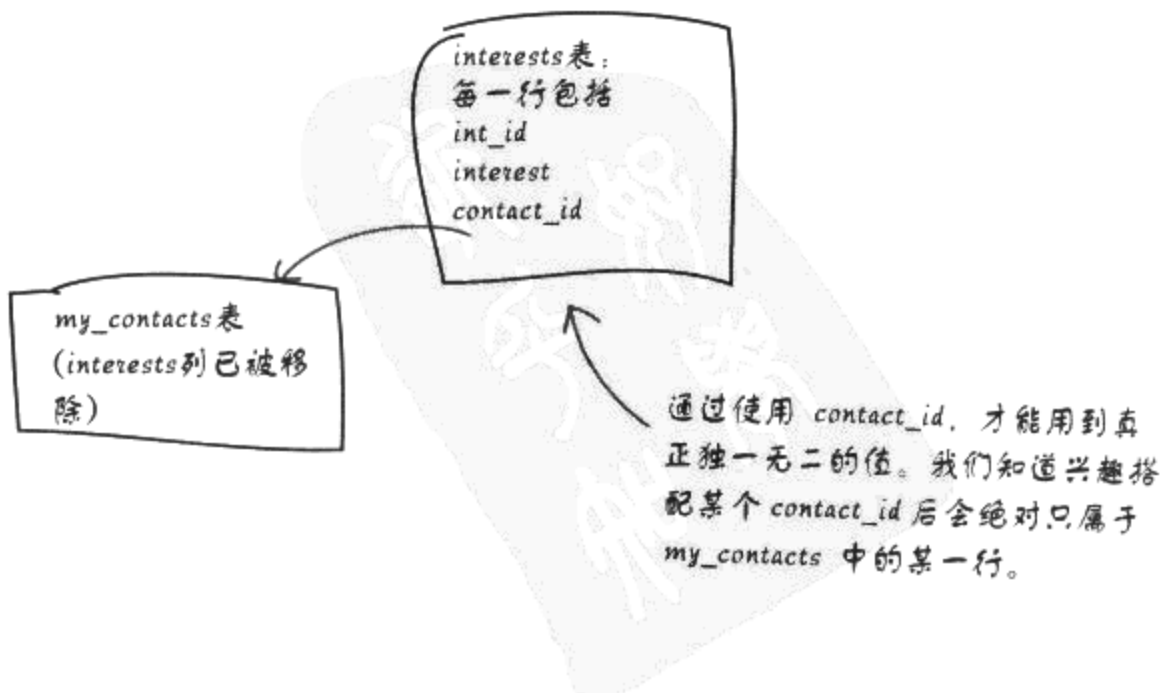
磨笔上阵解答

请用本页的空白设计更多可以添加进 greg_list 数据库的新表，以协助我们追踪更多兴趣。

别把心思放在把图画得整洁清楚上，现在的重点是要想出新主意。我们已经画出了示范，但它还有缺陷。

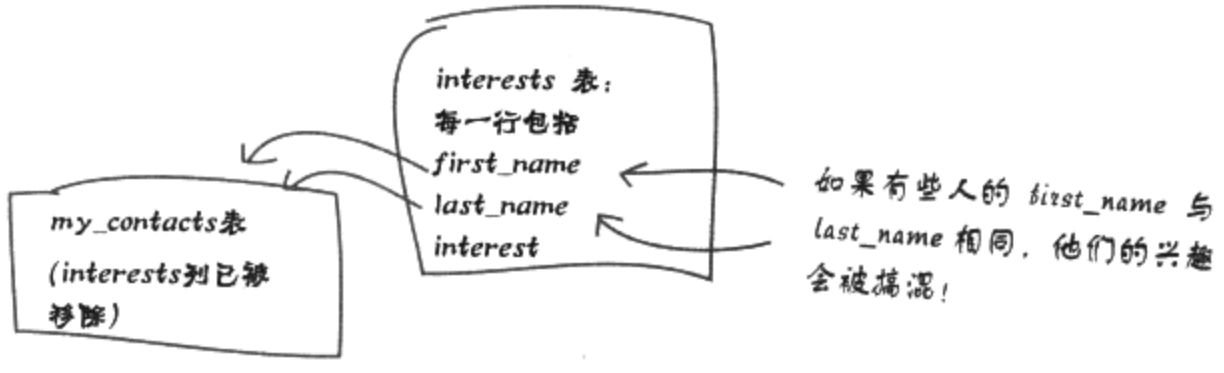


与其使用可能重复的 first_name 与 last_name，
不如使用 contact_id 来连接表：



连接你的表

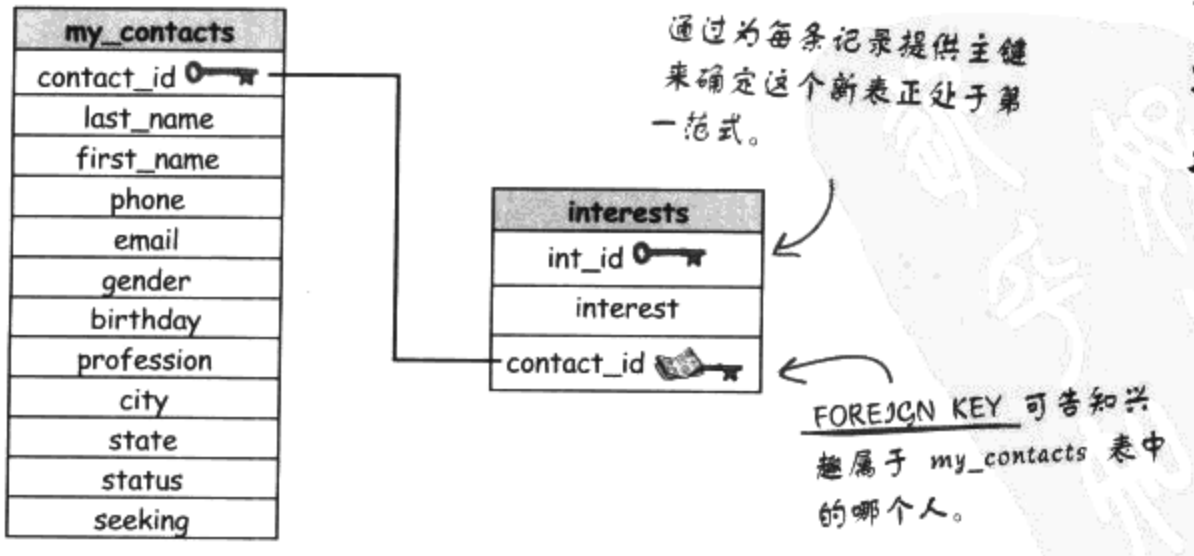
第一张草图的连接问题出在我们试着以first_name与last_name字段连接（connect）两张表。如果my_contacts表中有相同的first_name与last_name该怎么办？



我们需要独一无二的列来连接一切。幸好，我们已经开始规范化了，在my_contacts表中确实有独一无二的列：**主键**（primary key）contact_id。

可以把my_contacts表中的主键值作为interests表的一列。利用主键还有一项优点，就是我们可以能过主键列知道某个兴趣属于my_contacts表中的某个人。这种方式称为**外键**（foreign key）。

外键是表中的某一列，它引用到另一个表的主键。



外键二三事

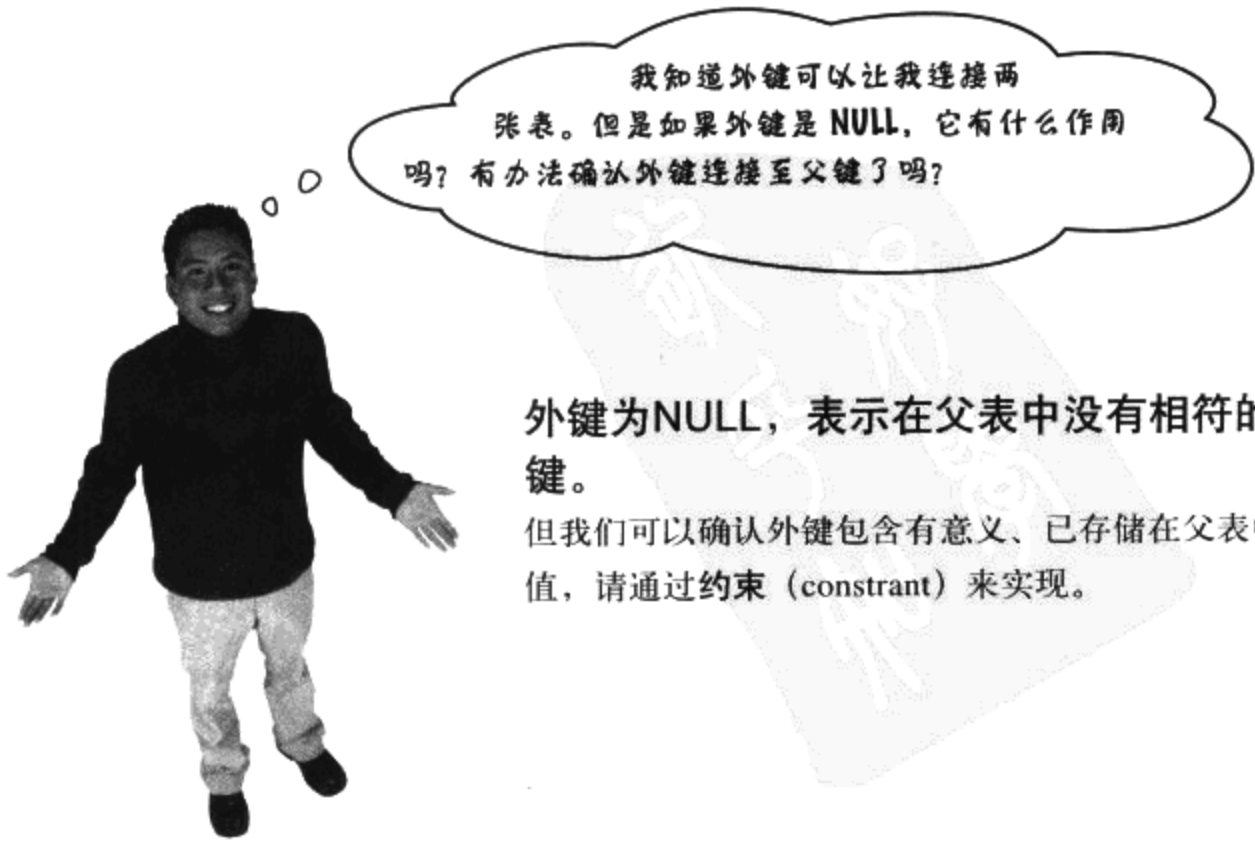
外键可能与它引用的主键名称不同。

外键使用的主键也被称为父键（parent key）。主键所在的表又被称为父表（parent table）。

外键能用于确认一张表中的行与另一张表中的行相对应。

外键的值可以是NULL，即使主键值不可为NULL。

外键值不需唯一——事实上，外键通常都没有唯一性。



外键为NULL，表示在父表中没有相符的主键。

但我们可以确认外键包含有意义、已存储在父表中的值，请通过**约束**（constrant）来实现。

外键约束

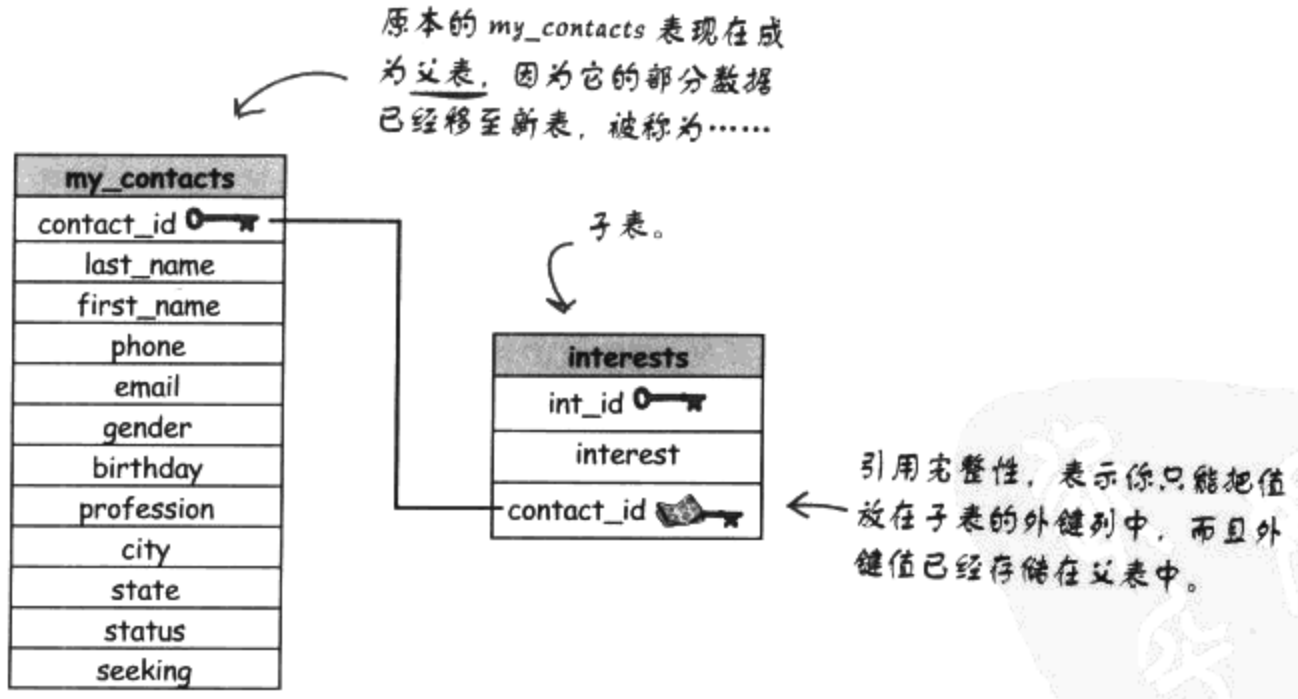
创建一张表并加上可作为外键的列虽然很简单，但除非你利用 CREATE 或 ALTER 语句来指定外键，否则都不算是真的外键。创建在结构内的键被称为**约束**（constraint）。

请把 CONSTRAINT 想象成表必须遵守的规则。

插入外键列的值必须已经存在于父表的来源列中，这是引用完整性（referential integrity）。

创建外键作为表的约束提供了明确的优势。

如果违反了规则，约束会阻止我们意外破坏表。



你可以使用外键来引用父表中某个唯一的值。
外键不一定必须是父表的主键，但必须有唯一性。

为什么要找外键的麻烦？

创建新表时你可以加入外键。

而且，你也可以用 ALTER TABLE 加入外键。语法很简单，只需要知道父表的主键列名，当然还要知道父表名。让我们创建带有外键contact_id的interests表，父表为my_contacts。

问： 当我把兴趣列抽离 my_contacts 后，我该怎么查询它们呢？

答： 下一章将提到这个问题。而且各位会从中发现：查询许多张表的方式真的很简单。现在，我们只需要重新设计 my_contacts，让查询简单且效率。

创建带有外键的表

知道为何需创建一个具有约束的外键后，接下来是实际操作。请注意 CONSTRAINT 的命名方式，这样才容易识别键的来源。

直接在设计列的同一行代码中加入 PRIMARY KEY 命令是另一种指定主键的方式（更快）。

创建外键就和创建索引列一样：把外键列设定为 INT 与 NOT NULL。

这部分称为 CONSTRAINT，它的命名方式能告知我们键的来源表（*my_contacts*）、键的名称（*contact_id*），还能说明它是一个外键（*fk*）。

如果我们日后改变了心意，还要用这个名称解除约束。本行为可选用的，但最好养成使用它的习惯。

括号中的列名就代表外键。可以随意命名。

```
CREATE TABLE interests (
int _ id INT NOT NULL AUTO _ INCREMENT PRIMARY KEY,
interest VARCHAR(50) NOT NULL,
contact _ id INT NOT NULL,
CONSTRAINT my _ contacts _ contact _ id _ fk
FOREIGN KEY (contact _ id)
REFERENCES my _ contacts (contact _ id)
);
```

这部分指定外键的来源……

……还有外键列在另一张表中的名称。

请大家动手做做看。打开你的数据库控制台界面，输入上述 SQL 代码来创建一张 interests 表。

创建完后，注意一下新表的结构。什么信息能够告诉你约束的位置？

请大家动手做做看。打开你的数据库控制台界面，输入上述 SQL 代码来创建一张 interests 表。

创建完成后，注意一下新表的结构。什么信息能够告诉你约束的位置？

```
File Edit Window Help
> DESC interests;
+------------+-------------+------+-----+---------+----------------+
| Field      | Type        | Null | Key | Default | Extra          |
+------------+-------------+------+-----+---------+----------------+
| int_id     | int(11)     | NO   | PRI | NULL    | auto_increment |
| interest   | varchar(50) | NO   |     |         |                |
| contact_id | int(11)     | NO   | MUL |         |                |
+------------+-------------+------+-----+---------+----------------+
```

MUL表示这一列可以存储多个相同的值，它也是追踪每个 *contact_id* 拥有什么兴趣的关键。

问： 我们做了这么多麻烦事来创建外键约束，但究竟是为了什么？不能单纯地使用另一张表的键，称之为外键，而不加上约束吗？

答： 其实可以，但创建成外键约束后，就只能插入已经存在于父表中的值，有助于加强两张表间的连接。

问： “加强连接”？这是什么意思？

答： 外键约束能确保引用完整性（换句话说，如果表中的某行有外键，约束能确保该行通过外键与另一张表中的某一行对应）。如果我们试着删除主键表中的行或试着改变主键值，而这个主键是其他表的外键约束时，你就会收到错误警告。

问： 所以说，my_contacts 中的数据行若具有主键，而且它的主键是其他表的外键时，我就没办法删除这个数据行？

答： 还是可以的，但必须先移除外键行。毕竟，如果你从 my_contacts 移除了某个人，也就不再需要知道他的兴趣了。

问： 但谁会在乎我把多余的兴趣行留在 interests 表中？

答： 多余的数据会拖慢查询速度。这种残留数据称为“孤儿”，它们会增加查询时间。孤儿记录变成必须搜索却无用的信息，因而拖慢了查询。

问： 好吧，我听你的。除了外键，还有其他约束吗？

答： 我们已经看到主键约束了。另外，（在创建列时）使用关键字 UNIQUE 也被视为约束的一种。还有另一种 MySQL 不支持的约束，称为 CHECK（检查）约束。它用于指定某个条件，列必须满足条件后才能插入新的值。关于 CHECK 的使用方式，请查询你的 RDBMS 说明文档。

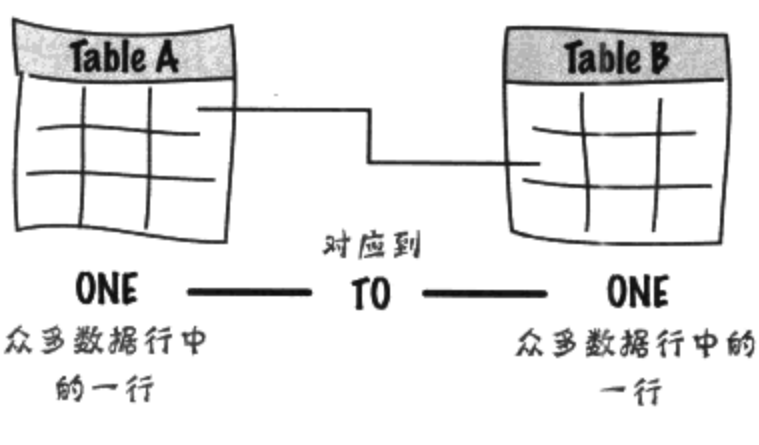
表间的关系

现在，我们知道如何通过外键连接表了，但我们仍然需要思考表之间产生关系的方式。以my_contacts表为例，我们的问题是要让**许多人**与**许多兴趣**产生关联。

接下来讨论的三种可能的模式，各位会重复在将来要接触到的数据上发现：**一对一**、**一对多**、**多对多**，找出数据所属的模式后，设计多张表的关系——设计**数据库模式**（schema），也就变得简单了。

数据模式：一对一

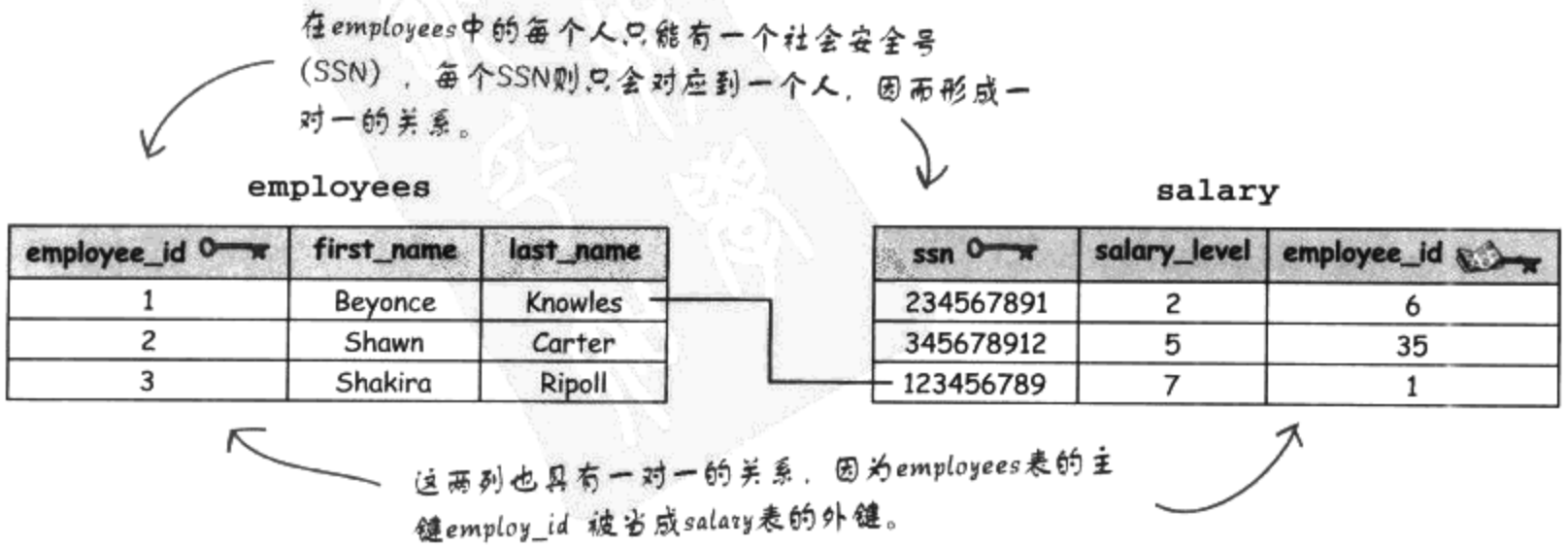
先看第一种模式（pattern），**一对一模式**。右图中 A 表的某条记录在 B 表中最多只能有一条相对应的记录。



假设 A 表包含你的姓名，B表则包含薪资信息和社会安全号，基于安全理由，所以要独立存储这些信息。

两张表都会包含你的ID编号，所以你可以拿到正确的薪资单。父表中的employee_id是主键，子表中的employee_id则是外键。

在模式（schema）图中，一对一关系的连接线是**单纯的实线**，表示连接**一件事物**与**另一件事物**。



数据模式：使用一对一的时机

事实上，不需要。我们其实不会经常用到一对一的表。

连接表时用到一对一关系的机会其实非常少。

使用一对一表的时机

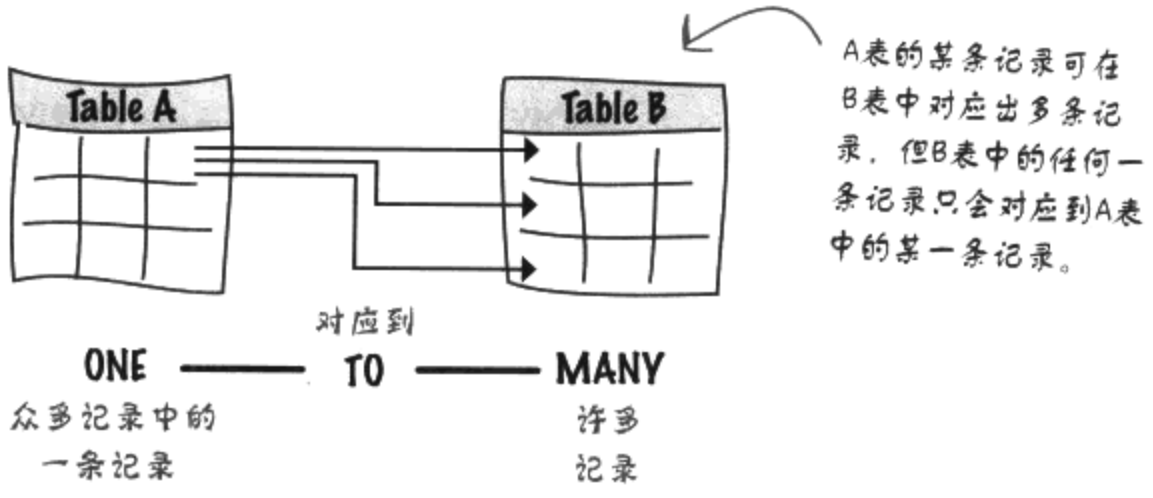
通常，把一对一的数据留在主表更合理，但也有适合把某些列拉出来的时候：

1. 抽出数据或许能让你写出更快速的查询。例如，如果大多数时候你只需要查询 SSN，就可以查询较小的 SSN 表。
2. 如果有列包含还不知道的值，可以单独存储这一列，以免主要表中出现 NULL。
3. 我们可能希望某些数据不要太常被访问。隔离这些数据即可管制访问次数。以员工表为例，他们的薪资信息最好存为另一张表。
4. 如果有一大块数据，例如BLOB类型，这段数据或许存为另一张表会更好。

一对一：父表只有一行与子表的某行相关。

数据模式：一对多

一对多，表示 A 表的某条记录在 B 表中可以对应到多条记录，但 B 表的每一条记录都只会对应到 A 表中的某一条记录。



一对多：A表中的某一条记录可以对应到B表中的多条记录，但B表中的某一条记录只能对应到A表中的某一条记录。

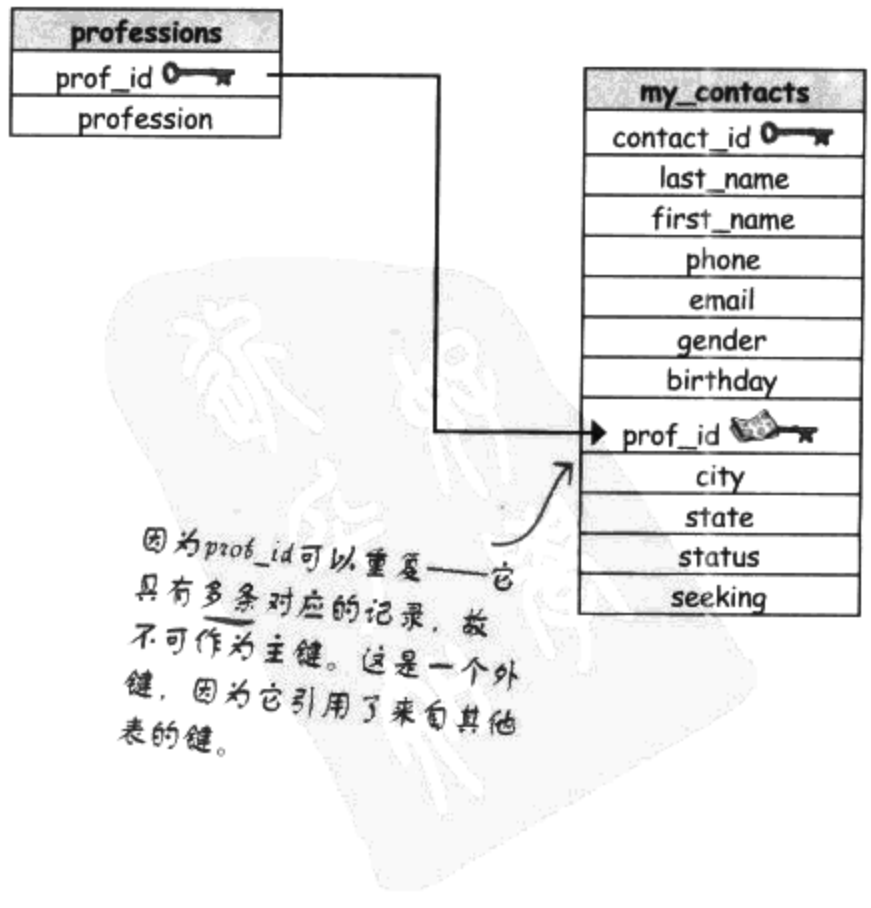
右图中my_contacts表的prof_id列就是一对多的好例子。每个人都有一个prof_id，但my_contacts表中却有很多人共用相同的prof_id。

在本例中，我们把profession列移进新表，父表的profession列则改为存储外键的 prof_id 列。因为它具有一对多关系，所以我们可以使用 prof_id 连接两张表。

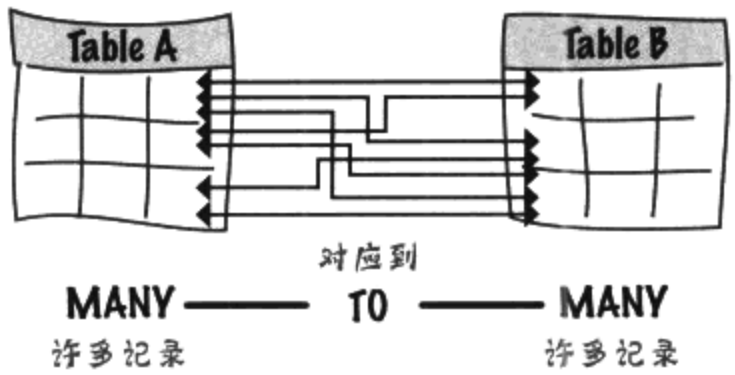
连接线应该带有**黑色箭头**来表示一对多的连接关系。

profession表中的每一行可能对应到my_contacts表中的许多行，但my_contacts表的每一行就只能对应到profession表中的一行。

例如，Programmer表中的prof_id可以在my_contacts表中出现很多次，但my_contacts表中的每个人却只会有一个prof_id。



数据模式：认清多对多



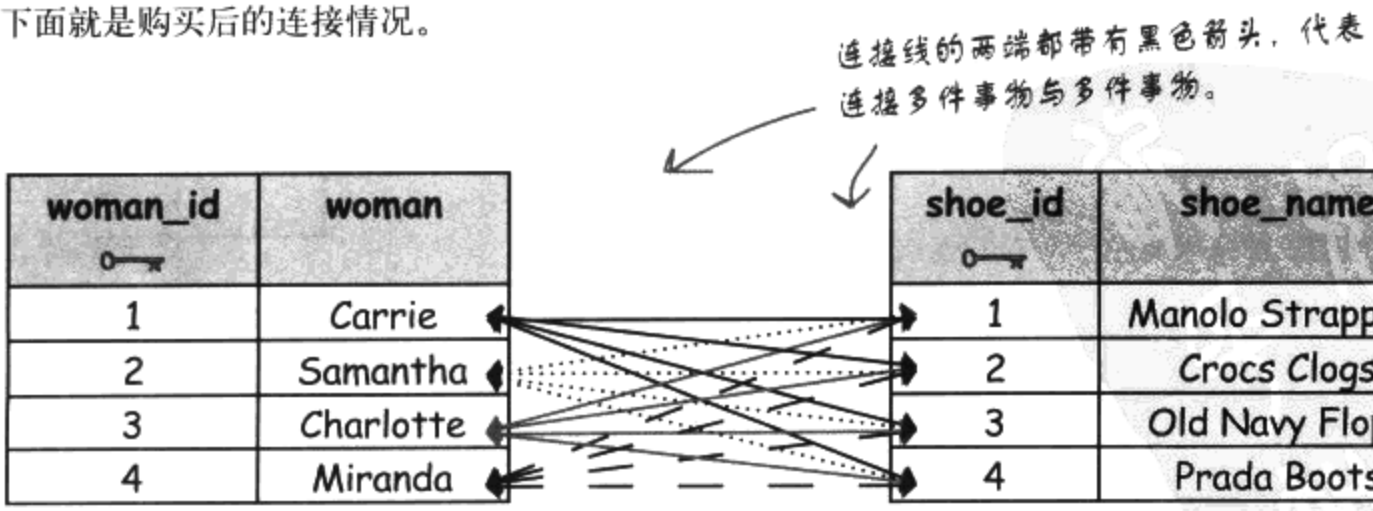
许多女士拥有**许多**双鞋。如果为了便于追踪而创建一张表来记录姓名，另建一张表来存储鞋名，此时就需要连接多条记录与多条记录，因为一种鞋可能不只是一名女士购买，每一位女士也不可能只有一双鞋。

假设 Carrie与Miranda 都买了 Old Navy Flops 和 Prada Boots，Samantha 与 Miranda 都拥有 Manolo Strappies，Charlotte 则每一款鞋都有一双。以下是women与shoes表之间的连接。

woman_id	woman
1	Carrie
2	Samantha
3	Charlotte
4	Miranda

shoe_id	shoe_name
1	Manolo Strappies
2	Crocs Clogs
3	Old Navy Flops
4	Prada Boots

试想女士们都很喜欢鞋子，每个人都买了一双自己还没有的鞋，下面就是购买后的连接情况。



该如何修改表，使得列的值不会超过一个（且最终可以实现类似 Greg 处理 interests 列问题的方式）？

磨笔上阵

查看前面关于鞋的第一组表，我们试着在记录女士姓名的表中加入外键shoe_id来解决这个问题。

woman_id (PK)	woman	shoe_id (FK)
1	Carrie	3
2	Samantha	1
3	Charlotte	1
4	Miranda	1
5	Carrie	4
6	Charlotte	2
7	Charlotte	3
8	Charlotte	4
9	Miranda	3
10	Miranda	4

shoe_id	shoe_name
1	Manolo Strappies
2	Crocs Clogs
3	Old Navy Flops
4	Prada boots

现在这两张表以 shoe_id 进行连接。

请试着自己画出表，但这一次把 woman_id当成外键放入shoes表。

画完表后，还要试着画出两张表之间的连接。

磨笔上阵解答

查看前面关于鞋的第一组表，我们试着在记录女士姓名的表中加入外键 shoe_id 来解决这个问题。

woman_id	woman	shoe_id
1	Carrie	3
2	Samantha	1
3	Charlotte	1
4	Miranda	1
5	Carrie	4
6	Charlotte	2
7	Charlotte	3
8	Charlotte	4
9	Miranda	3
10	Miranda	4

shoe_id	shoe_name
1	Manolo Strappies
2	Crocs Clogs
3	Old Navy Flops
4	Prada boots

现在这两张表以 shoe_id 进行连接。

请注意 woman_id 与 shoe_name 列的重复。

请试着自己画出表，但这一次把 woman_id当成外键放入 shoes 表。

画完表后，还要试着画出两张表之间的连接。

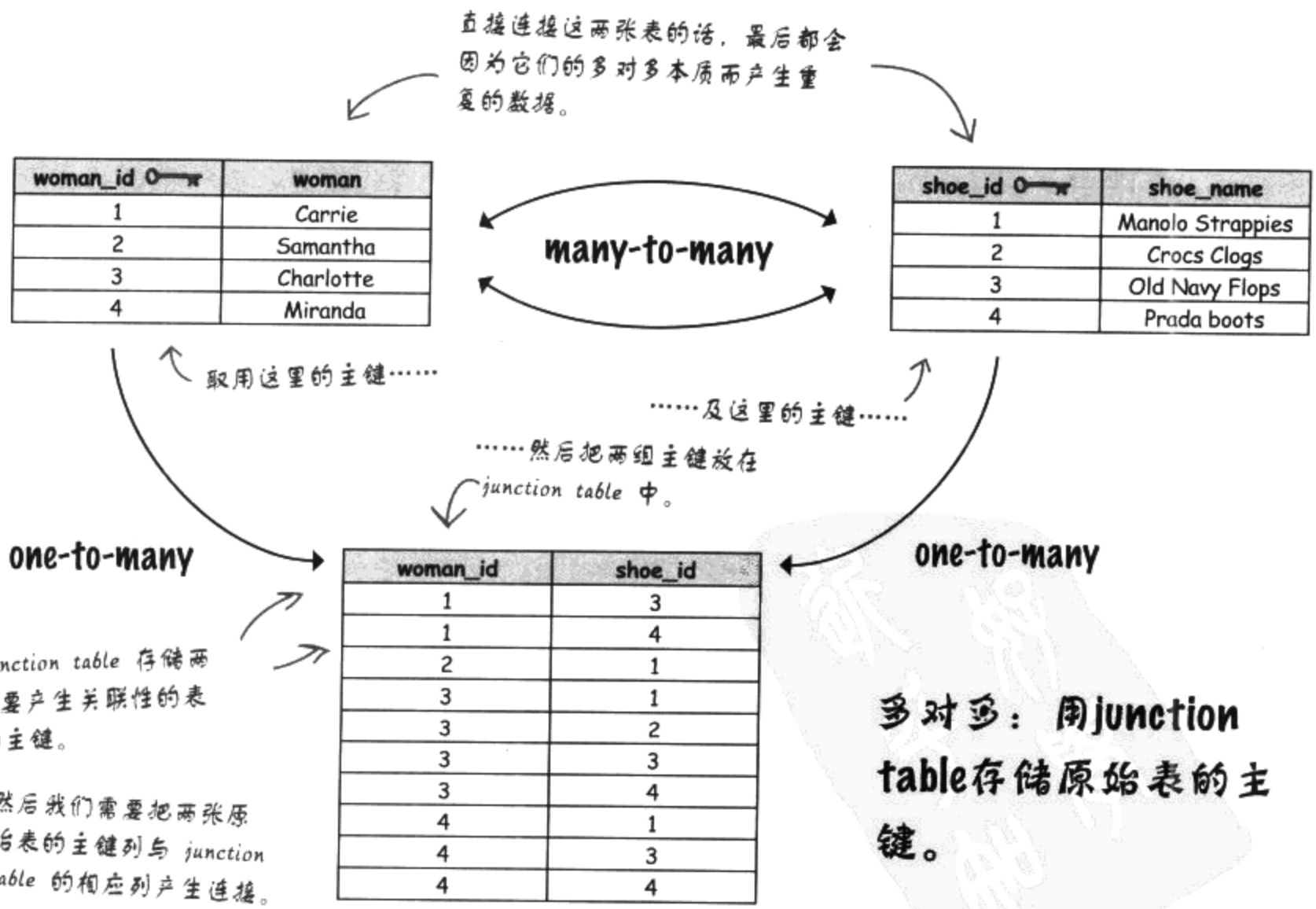
shoe_id	shoe_name	woman_id
1	Manolo Strappies	3
2	Crocs Clogs	2
3	Old Navy Flops	1
4	Prada boots	1
5	Crocs Clogs	3
6	Old Navy Flops	3
7	Prada boots	3
8	Manolo Strappies	4
9	Old Navy Flops	4
10	Prada boots	4

woman_id	woman
1	Carrie
2	Samantha
3	Charlotte
4	Miranda

数据模式：我们需要 junction table

如刚才所见，不管在哪个表中添加外键，都会造成表中出现重复的数据。请注意女士们的姓名重复出现的情况。我们应该只会看到一次姓名才对。

此时，在两个多对多的表之间需要一个中间桥梁来存储所有woman_id与shoe_id，从而把关系简化为一对多。这个中间桥梁就是所谓的junction table（连接表），用它来存储两个相关表的主键。

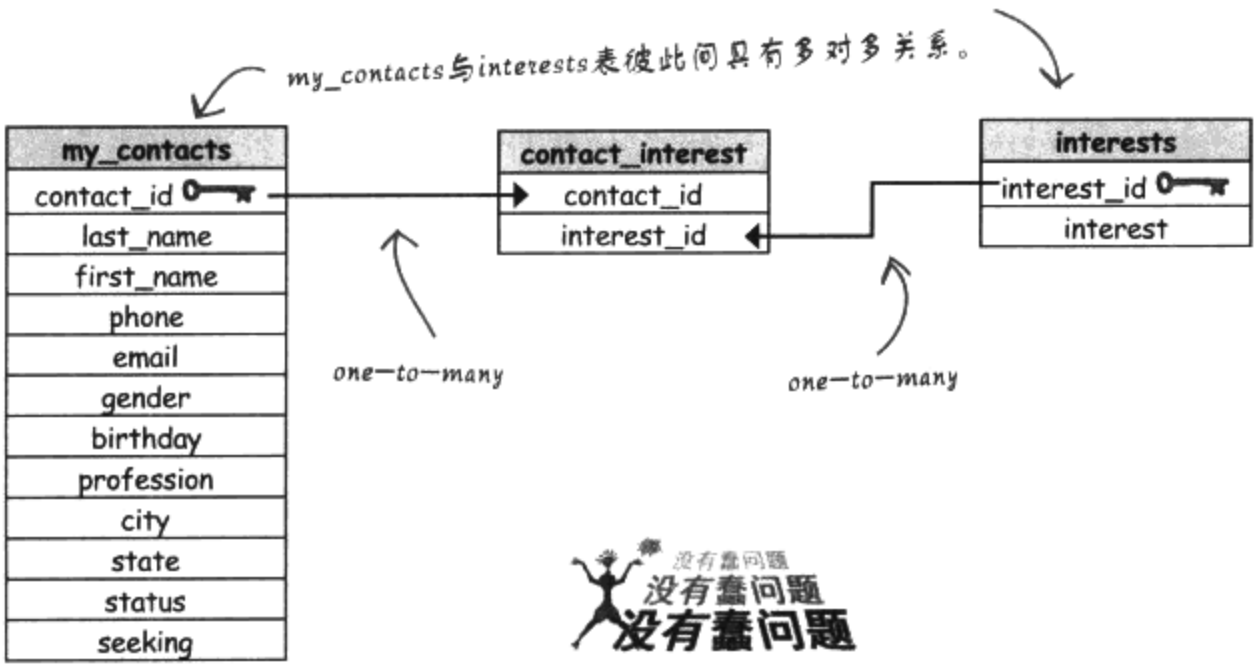


多对多：用junction table存储原始表的主键。

数据模式：多对多

现在你知道了多对多关系的秘密——它通常由两个一对多关系并利用junction table在中间连接而构成。在my_contacts表中，每一个人有多项兴趣，但每一项兴趣可能属于许多人，所以这种关系属于多对多的模式。

使用这种模式（schema），interests列也能转换为多对多关系。因为每个人有好几项兴趣，而每项兴趣则可能由许多人共享：



问： **遇到多对多关系的时候，一定要创建中间表吗？**

答： 是的，应该如此。如果两个表具有多对多关系，结果只会造成重复组，违反了第一范式。（过几页就会再次提到规范化。）

违反第一范式真的没什么好处，而且有太多不该违反的正当理由。最重要的一个原因就是重复的数据只会造成查询困难。

问： **把表改成上述形式后有什么优点？我可以把所有兴趣放在一个只有 contact_id 与 interest_name 列的表中。的确，数据有重复，但真的会有什么问题吗？**

答： 当你使用下一章的联接（join）查询多张表时就会体验到优点了。但是根据使用数据的方式，重复数据也可能有用处，像遇到重点在于多对多关系的表，而不是两张表中的数据的时候。

问： **如果我就是不在意数据重复呢？**

答： Junction table 有助于保持数据的完整性。如果必须删除某个存储在my _ contacts表中的联络人时，连接让我们不需分心注意interests表，只需管理contact _ interest表。若没有分开的表，你极有可能意外移除错误的记录。这种方式比较安全。

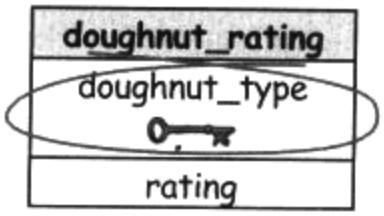
还有，在更新数据时，没有重复的数据能让工作更顺利。假设我们不小心拼错了某个冷门兴趣名，例如拼错了“spelunking”（洞穴探险）。当你修正它时，只要修改interests表中的一行记录，而不需修改contact _ interest或my _ contacts表。

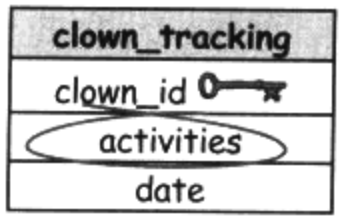
给关系命名

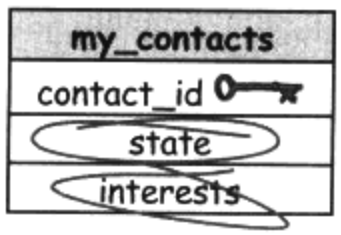
从下列表的简述中，判断圈起来的列是否最适合以一对多或多对多的关系来表示。

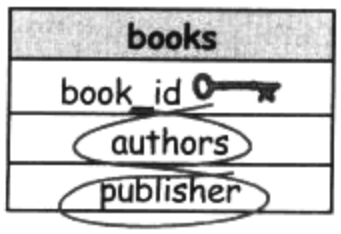
（请记住，在一对多或多对多的情况下，这些数据列会被抽离表，另外以 ID 字段来产生连接。）

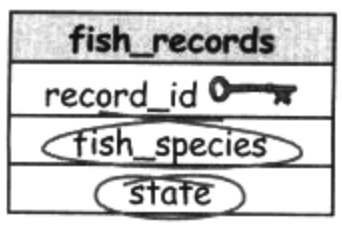
数据列











关系

.....

.....

.....

.....

.....

.....

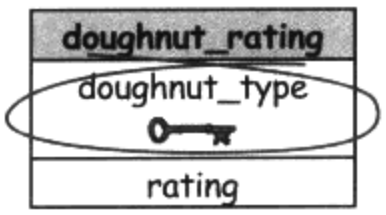
给关系命名

从下列表的简述中，判断圈起来的列是否最适合以一对多或多对多的关系来表示。

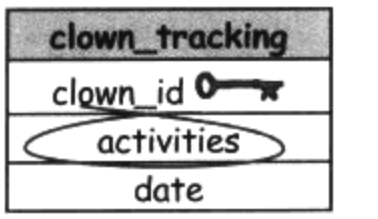
（请记住，在一对多或多对多的情况下，这些数据列会被抽离表，另外以 ID 字段来产生连接。）

数据列

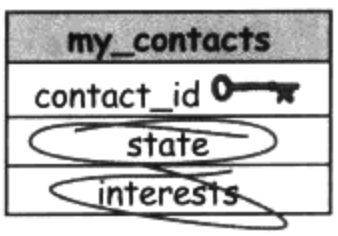
关系



一对多

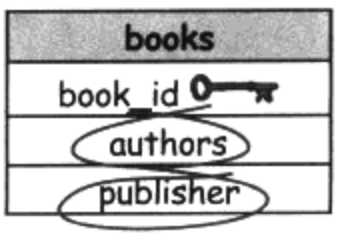


多对多



一对多

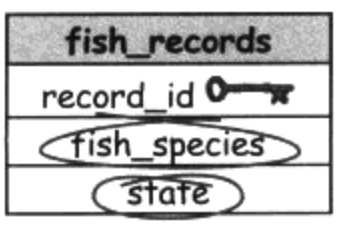
多对多



这一项要想一下，因为一本书的作者可能多于一位，所以是多对多。

多对多

一对多



一对多

一对多

数据模式：修正篇

我知道你接下来要做什么。一定是修改gregs_list数据库和my_contacts表，把它们改成多张表的形式，对不对？

差不多。在了解何为数据模式后，我们差不多可以重新设计 gregs_list 了。

我们现在已经知道兴趣列可以自己独立，并且与主要表形成一对多的关系。seeking 列也需要以相同方式修改。这些改变都会让表符合第一范式*。

但我们不能只停留在第一范式，还要进一步的规范化。数据越规范，以查询取得数据也就越容易，对下一章会谈到的联接（join）也越有帮助。为`gregs_list`数据库创建新的模式（schema）前，我们先一起了解规范化的更多级别。

my_contacts
contact_id
last_name
first_name
phone
email
gender
birthday
profession
city
state
status
interests
seeking

* 各位或许正打算翻回前几章温习第一范式。不需要，下一页就会再次提到它。

尚非第一范式

我们讨论过第一范式（1NF），这里先复习一遍，然后向前推动更多规范化，进入第二范式甚至第三范式。

回到正题。让我们先回忆一下，一个满足 1NF 的表应该具备哪些条件。

第一范式，或 1NF：
规则一：数据列只包含具有原子性的值
规则二：没有重复的数据组

下列表都不满足1NF。请注意，右侧的表已添加了额外的颜色列了，但颜色还是会在表中重复出现：

非1NF

toy_id	toy	colors
5	whiffleball	white, yellow, blue
6	frisbee	green, yellow
9	kite	red, blue, green
12	yoyo	white, yellow

想让数据有原子性，colors列就只能包含其中一个颜色，而不是在一列中记录两三个值。

仍非1NF

toy_id	toy	color1	color2	color3
5	whiffleball	white	yellow	blue
6	frisbee	green	yellow	
9	kite	red	blue	green
12	yoyo	white	yellow	

这个表还是不符合1NF的要求，因为各个颜色列都存储了相同分类中的数据——都是VARCHAR类型的玩具颜色描述。

终于符合1NF

观察一下我们做的改变。

In 1NF

主键。

toy_id	toy
5	whiffleball
6	frisbee
9	kite
12	yoyo

现在我们终于知道这些表会通过外键进行连接，我们不用帮每条记录画出连接线。

外键。

toy_id	color
5	white
5	yellow
5	blue
6	green
6	yellow
9	red
9	blue
9	green
12	white
12	yellow

现在，列内不再有重复的内容。每一行都是一条颜色记录，而每一行都是唯一的。

一起使用toy_id与color值，就能形成唯一的主键。

如果我们把toy_id作为外键加入另一张表中，那么因为另一张表不需要外键列具有唯一性，所以没有问题。但如果一起考虑颜色值，那么**所有行都会具有唯一性**，因为每个颜色值加上toy_id后就会形成**独一无二的组合**。

并非如此。由两列以上组成的键称为组合键。

接下来会以更多表为例，说明多个主键列如何运作。

组合键使用了多个列

到目前为止，我们讨论到表中的数据可以形成对其他表的关系（例如一对一、一对多）。但我们还未想过，**表中的数据列本身对其他列也有关系**。了解这层关系，就是了解第二范式与第三范式的关键。

一旦理解了列本身的关系，我们创建的**表数据库模式**（schema）就可使查询多张表的任务较为轻松简单。

下一章讨论到联接时，大家一定会希望有设计良好的表。

讲了这么多，组合键（composite key）到底是什么？

组合键就是由多个数据列构成的主键，组合各列后形成具有唯一性的键。

请看下面的超级英雄列表。它缺少独一无二的可作为主键的列，但我们可以利用name与power列创建一个组合主键。虽然有些超级英雄用了相同的名称，有些英雄具有相同的能力，但综合考量名称与能力，则形成了具有唯一性的键。

火柴人，火柴人千变万化他都能
我只需要NO.2
命令火柴器大能
画出你的火柴人

创建如下的表并指定这两列为组合主键。但前题是没有重复的name与power组合，这样才能满足唯一性。

super_heroes

name	power	weakness
Super Trashman	Cleans quickly	bleach
The Broker	Makes money from nothing	NULL
Super Guy	Flies	birds
Wonder Waiter	Never forgets an order	insects
Dirtman	Creates dust storms	bleach
Super Guy	Super strength	the other Super Guy
Furious Woman	Gets really, really angry	NULL
The Toad	Tongue of justice	insects
Librarian	Can find anything	NULL
Goose Girl	Flies	NULL
Stick Man	Stands in for humans	games of Hangman

即使是超级英雄，也要一点依靠

我们的超级英雄实在非常忙碌！下表是更新过的super_heroes，虽然已经符合 1NF，但却出现了其他问题。

看到initials列包含的name缩写了吗？如果有超级英雄决定改名，该怎么办？

没错，initials列也要一起改变。也就是说，initials列函数依赖（functional dependency）于name列。

当某列的数据必须随着另一列的数据的改变而改变时，表示第一列函数依赖于第二列。

这里有两个相同的英雄名称，但加上他们的能力后，即可形成具有唯一性的组合主键。

super_heroes

name 0+rr	power 0+rr	weakness	city	country	arch_enemy	initials
Super Trashman	Cleans quickly	bleach	Gotham	US	Verminator	ST
The Broker	Makes money from nothing	NULL	New York	US	Mister Taxman	TB
Super Guy	Flies	birds	Metropolis	US	Super Fella	SG
Wonder Waiter	Never forgets an order	insects	Paris	France	All You Can Eat Girl	WW
Dirtman	Creates dust storms	bleach	Tulsa	US	Hoover	D
Super Guy	Super strength	aluminum	Metropolis	US	Badman	SG
Furious Woman	Gets really, really angry	NULL	Rome	Italy	The Therapist	FW
The Toad	Tongue of justice	insects	London	England	Heron	T
Librarian	Can find anything	children	Springfield	US	Chaos Creep	L
Goose Girl	Flies	NULL	Minneapolis	US	The Quilter	GG
Stick Man	Stands in for humans	hang man	London	England	Eraserman	SM

磨笔上阵

从超级英雄列表中，我们已经知道initials列依赖于name列。这个表还有类似的依赖性吗？若还有依赖性，请写在下面。

.....

.....

.....

.....

从超级英雄列表中，我们已经知道initials列依赖于name列。这个表还有类似的依赖性吗？若还有依赖性，请写在下面。

initials 依赖于 *name*

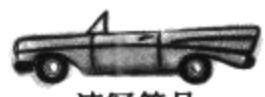
weakness 依赖于 *name*

arch_enemy 依赖于 *name*

city 依赖于 *country*

这里没有提到列的来源表，但随着表的增加，来源表也会变得重要。我们有表示依赖性与来源表的速记方式。

速记符号


速写符号

快速表示函数依赖的方式是：

T.x –> T.y ← 专有术语是速记符号（*shorthand notation*）

可以解释成“在关系表 T 中，y 列函数依赖于 x 列。”基本上，从右读到左就是解读依赖性的方式。

接下来把速记法套用到超级英雄表上：

super _ heroes.name –> super _ heroes.initials

“在关系表 super_heroes 中，initials列函数依赖于name列。”

super _ heroes.name –> super _ heroes.weakness

“在关系表super_heroes中，weakness列函数依赖于name列。”

super _ heroes.name–>super _ heroes.arch _ enemy

“在关系表 super_heroes中，arch_enemy列函数依赖于name列。”

super _ heroes.country –> super _ heroes.city

“在关系表super_heroes中，city列函数依赖于country 列。”

超级英雄的依赖性

如果超级英雄改了名字，initials 列也需随之修改，所以它**依赖于** name 列。

如果英雄们的超级天敌（arch-enemy）决定搬到新的城市，虽然敌人的位置改变了，但表中的其他内容并未改变，因此下表中的 `arch_enemy_city` 列是个完全**不依赖**的列。

依赖列中包含了**可能随其他列的改变而改变**的数据，**不依赖**的列则**完全置身事外**。

部分函数依赖

部分函数依赖（partial functional dependency）是指，非主键的列依赖于组合主键的某个部分（但不是完全依赖于组合主键）。

在超级英雄表中，`initials`列对`name`列的依赖正是**部分依赖**性的例子。如果超级英雄改名了，那么缩写列也要跟着修改，但英雄的能力如果变了，缩写并不需要跟着修改。

Name与power构成组合主键。

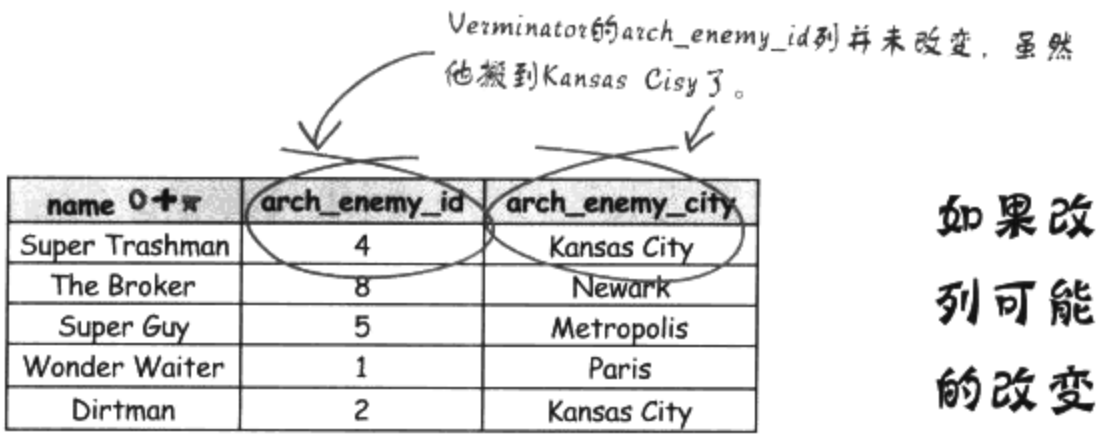
Initials依赖于name，但power与它无关，所以这个表具有部分函数依赖性。

super_heroes

name 0+键	power 0+键	weakness	city	initials	arch_enemy_id	arch_enemy_city
Super Trashman	Cleans quickly	bleach	Gotham	ST	4	Gotham
The Broker	Makes money from nothing	NULL	New York	TB	8	Newark
Super Guy	Flies	birds	Metropolis	SG	5	Metropolis
Wonder Waiter	Never forgets an order	insects	Paris	WW	1	Paris
Dirtman	Creates dust storms	bleach	Tulsa	D	2	Kansas City
Super Guy	Super strength	aluminum	Metropolis	SG	7	Gotham
Furious Woman	Gets really, really angry	NULL	Rome	FW	10	Rome
The Toad	Tongue of justice	insects	London	T	16	Bath
Librarian	Can find anything	children	Springfield	L	3	Louisville
Goose Girl	Flies	NULL	Minneapolis	GG	9	Minneapolis

传递函数依赖

另外，每个非键列的相互关联也一样需要考虑。假设有个天敌搬到新的城市，搬家的行动不会改变他的arch_ememy_id。

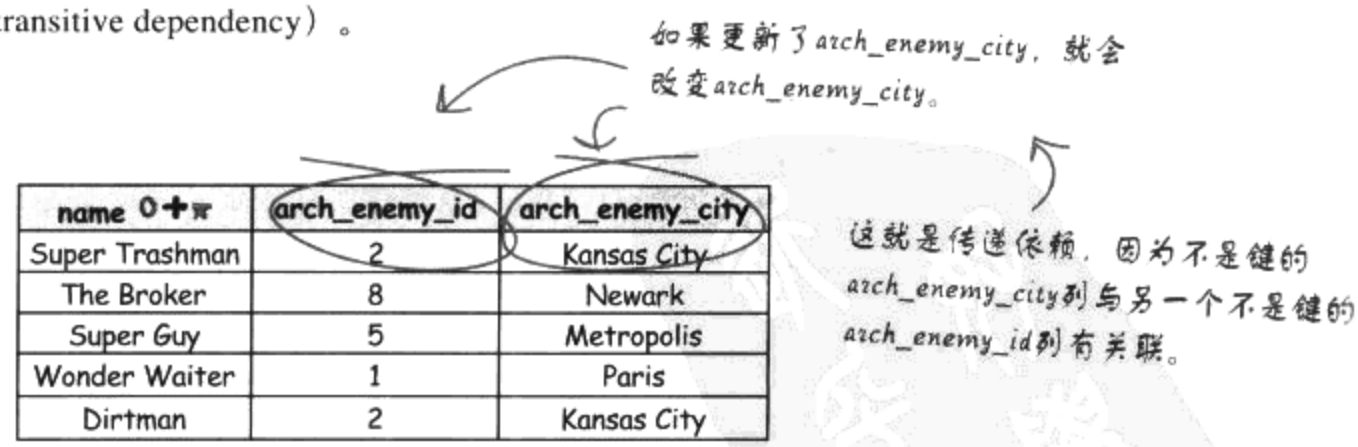


name 0+*	arch_enemy_id	arch_enemy_city
Super Trashman	4	Kansas City
The Broker	8	Newark
Super Guy	5	Metropolis
Wonder Waiter	1	Paris
Dirtman	2	Kansas City

如果改变任何非键列可能造成其他列的改变，即为传递依赖。

假设超级英雄改变了他的天敌对象，那么arch_enemy_id列可能会改变，arch_enemy_city列也可能改变。

如果改变任何非键列可能造成其他列的改变，即为**传递依赖**（transitive dependency）。



name 0+*	arch_enemy_id	arch_enemy_city
Super Trashman	2	Kansas City
The Broker	8	Newark
Super Guy	5	Metropolis
Wonder Waiter	1	Paris
Dirtman	2	Kansas City

传递函数依赖：

任何非键列与另一个非键列有关联。

请研究这个列出书籍标题的表。pub_id 表示出版商，pub_city 则是出版地点。

author 0+次	title 0+次	copyright	pub_id	pub_city
John Deere	Easy Being Green	1930	2	New York
Fred Mertz	I Hate Lucy	1968	5	Boston
Lassie	Help Timmy!	1950	3	San Francisco
Timmy	Lassie, Calm Down	1951	1	New York

如果第三行的书籍标题修改为“Help Timmy! I'm Stuck Down A Well”，请写下 copyright 列所受的影响。

如果 title 改变了，copyright 值也会随之改变。

copyright依赖于title，所以它的值也会改变。

如果第三行的作者修改为“Rin Tin Tin”，但标题不变，对 copyright 列会有何影响?

如果改变第一行的 pub_id 为 1，对“Easy Being Green”会有何影响?

如果“I Hate Lucy”的出版商搬到 Sebastopol，对于同一记录的 pub_id 值有何影响?

如果“I Hate Lucy”的 pub_id 值修改为 1，对于同一记录的 pub_city 值将有何影响?

请研究这个列出书籍标题的表。pub_id 表示出版商，pub_city 则是出版地点。

如果第三行的书籍标题修改为“Help Timmy! I'm Stuck Down A Well”，请写下 copyright 列所受的影响。

如果title列改变了，copyright值也会随之改变。

copyright 依赖于 title，所以它的值也会改变。

如果第三行的作者修改为“Rin Tin Tin”，但标题不变，对 copyright 列会有何影响?

如果author列改变，但title列不变，copyright列也会随之改变。

author列加上title列构成组合主键。

copyright列依赖于title列，也依赖于author列。

author 0+*	title 0+*	copyright	pub_id	pub_city
John Deere	Easy Being Green	1930	2	New York
Fred Mertz	I Hate Lucy	1968	5	Boston
Lassie	Help Timmy!	1950	3	San Francisco
Timmy	Lassie, Calm Down	1951	1	New York

如果改变第一行的 pub_id 为 1，对“Easy Being Green”列有何影响?

pub_city 不会改变。

pub_id 1与pud_id 2的pub_city值都是New York，所以城市的值不会改变（不过pub_city列对pub_id列具有传递依赖性。）

如果“I Hate Lucy”的出版商搬到 Sebastopol，对于同一记录的 pub_id 值有何影响?

pub_id列不依赖于pub_city列，所以pub_id值保持不变。

pub_id 值会保持不变。

如果“I Hate Lucy”的 pub_id 值修改为 1，对于同一记录的 pub_city 值有何影响?

pub_city列对pub_id列具有传递依赖性，所以城市值会改变。

pub_city值会变成New York。

pub_city值根据pub_id值而改变，但这两列都不是键，所以是传递函数依赖的例子。

author 0+*	title 0+*	copyright	pub_id	pub_city
John Deere	Easy Being Green	1930	2	New York
Fred Mertz	I Hate Lucy	1968	5	Boston
Lassie	Help Timmy!	1950	3	San Francisco
Timmy	Lassie, Calm Down	1951	1	New York

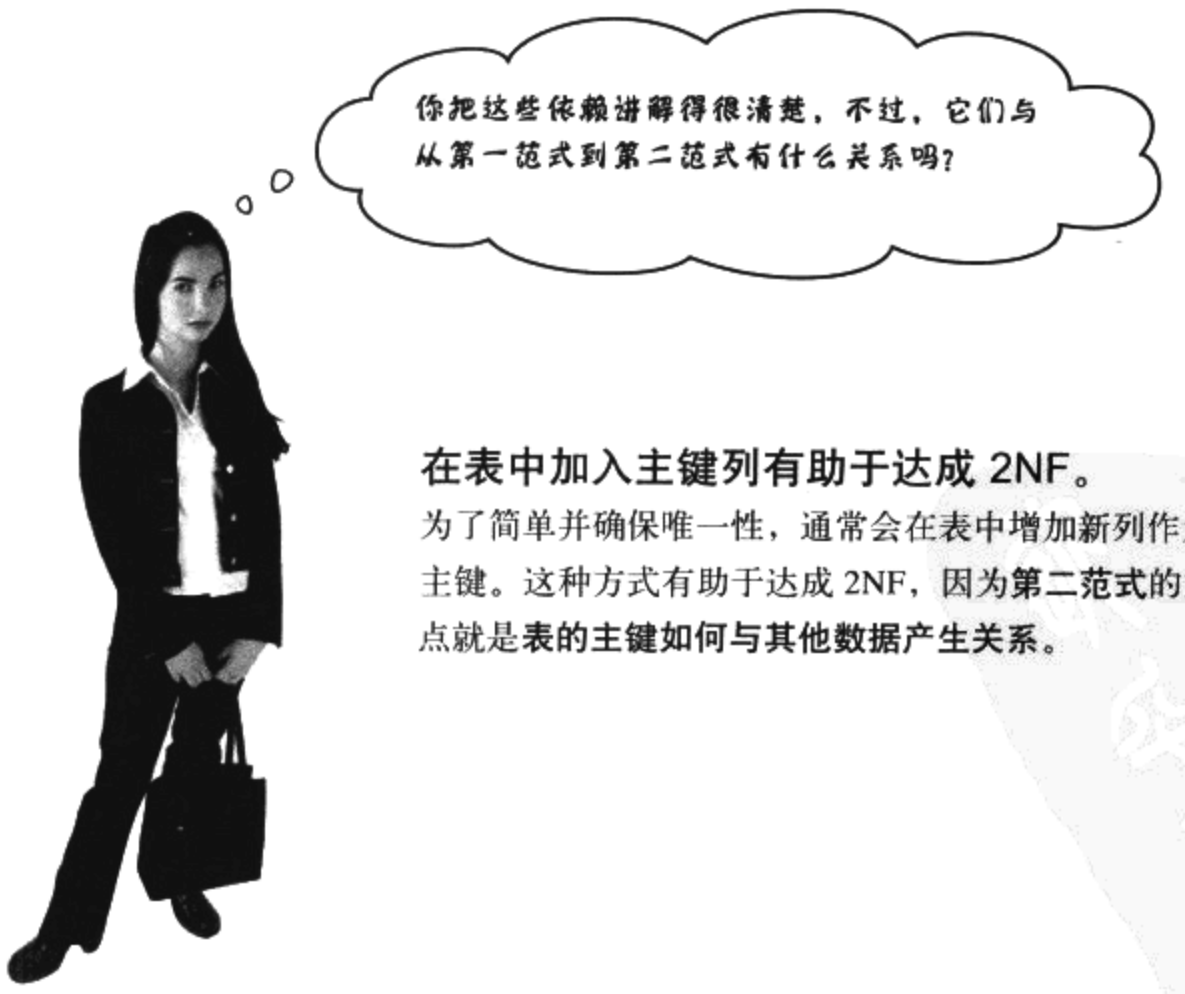
没有蠢问题

问： 有什么简单的方式可以避免部分函数依赖吗？

答： 采用类似于 my _ contacts 中用的 id 字段即可完全避免这类问题。既然这类 id 是新的专用于索引的字段，就表示没有字段依赖于它。

问： 除了创建 junction table 的需求，我真的会想用组合键吗？为什么不只是创建 id 字段？

答： 直接创建id 字段也是一种方式。但如果上网搜索“synthetic or natural key”（人造键或自然键），你会发现争论这个问题的两方都很有说服力，也会找到白热化状态中的辩论内容。关于这个议题，我们留给各位决定。本书主要采用单一人造主键，用于维持语法的简单，让大家能学到该学的概念，又不会因为实现约束而受困。



在表中加入主键列有助于达成 2NF。

为了简单并确保唯一性，通常会在表中增加新列作为主键。这种方式有助于达成 2NF，因为**第二范式**的重点就是**表的主键如何与其他数据产生关系。**

第二范式

接下来采用两个假想的玩具库存表来说明第二范式的重点——表的主键与表中其他数据之间的关系。

toy_id	toy
5	whiffleball
6	frisbee
9	kite
12	yoyo

组合键。

toy_id 0+	store_id 0+	color	inventory	store_address
5	1	white	34	23 Maple
5	3	yellow	12	100 E. North St.
5	1	blue	5	23 Maple
6	2	green	10	1902 Amber Ln.
6	4	yellow	24	17 Engleside
9	1	red	50	23 Maple
9	2	blue	2	1902 Amber Ln
9	2	green	18	1902 Amber Ln
12	4	white	28	17 Engleside
12	4	yellow	11	17 Engleside

这一列有很多重复，而且它对于库存的检索也没有什么帮助，它是与销售商店有关的信息。

这一列或许也需要重新考虑。它其实与玩具本身的关联较多，与库存的关联并不太多。我们的 *toy_id* 列应该就足以找出玩具类型和颜色了。

Inventory列依赖于被标示为组合主键的两列，所以它没有部分函数依赖性。

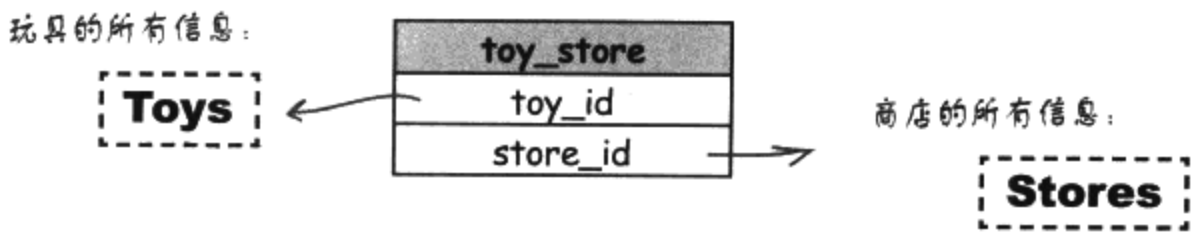
请注意，当玩具与商店（store _ id）相关联时，store_address 会一再重复。如果有修改store_address的需要，就必须改变每一条引用这张表的记录。需要更新的行越多，不知不觉在数据中出现错误的可能性也会越大。

或许已经达成2NF了……

已符合1NF的表只要其所有列都是主键的一部分，则它也就符合2NF。

创建新的表，以toy_id与store_id为组合主键。我们可以通过新表来连接原有的玩具信息表与商店信息表。

只要所有列都是主键的一部分
或者
表中有唯一主键列
符合 1NF 的表
也会符合 2NF



已符合1NF的表若只拥有一列主键，也会符合2NF。

这也是指定一个AUTO_INCREMENT ID列的好理由。

第二范式，又称2NF:
规则一：先符合1NF
规则二：没有部分函数依赖性。

所以说，游戏时间又到了……

与2NF表天人合一
别留下部分函数依赖性

你的任务是与表合而为一，并移除表中的部分函数依赖性。请观察下列各个图表，划掉应该从中移除的列。

这两列构成了具有唯一性的组合主键。

toy_inventory
toy_id
store_id

singers
singer_id
last_name
first_name
agency
agency_state

cookie_sales
amount
girl_id
date
girl_name
troop_leader
total_sales

movies
movie_id
title
genre
rented_by
due_date
rating

salary
employee_id
last_name
first_name
salary
manager
employee_email
hire_date

dog_breeds
breed
description
avg_weight
avg_height
club_id
club_state

磨笔上阵

重新设计下列表，改成三张都符合 2NF 的表。

其中一张表用来记录玩具信息，另一张用来记录商店信息，最后一张则记录库存量并可连接到另外两张表。

最后，为合适的表添加如下列：phone、manager、cost及weight。你可能必须创建新的tog_ids。

toy_id	toy
5	whiffleball
6	frisbee
9	kite
12	yoyo

toy_id	store_id	color	inventory	store_address
5	1	white	34	23 Maple
5	3	yellow	12	100 E. North St.
5	1	blue	5	23 Maple
6	2	green	10	1902 Amber Ln.
6	4	yellow	24	17 Engleside
9	1	red	50	23 Maple
9	2	blue	2	1902 Amber Ln
9	2	green	18	1902 Amber Ln
12	4	white	28	17 Engleside
12	4	yellow	11	17 Engleside

与 2NF 表天人合一
别留下部分函数依赖性解答

你的任任务是与表合而为一，并移除表中的部分函数依赖性。请观察下列各个图表，划掉应该从中移除的列。

这两列构成了具唯一性的组合主键。

toy_inventory
- toy_id
- store_id

主键。

singers
- singer_id
- last_name
- first_name
- agency
- agency_state

虽然这两列应该改为从另一张专门记录经纪公司的表中抽取出来的 ID 值，但它并不具有部分依赖性（因为可能有两家公司采用相同名称）。

cookie_sales
- amount
- girl_id
- date
- ~~girl_name~~
- ~~troop_leader~~
- ~~total_sales~~

一旦移出这些列，其余列就可以形成组合主键。

主键。

movies
- movie_id
- title
- genre
- rented_by
- due_date
- rating

这些列只有传递函数依赖性。

主键。

salary
- employee_id
- last_name
- first_name
- salary
- manager
- employee_email
- hire_date

虽然这些列应该移出，但它们并未具有部分函数依赖性。

dog_breeds
- breed
- description
- avg_weight
- avg_height
- club_id
- ~~club_state~~

组合主键。

club_id 或许属于这张表（如果它具有一对一的关系），但 *club_state* 绝对不该放在这里。即使如此，表中的列还是没有部分函数依赖性。

磨笔上阵 解答

重新设计下列表，改成三张都符合 2NF 的表。

其中一张表用来记录玩具，另一张用来记录商店信息，最后一张则记录库存量并可连接到另外两张表。

最后，为合适的表添加如下列：phone、manager、cost及weight。你可能必须创建新的tog_ids。

toy_id	toy
5	whiffleball
6	frisbee
9	kite
12	yoyo

toy_id 0+	store_id 0+	color	inventory	store_address
5	1	white	34	23 Maple
5	3	yellow	12	100 E. North St.
5	1	blue	5	23 Maple
6	2	green	10	1902 Amber Ln.
6	4	yellow	24	17 Engleside
9	1	red	50	23 Maple
9	2	blue	2	1902 Amber Ln
9	2	green	18	1902 Amber Ln
12	4	white	28	17 Engleside
12	4	yellow	11	17 Engleside

组合主键是toy_id和store_id。

store _ inventory

toy_id 0+	store_id 0+	inventory
1	1	34
2	3	12
3	1	5
4	2	10
5	4	24
6	1	50
7	2	2
8	2	18
9	4	28
10	4	11

toy _ info

toy_id 0	toy	color	cost	weight
1	whiffleball	white	1.95	0.3
2	whiffleball	yellow	2.20	0.4
3	whiffleball	blue	1.95	0.3
4	frisbee	green	3.50	0.5
5	frisbee	yellow	1.50	0.2
6	kite	red	5.75	1.2
7	kite	blue	5.75	1.2
8	kite	green	3.15	0.8
9	yoyo	white	4.25	0.4
10	yoyo	yellow	1.50	0.2

store _ info

store_id 0	address	phone	manager
1	23 Maple	555-6712	Joe
2	1902 Amber Ln.	555-3478	Susan
3	100 E. North St.	555-0987	Tara
4	17 Engleside	555-6554	Gordon

第三范式（终于到了这一步）

因为本书的表通常会加上人工主键（artificial primary key），所以让表符合第二范式并不算特别困难。任何具有人工主键且没有组合主键的表都符合 2NF。

如果你的表有人工主键且没有组合主键，则符合2NF。

不过，我们必须要确认表是否符合 3NF：

第三范式，又称3NF：
规则一：符合2NF
规则二：没有传递函数依赖性

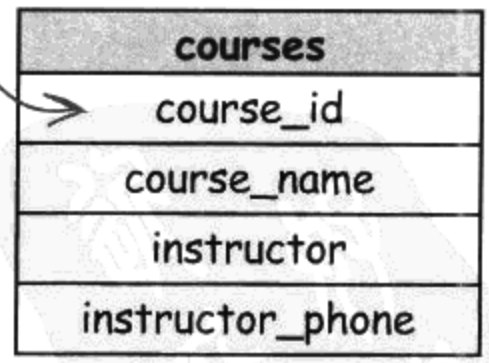
还记得吗？传递函数依赖表示任何非键列与相同表中的其他非键列有关联。

如果改变任何非键列可能造成其他任何非键列的改变，就是具有可传递依赖性。

如果修改了`course_name`、`instructor`、`instructor_phone`中任一列的值，请想想会有什么变化：

- 如果改变了`course_name`，`instructor`或`instructor_phone`就都没有改变的必要。
- 如果改变了`instructor_phone`，`instructor`或`course_name`就都没有改变的必要。
- 如果改变了`instructor`，`instructor_phone`也需要改变。找到了，这就是可传递依赖。

在考量3NF的关键时可以忽略主键。



现在应该很明显了，如果我们希望表符合3NF，就该把instructor_phone移到其他地方。

my_contacts 经得起考验吗？

my_contacts 的确需要一些改变。请利用本页的空白，以当前的 my_contacts 表为基础，画出新的 gregs_list 数据库模式（schema）。记得以单纯线段表示外键之间的关系，以箭头表示一对多关系，最后也要标出主键或组合键。

my_contacts
contact_id
last_name
first_name
phone
email
gender
birthday
profession
city
state
status
interests
seeking

提示：下一页的解答共有8张表（我们添加了邮编列，除此之外，共有7张表）。

my_contacts经得起考验吗？

my_contacts 的确需要一些改变。请利用本页的空白，以当前的 my_contacts 表为础，画出新的 gregs_list 数据库模式（schema）。记得以单纯线段表示外键之间的关系，以箭头表示一对多关系，最后也要标出主键或组合键。

my_contacts
contact_id
last_name
first_name
phone
email
gender
birthday
profession
city
state
status
interests
seeking

这三张表都有一对多关系。

这部分是多对多关系，现在由两个一对多关系加上一个连接表构成。

many-to-many

两个列构成组合主键。

one-to-many

one-to-many

one-to-many

在 contact_interest 中可能有很多相同的 interest_id，但在 interest 表中，这些 id 只会出现一次。

contact_interest
contact_id
0+
interest_id
0+

interests
interest_id
interest

my_contacts
contact_id
last_name
first_name
phone
email
gender
birthday
prof_id
zip_code
status_id

contact_seeking
contact_id
0+
seeking_id
0+

seeking
seeking_id
seeking

profession
prof_id
profession

zip_code
zip_code
city
state

status
status_id
status

one-to-many

one-to-many

两个列构成组合主键。

many-to-many

这部分是多对多关系，现在由两个一对多关系加上一个连接表构成。

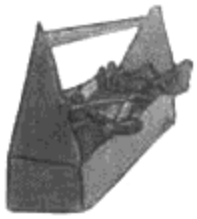
终于，Regis（还有 gregs_list）从此过着幸福美满的日子

使用经过规范化的数据库，Greg 终于为 Regis 找到了完美新娘。而且，他还能为更多好友更快地找到合适的对象，让他的“Greg's List 交友平台”不再只是梦想。

剧终

这就是联接登场的地方了。

下一章再会……



你的SQL工具包

为自己鼓掌吧！各位已经进入本书的后半部。请看你在第 7 章学到的 SQL 关键术语。如果需要本书工具的完整列表，请参考附录 3。

Schema

数据库模式。描述数据库中的数据、其他相关对象，以及这些对象相互连接的方式。

One-to-One relationship

一对一关系。父表中的一行记录只与子表中的一行记录相关联。

One-to-Many relationship

一对多关系。一张表中的一行记录可能与另一张表中的多行记录相关联，但后一张表中的任一行记录只会与前一张表中的一行记录相关联。

Many-to-Many relationship

多对多关系。两个通过 *junction table* 连接的表，让一张表中的多行记录能与另一张表中的多行记录相关联，反之亦然。

First normal form (1NF)

第一范式。列中只包含原子性数据，而且列内没有重复的数据组。

Transitive functional dependency

传递函数依赖。指任何非键列依赖于另一个非键列。

Second normal form (2NF)

第二范式。表必须先符合 1NF，同时不能包含部分函数依赖性，才算满足 2NF。

Third normal form (3NF)

第三范式。表必须先符合 2NF，同时不可包含可传递函数依赖。

Foreign key

外键。引用其他表的主键的列。

Composite key

组合键。由多个列构成的主键，这些列需形成唯一的键值。

磨笔上阵 解答

使用 ALTER 和 SUBSTRING_INDEX 函数把表修改成具有如下列。请把需要的查询都写出来。

```
contact _ id
last _ name
first _ name
phone
email
gender
birthday
profession
city
state
status
interest1
interest2
interest3
interest4
seeking
```

首先创建新的列：

```
ALTER TABLE my _ contacts
ADD COLUMN interest1 VARCHAR(50),
ADD COLUMN interest2 VARCHAR(50),
ADD COLUMN interest3 VARCHAR(50),
ADD COLUMN interest4 VARCHAR(50);
```

然后把第一项兴趣移至新的 *interest1* 列。移动方式如下：

```
UPDATE my _ contacts
SET interest1 = SUBSTRING _ INDEX(interests, ',', 1);
```

接下来要从原始的 *interest* 字段中移除第一项兴趣，因为它已经保存在 *interest1* 列中了。我们利用字符串函数移除第一个逗号左侧的所有内容：

在移除逗号前的所有内容后（左侧），TRIM 能清除字符串前的所有空格。

RIGHT 返回 *interest* 列的部分内容，从字符串右端（尾端）开始返回。

```
UPDATE my _ contacts SET interests = TRIM(RIGHT(interests,
(LENGTH(interests)-LENGTH(interest1) - 1)));
```

看起来让人头昏眼花的这个部分负责计算需要的兴趣列。它接收 *interests* 列的总长度，然后减去我们移到 *interest1* 的长度。最后再减 1，才会从逗号后开始。

之后，对 *interest* 列的其余部分重复上述步骤：

```
UPDATE my _ contacts SET interest2 = SUBSTRING _ INDEX(interests, ',', 1);
UPDATE my _ contacts SET interests = TRIM(RIGHT(interests, (LENGTH(interests)-
LENGTH(interest2) - 1)));
UPDATE my _ contacts SET interest3 = SUBSTRING _ INDEX(interests, ',', 1);
UPDATE my _ contacts SET interests = TRIM(RIGHT(interests, (LENGTH(interests)-
LENGTH(interest3) - 1)));
```

在最后一列中，我们只会得到单一值：

```
UPDATE my _ contacts SET interest4 = interests;
```

现在终于可以完全删除 *interests* 列了。我们也可将它重新命名为 *interest4*，不需要多加一次 ADD COLUMN（假设最多只有四项兴趣）。

习题解答

为 Regis 设计专属查询，但不使用 interests 列。

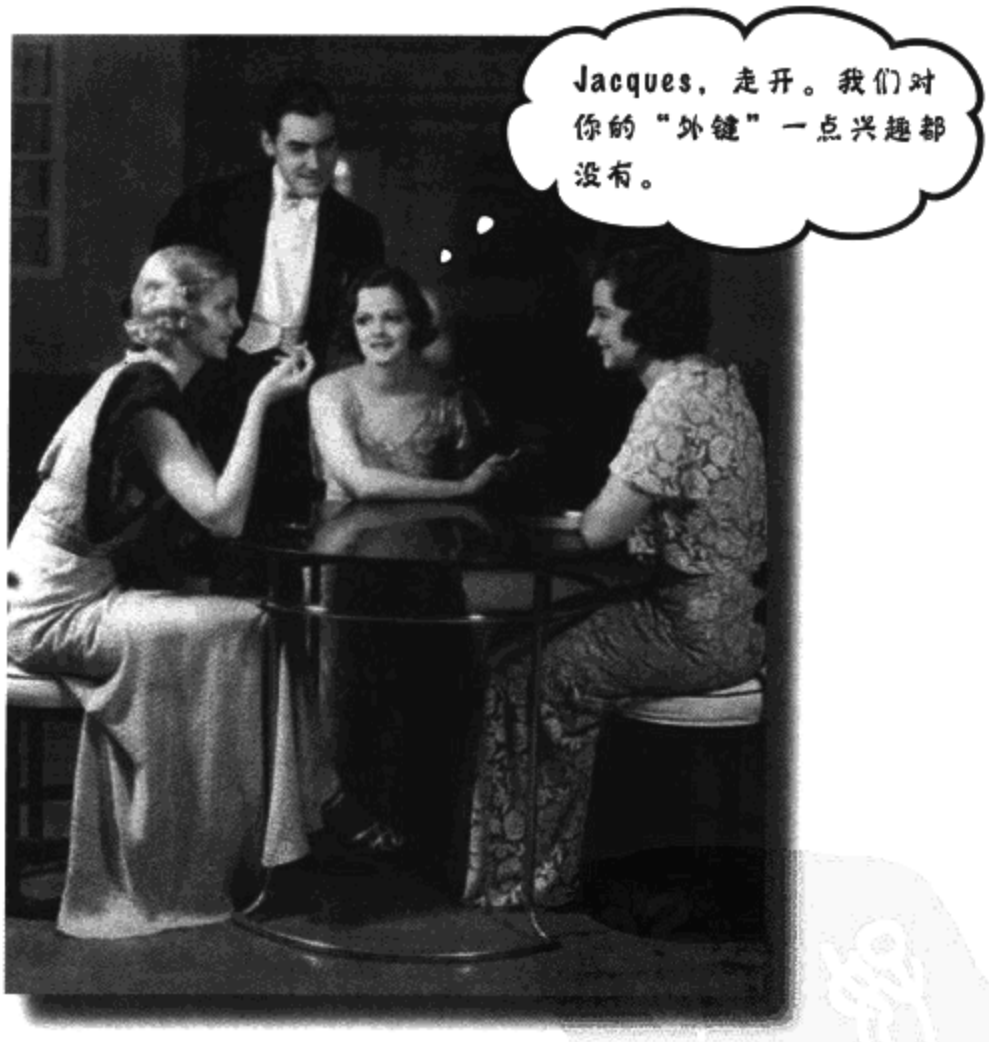
第286页上的习题。

```
SELECT * FROM my_contacts
WHERE gender = 'F'
AND status = 'single'
AND state='MA'
AND seeking LIKE '%single M%'
AND birthday > '1950-03-20'
AND birthday < '1960-03-20';
```

其实这个查询与 Greg 最初为 Nigel 设计的查询大致一样，只是没有加入兴趣条件。

8 联接与多张表的操作

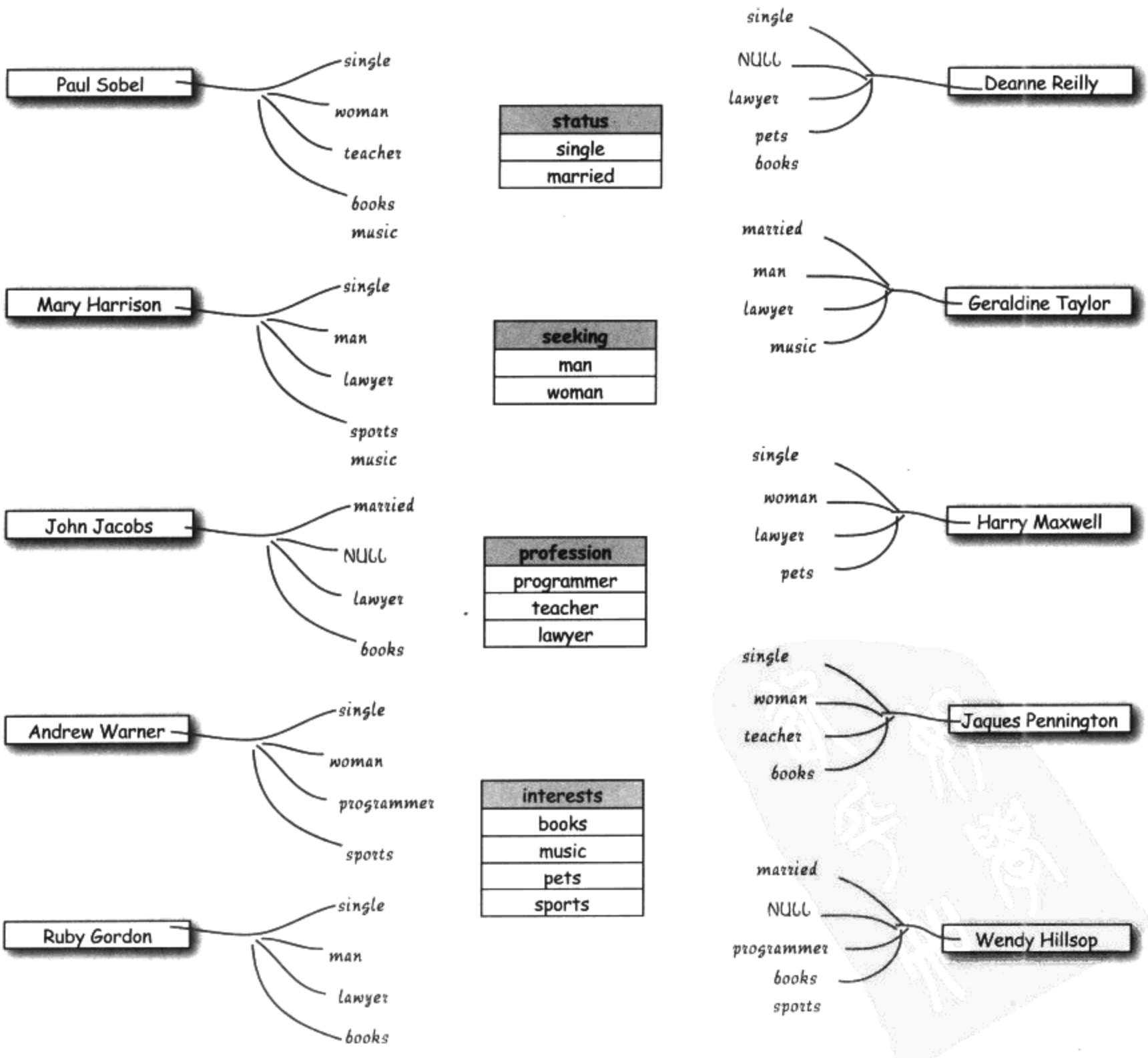
不能单独存在吗？



欢迎来到多张表的世界。数据库中有**多张表**是件好事，但我们也需要学习一些操控多张表的**新技术与工具**。混乱状态与多张表一起出现，所以你需要**别名**来让表更清楚简单。**联接**则有助于联系表，取得分布在各张表里的内容。准备好，再度**取回数据库的控制权**吧！

自我重复、自我重复……

Greg 发现，在 status、profession、interests、seeking 中都有相同的值一再出现。



预填充表

表中有很多重复的值，表示 status、profession、interests、seeking 的值适合预填充（prepopulating）。Greg 想把旧的 my_contacts 表中的内容载入这4张表中。

首先，Greg 需要查询表以找出已有的数据值，但他不想要一长串一再重复值的超级详细列表。

磨笔上阵

设计一段从 my_contacts 表中检索 status、profession、interests、seeking 值的查询，但是要把数据值过滤成不重复的形式。可参考第6章中关于 Girl Sprout 饼干销售问题的说明。

磨笔上阵解答

设计一段从 my_contacts 表中检索 status、profession、interests、seeking 值的查询，但是要把数据值过滤成不重复的形式。可参考第6章中关于 Girl Sprout 饼干销售问题的说明。

```
SELECT status FROM my_contacts
GROUP BY status
ORDER BY status;
```

```
SELECT profession FROM my_contacts
GROUP BY profession
ORDER BY profession;
```

使用 GROUP BY 把重复的数据值合成一个组。

然后使用 ORDER BY 整理出按照字母顺序排列的列表。

```
SELECT seeking FROM my_contacts
GROUP BY seeking
ORDER BY seeking;
```

如果不按照这里的顺序执行，你就会得到错误信息。ORDER BY 永远都要最后出现。

```
SELECT interests
FROM my_contacts
GROUP BY interests
ORDER BY interest;
```

我们无法只靠单一简单的 SELECT 来取出所有兴趣。

若对兴趣列使用 SELECT 语句，你会无法处理如下表中的兴趣：

interests
books, sports
music, pets, books
pets, books
sports, music

我得了“表难以规范化”的忧郁症

就像尝到了甜头的小狗，一旦看到兴趣列这种没被规范化的设计，我们就开始觉得不舒服。若把这些值抽离出兴趣列，又没办法一次检索出所有兴趣。

我们需要从这种形式

interests
first, second, third, fourth

my_contacts 的某一列。

变成这种形式

interests
first
second
third
fourth

新*interests*表的某一列。

动动脑

如何把这些兴趣都存储在 interests 表的某一列里呢？

首先，上述提议将是一项浩大的工程。如果表已有成千上万条记录，你试想一下要做多久？

而且，手动输入会难以找出重复的数据。当联络人提供了几百种兴趣，每次输入新的兴趣时，你都得先查看以前是否记录过相同的数据。

与其自己辛苦动手做，又要冒着打错字的危险，不如交给 SQL 代劳。

特殊的兴趣列

直接在my_contacts里加入新列，用来暂时存储从原始兴趣列抽出的值，算是容易想到的解决方法。兴趣值都存储完毕后，即可删除这些临时列。

磨笔上阵

各位现在都已知道如何修改表，所以请用 ALTER 修改 my_contacts，为它添加4个新列，分别命名为 interest1、interest2、interest3、interes4。

答案请见第 378 页。

下表是对 my_contacts 运行 ALTER 后新旧兴趣列的示意图。

interests	interest1	interest2	interest3	interest4
first, second, third, fourth				

利用第 5 章的 SUBSTRING_INDEX 函数，即可轻易地复制第一项兴趣并将其存储到 interest1 列：

```
UPDATE my _ contacts
SET interest1 = SUBSTRING _ INDEX(interests, ',', 1);
```

要截取数据的列名

要查找的分隔字符，本处为逗号

……查找第一个逗号

运行上述查询，结果如下：

interests	interest1	interest2	interest3	interest4
first, second, third, fourth	first			

保存兴趣

接下来是麻烦的部分：要用另一个子字符串函数把interests列中已经被我们存储到interest1列的数据移除。然后才能以相同方式填充其他兴趣列。

interests	interest1	interest2	interest3	interest4
first, second, third, fourth	first			

现在要从 *interests* 列中移除第一项兴趣以及它后面的逗号和空格。

我们会使用SUBSTR函数，它能抓取兴趣列中的字符串并返回部分字符串值。

下例查询的翻译：把 *interests* 列的值改变为这个查询指定的任何内容，但要去除 *interest1* 列存储的值、有逗号与空格。

Interest1 字段中存储的文本字符串的长度……

……再加上 2：逗号与空格的长度。

```
UPDATE my_contacts
SET interests = SUBSTR(interests, LENGTH(interest1)+2 );
```

SUBSTR 返回本列内原始字符串的一部分。它接受字符串并切除我们用括号指定的第一部分，然后返回剩下的部分。

LENGTH 返回括号中参数字符串的长度。

本例中，字符串 "first" 的长度为5个字符。

所以，由 LENGTH 返回的数字是 5 再加 2，也就是 7，所以会从原本的 *interests* 列的左侧（字符串开始处）开始移除 7 个字符。

有些函数会因为每个人使用的 SQL 产品不同而略有差异，还记得这点吗？SUBSTR 也是其中之一。请参考非常有用的参考书籍——例如 O'Reilly 出版的《SQL 技术手册》，寻找你习惯的 SQL 产品。

UPDATE所有兴趣列

运行UPDATE语句后，表会如下所示。但工作还没完成，interest2、interest3、interes4 列都需要相同的处理。

interests	interest1	interest2	interest3	interest4
second, third, fourth	first			

磨笔上阵

补齐下列 UPDATE 语句，帮 Greg 完成查询。旁边附有提示。

提示：interests列每次都会改变，因为每次执行后，它存储的字符串值都会被SUBSTR截短一点。

```
UPDATE my_contacts SET
interest1 = SUBSTRING_INDEX(interests, ',', 1),
interests = SUBSTR(interests, LENGTH(interest1)+2),
interest2 = SUBSTRING_INDEX(.....),
interests = SUBSTR(.....),
interest3 = SUBSTRING_INDEX(.....),
interests = SUBSTR(.....),
interest4 = .....
```

在移除前三项兴趣后，interests列只剩下第四项兴趣。此时该如何处理？

执行过上述的复杂查询后，请在下表中填入执行结果。

interests	interest1	interest2	interest3	interest4
second, third, fourth	first			

答案请见第378页。

取得所有兴趣

终于，每一项兴趣都存储在不同列中了。使用简单的 SELECT 语句即可看到所有兴趣，但无法同时取得它们。而且我们想把所有兴趣整理成单一结果集也并非简单的事。我们的尝试结果是：

```
File Edit Window Help TooManyColumns
> SELECT interest1, interest2, interest3, interest4 FROM my _ contacts;
+-----------+-----------+-----------+-----------+
| interest1 | interest2 | interest3 | interest4 |
+-----------+-----------+-----------+-----------+
|  first    |  second   |  third    |  fourth   |
|  horses   |  pets     |           |           |
|  music    |  fishing  |  books    |  movies   |
|  painting |           |           |           |
|  horses   |  pets     |           |           |
|  music    |  sports   |  books    |  boating  |
|  travel   |  music    |           |           |
|  horses   |  pets     |           |           |
|  music    |  sports   |  books    |  knitting |
|  pets     |  writing  |  travel   |           |
|  dogs     |  hiking   |           |           |
|  movies   |  sports   |           |           |
+-----------+-----------+-----------+-----------+
```

至少我们还可以分开编写4条 SELECT 语句来取得所有兴趣值：

```
SELECT interest1 FROM my _ contacts;  SELECT interest3 FROM my _ contacts;
SELECT interest2 FROM my _ contacts;  SELECT interest4 FROM my _ contacts;
```

我们现在缺少接受这些 SELECT 语句并把内容直接填入新表的工具。我们至少有三种能选择的达成这个目标的方式！

回家试试看

看一下你在第 345 页为 profession 列设计的 SELECT 语句：

```
SELECT profession FROM my_contacts GROUP BY profession
ORDER BY profession;
```

下一页我们会解说三种利用这里的 SELECT 语句的方式，以便给新的兴趣表填入现有内容。

尽情地利用 SELECT、INSERT、CREATE，看看各种查询结果。然后翻到下一页，研究我们提供的三种方式。

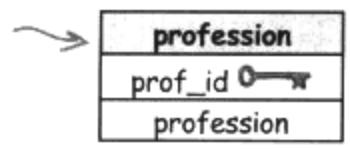
重点不是一次就做对事情，而是思考一下现有工具的潜力。

条条大路通罗马

对于疯狂小丑而言，同样的事情可以有三种方式来完成或许的确非常有趣，但对我们一般人而言可能就有点头大了。

不过，这么多选择自然有它们的用处。当我们知道有三种方式可以做同一件事情时，也就表示可以选用最适合自己的方式。随着数据的增加，某些查询可能表现得比其他方式快（根据各人使用的RDBMS而定）。当表变得非常庞大后，你肯定希望优化你的查询，此时，知道各种达成目标的方式会有助于最优化的工作。

接下来的几页，就是创建这个表，并填满没有重复、按字母顺序排列的内容的三种方式。



同时（几乎同时啦）CREATE、SELECT、INSERT

1. CREATE TABLE，然后利用 SELECT 进行 INSERT

我们已经知道这种方式了！首先创建（CREATE）profession表，然后填入第 345 页上的 SELECT 的查询结果。

```
CREATE TABLE profession
(
  id INT(11) NOT NULL AUTO_INCREMENT PRIMARY KEY,
  profession varchar(20)
);

INSERT INTO profession (profession)
  SELECT profession FROM my_contacts
  GROUP BY profession
  ORDER BY profession;
```

创建带有主键列的profession表，并用VARCHAR类型的列存储职业。

现在以SELECT的查询结果填满profession表的profession列。

2. 利用 SELECT 进行 CREATE TABLE，然后 ALTER 以添加主键

第二种方式：利用SELECT从my_contacts表的职业列抓出来的数据创建新的profession表，再用 ALTER 修改新表并添加（ ADD）主键字段。

创建只有一列的 *profession* 表，并填入 SELECT 的查询结果……

```
CREATE TABLE profession AS
  SELECT profession FROM my _ contacts
  GROUP BY profession
  ORDER BY profession;
```

……然后用 ALTER 修改表以添加主键字段。

```
ALTER TABLE profession
ADD COLUMN id INT NOT NULL AUTO _ INCREMENT FIRST,
ADD PRIMARY KEY (id);
```

同一时间 CREATE、SELECT、INSERT

3. CREATE TABLE 的同时设置主键并利用 SELECT 填入数据

这是只需一个步骤的方式：创建profession表的同时设置主键列以及另一个VARCHAR类型的列来存储职业，同时还要填入SELECT 的查询结果。SQL 具有 AUTO_INCREMENT 功能，所以 RDBMS 知道 ID 列需要自动填入，因此只剩一列，也就是 SELECT 的数据应该填入的地方。

创建 *profession* 表时一并创建主键与 *profession* 列，并以 SELECT 的查询结果填满 *profession* 列。

```
CREATE TABLE profession
(
  id INT(11) NOT NULL AUTO _ INCREMENT PRIMARY KEY,
  profession varchar(20)
) AS
  SELECT profession FROM my _ contacts
  GROUP BY profession
  ORDER BY profession;
```

没错。关键字 AS 的作用正是这样。

一切都与别名（aliasing）有关，接下来我们就要进入这个主题了！

AS到底是怎么一回事？

AS能把SELECT的查询结果填入新表中。我们在第二和第三个范例中使用　AS时，其实是要求软件把来自my_contacts表的内容当成SELECT的查询结果，并把结果值存入新建的profession表中。

如果不指定新表具有带有新名称的两列，AS只会创建一列，且该列的列名及数据类型与SELECT的查询结果相同。

如果我们没有给新表设计两个列，AS只会创建一列，并采用与SELECT的查询结果相同的列名与数据类型。

我们在新表里创建了一个VARCHAR列，命名为profession。

```
CREATE TABLE profession
(
  id INT(11) NOT NULL AUTO _ INCREMENT PRIMARY KEY,
  profession varchar(20)
) AS
SELECT profession FROM my _ contacts
GROUP BY profession
ORDER BY profession;
```

这个.)..)的关键字有很大的作用。它就像隧道一样，把所有SELECT的查询结果输出到新的表中。

这些都是my_contacts表的profession列，都是SELECT的一部分。

既然我们创建了职业表，而且有AUTO_INCREMENT主键，所以只要插入第二列的值就够了，这一列名为profession。

我都搞糊涂了。一个查询中“profession”出现了5次。SQL软件或许能区分具体的profession，但我该怎么分辨？

所以SQL才会提供别名功能，以免各种名称把我们搞糊涂。

这只是SQL允许我们暂时对表与列赋予新名称的原因之一，这项功能称为**别名**（alias）。

列的别名

创建别名真的很简单。在查询中首次使用原始列名的地方后接 AS 并设定要采用的别名，告诉软件现在开始以另一个名称引用my_contacs表的profession列，这样可以让查询更容易被我们理解。

我们将由my_contacts表选取的职业值称为**mc_prof**（mc是my_contacts的缩写）。

```
CREATE TABLE profession
(
  id INT(11) NOT NULL AUTO_INCREMENT PRIMARY KEY,
  mc_prof varchar(20)
) AS
  SELECT profession AS mc_prof FROM my_contacts
  GROUP BY mc_prof
  ORDER BY mc_prof;
```

查询里首次提到原始列名的地方后接 AS 并设定别名，让软件从这里开始用别名引用数据。

这个查询与原先的查询效果完全一样，但因为有别名，所以它更容易理解。

这两个查询有一点小小的不同。所有查询都以表的形式返回。别名改变了查询结果中的列名，但并未改变来源列的名称。别名只是临时的。

但因为指定了新表有两列——主键和职业列相当于覆盖了原始查询结果，所以新表还是会有名为 profession 的列，而非mc_prof。

表的别名，谁会需要？

你会需要！我们很快就要进入联接（join）领域，一个从多张表里选取数据的世界。若是没有别名，众多一再出现的表名会把我们搞得头昏眼花。

创建表别名的方式与创建列别名的方式几乎一样。在查询中首次出现表名的地方后接 AS 并设定别名，告诉软件现在开始以 mc 引用原有的 my_contacts。

表别名又称为 correlation name（相关名称）。

```
SELECT profession AS mc_prof
FROM my_contacts AS mc
GROUP BY mc_prof
ORDER BY mc_prof;
```

创建表别名的方式几乎与创建列别名的方式相同。

不用，另有设定别名的简短方式。

可以省略 AS。下列查询也会得到与上例查询相同的结果。

这两个查询的结果并无不同。

```
SELECT profession mc_prof
FROM my_contacts mc
GROUP BY mc_prof
ORDER BY mc_prof;
```

我们拿掉了AS。只要别名紧接在原始表名或列名后，就能直接设定别名。

关于内联接的二三事

如果各位曾经听过关于SQL的讨论，可能常常会听到“联接”（join）这个词。不过，联接并没有你想象的那么复杂。接下来我们就要开始了解联接，研究它的运作方式，并提供很多机会，让大家研究使用联接的时机以及应该使用何种联接。

不过，开始联接的深度之旅前，我们先看看最简单的联接（它甚至不算真正的联接）。

它有很多不同的名称。本书称之为交叉联接（cross join）（译注1），但各位或许也听过它的其他名字，例如笛卡尔积、交叉积……还有最奇怪的“没有联接”（no join）。

▶ 译注1：为统一本章末及附录3的用词，故修改原文的“Cartesian Join”为“cross join”。

假设你有一个存储男孩姓名的表及一个记录男孩们分别拥有哪些玩具的表。现在我们要试着找出每个男孩拥有的玩具。

toys

toy_id	toy
1	hula hoop
2	balsa glider
3	toy soldiers
4	harmonica
5	baseball cards

boys

boy_id	boy
1	Davey
2	Bobby
3	Beaver
4	Richie

交叉联接

下例同时查询玩具表的 toy 列与男孩表的 boy 列，这个方法的查询结果会是交叉联接。

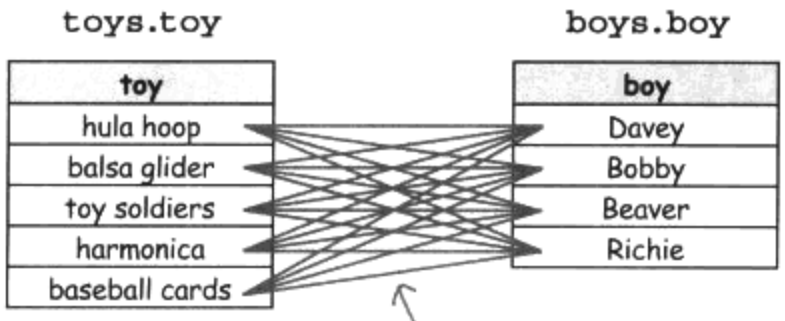
```
SELECT t.toy, b.boy
FROM toys AS t
  CROSS JOIN
  boys AS b;
```

还记得上一章提过的速记符号吗？点号前是表名，点号后是表内的列名。只是这里以别名代替表的全名。

这一行的意思是：从 boy 表里 SELECT 'boy' 列、从 toy 表里 SELECT "toy" 列。至于查询的其他部分，则会把查询结果联接成一张新的结果表。

这里也使用表别名。

交叉联接把第一张表的每个值都与第二张表的每个值配成对。



CROSS JOIN返回两张表的每一行相乘的结果。

这些线显示本例的联接结果。每种玩具与每个男孩的搭配。没有重复的数据。

本例联接出 20 条记录。5 个玩具乘以 4 个男孩的结果呈现了所有可能的组合。

因为 toys.toy 的查询结果比较多，所以结果呈现如右表所示的组。如果男孩有 5 个，玩具只有 4 种，则会以男孩姓名为划分组的依据。但请记住，结果的顺序在这个查询里没有意义。

toy	boy
hula hoop	Davey
hula hoop	Bobby
hula hoop	Beaver
hula hoop	Richie
balsa glider	Davey
balsa glider	Bobby
balsa glider	Beaver
balsa glider	Richie
toy soldiers	Davey
toy soldie	Bobby

问： **我为什么需要交叉联接？**

答： 知道这一点很重要，因为当我们乱玩联接时可能意外造成交叉联接。知道交叉联接的存在有益于找出修正联接的方式。这种事情有时真的会发生。还有，交叉联接有时可用于检测 RDBMS 软件及其配置的运行速度。运行交叉联接所需的时间可以轻易地检测与比较出速度慢的查询。

问： **如下所示，假设改用 SELECT * 写查询，会有什么不同？**

```
SELECT * FROM toys CROSS JOIN boys;
```

答： 你可以动手试试看。不过还是会得出20行，只不过会包含 4 个数据列。

内联接就是通过查询中的条件移除了某些结果数据行后的交叉连接。

问： **如果对两个很大的表做了交叉联接，会发生什么事？**

答： 查询结果将会非常的庞大。最好别对数据量大的表进行交叉联接，否则会因为返回的数据过多而让机器冒着停滞不动的风险。

问： **这个查询还有其他同义语法吗？**

答： 当然有！CROSS JOIN可以省略不写，只用逗号代替，就像这样：

```
SELECT toys.toy, boys.boy
FROM toys, boys;
```

问： **我听说过“内联接”与“外联接”这两个词，交叉联接是相同的东西吗？**

答： 交叉联接是内联接的一种。内联接基本上就是通过查询中的条件移除了某些结果数据行后的交叉联接。再过几页我们就要讨论内联接，记住这一点！

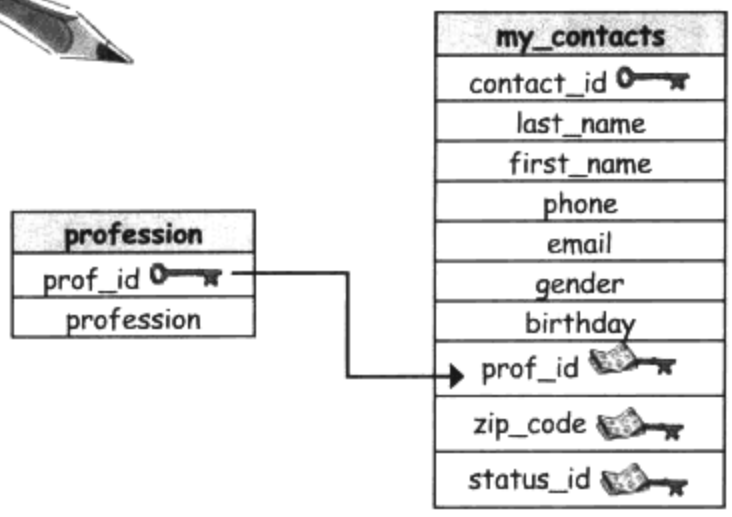
脑力锻炼

你觉得下列查询会产生什么结果？

```
SELECT b1.boy, b2.boy
FROM boys AS b1 CROSS JOIN boys AS b2;
```

动手试试看。

磨笔上阵



这里有 gregs_list 数据库中的两张表：profession 与 my_contacts。请观察我们提供的查询，并在旁边的空白处写下每一行的用途。

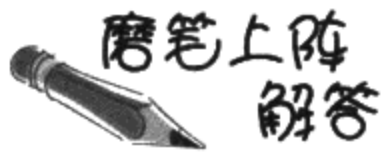
```
SELECT mc.last _ name,
mc.first _ name,
p.profession
FROM my _ contacts AS mc
  INNER JOIN
  profession AS p
ON mc.prof _ id = p.prof _ id;
```

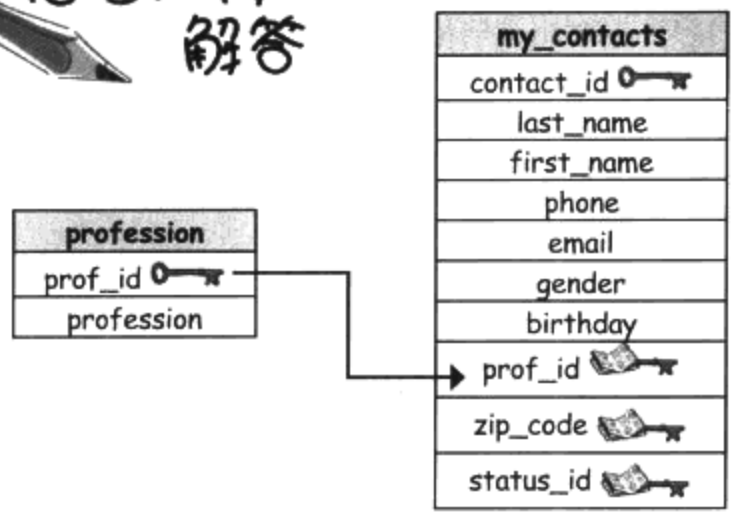
假设左页上的表中存储了以下三张便笺的内容，请利用这些信息画出查询结果表。

Joan Everett
single
3-4-1978
Salt Lake City, UT
Artist
Female
jeverett@mightygumball.net
sailing, hiking, cooking
555 555-9870

Tara Baldwin
married
1-9-1970
Boston, MA
Chef
female
tara@breakneckpizza.com
movies, reading, cooking
555 555-3432

Paul Singh
married
10-12-1980
New York City, NY
Professor
male
ps@tikibeanlounge.com
dogs, spelunking
555 555-8222





这里有 gregs_list数据库中的两张表：profession 与 my_contacts。请观察我们提供的查询，并在旁边的空白处写下每一行的用途。

```
SELECT mc.last _ name,
mc.first _ name,
p.profession
FROM my _ contacts AS mc
  INNER JOIN
  profession AS p
ON mc.prof _ id = p.prof _ id;
```

从 *my_contacts* 表（别名为 *mc*）中选取出 *last_name* 列

也选取 *my_contacts* 表中的 *first_name* 列

还要选取 *profession* 表（别名为 *p*）中的 *profession* 列

前述 SELECT 的选取对象是 *my_contact*表（别名为 *mc*）

使用 INNER JOIN ……

…… 联接 *profession*表（别名为 *p*）后的 SELECT 结果

联接的条件是用*my_contacts* 和 *profession* 表的 *prof_id* 字段找出的相符记录

假设表中存储了前页上的三张便笺的内容，请利用这些信息画出查询结果表。

last_name	first_name	profession
Everett	Joan	artist
Singh	Paul	professor
Baldwin	Tara	chef

释放你的内联接

我知道了！这就是如何把所有新表与新的 my_contacts 连接的方式。我不再需要编写许多 SELECT，只要用那个 INNER JOIN 连接所有表，一切就大功告成了！

要学的事情还很多。

各位现在看到的只不过是联接种类中的一小部分。关于内联接以及其他联接，还有很多要学的事，然后你才能适当且有效地运用这项新技巧。

INNER JOIN 利用条件判断中的比较运算符结合两张表的记录。只有联接记录符合条件时才会返回列。接下来，让我们仔细观察语法。

```
SELECT somecolumns
FROM table1
  INNER JOIN
  table2
ON somecondition;
```

我们需要的列。

这里省略指定别名的部分，让语法简单一点。

也可改用关键字 WHERE。

条件式里可采用任何一个比较运算符。

INNER JOIN利用条件式里的比较运算符结合两张表

内联接上场了：相等联接（equijoin）

观察下列表。每个男孩都只有一个玩具，表之间为一对一关系，`toy_id` 是外键。

boys

boy_id	boy	toy_id
1	Davey	3
2	Bobby	5
3	Beaver	2
4	Richie	1

toys

toy_id	toy
1	hula hoop
2	balsa glider
3	toy soldiers
4	harmonica
5	baseball cards

我们只是想找出每个男孩拥有什么玩具。我们可以使用内联接加上相等运算符（=），用boys表中的外键`toy _ id`与`toys`表里的主键进行比对，看看会找出什么玩具。

```
SELECT boys.boy, toys.toy
FROM boys
  INNER JOIN
  toys
ON boys.toy _ id = toys.toy _ id;
```

EQUIJOIN

测试相等性的内联接

boys

boy_id	boy	toy_id
1	Davey	3
2	Bobby	5
3	Beaver	2
4	Richie	1

toys

toy_id	toy
1	hula hoop
2	balsa glider
3	toy soldiers
4	harmonica
5	baseball cards

我们的查询结果表。有需要时可以加上 ORDER BY boys.toy。

boy	toy
Richie	hula hoop
Beaver	balsa glider
Davey	toy soldiers
Bobby	baseball cards

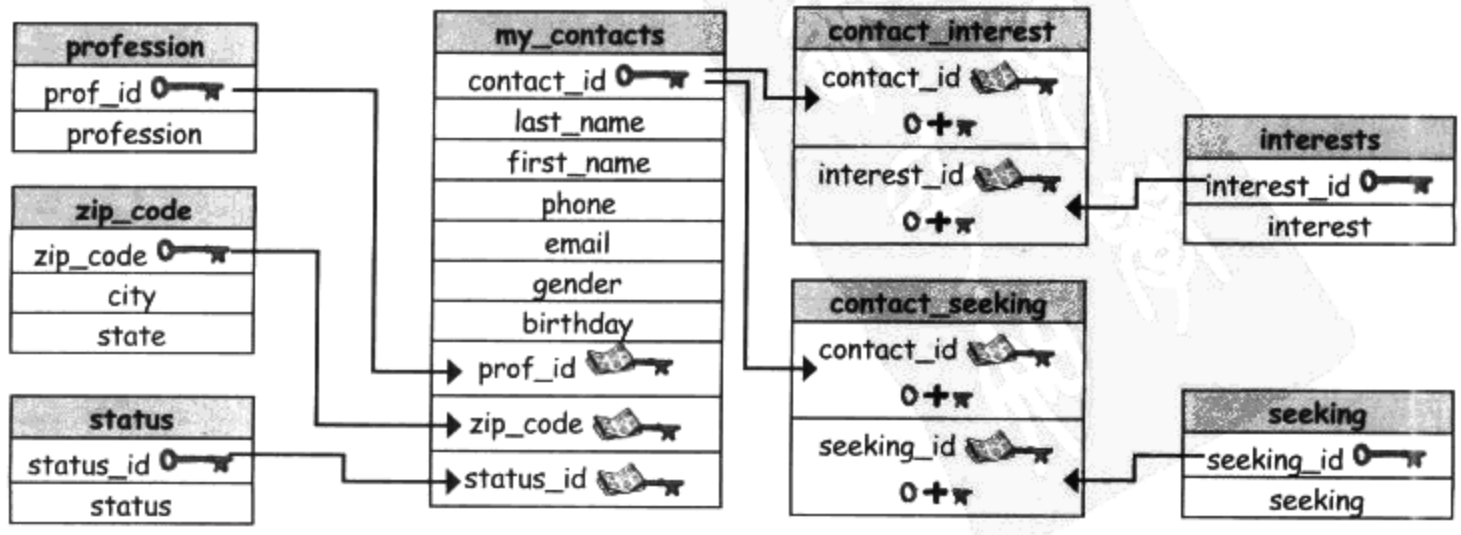
磨笔上阵

为 gregs_list 表设计 equijoin 查询。

返回 my_contacts 表中每个人的电子邮件地址与职业的查询。

返回 my_contacts 表中每个人的姓、名与婚姻状况的查询。

返回 my_contacts 表中每个人的姓、名与所在位置（州名）的查询。



磨笔上阵解答

为 gregs_list 表设计 equijoin 查询。

返回 my_contacts 表中每个人的电子邮件地址与职业的查询。

```
SELECT mc.email, p.profession FROM my_contacts mc
INNER JOIN profession p ON mc.prof_id = p.prof_id;
```

外键 prof_id 连接至 profession 表中的 prof_id。

返回 my_contacts 表中每个人的姓、名与婚姻状况的查询。

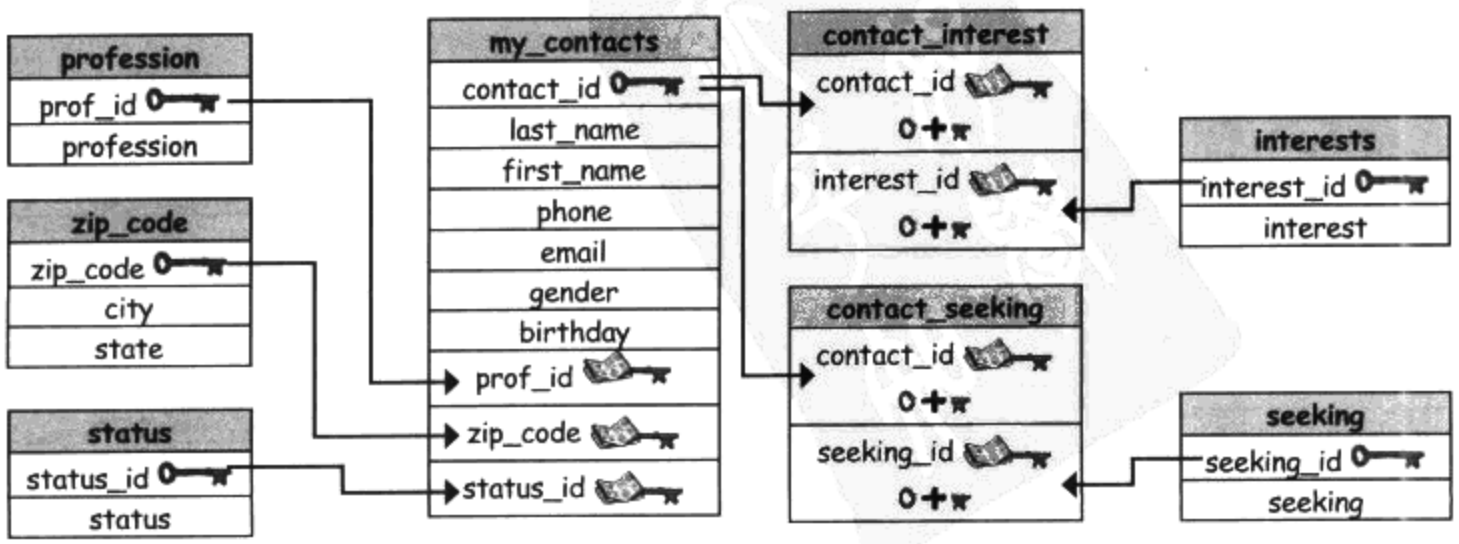
```
SELECT mc.first_name, mc.last_name, s.status FROM my_contacts mc
INNER JOIN status s ON mc.status_id = s.status_id;
```

外键 status_id 连接至 status 表中的 status_id。

返回 my_contacts 表中每个人的姓、名与所在位置（州名）的查询。

```
SELECT mc.first_name, mc.last_name, z.state FROM my_contacts mc
INNER JOIN zip_code z ON mc.zip_code = z.zip_code;
```

这次使用 zip_code 作为连接两张表的键。



内联接上场了：不等联接（non-equijoin）

不等联接则返回任何不相等的记录。仍然以 boys 和 toys 两张表为例，使用不等联接，我们可以看出每个男孩没有的玩具（在他们生日时或许会很有用）。

```
SELECT boys.boy, toys.toy
FROM boys
  INNER JOIN
  toys
ON boys.toy _ id <> toys.toy _ id
ORDER BY boys.boy;
```

排列查询结果的顺序会使结果更容易阅读。

不等于运算。这是联接的"non-equi"部分。

boys

boy_id 0🔑	boy	toy_id 🔑
1	Davey	3
2	Bobby	5
3	Beaver	2
4	Richie	1

toys

toy_id 0🔑	toy
1	hula hoop
2	balsa glider
3	toy soldiers
4	harmonica
5	baseball cards

boy	toy
Beaver	hula hoop
Beaver	toy soldiers
Beaver	harmonica
Beaver	baseball cards
Bobby	toy soldiers
Bobby	harmonica
Bobby	hula hoop
Bobby	balsa glider
Davey	hula hoop
Davey	balsa glider
Davey	harmonica
Davey	baseball cards
Richie	balsa glider
Richie	toy soldiers
Richie	harmonica
Richie	baseball cards

这4个是Beaver没有的玩具。

NON-EQUIJOIN

测试不相等性的内联接

最后一种内联接：自然联接（natural join）

只剩下最后一种内联接，那就是自然联接。自然联接只有在联接的列在两张表中的名称都相同时才会有用。仍然以这两张表为例。

相同的列名

boys

boy_id	boy	toy_id
1	Davey	3
2	Bobby	5
3	Beaver	2
4	Richie	1

toys

toy_id	toy
1	hula hoop
2	balsa glider
3	toy soldiers
4	harmonica
5	baseball cards

和前面的例子一样，我们想知道每个男孩拥有什么玩具。我们的自然联接会识别出每个表里的相同名称并返回相符的记录。

```
SELECT boys.boy, toys.toy
FROM boys
  NATURAL JOIN
  toys;
```

boys

boy_id	boy	toy_id
1	Davey	3
2	Bobby	5
3	Beaver	2
4	Richie	1

toys

toy_id	toy
1	hula hoop
2	balsa glider
3	toy soldiers
4	harmonica
5	baseball cards

我们取得的结果集，与第一种内联接（相等联接）范例的结果完全一样。

boy	toy
Richie	hula hoop
Beaver	balsa glider
Davey	toy soldiers
Bobby	harmonica

NATURAL JOIN
利用相同列名的内联接

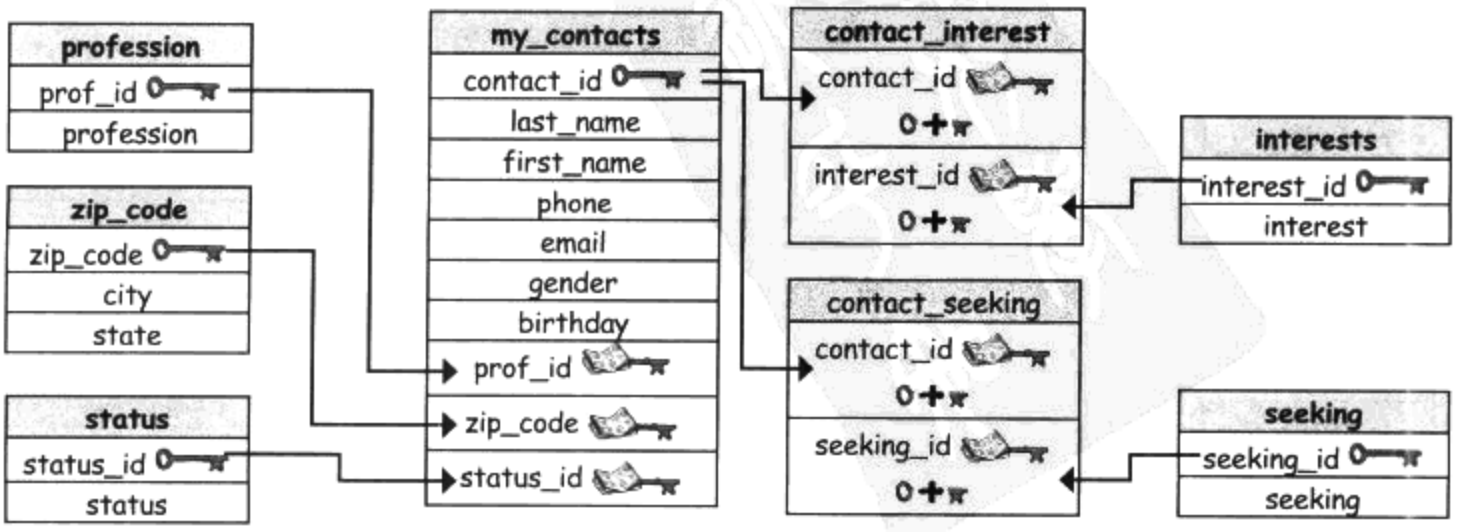
磨笔上阵

请为 gregs_list 数据库设计自然联接或不等联接的查询。

返回 my_contacts 表中每个人的电子邮件地址与职业的查询。

返回 my_contacts 表中每个人的姓、名与他们没有的状态的查询。

返回 my_contacts 表中每个人的姓、名与所在位置（州名）的查询。



请为 gregs_list 数据库设计自然联接或不等联接的查询。

返回 my_contacts 表中每个人的电子邮件地址与职业的查询。

```
SELECT mc.email, p.profession FROM my_contacts mc
NATURAL JOIN profession p;
```

返回 my_contacts 表中每个的人的姓、名与他们没有的状态的查询。

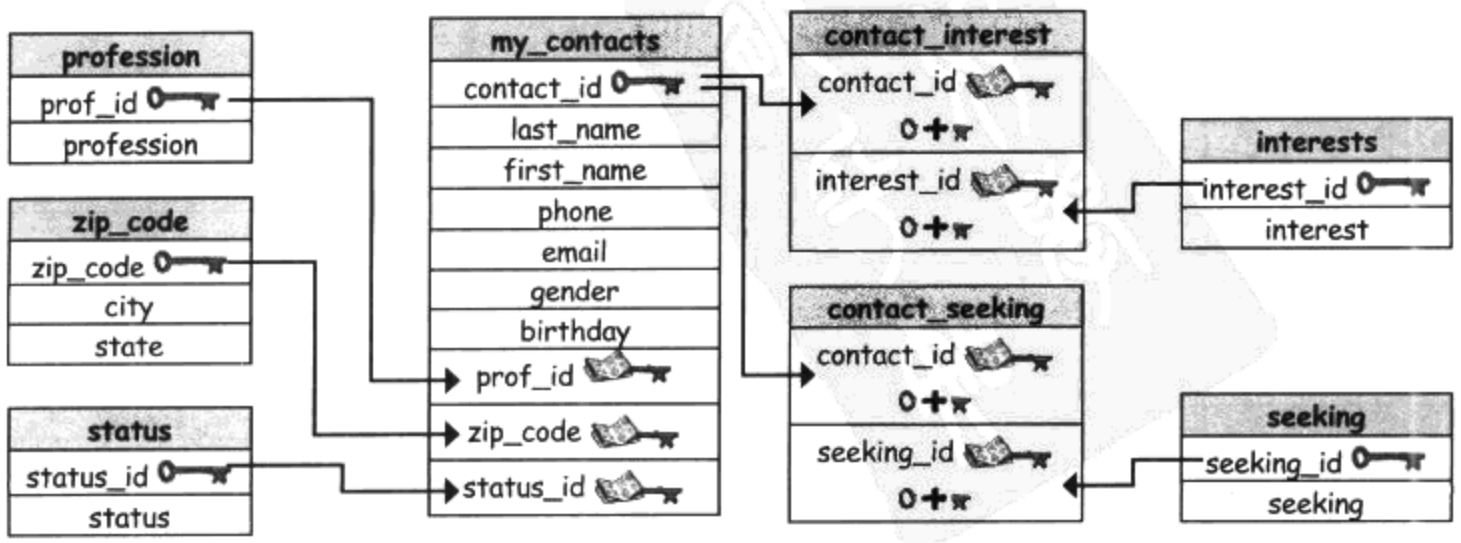
```
SELECT mc.first_name, mc.last_name, s.status FROM my_contacts mc
INNER JOIN status s ON mc.status_id <> s.status_id;
```

这会返回每个人的数条记录，记录里包含的状态是并未链接至status_id的状态。

返回 my_contacts 里每个人的姓、名与所在位置（州名）的查询。

```
SELECT mc.first_name, mc.last_name, z.state FROM my_contacts mc
NATURAL JOIN zip_code z;
```

第一个与第三个查询不需要 ON，因为这两张表的外键和主键名称相符合。



连连看

为每个联接与说明连线。可能会有多种联接匹配说明。

自然联接（natural join）

相等联接（equijoin）

交叉联接（cross join）

外联接（outer join）

不等联接（non-equijoin）

内联接（inner join）

笛卡尔联接（Cartesian join）

叉积（cross product）

我返回两张表里联接列内容不符合条件的所有记录。

联接表的顺序对我来说很重要。

我返回两张表里联接列内容符合条件的所有记录，而且我使用关键字ON。

我能结合两个共享相同列名的表。

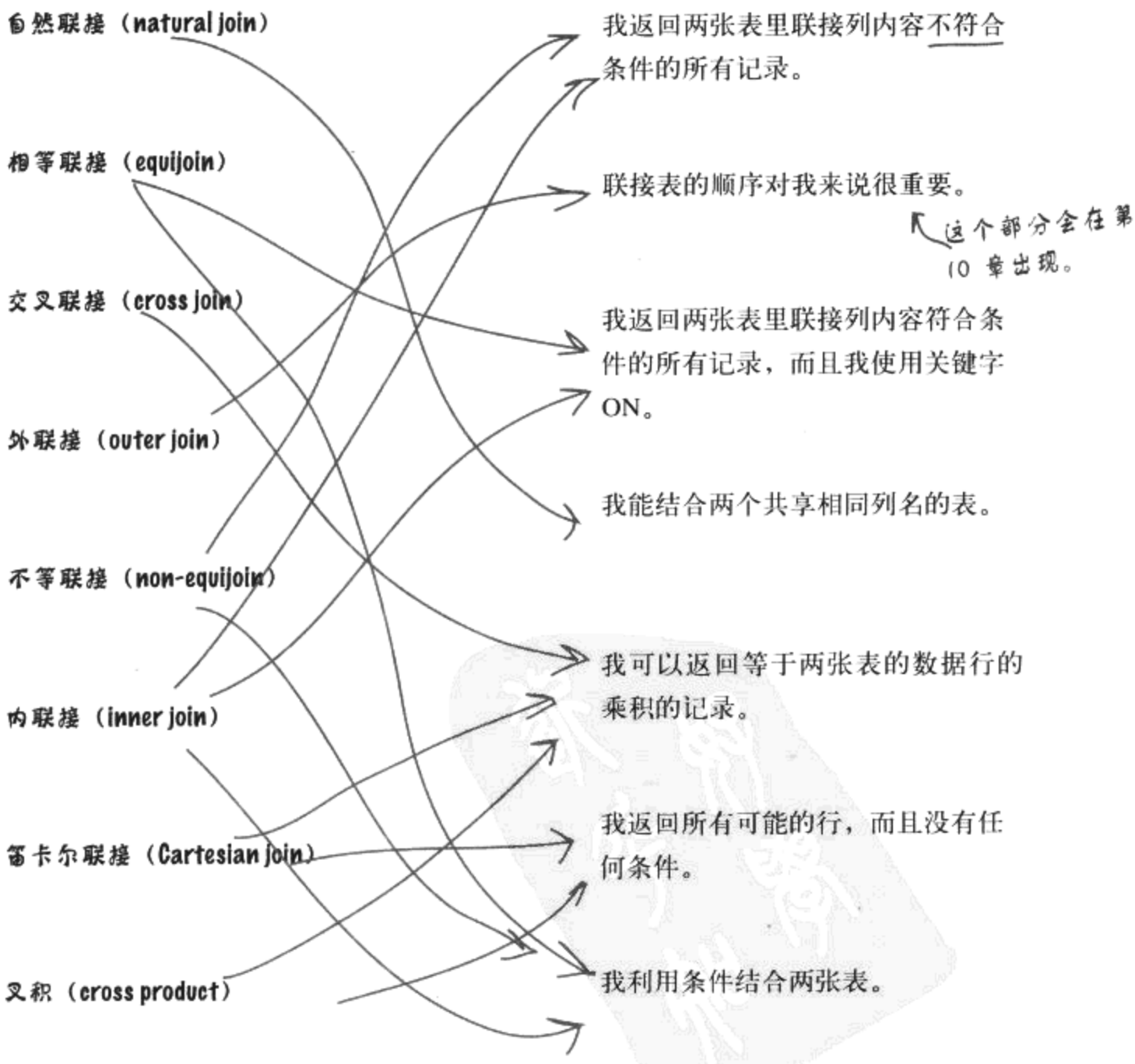
我可以返回等于两张表的数据行的乘积的记录。

我返回所有可能的行，而且没有任何条件。

我利用条件结合两张表。

连连看

为每个联接与说明连线。可能会有多种联接匹配说明。

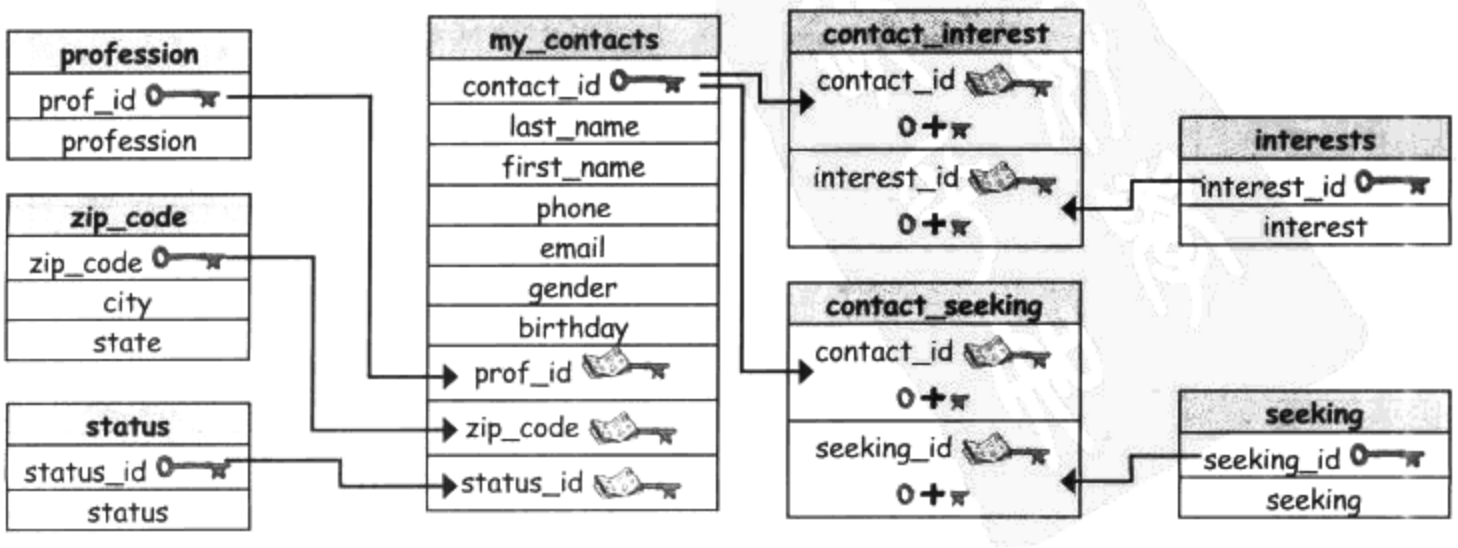


使用下面的 gregs_list 数据库图表设计取得所需信息的查询。

使用不同联接方式设计两个查询，取得 my_contacts 与 contact_interests 里相符的记录。

设计一个查询，返回 contact_seeking 与 seeking 所有可能的合并结果。

列出 my_contacts 表中每个人的职业，但职业不能重复列出，而且要按字母顺序排列。



使用下面的 gregs_list 数据库图表设计取得所需信息的查询。

使用不同联接方式设计两个查询，取得 my_contacts 与 contact_interests 里相符的记录。

```
SELECT mc.first_name, mc.last_name, ci.interest_id FROM my_contacts mc
INNER JOIN contact_interest ci ON mc.contact_id = ci.contact_id;

SELECT mc.first_name, mc.last_name, ci.interest_id FROM my_contacts mc
NATURAL JOIN contact_interest ci;
```

设计一个查询，返回 contact_seeking 与 seeking 所有可能的合并结果。

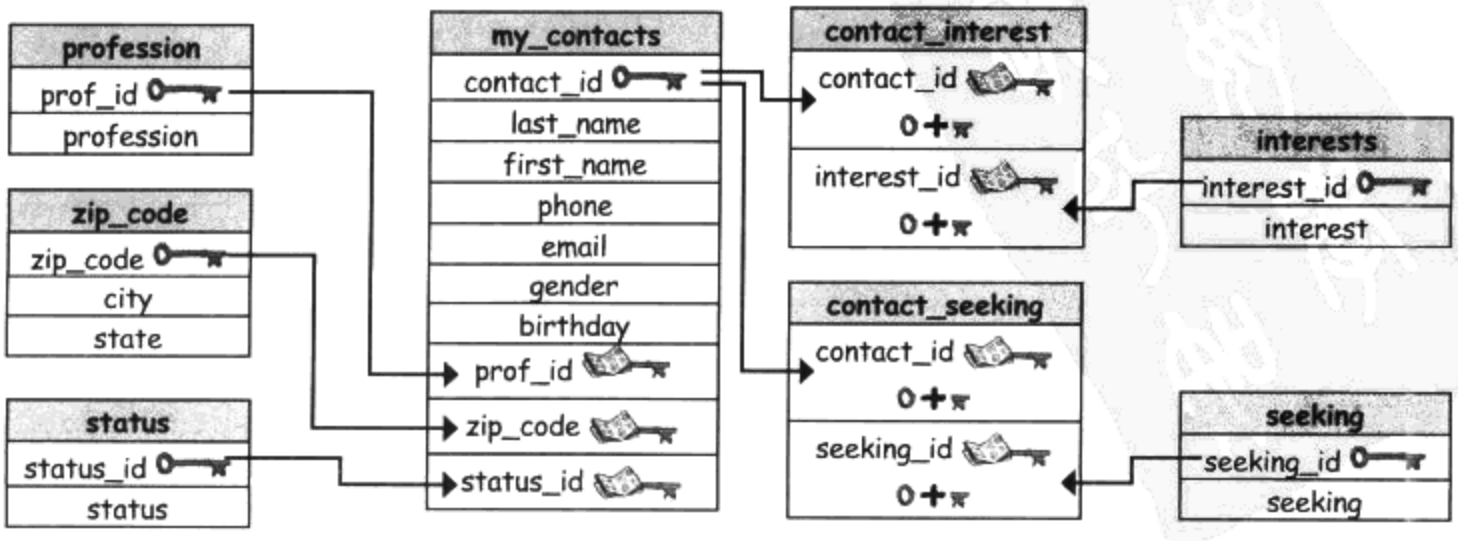
```
SELECT * FROM contact_seeking CROSS JOIN seeking;

SELECT * FROM contact_seeking, seeking;
```

可以有两种方法来进行交叉联接。

列出 my_contacts 表中每个人的职业，但职业不能重复列出，而且要按字母顺序排列。

```
SELECT p.profession FROM my_contacts mc
INNER JOIN profession p ON mc.prof_id = p.prof _ id GROUP BY profession ORDER BY profession;
```



问： 可以联接多于两张表吗？

答： 可以，而且稍后也会略微提到这个主题。现在，我们只需专心于联接的概念。

问： 联接不是应该更难一点吗？

答： 一旦进入联接与别名的世界，SQL查询就开始像是火星文。再加上使用速记法（例如以逗号取代关键字 INNER JOIN），会让我们更加混乱。因此，本书选用比较啰嗦的SQL查询写法，少用一点速记法。

问： 你是说，还有其他设计内联接的方式吗？

答： 是的，确实还有。但如果理解了本书讲到的方式与语法，再理解其他语法就会轻而易举。概念其实远比使用WHERE或ON设计联接更重要。

问： 我发现你在联接中使用了ORDER BY，这是说其他东西也能与联接一起使用吗？

答： 没错。请放心地使用 GROUP BY、WHERE子句以及SUM、AVG等函数。

联合查询？

Greg真的由衷地爱上了联接。他开始明白多张表设计的道理，也发现这些表若设计得当，仍然很好操作。
他甚至想要进一步扩充 gregs_list 数据库。

在另一个查询里的查询？

有可能吗？

SQL真情指数：表与列的别名篇

本周主题：
你们在隐藏什么？

HeadFirst：让我们欢迎 Table Alias 与 Column Alias。很高兴能请到两位一起上节目，我们都很希望澄清一些疑惑。

Table Alias：我也很高兴来做专访。你可以叫我们 TA 与 CA，比较简短（笑）。

HeadFirst：哈哈哈，听起来很合适。好的，CA，我想先请教你一些问题。为什么你这么神秘呢？你在隐藏什么事情吗？

Column Alias：当然没有！如果硬要说的话，我只是想让一切看来更清楚。这个回答应该适用于我们两个，对吧？TA？

TA：没错。以 CA 而言，他的意图应该已经很明显了。CA 把冗长累赘的列名变成容易理解、更好存取的别名，而且还会为查询结果加上有用的列名。我的部分则有点不一样。

HeadFirst：我得承认，TA，我真的对你比较陌生。虽然看过你工作的样子，但我还是不太确定你的工作内容是什么。当我们在查询中请你帮忙时，你不会出现在所有结果中。

TA：确实如此。不过，你还没有抓到我的工作精髓。

HeadFirst：精髓？听起来很有趣，请说。

TA：我存在，是为了让联接更容易设计。

TA：而且TA 也在有我的联接里，助我一臂之力。

HeadFirst：我还是搞不清楚。可以做个示范吗？

TA：我可以提供语法，应该可以从中清楚地看出我的工作了：

```
SELECT mc.last_name, mc.first_name, p.profession
FROM my_contacts AS mc
  INNER JOIN
  profession AS p
WHERE mc.contact_id = p.id;
```

HeadFirst：我看到了！每个必须输入 `my_contacts` 的地方现在只要输入 `mc`，而 `p` 则表示 `profession` 表。看起来简单多了，而且在查询中用到两张表时非常好用。

TA：尤其在表名相近时特别好用。让查询更容易被理解，不只有助于查询的设计，也对日后回想查询的作用大有帮助。

HeadFirst：真是太感谢两位来宾——TA 与 CA 的光临。接下来……咦……他们去哪里了？

你的SQL工具包

大家刚结束第8章，我们已经能像SQL专家一样运用联接（JOIN）了。让我们回顾本章学到的新技巧。若想浏览本书的所有工具，请参考附录3。

INNER JOIN

内联接。任何使用条件结合来自两张表的记录的联接。

NATURAL JOIN

自然联接不使用“ON”子句的内联接。只有在联接的两张表中有同名列时才能顺利运作。

EQUIJOIN 与 NON-EQUI-JOIN

相等联接与不等联接。两者均为内联接的一种。相等联接返回相等的行，不等联接则返回不相等的行。

CROSS JOIN

交叉联接。返回一张表的每一行与另一张表的每一行的所有可能的搭配结果。其他常见名称还包括笛卡尔联接（CARTESIAN JOIN）与NO JOIN。

COMMA JOIN

与CROSS JOIN相同，只不过以逗号取代关键字CROSS JOIN。

磨笔上阵解答

第348页上的习题

各位现在都已知道如何修改表，所以请用 ALTER 修改 my_contacts，为它添加4个新列，分别命名为 interest1、interest2、interest3、interes4。

```
ALTER TABLE my_contacts
ADD (interest1 VARCHAR(20), interest2 VARCHAR(20), interest3 VARCHAR(20),
interest4 VARCHAR(20));
```

磨笔上阵解答

第350页上的习题

补齐下列 UPDATE 语句，帮 Greg 完成查询。旁边附有提示。

SUBSTRING_INDEX 与 SUBSTR 的差别在于 SUBSTRING_INDEX 寻找位于 interests 列中的字符串 — 本例为逗号，然后返回它前面的所有内容；SUBSTR 则会缩短 interests 列的长度 — 减去第一项兴趣、再减去一个逗号与一个空格。

```
UPDATE my_contacts SET
interest1 = SUBSTRING_INDEX(interests, ',', 1),
interests = SUBSTR(interests, LENGTH(interest1)+2),
interest2 = SUBSTRING_INDEX(interests, ',', 1),
interests = SUBSTR(interests, LENGTH(interest2)+2),
interest3 = SUBSTRING_INDEX(interests, ',', 1),
interests = SUBSTR(interests, LENGTH(interest3)+2),
interest4 = interests;
```

在移出前三项兴趣后，interests 列只剩下第四项兴趣。这一行只是把剩下的内容移到新列中。此时，也可以简单地把 interests 列改名为 interest4。

interests列此时只包含最后一个兴趣。

interests	interest1	interest2	interest3	interest4
~~second, third~~, fourth	first	second	third	fourth

9 子查询

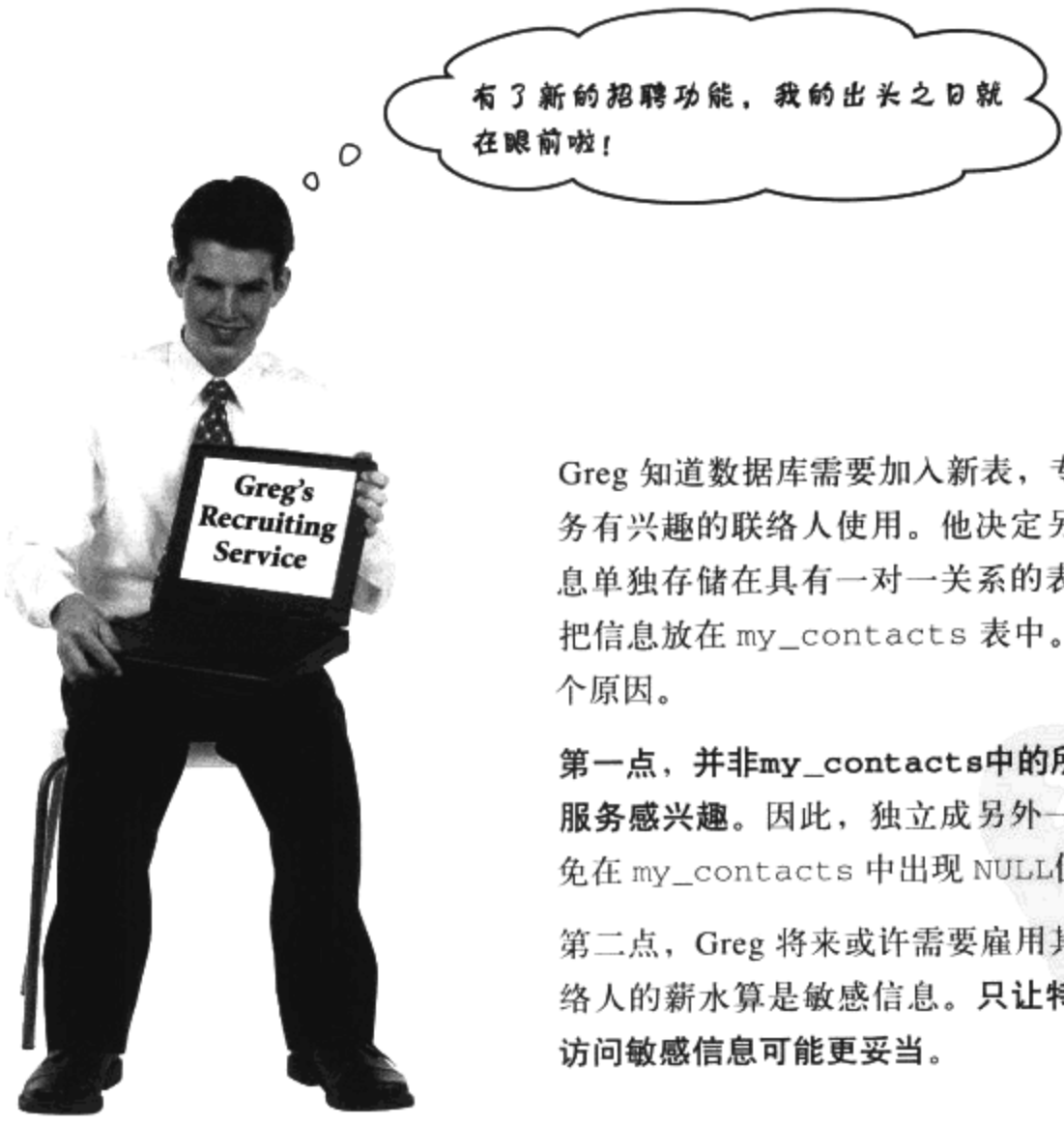
查询中的查询

Jack，**请给我被分成两部分的问题，谢谢**。有了联接的确很好，但有时要问数据库的问题不只一个。或者需要**把甲查询的结果作为乙查询的输入**。这时候就该是**子查询**出场了。子查询有助于**避免数据重复，让查询更加动态灵活**，甚至能引入高端概念。（最后一项不一定会成真，不过三项好处中有两项是真的也很好嘛！）

Greg踏入招聘服务行列

到目前为止，gregs_list数据库是名符其实的月下老人，Greg利用数据库帮助了许多单身的朋友，但是自己却分文未取。

Greg想到他可以开始经营招聘业务，只要把数据库中的联络人与可能的工作进行配对。



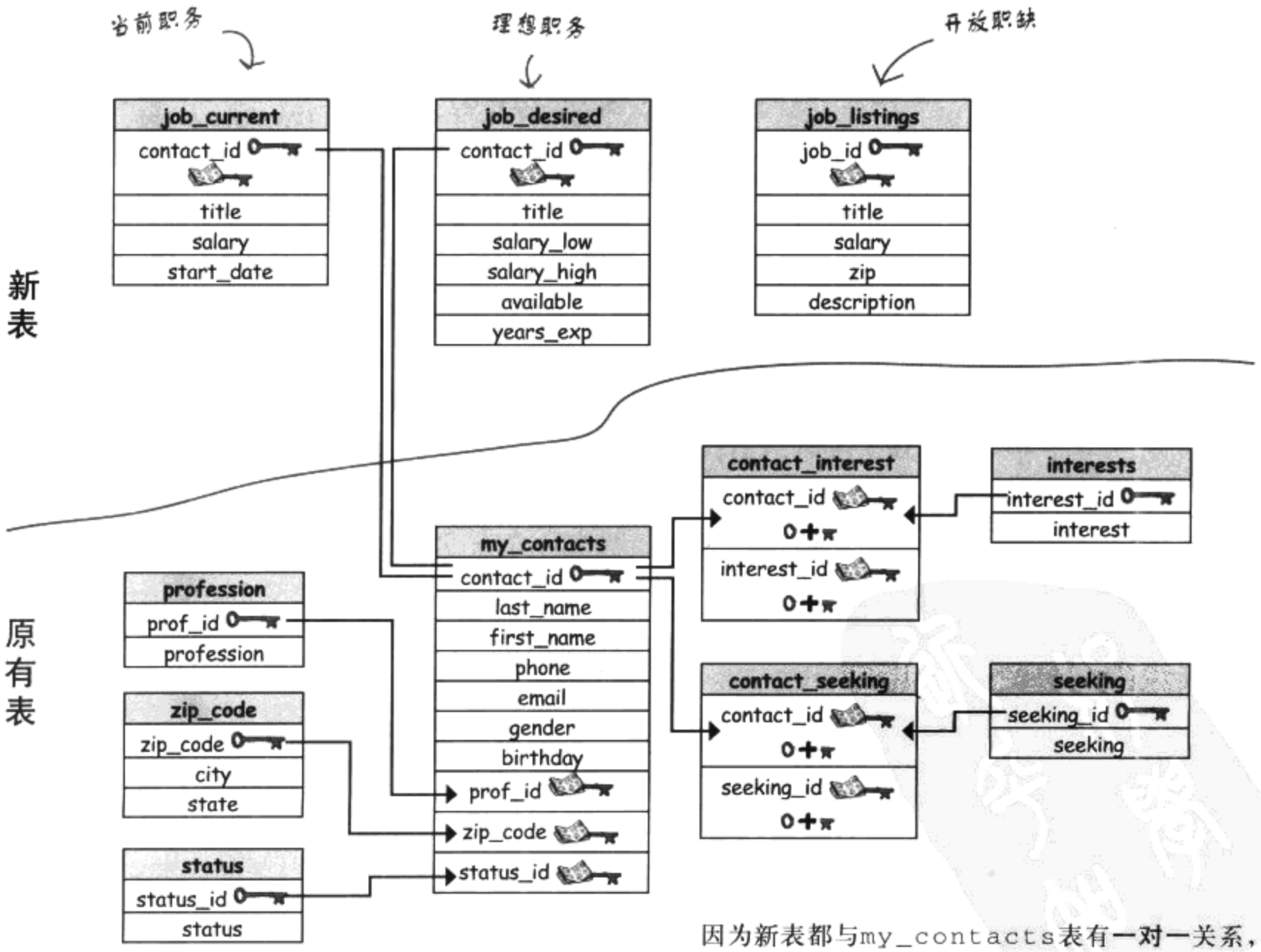
Greg 知道数据库需要加入新表，专供对招聘服务有兴趣的联络人使用。他决定另外把这些信息单独存储在具有一对一关系的表中，而不是把信息放在 my_contacts 表中。这样做有两个原因。

第一点，并非my_contacts中的所有人都对此服务感兴趣。因此，独立成另外一张表可以避免在 my_contacts 中出现 NULL值。

第二点，Greg 将来或许需要雇用其他帮手，联络人的薪水算是敏感信息。**只让特定人士有权访问敏感信息可能更妥当。**

Greg 加入了更多表

Greg 在他数据库中添加了追踪**理想职务**与**理想薪资**的表，还有追踪联络人的**当前职务**和**当前薪资**的表。他还创建了一个包含**职缺信息**的简单表。



因为新表都与my_contacts表有**一对一**关系，所以使用**自然联接**（natural join）就能轻松地完成工作。

Greg使用内联接

Greg列出一份职缺列表，并试着将列表上的职务与他的联络人比对。他希望为每个职务找到最合适的人选，如果他推荐的候选人被录用了，才会有资金流进他的口袋。

职务：网站开发员（Web Developer）

我们的互动与视觉设计团队正在寻找精通HTML和CSS的一流网站开发员。对于非常关注网站标准的人而言，这是一个进入高知名度的公司并在其中发光发热的大好机会。加入极有影响力的公司，与一群乐在工作的人一起工作吧！

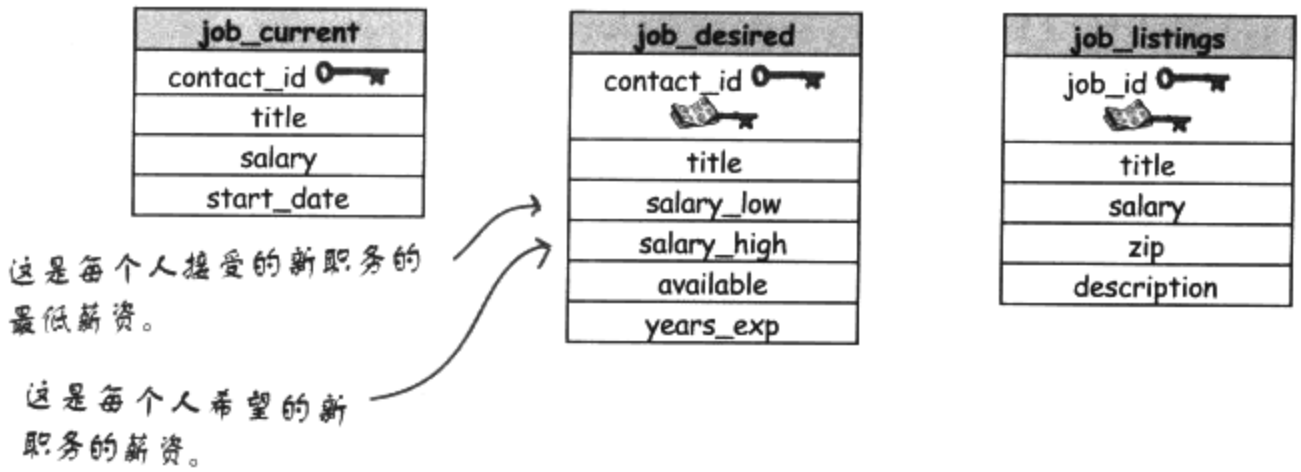
薪资：$95,000~$105,000

经验：5年以上

一旦找出最适合的候选人，Greg就能联络他们并进一步筛选。但是，首先要找出所有具有5年以上经验的网站开发员，而且他们对薪资的期待不能高于$105,000。

磨笔上阵

设计从数据库中取得合适候选人的查询。



但是他想试试其他查询

Greg 收到的招聘需求比他知道的人才多。他必须从自己的求职表中找出适合每项职缺的人才。然后才能利用`my_contacts`取得合适人才的联络方式，并询问他们是否有意应征。

首先选出 job_listings 表中的所有职缺。

```
SELECT title FROM job_listings
GROUP BY title ORDER BY title;
```

我们使用GROUP BY让查询结果中的每种职缺只占用一行，同时按字母顺序排列。

查询结果

title
Cook
Hairdresser
Waiter
Web Designer
Web Developer

这里只是 job_listings 表中存储的部分职缺。

磨笔上阵

设计从数据库中取得合适候选人的查询。

```
SELECT mc.last_name, mc.first_name, mc.phone
FROM my_contacts AS mc
    NATURAL JOIN
    job_desired AS jd
WHERE jd.title = 'Web Developer'
AND jd.salary_low < 105000;
```

此例只需取得联络信息，因为搜索的对象一定都在寻找“Web Developer”职缺。

因为我们的 my_contacts 与 job_desired 表都以 contact_id 作为主键，所以自然联接直接连接两张表。

我们只对有意考虑这份薪资的人选有兴趣。所以要查询 salary_low，检查薪资是否低于候选人的底限。

现在Greg使用关键字IN，检查他的联络人清单中是否存储了
符合职务需求的人才。

```
SELECT mc.first_name, mc.last_name, mc.phone, jc.title
FROM job_current AS jc NATURAL JOIN my_contacts AS mc
WHERE
jc.title IN ('Cook', 'Hairdresser', 'Waiter', 'Web Designer', 'Web Developer');
```

还记得关键字IN吗？如果列中的jc.title属于括号里的职缺之一，IN就会返回该行。

来自上一个查询的结果。

mc.first_name	mc.last_name	mc.phone	jc.title
Joe	Lonnigan	(555) 555-3214	Cook
Wendy	Hillerman	(555) 555-8976	Waiter
Sean	Miller	(555) 555-4443	Web Designer
Jared	Callaway	(555) 555-5674	Web Developer
Juan	Garza	(555) 555-0098	Web Developer

找出候补人选了！

不过，Greg 还是要设计两段查询才能办到……

动动脑

试着把上述查询结合成一个查询。请把你的答案写在下面。

子查询

想用一个查询来完成两个查询的工作，我们需要在查询中添加**子查询**（subquery）。

前一页的第二个查询从my_contacts与job_current表中取出职务符合所需职缺的联络人的信息，此处的查询称为外层查询（OUTER query），它里面另有一个内层查询（INNER query）。现在就来看看其运作方式：

外层查询

```
SELECT mc.first_name, mc.last_name, mc.phone, jc.title
FROM job_current AS jc NATURAL JOIN my_contacts AS mc
WHERE
jc.title IN ('Cook', 'Hairdresser', 'Waiter', 'Web Designer', 'Web Developer');
```

这部分称为外层查询。

这部分能以第一个查询的某部分取代，因而成为内层查询。

列在括号内的所有职务都来自前一页的第一个查询，从job_current表中取出所有职务的查询。接下来是 SQL 聪明的地方，仔细看：我们可以把属于内层查询的部分用第一个查询的一部分取代。这样仍可产生上述括号里的内容，但第一个查询已经被压缩成一个子查询了：

子查询，是被另一个查询包围的查询，也可称为内层查询。

内层查询

```
SELECT title FROM job_listings
GROUP BY title ORDER BY title;
```

第一个查询的这个部分将变成内层查询，也称为子查询。

以子查询合二为一

接下来只要把两个查询合并为一个。首先要有**外层查询**，另一个查询中的查询则是**内层查询**。

外层查询 + 内层查询 = 以子查询进行查询

两个查询结合而成的一个查询就是包含子查询的查询。

外层查询

```
SELECT mc.first _ name, mc.last _ name, mc.phone, jc.title
FROM job _ current AS jc NATURAL JOIN my _ contacts AS mc
WHERE jc.title IN (SELECT title FROM job _ listings);
```

前一页的第一个查询不需原封不动地复制到这里，内层查询会为我们照料其他细节！

下表就是运行查询后的结果，与加上列出**所有**职缺的 WHERE 子句的效果一样，但可以少打很多字。

与稍早的查询结果相同，但只用了一个查询。

mc.first_name	mc.last_name	mc.phone	jc.title
Joe	Lonnigan	(555) 555-3214	Cook
Wendy	Hillerman	(555) 555-8976	Waiter
Sean	Miller	(555) 555-4443	Web Designer
Jared	Callaway	(555) 555-5674	Web Developer
Juan	Garza	(555) 555-0098	Web Developer

子查询解剖课

在单一查询不够用的时候：请用子查询

子查询只不过是查询里的查询。

外部的查询称为**包含查询**（containing query）或**外层查询**。内部的查询就是内层查询，或称为**子查询**。

外层查询，有时称为包含查询。

```
SELECT some _ column, another _ column
FROM table
WHERE column = (SELECT column FROM table);
```

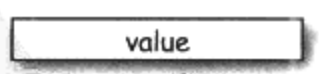
内层查询，又称子查询。

外层查询

```
SELECT some _ column, another _ column
FROM table
WHERE column = (SELECT column FROM table);
```

内层查询

因为查询里使用了=运算符，所以子查询只会返回**单一值，特定行和列的交叉点**（有时称为 *cell*，但 SQL 称之为标量值，*scalar value*），这一个值将是 WHERE 子句中比对数据列的条件。



子查询返回标量值（来自某个列的某一行）作为 WHERE 子句比对数据列内容的条件。

子查询示范

让我们以 my_contacts 为例，示范一个可比较的查询实例。首先，RDBMS 接受来自 zip_code 表的标量值，然后在 WHERE 子句指定的列中寻找该值。

```
SELECT last _ name, first _ name
FROM my _ contacts
WHERE zip _ code =  (SELECT zip _ code FROM
                    zip _ code WHERE city =
                    'Memphis' AND state = 'TN')
```

value

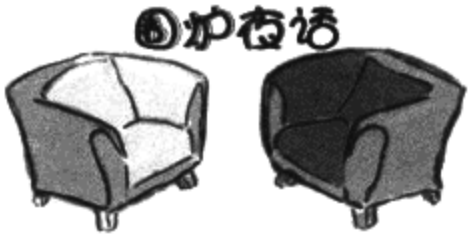
这个查询将从my_contacts中选出住在 Memphis, Tennessee的联络人的姓名。

没有蠢问题

问： 为什么我不能只用联接呢？

答： 也可以用联接，但有些人觉得子查询设计起来比联接简单。在语法上能有其他选择是件好事。相同查询也能以下列方式实现：

```
SELECT last _ name, first _ name
FROM my _ contacts mc
NATURAL JOIN zip _ code zc
WHERE zc.city = 'Memphis'
AND zc.state = 'TN'
```



今夜主题：你是 INNER 还是 OUTER？

外层查询

内层查询，你知道的，我不需要你。就算没有你，我的生活一样可以过得很好。

（大喊）你每次都只给我一个小小的结果。用户想要数据！数据！很多数据！！这就是我给用户的东西：很多很多数据。我看，你不在那搅局的话，用户反而更高兴。

只要加上 WHERE 子句就不会。

是哦，你不需要我！一个只有一行、一列的答案能做什么？信息量根本不够！

但我不需要你。

内层查询

彼此彼此，我也不需要你。你以为你每次都要找出限定目标的查询结果很好玩吗？我辛辛苦苦找出来的查询结果却只是给你用来找出一堆相符的行！质不等于量，知道吗？

才不是，我为你的查询结果赋予特定目的。没有我，你就等着喷出表中的所有内容吧！

对，没错，我就是你的WHERE子句。要我形容自己的话，那就是我是非常精确的 WHERE 子句。事实上，我一点都不需要你嘛！

我们一起工作的效果或许真的很好，那是因为我为你的结果赋予了一些方向。

我也不需要你，大部分的时候。

子查询规则

所有子查询都会遵循一些规则。请使用我们提供的词填入规则里的空格（有些词可能不只用到一次）。

SEMICOLON

SELECT

COLUMN LIST

END

PARENTHESES

DELETE

UPDATE

FROM

INSERT

HAVING

SQL 的子查询规则

子查询都是单一 ____ 语句。

子查询总是位于 ____ 里。

子查询没有属于自己的 ____。就像一般查询一样，要等到整个查询结束，出现 _____，它后面才会有 ____。

SQL 的子查询规则

子查询可能出现在查询中的四个地方：

____ 子句、选出 ____ 作为其中一列、____ 子句与 ____ 子句中。

子查询能与_____、_____、____，当然还有 _____ 一起使用。

子查询规则

在本章的后续内容中看到子查询时，请把这些规则牢记于心。

SQL 的子查询规则

子查询都是单一 **SELECT** 语句。

子查询总是位于 **PARENTHESES** 里。

子查询没有属于自己的 **SEMICOLON**。就像一般查询一样，要等到整个查询结束，出现 **END**，它后面才会有 **SEMICOLON**。

SQL 的子查询规则

子查询可能出现在查询中的四个地方：

SELECT 子句、选出 **COLUMN LIST** 作为其中一列、**FROM** 子句与 **HAVING** 子句中。

子查询能与 **INSERT**、**DELETE**、**UPDATE**，当然还有 **SELECT** 一起使用。

问： 内层查询究竟可以返回什么？外层查询呢？

答： 在大多数情况下，内层查询只能返回单一值——也就是一个列里的一行。而后，外层查询才能利用这个值与列中的其他值进行比较。

问： 为什么你要在第 387 页上的范例中强调“单一值”，范例不是返回了装满结果的表了吗？

答： 因为 IN 运算符会寻找一组值的集合。如果使用比较运算符，例如 =，就只能接受一个值来与列中的其他值进行比较。

问： 我还是不太清楚子查询究竟能返回几个值！只能返回一个？还是可以返回多个？有什么正式规定吗？

答： 一般而言，子查询必须返回一个值。使用 IN 是例外情况。大部分时候，子查询只需要返回单一值。

问： 如果你的子查询真的返回多个值，但使用的 WHERE 子句却不可以接受多个值，会发生什么事？

答： 大混乱！大破坏！事实上，只会得到错误信息。

是啊，这些规则是很闪亮动人，不过我真正想要知道的是如何让查询结果中的名称短一点，我不想看到mc.last_name这类名称。你有适合的规则吗？

事实上，有两件事可以协助你清理这些杂乱的名称。

你可以为 SELECT 选择的列创建别名，查询所产生的结果表立马就会变得干净许多。

以下是我们刚创建的子查询，但加上了简化版的**列别名**。

my_contacts的first_name列在查询结果里的别名是"firstname"。

……my_contacts的last_name列的别名则是"lastname"。

my_contacts的phone列在查询结果里的别名将是"phone"……依此类推。你知道其他列的变化方式了！

```
SELECT mc.first_name AS firstname, mc.last_name AS lastname,
mc.phone AS phone, jc.title AS jobtitle
FROM job_current AS jc NATURAL JOIN my_contacts AS mc
WHERE jobtitle IN (SELECT title FROM job_listings);
```

请记住，AS 是可选的关键字，创建别名时也可以省略 AS。

这里就是查询给我们的结果。

请注意，使用列别名后，查询结果显得更容易理解了。

既然只是临时性的别名，就表示不会影响原始表或列的名称。

firstname	lastname	phone	jobtitle
Joe	Lonnigan	(555) 555-3214	Cook
Wendy	Hillerman	(555) 555-8976	Waiter
Sean	Miller	(555) 555-4443	Web Designer
Jared	Callaway	(555) 555-5674	Web Developer
Juan	Garza	(555) 555-0098	Web Developer

子查询的构造流程

子查询的棘手部分并非结构，而是找出需要作为子查询的查询，甚至是是否需要子查询。

分析查询就像分析一个问题，从问题中找出自己知道的事物（例如表名与列名），然后分解问题。

我们来尝试分析一个想要询问数据库的问题以及如何从分析中构建查询。首先，问问自己：

在我所有的联络人里，谁赚的钱最多？

分解问题。

重新描述这个问题，换成表与列的名称。

“谁”表示需要 my _ contacts里的first _ name与last _ name 列。“钱最多”表示需要 job _ currenet 表的MAX值。

在我所有的联络人里，谁赚的钱最多？

My_contacts 表的 first_name 与 last_name 列

Job_currenet 表的 MAX(salary)

找出能够回答部分问题的查询。

既然我们创建的是非关联子查询（noncorrelated subquery），我们可以挑出部分问题并建立回答该部分的查询。

MAX(salary) 似乎是个不错的选择。

```
SELECT MAX(salary) FROM job _ current;
```

还记得关键字MAX吗？这个函数返回括号中指定列的最大值。

继续分解查询。

查询的第一个部分也很简单，只需要选出姓（last_name）与名（first_name）：

```
SELECT mc.first _ name, mc.last _ name
FROM my _ contacts AS mc;
```

选出姓与名

最后，找出串起两个查询的方式。

我们不仅需要 my_contacts 里记载的人名，还需要知道他们的薪水，才能比较出最高的薪资（MAX(salary)）。我们需要一个自然内联接来找出每个人的薪资信息：

```
SELECT mc.first _ name, mc.last _
name, jc.salary
FROM my _ contacts AS mc
NATURAL JOIN job _ current AS jc;
```

使用NATURAL JOIN，找出每个人的薪资。

接下来加上 WHERE 子句以连接两段查询。

我们创建了一个回答这个问题的大型查询：“谁赚的钱最多？”

这就是我们刚才做的事：找出每个人的薪资。

```
SELECT mc.first _ name, mc.last _ name, jc.salary
FROM my _ contacts AS mc NATURAL JOIN job _ current AS jc
WHERE jc.salary =
(SELECT MAX(jc.salary) FROM job _ current jc);
```

这里是问题的第一部分，现在成为子查询，用于找出最高薪资值。来自子查询的值将与外层查询进行比较，最后取得结果。

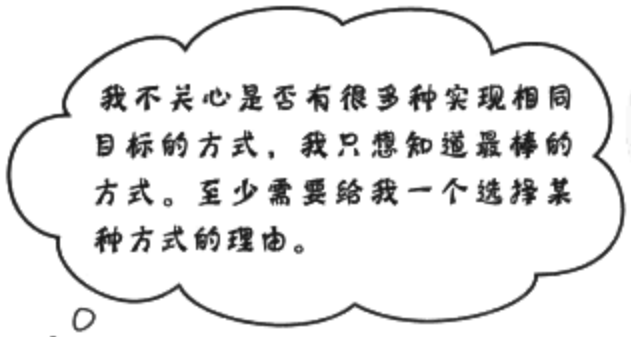
哦，原来是Mike？我应该早点想到的。他不曾领过支票。

mc.first_name	mc.last_name	jc.salary
Mike	Scala	187000

没错，子查询不是唯一的方式。

同样目标也能利用自然内联接和LIMIT命令达成。就像SQL的大部分功能一样，总是有许多方式能实现相同目标。

设计另一个查询，一样能找出Greg所有的联络人里最会赚钱的人。



好问题。

你可以先看一下第 400 页上的“SQL 真情指数”。

作为欲选取列的子查询

子查询能用作 SELECT 语句中选取的列之一。请见下例。

```
SELECT mc.first _ name, mc.last _ name,
(SELECT state
FROM zip _ code
WHERE mc.zip _ code = zip _ code) AS state
FROM my _ contacts mc;
```

我们设定了列别名"state"。

分解上述查询要先从子查询着手。子查询只是单纯地从 zip _ code 表中比对出邮政编码（zip code）与相应的州名而已。

简单地说，这段查询的用途是：

> 查找 my_contacts 表的每一行，取出每一行的姓、名、州名等信息（关于州名的部分，我们利用子查询比对 my_contacts 与 zip_code 表记录的邮政编码，再从 zip_code 表中取出州名信息）。

子查询应该只能返回一个值，所以每次子查询返回一个值，而整个查询也跟着返回一行。以下即为查询结果：

mc.first_name	mc.last_name	state
Joe	Lonnigan	TX
Wendy	Hillerman	CA
Sean	Miller	NY
Jared	Callaway	NJ
Juan	Garza	CA

如果子查询放在SELECT语句中，用于表示某个欲选取的列，则一次只能从一列返回一个值。

范例：子查询搭配自然联接

Greg 的朋友Andy 最近在吹嘘他的待遇有多么好。虽然 Andy 没有讲出具体数字，但 Greg 觉得他的数据库里应该有这项信息。他利用 Andy 的电子邮件地址做了一次快速的 NATURAL JOIN 查询。

```
SELECT jc.salary
FROM my_contacts mc NATURAL JOIN job_current jc
WHERE email = 'andy@weatherorama.com';
```

这个查询将返回Andy的薪资，是个单一值。

这个部分是内层查询。

Greg发现这个查询只返回单一值。与其直接运行查询，再把查询结果放到另一个查询里，他决定把它变成子查询。

所以 Greg 写了一个查询，作用包括：

- 取得 Andy 的薪资
- 与其他人的薪资进行比较
- 返回某些人的姓名
- 哪些人的薪资比 Andy 的高。

这个部分要用到比较运算符“>”。

比Andy的薪资更高。

以下则是外层查询

```
SELECT mc.first_name, mc.last_name, jc.salary
FROM
my_contacts AS mc NATURAL JOIN job_current AS jc
WHERE
jc.salary > (ANDY'S SALARY QUERY WILL GO HERE)
```

非关联子查询

拼图一块一块成型，构成整个查询。数据库软件先处理一次内层查询，然后使用内层查询返回的值找出外层查询的结果。

查询的目的是取得姓名与薪资。

RDBMS会第二个处理这个部分。

只呈现薪资高于Andy的人。

这两个查询会分别被RDBMS处理。

```
SELECT mc.first_name, mc.last_name, jc.salary
FROM
my_contacts AS mc NATURAL JOIN job_current AS jc
WHERE
jc.salary > (SELECT jc.salary
FROM my_contacts mc NATURAL JOIN job_current jc
WHERE email = 'andy@weatherorama.com');
```

能取得Andy的薪资的子查询，以供外层查询比较。

这部分会首先处理。

下表是部分查询结果。我们并未使用 ORDER BY，所以查询结果并未按任何顺序排列。

mc.first_name	mc.last_name	jc.salary
Gus	Logan	46500
Bruce	Hill	78000
Teresa	Semel	48000
Randy	Wright	49000
Julie	Moore	120000

目前我们看到的所有子查询都是**非关联子查询**（noncorrelated subquery）。软件先处理内层查询，查询结果再用于外层查询的 WHERE 子句。但是内层查询完全不需依赖外层查询的值，它本身就是一个可以完全独立运行的查询。

如果子查询可以独立运行且不会引用外层查询的任何结果，即称为非关联子查询。

（如果你在谈话中提到“非关联子查询”，没有接触过SQL的人会觉得你很神）

外层查询较晚处理。它的查询结果取决于内层查询的值。

内层查询可以单独运行，通常会先处理这个部分。

SQL 真情指数：

本周主题：
在众多选择中挑出最好的查询方式

Head First SQL：欢迎本周的来宾，SQL。谢谢你今天抽空来做独家访谈，我知道最近你面对不少难题。

SQL：难题？只是你这么形容的吧？我会说一切很麻烦、很让人头痛、很难量化同时又不断地反复。

Head First SQL：嗯，是啊。其实这也是本周访谈的重点。你一直收到关于 SQL 太灵活的抱怨，当我们问你问题时，你给了我们太多可用的询问方式。

SQL：我承认自己很灵活，而且大家可以用不同方式问我问题，最后都能得出相同答案。

Head First SQL：有人认为这是优柔寡断。

SQL：我不想为自己辩解，我又不是坏人。

Head First SQL：当然不是，我们都知道你不是坏人，只不过有点……不够严谨。

SQL：哼！不严谨！我受够了。（起身准备走人）

Head First SQL：别走、别走！我们只是想要一些答案。有时候你会让我们用不同方式询问相同问题。

SQL：是啊，有什么不可以吗？

Head First SQL：没有，没什么不可以的，我们只是想知道应该问什么。在你会提供相同答案的前提下，询问的方式真的很重要吗？

SQL：当然重要啦！还用问吗？有时候你们的问题会让我思考很久才找到答案；有时候，砰！答案突然就找到了。重点在于问对方式。

Head First SQL：所以说，重点其实是响应的速度。速度就是我们选择询问方式的关键吗？

SQL：当然！一切都取决于你们问我的问题。我只是等在这边准备回答大家的问题，只要问题精确。

Head First SQL：速度？速度就是秘密？

SQL：听好，我已经给你提示了。数据库会增长。大家都希望自己的查询的响应方式越简单越好。例如，如果有人问我“Whodunnit”，就会让我思考半天才能转换成“这件事是谁做的”。让我尽可能不用思考，给我简单的问题，答案自然就会很快出现。

Head First SQL：原来如此。不过该怎么判断问题是否简单呢？

SQL：这个嘛，交叉联接是件非常浪费时间的事，关联子查询也会拖慢速度。

Head First SQL：还有吗？

SQL：呃……

Head First SQL：拜托，请你多说一点。

SQL：经验很重要。有时候最好创建测试数据库来尝试各种查询方式，比较查询运行的时间。对了，联接比子查询更有效率。

Head First SQL：谢谢你，SQL。没想到这就是天大的秘密……

SQL：对啦，谢谢你浪费我的时间。

子查询工坊

阅读下列情境。遵循指示设计两个查询，然后把它们结合成一个子查询。

1. Greg 希望从 `job _ current` 表中计算出 Web Developer 的平均薪资。然后他想比较每位网站设计员的实际薪资与平均薪资。薪资少于平均水平的人或许更有兴趣换新工作，他们是 Greg 想要锁定的潜在客户。

   请设计一个查询，从 `job_current` 表中取得网站设计员的平均薪资。

2. Greg 需要知道 `job _ current` 表中每位网站设计员的姓、名与薪资。

   请设计一个查询，取得每位网站设计员的姓、名及记录在 `job_current` 表中的薪资。

3. Greg 使用平均薪资（有一点数学计算）作为子查询，列出每位网站设计员及其薪资与平均薪资间的差距。

   结合上面的两个查询。使用子查询作为欲选择的列之一。

子查询工坊解答

阅读下列情境。遵循指示设计两个查询，然后把它们结合成一个子查询。

1. Greg 希望从 job _ current 表中计算出 Web Developer 的平均薪资。然后他想比较每位网站设计员的实际薪资与平均薪资。薪资少于平均水平的人或许更有兴趣换新工作，他们是 Greg 想要锁定的潜在客户。

   请设计一个查询，从 job_current 表中取得网站设计员的平均薪资。

```
SELECT AVG(salary) FROM job_current WHERE title = 'Web Developer';
```

需要关键字 AVG。

2. Greg 需要知道 job _ current 表中每位网站设计员的姓、名与薪资。

   请设计一个查询，取得每位网站设计员的姓、名及记录在 job_current 表中的薪资。

```
SELECT mc.first_name, mc.last_name, jc.salary
FROM my_contacts mc NATURAL JOIN job_current jc
WHERE jc.title = 'Web Developer';
```

3. Greg 使用平均薪资（有一点数学计算）作为子查询，列出每位网站设计员及其薪资与平均薪资间的差距。

   结合上面的两个查询。使用子查询作为欲选择的列之一。

这部分是子查询。

```
SELECT mc.first_name, mc.last_name, jc.salary,
jc.salary - (SELECT AVG(salary) FROM job_current WHERE title = 'Web Developer')
FROM my_contacts mc NATURAL JOIN job_current jc
WHERE jc.title = 'Web Developer';
```

有多个值的非关联子查询：IN、NOT IN

回头看看第 387 页上 Greg尝试的第一个查询，协助他找出**符合**其职缺列表（job _ listings）需求的职务。这个查询接受整个title集（由子查询里的 SELECT 返回）并评估 job _ current 表中的每一行，寻找潜在的匹配记录。

```
SELECT mc.first_name, mc.last_name, mc.phone, jc.title
FROM job_current AS jc NATURAL JOIN my_contacts AS mc
WHERE jc.title IN (SELECT title FROM job_listings);
```

IN根据子查询返回的整个结果集来评估 jc.title 每一行的值。

使用**NOT IN**可协助Greg找出**不符合**职缺列表的职务。同样接受由子查询里的 SELECT 返回的完整 title 集，并根据这个数据集评估 job _ current 表中的每一行，返回不匹配 job _ current 的记录。现在 Greg 可以专心为这些职务类型寻找更多职缺。

```
SELECT mc.first_name, mc.last_name, mc.phone, jc.title
FROM job_current jc NATURAL JOIN my_contacts mc
WHERE jc.title NOT IN (SELECT title FROM job_listings);
```

NOT IN 返回任何 job_current 里未存储在职缺列表中的职务。

这种查询称为非关联子查询，IN 或 NOT IN 在其中根据子查询的结果检查外层查询，比较两者是否相符。

非关联子查询使用 IN 或 NOT IN来检查子查询返回的值是否为集合的成员之一。

动动脑

直接输入要比较的值不就行了，何必使用子查询？

动手为下列问题设计查询，若有需要，可利用联接与非关联子查询。请参考右页的 `gregs_list` 数据库模式（schema）。

需要我们从Girl Sprout饼干销售范例学到的统计函数。

列出薪资等于 job_listings 表中最高薪资的职务名称。

答案请见第 406 页。

列出薪资高于平均薪资者的姓名。

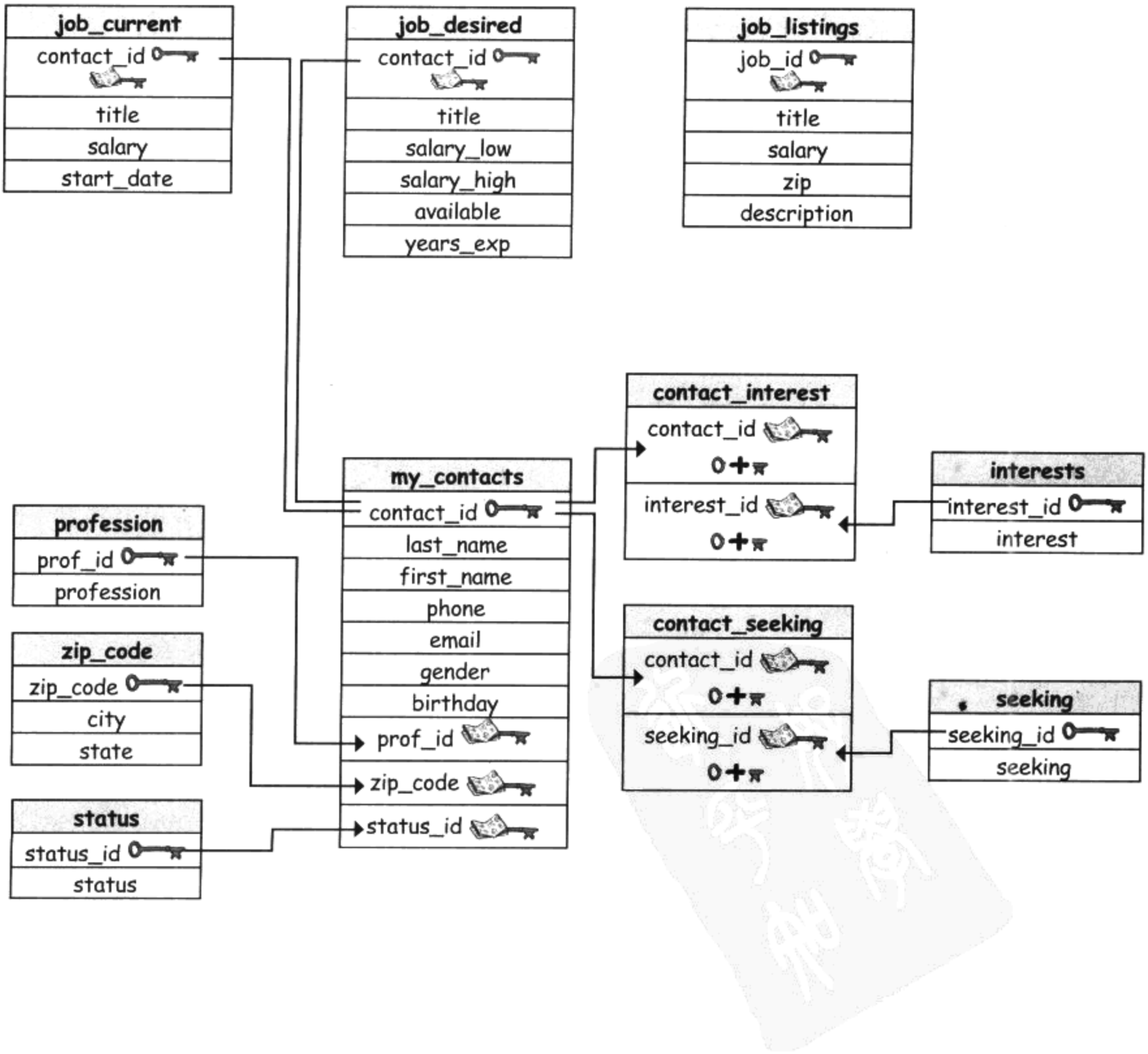
答案请见第 406 页。

请寻找网站设计员，但只列出其邮政编码与任何一个网站设计职缺的邮政编码相同的设计员。

答案请见第 407 页。

列出每个邮政编码涵盖地区中当前薪资最高的人。

答案请见第 407 页。


job_current
contact_id
title
salary
start_date
job_desired
contact_id
title
salary_low
salary_high
available
years_exp
job_listings
job_id
title
salary
zip
description
contact_interest
contact_id
interest_id
interests
interest_id
interest
my_contacts
contact_id
last_name
first_name
phone
email
gender
birthday
prof_id
zip_code
status_id
profession
prof_id
profession
zip_code
zip_code
city
state
status
status_id
status
contact_seeking
contact_id
seeking_id
seeking
seeking_id
seeking

动手为下列问题设计查询，若有需要，可利用联接与非关联子查询。请参考右页的 `gregs _ list` 数据库模式（schema）。

列出薪资等于 job_listings 表中最高薪资的职务名称。

外层查询根据最高薪资值（MAX(*salary*)）进行比较。

子查询返回单一值。

```
SELECT title FROM job_listings
WHERE salary = (SELECT MAX(salary)
FROM job_listings);
```

MAX 返回表中最高的薪资值。

列出薪资高于平均薪资者的姓名。

外层查询接收子查询的结果并返回大于前述结果的记录。

```
SELECT mc.first_name, mc.last_name
FROM my_contacts mc
NATURAL JOIN job_current jc
WHERE jc.salary > (SELECT AVG(salary) FROM job_current);
```

自然联接给了我们人名，他们都是薪资高于内层查询返回结果的人。

子查询返回平均薪资。

请寻找网站设计员，但只列出其邮政编码与任何一个网站设计职缺的邮政编码相同的设计员。

我们需要使用自然联接来取得符合对象的有用信息，例如姓名与电话号码。

```
SELECT mc.first_name, mc.last_name, mc.phone FROM my_contacts mc
NATURAL JOIN job_current jc WHERE jc.title = 'web designer' AND mc.zip_code IN
(SELECT zip FROM job_listings WHERE title = 'web designer');
```

因为可能不只返回一个邮政编码，所以我们把结果当成一个集合并使用“IN”寻找相符信息。

内层查询返回所有提供网站设计员职缺的地区的邮政编码。

列出每个邮政编码涵盖的地区中当前薪资最高的人。

这是个有点困难的问题，因为可能不只一个人具有最高薪资，我们需要使用 IN。我们也需要用到两个子查询。

外层查询会接收邮政编码并从 my_contacts 表中找出相符信息。因为中间的子查询可能返回多个邮政编码，所以此时需要 IN。

中间的子查询负责寻找薪资最高者的邮政编码。

```
SELECT last_name, first_name FROM my_contacts
WHERE zip_code IN (SELECT mc.zip_code FROM my_contacts mc
NATURAL JOIN job_current jc
WHERE jc.salary = (SELECT MAX(salary) FROM job_current));
```

最内层的子查询从 job_current 表取得最高薪资值。这是个单一值，所以使用“=”。

关联子查询

如果非关联子查询表示有个可以独立运行的子查询，那么我相信一定还有关联子查询，而且它以某种方式依赖于外层查询。

没错。在非关联子查询中，内层查询（子查询）先被 RDBMS 解释，然后才输到外层查询。

所以接下来要面对关联子查询。关联子查询是指内层查询的解析需要依赖外层查询的结果。

下例查询会计算 `my_contacts` 里的每个人各有几项兴趣，然后返回具有三项兴趣的人（返回其姓名）。

```
SELECT mc.first _ name, mc.last _ name
FROM my _ contacts AS mc
WHERE
3 = (
SELECT COUNT(*) FROM contact _ interest
WHERE contact _ id = mc.contact _ id
);
```

my_contacts 的别名创建在外层查询中。

子查询也引用了别名 mc。

外层查询必须先执行，执行完后我们才能知道 mc.contacts_id 的值。

子查询依赖外层查询，它需要来自外层查询的结果、`contacts _ id` 的值，然后才能解析内层查询。

本例的子查询中使用的**别名（关联名称）** `mc`，是在外层查询中创建的。

一个搭配 NOT EXISTS 的（好用）关联子查询

关联子查询的常见用法，是找出所有外层查询结果里不存在于关联表里的记录。

假设 Greg 想为日渐增长的招聘业务找到更多客户，所以打算寄电子邮件给 my_contacts 里的目前不在 **job_current** 表中的每个人。他可以使用 NOT EXISTS 找出这些人。

```
SELECT mc.first _ name firstname, mc.last _ name lastname, mc.email email
FROM my _ contacts mc
WHERE NOT EXISTS
(SELECT * FROM job _ current jc
WHERE mc.contact _ id = jc.contact _ id );
```

NOT EXISTS 负责从 my_contacts 表找出姓名与电子邮件地址，他们都未列在 job_current 表中。

连连看

连接查询的每个部分与其功能。

`mc.first _ name firstname`	为 mc.last_name 设定别名
`WHERE NOT EXISTS`	如果两个 contact_id 都是 true，则符合条件
`WHERE mc.contact _ id = jc.contact _ id`	为“firstname”列设定别名
`FROM my _ contacts mc`	选择别名为“jc”的表中的所有列
`mc.last _ name lastname`	设定某列的别名为“email”
`SELECT * FROM job _ current jc`	如果没有找到指定的内容，才视为条件成立
`mc.email email`	为 my_contacts 设定别名

EXISTS 与 NOT EXISTS

就像 IN 与 NOT IN，子查询也能搭配 EXISTS与 NOT EXISTS 一起使用。下列查询将返回来自 my _ contacts 表的数据，其中 contacts _ id 曾出现在 contact _ interest 表里。

```
SELECT mc.first _ name firstname, mc.last _ name lastname, mc.email email
FROM my _ contacts mc
WHERE EXISTS
(SELECT * FROM contact _ interest ci WHERE mc.contact _ id = ci.contact _ id );
```

EXISTS 负责从 *my_contacts* 表中找出姓名和电子邮件地址，这些记录的 *contact_id* 曾出现在 *contact_interest* 表中。

连连看

连接查询的每个部分与其功能。

```
mc.first _ name firstname

WHERE NOT EXISTS

WHERE mc.contact _ id =
jc.contact _ id

FROM my _ contacts mc

mc.last _ name lastname

SELECT * FROM
job _ current jc

mc.email email
```

为 mc.last_name 设定别名

如果两个 contact_id 都是 true，则符合条件

为“firstname”列设定别名

选择别名为“jc”的表中的所有列

设定某列的别名为“email”

如果没有找到指定的内容，才视为条件成立

为 my_contacts 设定别名

磨笔上阵

设计返回每个人的电子邮件地址的查询，电子邮件的主人至少拥有一项兴趣，但其在 job_current 表没有相应记录。

答案在第 416 页。

Greg 的Recruiting Service正式开业

现在，Greg 对于利用子查询取得数据已经非常熟悉了。他甚至还发现这些工具也能用在 INSERT、UPDATE、DELETE 语句中。

他租了一个小办公室来开展他的新业务，并决定举行一个盛大的启动派对。

问： 所以说，子查询可以放在子查询里吗?

答： 当然可以。嵌套查询的层次的确有其限制，但大多数 RDBMS 系统的支持已经超过我们能轻松使用的程度。

问： 在试着于子查询里构造子查询时，什么是最佳方案?

答： 最好的尝试是为问题里的各个部分设计小型查询，然后研究这些查询并找出结合它们的方式。如果想试着找出其薪资同为最高级别的网站设计员，应该把问题分解成：

寻找最高薪资的网站设计员

寻找薪资为 x 的人

然后把第一个答案填入 **x** 处。

问： 如果我不想使用子查询，可以改用联接吗?

答： 大多数情况下都可以。只需要多学一些联接，讲到这一点……

前往派对的路上

Greg 发现了一则让他有点困惑的小报新闻：

THE WEEKLY
INQUERYER

独家报导！关于子查询的惊人事实！

联接隐藏了实力

街头巷尾正在流传一种说法，子查询的能力其实“并未优于”联接，而且“需要有人说出这个事实”。

INQUERYER 编辑室

Troy Armstrong

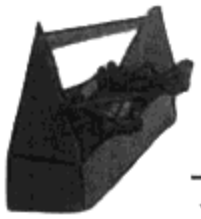
DATAVILLE 有个多年来都只是谣传的说法，现在终于由我们 INQUERYER 周刊的消息来源证实了——联接和子查询可以形成完全相同的查询。关于本地民众的最大困扰也已经确认了，任何子查询能实现的事情都能以相同类型的联接来实现。

本地教师 Heidi Musgrove 不禁泣诉：“太可怕了！我们该怎么告诉孩子们？他们认识的子查询，这么多花在学习使用子查询上的时间，结果只要用联接就可以了。真是太令人痛心疾首了！”

这个惊人真相造成的余波势必会继续影响下一章的走向，下一章，外联接也将于大众面前曝光。

校长 Heidi Musgrove 对于子查询的新发展也非常震惊。

一切都是浪费时间？子查询真的与联接没什么不同吗？请继续翻阅下一章。



你的SQL工具包

各位已经完成了第9章，而且精通了子查询的艺术，看一下你刚刚学会的技巧。若想复习本书中的所有技巧，请参考附录 3。

Outer query

外层查询。包含inner query/ subquery的查询。

Inner query

内层查询。在查询内的查询，也称为subquery。

Subquery

子查询。被另一个查询包围的查询，也称为inner query。

Noncorrelated sub-query

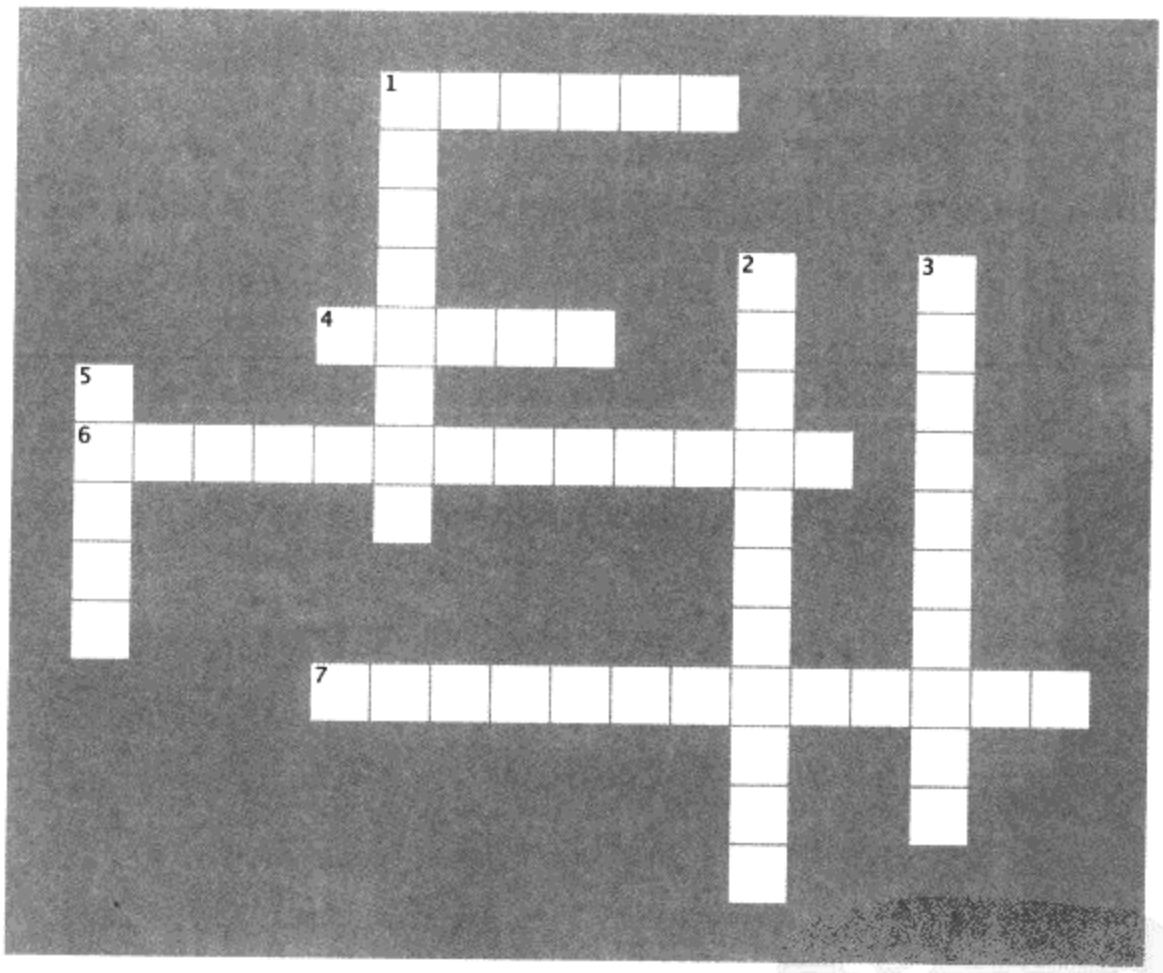
非关联子查询。一个独立而且不引用outer query的任何部分的subquery。

Correlated Subquery

关联子查询。一个依赖outer query的返回结果的subquery。

子查询填字游戏

各位可以从外层查询中分辨出内层查询，但能够解决这个字迷吗？所有解决字迷的关键词都在本章中。



横向

1. 子查询一定是个单一的 ____ 语句。

4. ____ 查询（____ query）包围着内层查询（或称子查询）。

6. 如果子查询可以独立运行而且并未引用来自外层查询的任何信息，即称为 ____ 子查询（____ subquery）。

7. 在 ____ 子查询（____ subquery）里，RDBMS先解释内层查询（子查询），再解释外层查询。

纵向

1. 在另一个查询里的查询被称为 ____。

2. 子查询一定在 ____ 里。

3. ____ 子查询（____ subquery）是指内层查询需依赖外层查询的结果才能被解析。

5. ____ 查询（____ query）被称为子查询。

磨笔上阵解答

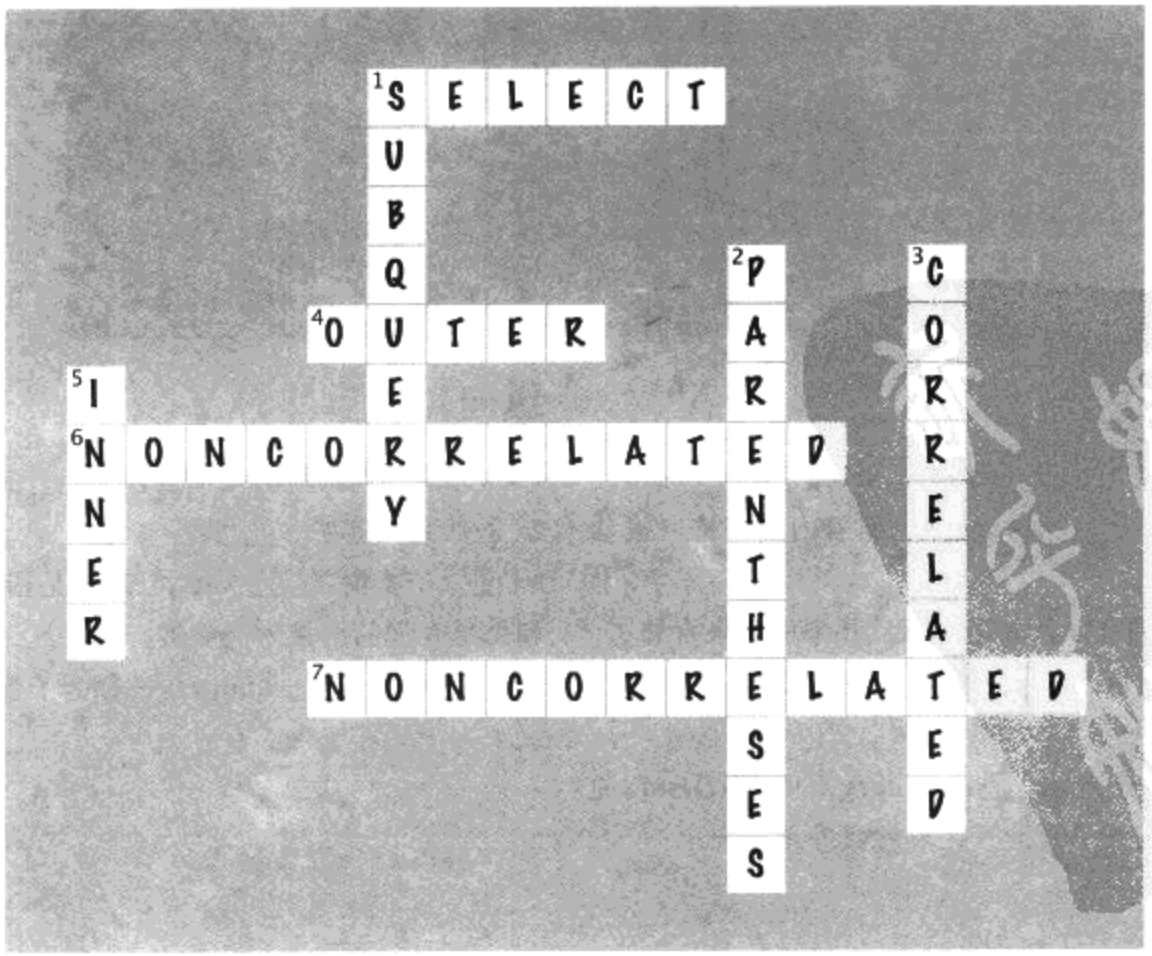
第411页上的练习。

设计返回每个人的电子邮件地址的查询，电子邮件的主人至少拥有一项兴趣，但其在 job_current 表中没有相应记录。

```
SELECT mc.email FROM my_contacts mc WHERE
EXISTS
(SELECT * FROM contact_interest ci WHERE mc.contact_ID = ci.contact_ID)
AND
NOT EXISTS
(SELECT * FROM job_current jc
WHERE mc.contact_id = jc.contact_id );
```

就像任意两项都需成立的条件一样，应在 WHERE 子句里使用 AND。

子查询填字游戏解答



10 外联接、自联接与联合

新策略

关于联接，我们只认识了一半。我们已经看过返回每个可能行的交叉联接，返回两张表中相符记录的内联接。但我们还没见过**外联接**，它可以在表中没有匹配记录的情况下返回记录；**自联接**（光听名称就很奇怪了），它可以联接表本身；还有**联合**，它可以合并查询结果。学会这些技巧后，你就可以采用自己需要的方式取得所有数据（而且本章也没有忘记探讨关于子查询的真相！）

清理旧数据

我想整理一下存储职业的表。我想表里面可能有些我不再需要的值。该怎样轻松找出未与 my_contacts 表中的任何记录连接的职业值呢？内联接无法完成我的任务。

这种信息可通过外联接（outer join）取得。

让我们一起了解外联接的用途，然后再说明如何找出不再需要的职业。

外联接返回某张表的所有行，并带有来自另一张表的条件相符的行。

使用内联接时，虽然要**比对来自两张表的行，但表的顺序并无影响**。

先简单回顾一下 equijoin 的运作方式。下例从两张表中取出 `toy _ id` 列同时出现于两张表中的列：

```
SELECT g.girl, t.toy
FROM girls g
INNER JOIN toys t
ON g.toy _ id = t.toy _ id;
```

equijoin 比较这两张表的记录以取得结果，本例比较 id 的值。

girls

girl_id	girl	toy_id
1	Jane	3
2	Sally	4
3	Cindy	1

toys

toy_id	toy
1	hula hoop
2	balsa glider
3	toy soldiers
4	harmonica
5	baseball cards
6	tinker toys
7	etch-a-sketch
8	slinky

查询结果

girl	toy
Cindy	hula hoop
Jane	toy soldiers
Sally	harmonica

一切都跟左、右有关

通过比较，可见外联接比我们学过的所有联接更加注重**两张表之间的关系**。

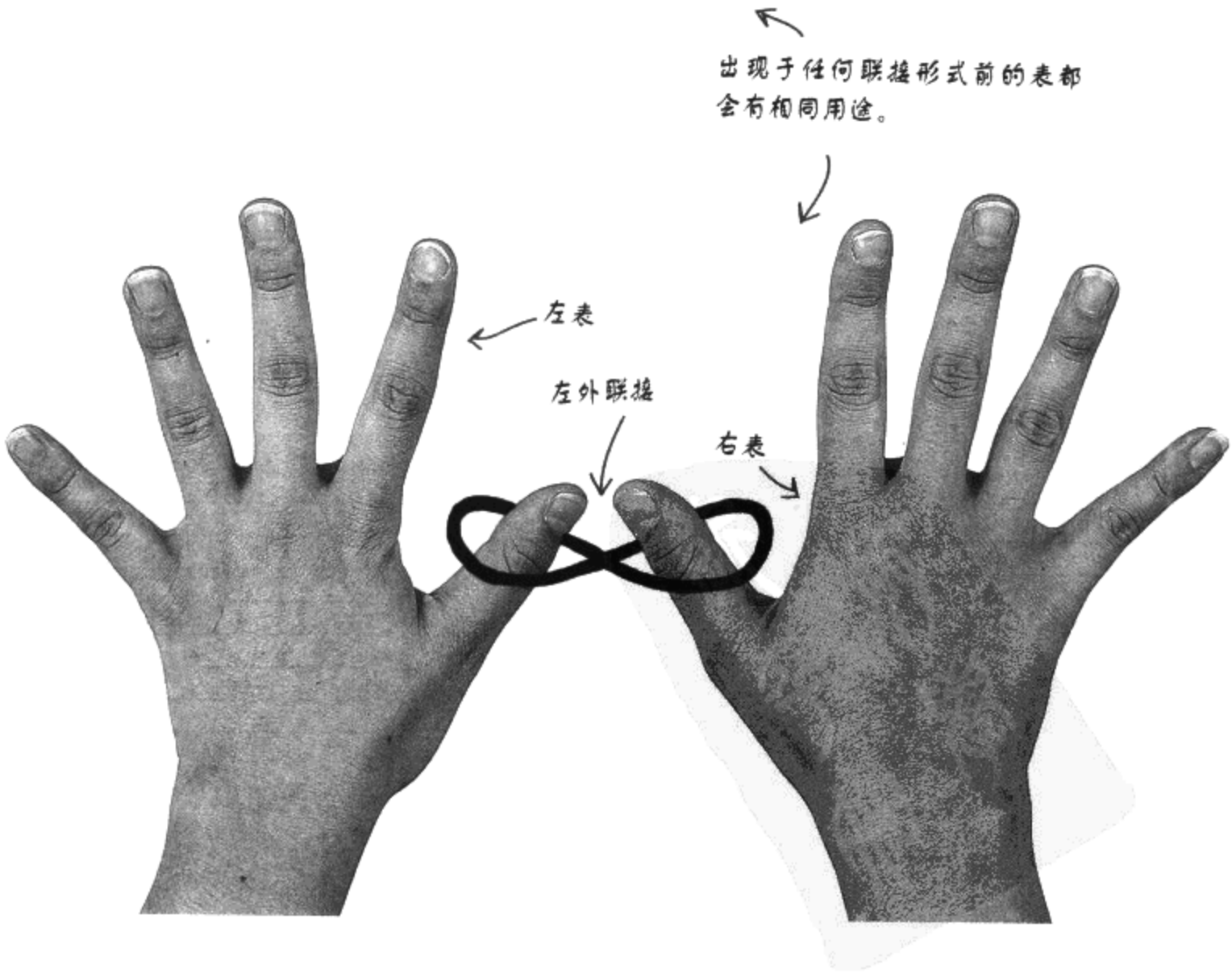
LEFT OUTER JOIN（左外联接）接收**左表的所有行**，并**用这些行与右表中的行匹配**。当左表与右表具有一对多关系时，左外联接特别有用。

LEFT OUTER JOIN 会匹配左表中的每一行及右表中符合条件的行。

理解外联接的最大秘密在于知道表在左边还是在右边。

在 **LEFT OUTER JOIN** 中，出现在 FROM 后、联接前的表称为**左表**，而出现在联接**后**的表当然就是**右表**。

出现于任何联接形式前的表都会有相同用途。



请看左外联接

我们可以利用左外联接找出每个女孩拥有的玩具。

下面即为左外联接的语法，使用了前一页的范例表。由于girls 表紧接在 FROM 后，所以它是左表；然后是 LEFT OUTER JOIN；最后列出 toys 表，它是右表：

所以，LEFT OUTER JOIN会取得左表(girls)的所有行，并把这些行与右表(toys)的行进行匹配。

```
SELECT g.girl, t.toy
FROM girls g
LEFT OUTER JOIN toys t
ON g.toy _ id = t.toy _ id;
```

位于LEFT OUTER JOIN前，所以girls是左表……

……位于LEFT OUTER JOIN后，所以toys是右表。

位于left outer join前，所以girls是左表……

……位于left outer join后，所以 toys 是右表……

girls

girl_id	girl	toy_id
1	Jane	3
2	Sally	4
3	Cindy	1

toys

toy_id	toy
1	hula hoop
2	balsa glider
3	toy soldiers
4	harmonica
5	baseball cards
6	tinker toys
7	etch-a-sketch
8	slinky

左外联接的查询结果

我们的查询结果与使用内联接时的一样。

查询结果

girl	toy
Cindy	hula hoop
Jane	toy soldiers
Sally	harmonica

就这样吗？神奇的地方在哪？外联接似乎跟内联接没有差别。

差别是：外联接一定会提供数据行，无论该行能否在另一个表中找出相匹配的行。

出现 NULL 是告诉我们没有相匹配的行。以女孩们拥有的玩具为例，结果集中的 NULL 表示没人拥有该玩具。这是个非常重要的信息！

左外联接的结果集中的NULL表示右表中没有找到与左表相符的记录。

磨笔上阵

试着画出下列查询的结果。

```
SELECT g.girl, t.toy
FROM toys t
LEFT OUTER JOIN girls g
ON g.toy _ id = t.toy _ id;
```

（提示：查询结果应该有8列。）

磨笔上阵解答

下面的查询更改了稍早的查询中表的顺序。请试着画出下列查询的结果。

```
SELECT g.girl, t.toy
FROM toys t
LEFT OUTER JOIN girls g
ON g.toy_id = t.toy_id;
```

左表。

右表。

这一次，*toys*（左表）的每一行均与*girls*（右表）的每一行进行匹配。

左表

toys

toy_id	toy
1	hula hoop
2	balsa glider
3	toy soldiers
4	harmonica
5	baseball cards
6	tinker toys
7	etch-a-sketch
8	slinky

右表

girls

girl_id	girl	toy_id
1	Jane	3
2	Sally	4
3	Cindy	1

表出现的顺序改变后，得到的结果如下：

girl	toy
Cindy	hula hoop
NULL	balsa glider
Jane	toy soldiers
Sally	harmonica
NULL	baseball cards
NULL	tinker toys
NULL	etch-a-sketch
NULL	slinky

如果匹配出相符数据，则呈现在查询结果的表格中。如果没有相符数据，结果表中还是会有该行，但会于不相符的行中填入 NULL。

表中列的顺序就是 SELECT 指定的顺序。左联接对于结果行的排列顺序没有影响。

以下为两组查询结果。请为各组结果写出它们所采用的左外联接，并画出符合该结果的 girls 表与 toys 表。

查询

左外联接的结果

girl	toy
Jen	squirt gun
Cleo	crazy straw
Mandy	NULL

左表

我们为大家填入了这个部分。

girls

girl_id	girl	toy_id
1	Jen	1
2	Cleo	2
3	Mandy	3

右表

查询

这个问题比较难！

左外联接的结果

girl	toy
Jen	squirt gun
Cleo	squirt gun
NULL	crazy straw
Sally	slinky
Martha	slinky

左表

右表

以下为两组查询结果。请为各组结果写出它们所采用的左外联接，并画出符合该结果的 girls 表与 toys 表。

查询

```
SELECT g.girl, t.toy
FROM girls g
LEFT OUTER JOIN toys t
ON g.toy_id = t.toy_id;
```

左外联接的结果

girl	toy
Jen	squirt gun
Cleo	crazy straw
Mandy	NULL

这些是出现在查询结果中的玩具。

左表

girls

girl_id	girl	toy_id
1	Jen	1
2	Cleo	2
3	Mandy	3

这可能是任何不存在 toys 表中的 toy_id，因为查询结果的 toy 列为 NULL。

右表

toys

toy_id	toy
1	squirt gun
2	crazy straw

查询

```
SELECT g.girl, t.toy
FROM toys t
LEFT OUTER JOIN girls g
ON g.toy_id = t.toy_id;
```

左外联接的结果

重复值表示多个女孩拥有相同的玩具。

NULL表示没人拥有 crazy straw。

girl	toy
Jen	squirt gun
Cleo	squirt gun
NULL	crazy straw
Sally	slinky
Martha	slinky

左表

toys

toy_id	toy
1	squirt gun
2	crazy straw
3	slinky

右表

girls

girl_id	girl	toy_id
1	Jen	1
2	Cleo	1
3	Sally	3
4	Martha	3

外联接与多个相符结果

你可能在习题中注意到了，虽然在另一个表中没有相符的记录，但你还是会取得数据行，在匹配出多条记录时就会取出多行。以下是左外连接的实际行动：

```
SELECT g.girl, t.toy
FROM toys t
LEFT OUTER JOIN girls g
ON g.toy _ id = t.toy _ id;
```

toys

toy_id	toy
1	squirt gun
2	crazy straw
3	slinky

girls

girl_id	girl	toy_id
1	Jen	1
2	Cleo	1
3	Sally	3
4	Martha	3

toys 中的玩具 squirt gun 与 girls 中 Jen 的记录比对：toys.toy_id = 1, girls.toy_id = 1
找到相符记录。
toys 中的玩具 squirt gun 与 girls 中 Clea 的记录比对：toys.toy_id = 1, girls.toy_id = 1
找到相符记录。
toys 中的玩具 squirt gun 与 girls 中 Sally 的记录比对：toys.toy_id = 1, girls.toy_id = 3
没有相符记录。
toys 中的玩具 squirt gun 与 girls 中 Martha 的记录比对：toys.toy_id = 1, girls.toy_id = 3
没有相符记录。
toys 中的玩具 crazy straw 与 girls 中 Jen 的记录比对：toys.toy_id = 2, girls.toy_id = 1
没有相符记录。
toys 中的玩具 crazy straw 与 girls 中 Clea 的记录比对：toys.toy_id = 2, girls.toy_id = 1
没有相符记录。
toys 中的玩具 crazy straw 与 girls 中 Sally 的记录比对：toys.toy_id = 2, girls.toy_id = 3
没有相符记录。
toys 中的玩具 crazy straw 与 girls 中 Martha 的记录比对：toys.toy_id = 2, girls.toy_id = 3
没有相符记录。
表已查找完毕，创建带有 NULL 值的行。
toys 中的玩具 slinky与 girls 中 Jen 的记录比对：toys.toy_id = 3, girls.toy_id = 1
没有相符记录。
toys 中的玩具 slinky与 girls 中 Jen 的记录比对：toys.toy_id = 3, girls.toy_id = 1
没有相符记录。
toys 中的玩具 slinky与 girls 中 Jen 的记录比对：toys.toy_id = 3, girls.toy_id = 3
找到相符记录。
toys 中的玩具 slinky与 girls 中 Jen 的记录比对：toys.toy_id = 3, girls.toy_id = 3
找到相符记录。

girl	toy
Jen	squirt gun
Cleo	squirt gun
NULL	crazy straw
Sally	slinky
Martha	slinky

右外联接

右外连接与左外联接完全一样，除了它是用右表与左表比对。

右外联接会根据左表评估右表。

```
SELECT g.girl, t.toy
FROM toys t
RIGHT OUTER JOIN girls g
ON g.toy_id = t.toy_id;
```

右表

左表

```
SELECT g.girl, t.toy
FROM girls g
LEFT OUTER JOIN toys t
ON g.toy_id = t.toy_id;
```

左表

右表

你已经在第420页看过这个查询了。

这两个查询都以girls为左表。

左表（用于两个查询）

girls

girl_id	girl	toy_id
1	Jane	3
2	Sally	4
3	Cindy	1

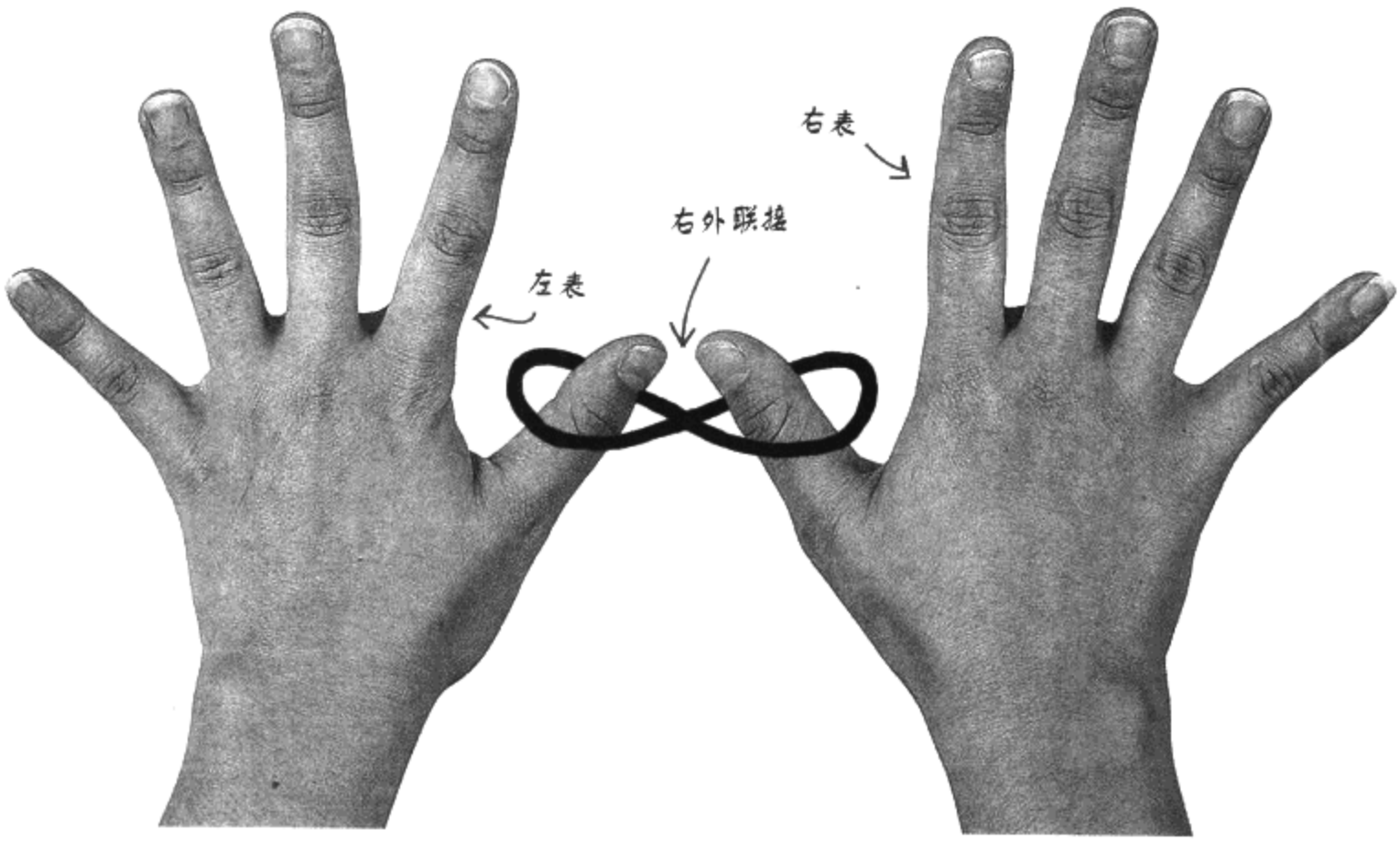
右表（用于两个查询）

toys

toy_id	toy
1	hula hoop
2	balsa glider
3	toy soldiers
4	harmonica
5	baseball cards
6	tinker toys
7	etch-a-sketch
8	slinky

查询结果

girl	toy
Cindy	hula hoop
Jane	toy soldiers
Sally	harmonica



问： 有使用左外联接取代右联接的理由吗？

答： 把LEFT改为RIGHT比改变查询中的表顺序简单多了。你只要改变一个词，不用交换两个表名与其别名的位置。

不过，一般来说，固定使用某一种联接的习惯会让事情更简单，例如固定只使用左外联接，需要时则搬动左右表的位置。这样较不容易搞混。

问： 如果有左外联接与右外联接，有可以返回两种联接结果的联接吗？

答： 有些RDBMS系统可以做到，称之为全外联接（FULLOUTERJOIN）。不可以做到的系统包括MySQL、SQL Server、Access。

同一个表可以同时作为外联接的左右表。

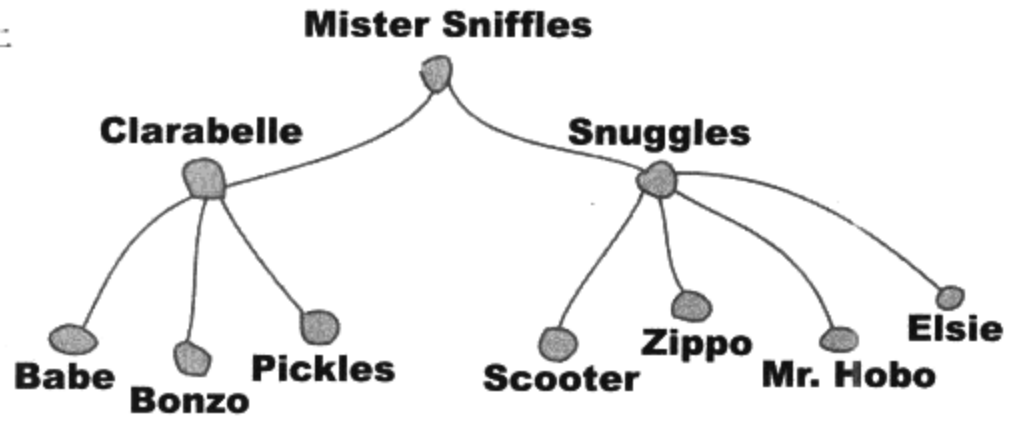
虽然听起来很奇怪，不过却很好用。让我们一起看看你或许需要将表外联接它自己的情况吧。

不过，首要的问题是 Dataville 的小丑。

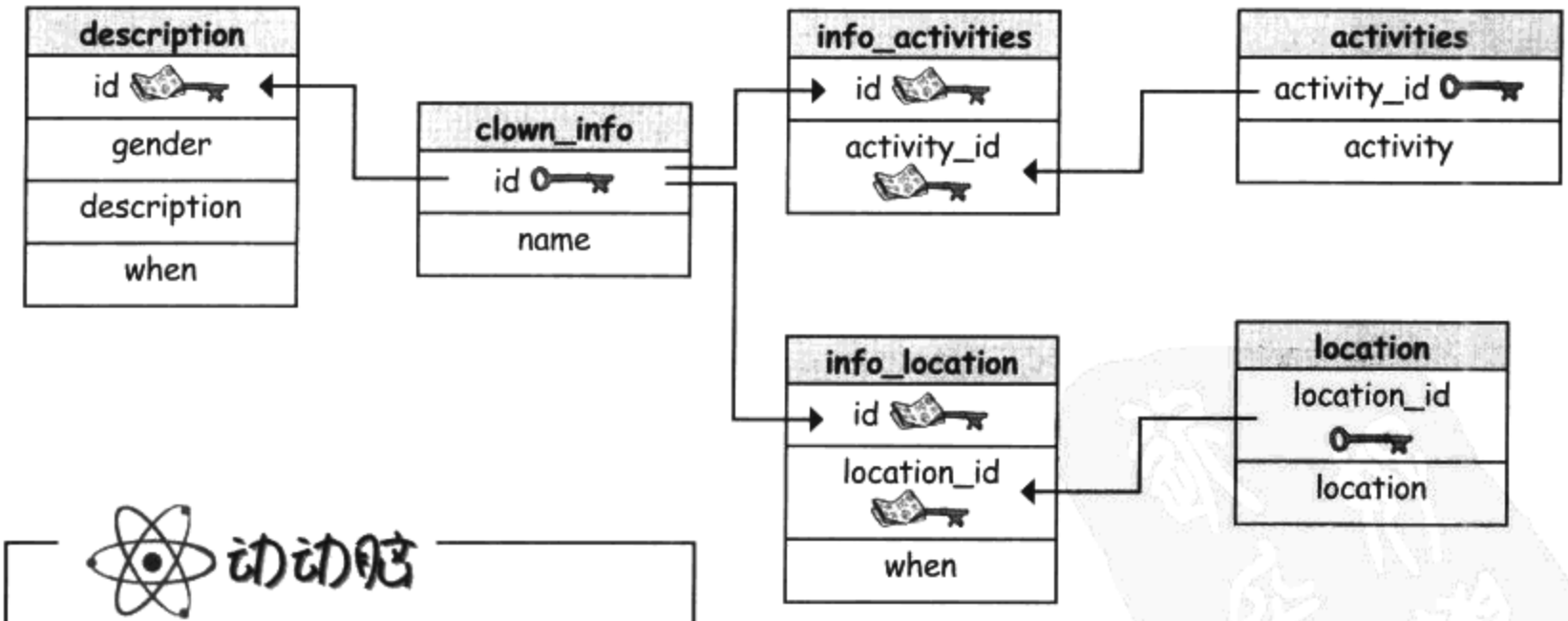
利用外联接……

我们又回到了Dataville，小丑被组织起来，由小丑头领负责管理他们的行动。事态的发展越来越严重，我们需要追踪这些小丑头领的行踪，还要找出向他们汇报的小丑。

右图是小丑的组织图。每个小丑都有一个上级头领，只有大头领Mister Sniffles例外。



先看一下当前的数据库模式（schema），想出最适用于当前情况的方式：



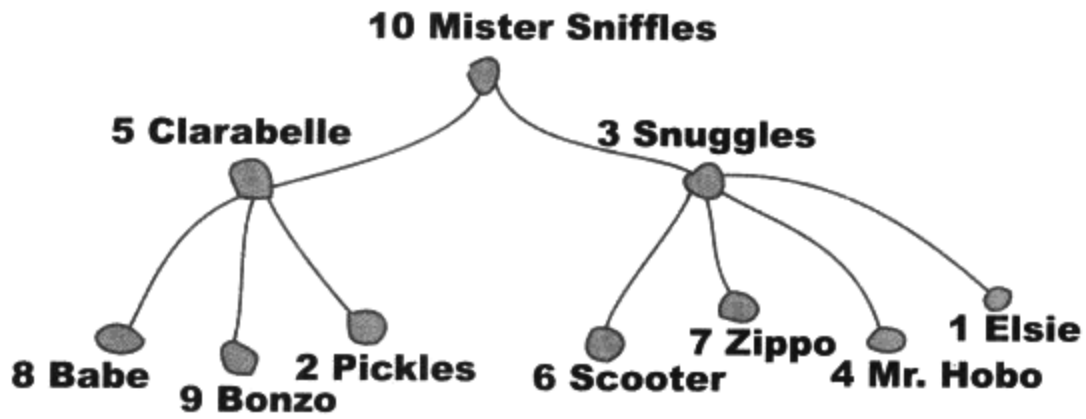
动动脑

该如何重新组织数据库模式（schmea），以便存储小丑头领们的信息？

Snibbles，他是Clarabelle与Snuggles的头领。

可以创建新表

我们可以创建一张表，列出每个小丑与他的头领的 ID。下图是小丑组织图并带有每个小丑的 ID。



下表列出了每个小丑的 ID以及从`clown _ info`表中得知的头领 ID。

clown _ boss

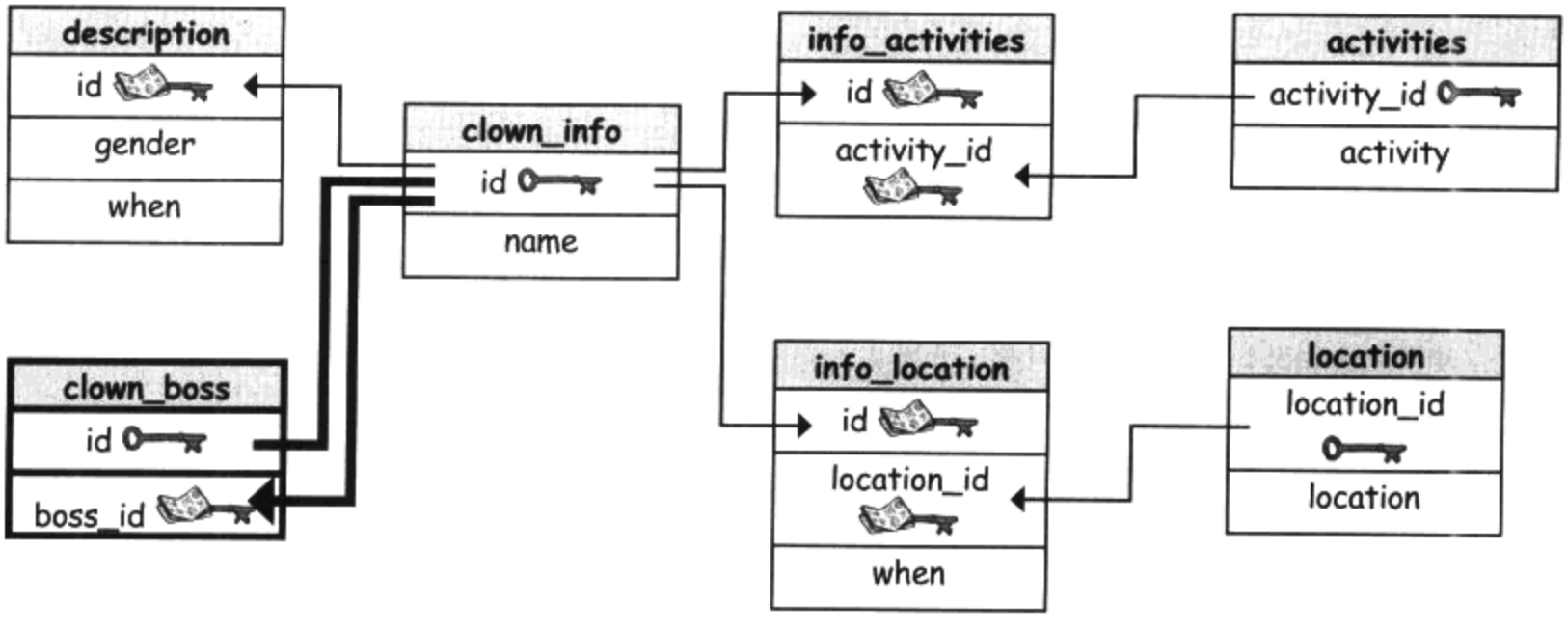
id	boss_id
1	3
2	5
3	10
4	3
5	10
6	3
7	3
8	5
9	5
10	10

我们的`clown _ boss`表与`clown _ info`表具有一对一关系。

Mister Snibbles 没有头领，不过他也需要一个 id，所以我们把他的 id 填入 boss_id 列以避免 NULL 的出现。

新表的位置

再看一下现在的数据库模式（schema），并研究新表如何放入模式（schema）中：



有点奇怪。新表具有一对一关系——主键 id列，同时还有一对多关系——boss _ id列。这张表的主键与外键都来自 clown _ info 表。

动动脑

有能追踪小丑头领且不需创建新表的办法吗？

自引用外键

其实，我们只需在 clown_info 表中加上新列来记录每个小丑头领的 ID。这一列就称为 boss_id，与 clown_boss 表中的设计一样。

在 clown_boss 表中boss_id 是外键。当我们把这一列加入 clown_info 时，它仍然是外键，虽然它也在同一张表中。这称为**自引用外键**（self-referencing foreign key）。“自引用”表示它是引用同一张表内另一列的键。

我们假设 Mister Sniffles 是他自己的头领，所以他的 boss_id 与他的 id 一样。这表示我们可以使用自引用外键作为 boss_id.

自引用外键是出于其他目的而用于同一张表的主键。

自引用外键是个出于其他目的而用于同一张表的主键。

这是新设立的boss_id列，我们直接把它加入clown_info表。这部分存储了自引用外键。

clown _ info

id	name	boss_id
1	Elsie	3
2	Pickles	5
3	Snuggles	10
4	Mr. Hobo	3
5	Clarabelle	10
6	Scooter	3
7	Zippo	3
8	Babe	5
9	Bonzo	5
10	Mister Sniffles	10

这里引用同一张表中的id字段，所以能找出谁是Elsie的头领。

又出现了，Mister Sniffles的boss_id 就是他自己的 id。

联接表与它自己

假设需要列出每个小丑及他们的头领。用下面这段 SELECT 可以轻易地列出每个小丑的姓名及他们头领的 id：

```
SELECT name, boss_id FROM clown_info;
```

但我们其实需要小丑的姓名与他们头领的姓名：

name	boss
Elsie	Snuggles
Pickles	Clarabelle
Snuggles	Mister Sniffles
Mr. Hobo	Snuggles
Clarabelle	Mister Sniffles
Scooter	Snuggles
Zippo	Snuggles
Babe	Clarabelle
Bonzo	Clarabelle
Mister Sniffles	Mister Sniffles

磨笔上阵

假设有两个完全相同的表 clown_info1 与 clown_info2。请设计一个联接，以取得包含每个小丑的姓名与他们头领的姓名的结果集。

clown _ info1

id	name	boss_id
1	Elsie	3
2	Pickles	5
3	Snuggles	10
4	Mr. Hobo	3
5	Clarabelle	10
6	Scooter	3
7	Zippo	3
8	Babe	5
9	Bonzo	5
10	Mister Sniffles	10

clown _ info2

id	name	boss_id
1	Elsie	3
2	Pickles	5
3	Snuggles	10
4	Mr. Hobo	3
5	Clarabelle	10
6	Scooter	3
7	Zippo	3
8	Babe	5
9	Bonzo	5
10	Mister Sniffles	10

假设有两个完全相同的表 clown_info1 与 clown_info2。请设计一个联接，以取得包含每个小丑的姓名与他们头领的姓名的结果集。

clown _ info1

id	name	boss_id
1	Elsie	3
2	Pickles	5
3	Snuggles	10
4	Mr. Hobo	3
5	Clarabelle	10
6	Scooter	3
7	Zippo	3
8	Babe	5
9	Bonzo	5
10	Mister Sniffles	10

clown _ info2

id	name	boss_id
1	Elsie	3
2	Pickles	5
3	Snuggles	10
4	Mr. Hobo	3
5	Clarabelle	10
6	Scooter	3
7	Zippo	3
8	Babe	5
9	Bonzo	5
10	Mister Sniffles	10

我们给第二列取别名为“boss”，这样就不会因为有两列都名为“name”而搞混了。

```
SELECT c1.name, c2.name AS boss
FROM clown_info1 c1
INNER JOIN clown_info2 c2
ON c1.boss_id = c2.id;
```

这是用于匹配来自 clown_info1 的 boss_id 列与 clown_info2 的 id 列。

我们需要自联接

在刚才的“磨笔上阵”练习中，我们列出了两张相同的表以供联接。但在规范的数据库中，相同的表不会出现两次。于是，我们可以**改用自联接（self-join）以模拟联接两张表的效果。**

clown _ info

id	name	boss_id
1	Elsie	3
2	Pickles	5
3	Snuggles	10
4	Mr. Hobo	3
5	Clarabelle	10
6	Scooter	3
7	Zippo	3
8	Babe	5
9	Bonzo	5
10	Mister Sniffles	10

```
SELECT c1.name, c2.name AS boss
FROM clown _ info c1
INNER JOIN clown _ info c2
ON c1.boss _ id = c2.id;
```

clown_info表会被使用两次。它的别名分别是“c1”（由此取得 boss_id）与“c2”（由此取得小丑头领的姓名：name）。

不需要两张相同的表，而是使用clown_info 表两次，第一次设定它的别名为c1，第二次则设定别名为 c2。然后利用内联接来连接 boss_id（来自c1）与头领的姓名（name，来自 c2）。

name	boss
Elsie	Snuggles
Pickles	Clarabelle
Snuggles	Mister Sniffles
Mr. Hobo	Snuggles
Clarabelle	Mister Sniffles
Scooter	Snuggles
Zippo	Snuggles
Babe	Clarabelle
Bonzo	Clarabelle
Mister Sniffles	Mister Sniffles

这一列由INNER JOIN得到，联接的两方分别是clown_info 表的第一个实例（c1）的 boss_id 列和它的第二个实例（c2）的头领姓名列。

自联接能把单一表当成两张具有完全相同的信息的表来进行查询。

另一种取得多张表内容的方式

我想试着列出 gregs_list 中存储的所有职务，但我不知道该怎么同时列出这三张表中的所有职务。

以下即为 Greg 提到的三张表。

目前，Greg 创建了三条独立的 SELECT 语句：

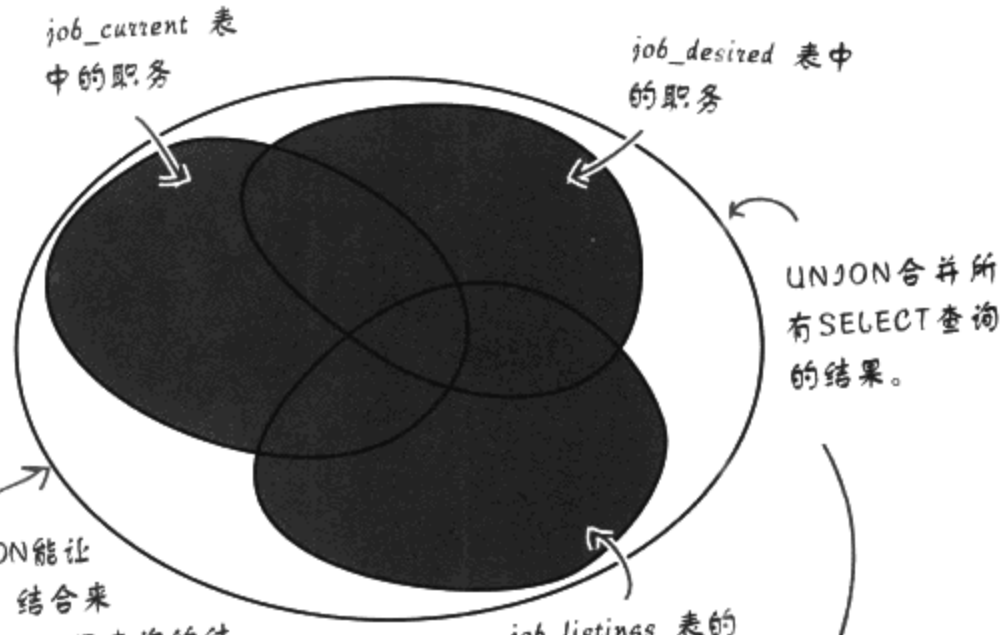
```
SELECT title FROM job _ current;
SELECT title FROM job _ desired;
SELECT title FROM job _ listings;
```

上述查询当然会成功，但 Greg 想把三份查询结果合并，他想只用一个查询取得三张表中记录的所有职务。

可以利用 UNION

还有另一种取得多张表的查询结果的方式：UNION，联合。

UNION根据我们在SELECT中指定的列，把两张或更多张表的查询结果合并至一个表中。可以把UNION的查询结果想成“重叠了”每个SELECT的查询结果。




```
SELECT title FROM job _ current
UNION
SELECT title FROM job _ desired
UNION
SELECT title FROM job _ listings;
```

UNION能让Greg结合来自三个不同查询的结果为一张结果表。

这只是Greg结合查询后得到的数百条职务记录的示例。

title
Accountant
Lawyer
Programmer
Web Designer
Cat Herder
Chef
Psychologist
Barber
Teacher
Writer

Greg 发现结果集中没有重复的记录，但职务没有依序排列，所以他再次执行这个查询，但在每个 SELECT 语句中加入了 ORDER BY 子句。

```
SELECT title FROM job _ current ORDER BY title
UNION
SELECT title FROM job _ desired ORDER BY title
UNION
SELECT title FROM job _ listings ORDER BY title;
```

Greg 在每条语句中加入了ORDER BY 子句，希望查询到的职务会依字母排序。

动动脑

你觉得 Greg 的新查询会产生什么结果？

UNION的使用限制

Greg 的新查询无法使用！他只得到一个错误信息，因为数据库软件不知道如何解释多个ORDER BY。

UNION只能接受一个ORDER BY且必须位于语句末端。这是因为UNION 已经把多个SELECT语句的查询结果串联起来并分组了。

联合的规则大家都应该知道。

SQL 联合规则

每个 SELECT 语句中列的数量必须一致。不可以由第一条语句选取了两列，由其他语句却只选取一列。

每个 SELECT 语句包含的表达式与统计函数也必须相同。

SELECT 语句的顺序不重要，不会改变结果。

SQL 联合规则

SQL 默认会清除联合的结果中的重复值。

列的数据类型必须相同或者可以互相转换。

如果出于某些原因而需要看到重复数据，可以使用UNION ALL 运算符。这个运算符返回每个相符的记录，而不只是没有重复的记录。

UNION规则的运作

使用 UNION 合并的 SELECT 语句中列的数量必须一致。不可以由第一条语句选取了两列，由其他语句却只选取一列。

每条SELECT语句选取的列的数量必须相同。

```
SELECT title FROM job_current
UNION
SELECT title FROM job_desired
UNION
SELECT title FROM job_listings
ORDER BY title;
```

如果想对查询结果排序，请在合并的最后一条SELECT语句中加入 ORDER BY，这样就能排列整个结果集的顺序。

title
Baker
Cat Herder
Cat Wrangler
Clown
Dog Trainer
Hairdresser
Jeweler
Lawyer
Mechanic
Neurosurgeon

预期的查询结果示例。

在本例中，三列都有相同的数据类型，VARCHAR。所以，查询返回的列也是 VARCHAR 类型的。

动动脑

如果联合的列有不同数据类型，你觉得会发生什么事？

UNION ALL

UNION ALL的运作方式与UNION相同，只不过它会返回列的所有内容，而不是每个值的复制实例。

这一次，我们希望看到三张表中所有存储在 title 列中的值。

```
SELECT title FROM job_current
UNION ALL
SELECT title FROM job_desired
UNION ALL
SELECT title FROM job_listings
ORDER BY title;
```

title
Baker
Baker
Cat Herder
Cat Wrangler
Clown
Clown
Clown
Dog Trainer
Dog Trainer
Hairdresser
Jeweler
Lawyer
Lawyer
Lawyer
Lawyer
Mechanic
Neurosurgeon

相同职务不只列出一次。

到目前为止，UNION家族都使用相同数据类型的列。但我们也可能需要结合不同类型列的UNION。

联合规则说，“选取的列的数据类型必须可以互相转换”，也就是说，数据会试着转换为相容类型，如果无法转换，查询就会失败。

以联合 INTERGER与VARCHAR类型为例，因为VARCHAR无法转换成整型，所以查询结果会把INTERGER 转换为 VARCHAR。

从联合创建表

由UNION返回的数据类型其实不太容易分辨，除非用某种方式捕获类型。使用CREATE TABLE AS可以捕获UNION的结果，以便仔细观察。

CREATE TABLE AS接收来自SELECT查询的结果并把结果制作成一张表。下例即以UNION结果制成新表 my_union：

新表名称

```
CREATE TABLE my_union AS
SELECT title FROM job_current UNION
SELECT title FROM job_desired
UNION SELECT title FROM job_listings;
```

这部分是我们刚学到的 UNION。任何一种 SELECT 查询都可用于创建新表。

磨笔上阵

创建job_current的contact_id列与job_listings的salary列的UNION。

.....
.....

猜猜联接结果的数据类型，然后利用刚刚设计的UNION写出 CREATE TABLE AS 语句。

.....
.....
.....

请用 DESC 观察表的类型，看看你的猜测是否正确。

.....

答案请见第 453 页。

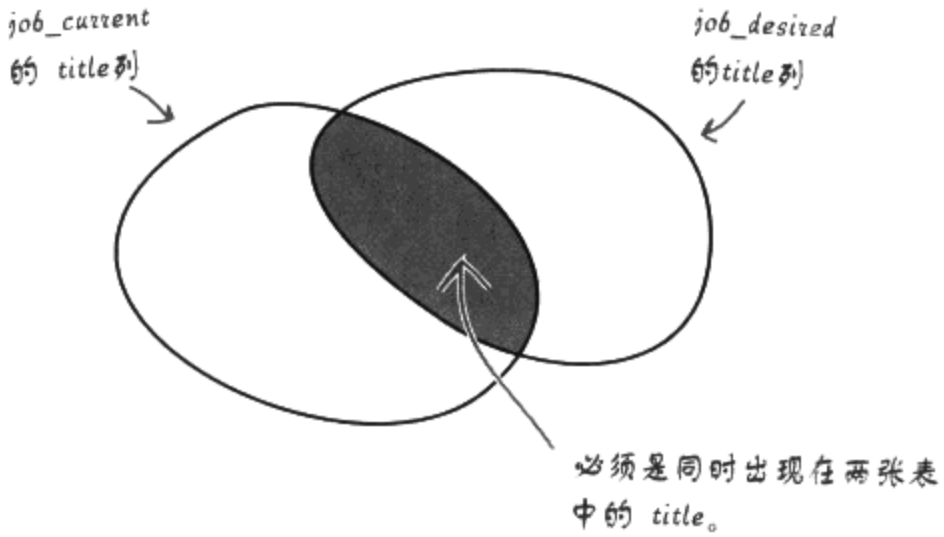
INTERSECT与EXCEPT

INTERSECT（交集）与 EXCEPT（差集）的使用方式与 UNION 大致相同——都是找出查询结果重叠的部分。

INTERSECT只会返回同时在第一个与第二个查询中的列。

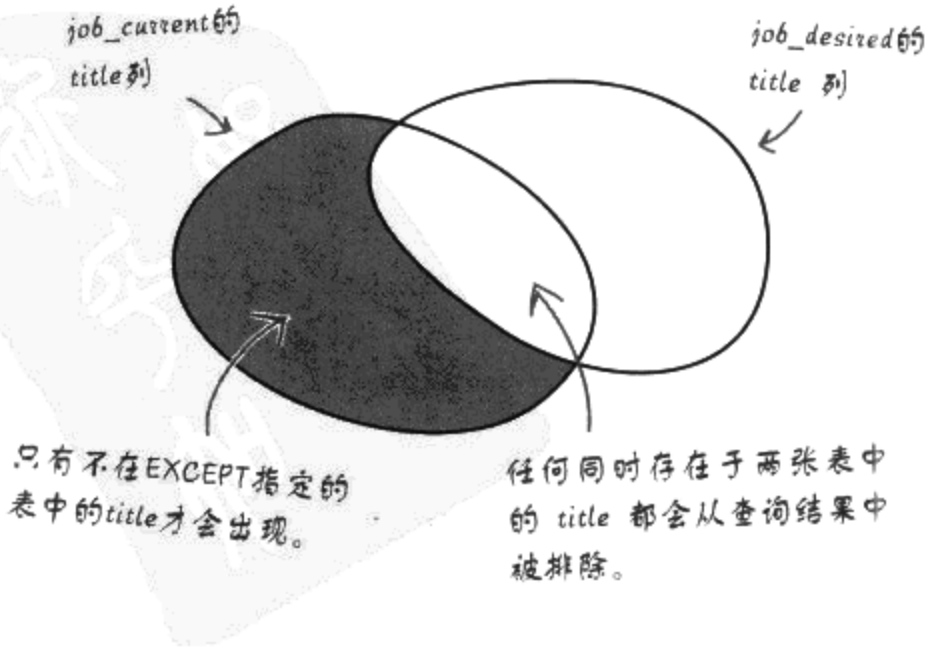
这两个运算符不在我的 MySQL 中。

```
SELECT title FROM job _ current
INTERSECT
SELECT title FROM job _ desired;
```



EXCEPT 返回只出现在第一个查询，而不在第二个查询中的列。

```
SELECT title FROM job _ current
EXCEPT
SELECT title FROM job _ desired;
```

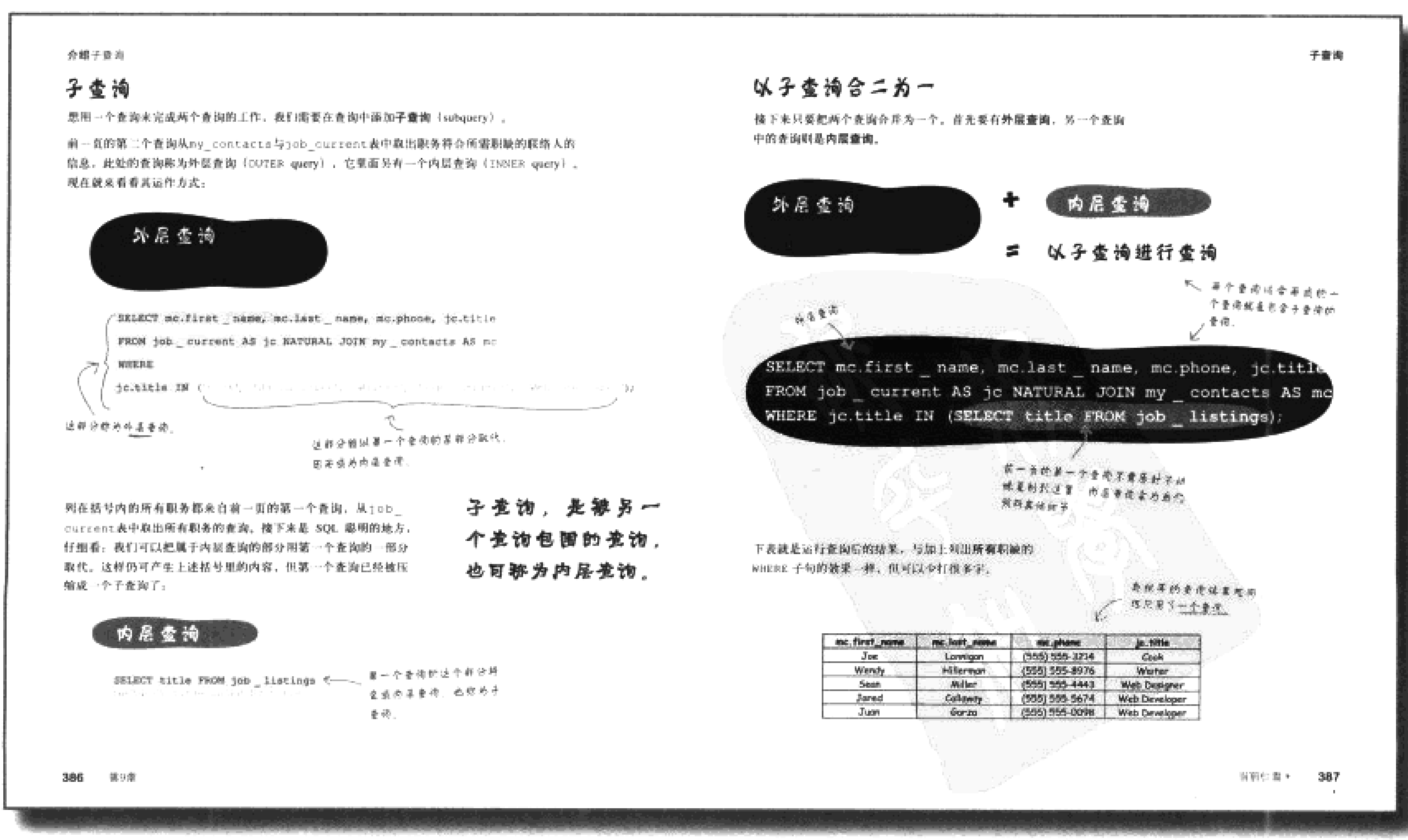


我们已经解决了联接，应该进入……

（呃，没错，我刚刚想说的就是……）

应该进入子查询与联接的比较了

几乎所有能用子查询办到的事都能用联接实现。让我们回到第9章的开头部分。



介绍子查询

子查询

想用一个查询来完成两个查询的工作，我们需要在查询中添加**子查询**（subquery）。

前一页的第二个查询从my_contacts与job_current表中取出职务符合所需职缺的联络人的信息。此处的查询称为外层查询（OUTER query），它里面另有一个内层查询（INNER query）。现在就来看看其运作方式：

外层查询

```
SELECT mc.first _ name, mc.last _ name, mc.phone, jc.title
FROM job _ current AS jc NATURAL JOIN my _ contacts AS mc
WHERE
jc.title IN (...);
```

则在括号内的所有职务都来自前一页的第一个查询，从job_current表中取出所有职务的查询。接下来是SQL聪明的地方，仔细看：我们可以把属于内层查询的部分用第一个查询的一部分取代。这样仍可产生上述括号里的内容，但第一个查询已经被压缩成一个子查询了：

子查询，是被另一个查询包围的查询，也可称为内层查询。

内层查询

```
SELECT title FROM job _ listings
```

386　第9章

子查询

以子查询合二为一

接下来只要把两个查询合并为一个。首先要有**外层查询**，另一个查询中的查询则是**内层查询**。

外层查询 + 内层查询 = 以子查询进行查询

```
SELECT mc.first _ name, mc.last _ name, mc.phone, jc.title
FROM job _ current AS jc NATURAL JOIN my _ contacts AS mc
WHERE jc.title IN (SELECT title FROM job _ listings);
```

下表就是运行查询后的结果，与加上列出所有职缺的WHERE子句的效果一样，但可以少打很多字。

mc.first_name	mc.last_name	mc.phone	jc.title
Joe	Lonnigan	(555) 555-3214	Cook
Wendy	Hillerman	(555) 555-8976	Waiter
Sean	Miller	(555) 555-4443	Web Designer
Jared	Callaway	(555) 555-5674	Web Developer
Juan	Garza	(555) 555-0098	Web Developer

387

把子查询转换为联接

回到第 9 章，这是我们创建的第一个子查询：

外层查询

```
SELECT mc.first _ name, mc.last _ name, mc.phone, jc.title
FROM job _ current AS jc NATURAL JOIN my _ contacts AS mc
WHERE jc.title IN (SELECT title FROM job _ listings);
```

内层查询

下表则是运行查询后得到的结果：

mc.first_name	mc.last_name	mc.phone	jc.title
Joe	Lonnigan	(555) 555-3214	Cook
Wendy	Hillerman	(555) 555-8976	Waiter
Sean	Miller	(555) 555-4443	Web Designer
Jared	Callaway	(555) 555-5674	Web Developer
Juan	Garza	(555) 555-0098	Web Developer

磨笔上阵

以下是加上子查询的 WHERE 子句，但以内联接的方式重写：

```
SELECT mc.first _ name, mc.last _ name, mc.phone, jc.title
FROM job _ current AS jc NATURAL JOIN my _ contacts AS mc
INNER JOIN job _ listings jl
ON jc.title = jl.title;
```

可使用 INNER JOIN 替代包含子查询的 WHERE 子句。

解释查询中的 INNER JOIN 部分将如何像子查询那样取得相同结果。

.....

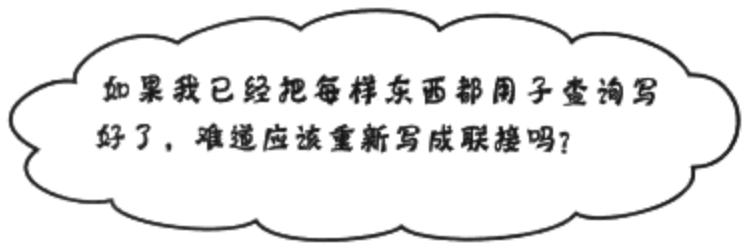
.....

.....

你觉得哪个查询更容易理解？

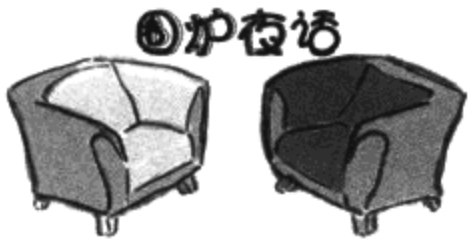
.....

答案请见第 453 页。



不用，如果子查询已经实现了你的目标，就没有重写的必要。

但是总有选择其中一种的理由吧……



今晚主题：**联接 v.s. 子查询，谁能胜出**

联接

很明显，我才是大多数情况下的最佳选择。我很容易理解，而且我的执行速度比那边的“老”子查询快多了。

你能完成的工作我也能办到，而且我还比你容易理解。

听你在那边瞎说。难道你的CORRELATED与NONCORRELATED 有好到哪里去了吗？

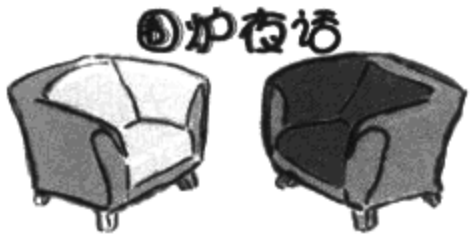
子查询

什么？你说谁“老”啊？直到最近，我才出现在每种 RDBMS 中。因为很多程序设计师需要我，所以才会有我的加入。

你以为大家比小孩还好骗吗？用你那些讨好观众的噱头——INNER 和 OUTER 来骗人吗？这些鬼东西只会徒增使用上的困扰……

好吧好吧！我们都有自己的行话，没错。不过用我的话，通常只要分别搞清楚内层查询和外层查询的部分就可以了。

下页待续。



今晚主题：**联接 v.s. 子查询，谁能胜出**

联接

不见得吧，关联子查询大人！好吧，就先把这个问题放到一边。但在结果集需要取得来自多张表的列时，我仍然是比较好的选择。事实上，我是唯一的选择。

这可能是真的，但是大家比较容易理解我的运作方式。你看，甚至可以使用别名，而不用重复输入相同的表名。

哈哈哈，真好笑。太优秀了，用不到别名是吧？你自以为比我简单很多，不过关联子查询又该怎么说？它是我见过最拐弯抹角的玩意。

炫耀鬼！

子查询

所以你才不擅长聚合值／统计值。在一个没有子查询的 WHERE 子句中无法使用统计函数。统计运算弥补了我不能返回多列的缺点。你实在太复杂了。

是，讲到这些别名，它们不是让事态更难追踪吗？而且，我也可以为记录使用别名啊，你知道的。只不过我在使用别名时太直接了。我根本懒得用别名。

呃……嗯……对。不过我能与 UPDATE、INSERT 与 DELETE 一起使用，这个你就不行了吧！

请查看下列来自第9章的子查询，并研究它们是否能写成不用子查询的形式，还是维持子查询的使用会较好。可以使用联接。

列出薪资等于 job_listings 表中最高薪资的职务名称。

```
SELECT title FROM job_listings WHERE salary = (SELECT
MAX(salary) FROM job_listings);
```

.....

.....

.....

最好使用子查询吗？.....

列出薪资高于平均薪资者的姓名。

```
SELECT mc.first_name, mc.last_name FROM my_contacts
mc NATURAL JOIN job_current jc WHERE jc.salary >
(SELECT AVG(salary) FROM job_current);
```

.....

.....

.....

最好使用子查询吗？.....

请查看下列来自第9章的子查询，并研究它们是否能写成不用子查询的形式，还是维持子查询的使用会较好。可以使用联接。

列出薪资等于 job_listings 表中最高薪资的职务名称。

```
SELECT title FROM job_listings WHERE salary = (SELECT
MAX(salary) FROM job_listings);
```

SELECT *title* FROM *job_listings* ORDER BY *salary* DESC LIMIT 1;

修改后查询只返回一个结果，也就是薪资最高的行。

最好使用子查询吗？ 不是。

列出薪资高于平均薪资者的姓名。

```
SELECT mc.first_name, mc.last_name FROM my_contacts mc
NATURAL JOIN job_current jc WHERE jc.salary > (SELECT
AVG(salary) FROM job_current);
```

嗯……不能使用 LIMIT 和 ORDER BY，就无法取得薪资的平均值。

最好使用子查询吗？ 是的。

在上个问题的解法中，我们可以使用 LIMIT 从有序的薪资表中取出最大的薪资值。但这个大于薪资平均值的问题无法排序，所以无法通过 LIMIT 取得所需结果。

把自联接变成子查询

我们看过子查询如何转变成联接，现在来看看转变成子查询的自联接。

还记得新加入 clown_info表的boss_id列吗？以下是我们使用的自联接，其中把clown_info的一个实例称为c1，另一个称为c2。

这一栏指出每个小丑的头领

clown _ info

id	name	boss_id
1	Elsie	3
2	Pickles	5
3	Snuggles	10
4	Mr. Hobo	3
5	Clarabelle	10
6	Scooter	3
7	Zippo	3
8	Babe	5
9	Bonzo	5
10	Mister Sniffles	10

变身前

```
SELECT c1.name, c2.name AS boss
FROM clown _ info c1
INNER JOIN clown _ info c2
ON c1.boss _ id = c2.id;
```

clown_info 的第一个实例

clown_info 的第二个实例

变身后

当我们把自联接转变为子查询时，因为子查询需依赖外层查询的结果才能取得正确的boss_id，所以子查询将是关联子查询，而关键性可由SELECT选取的列揭示。

外层查询。

子查询出现在SELECT的选取列表中。

```
SELECT c1.name,
(SELECT name FROM clown _ info
WHERE c1.boss _ id = id) AS boss
FROM clown _ info c1;
```

子查询依赖外层查询的结果才能取得正确的 *boss_id*，所以它是关联子查询。

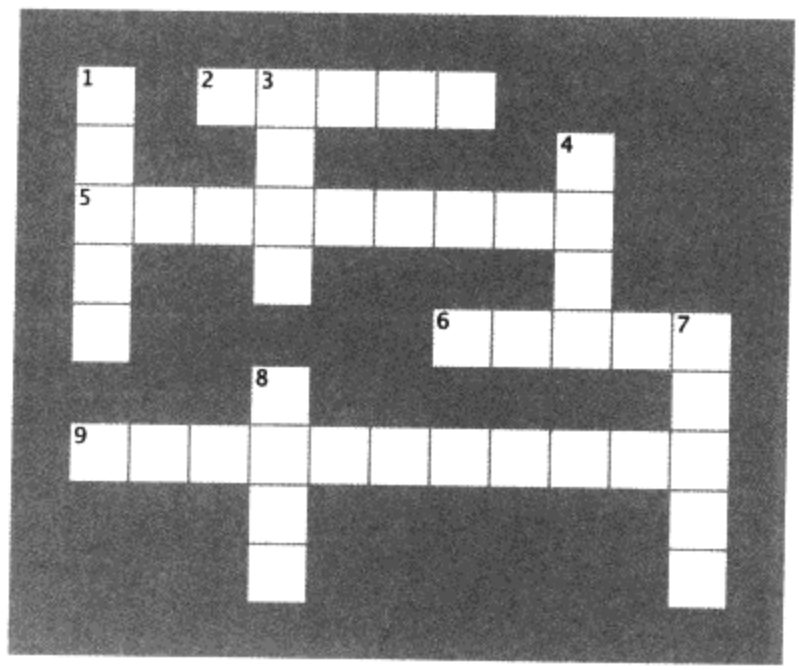
Greg 的公司正在成长

Greg最近忙着学习联接与子查询，他请了两位朋友来帮他处理较不复杂的查询。

但是他们都不知道自己在做什么。当很多人，而且是没有可靠的 SQL 技巧的人同时在一个数据库中工作时，Greg 打算找出到底会发生什么事。

联接与联合填字游戏

这一章就像装了涡轮增压引擎一样进展飞快，当然也有很多等待学习的要点。这个填字游戏有助于牢记本章要点。所有答案都出自本章内容。



横向

2. 把两个或多个查询的结果合并为一张表，合并的依据为 SELECT 语句中指定的列。

5. 默认情况下，SQL 会抑制联合结果中的 _____值。

6. _____ JOIN，不论另一张表中是否有相符记录，都会返回结果。

9. self-_____ 外键其实是表的主键，因其他用途而再次用于同一张表。

纵向

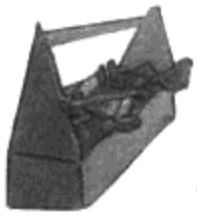
1. 利用内联接可以比对两张表的记录，但表的 _____ 并不重要。

3. 如果 _____ 在左外联接的结果中，表示右表中没有与左表相符的值。

4. _____OUTER JOIN 会把左表中的所有记录都拿来与右表比对。

7. _____OUTER JOIN 会把右表拿来与左表比对。

8. 使用 _____-JOIN，我们可以模拟两张表的效果。



你的SQL工具包

各位现在已经进入最省力的状态了。我们学习了外联接、自联接与联合，甚至还知道如何把联接转换为子查询，反向转换也已经会了。如果需要本书工具的完整列表，请参考附录 3。

LEFT OUTER JOIN

左外联接。LEFT OUTER JOIN接受左表中的所有记录，并从右表比对出相符记录。

SELF-REFERENCING FOREIGN KEY

自引用外键。这种外键就是同一张表的主键，但作为其他用途。

RIGHT OUTER JOIN

右外联接。RIGHT OUTER JOIN接受右表中的所有记录，并从左表比对出相符记录。

SELF-JOIN

自联接。SELF-JOIN能用一张表做出联接两张完全相同表的效果。

UNION 与 UNION ALL

UNION（联合）根据SELECT指定的列合并两个或多个查询的结果为一张表。

UNION 默认为隐藏重复的值，UNION ALL 则可包含重复的值。

INTERSECT

使用这个关键字返回同时存在于第一个与第二个查询中的值。

EXCEPT

使用这个关键字返回在第一个查询中但不在第二个查询中的值。

CREATE TABLE AS

使用本命令从任何SELECT语句的结果创建表。

磨笔上阵 解答

第441页上的习题。

创建job_current的contact_id列与job_listings的salary列的UNION。

```
SELECT contact_id FROM job_current UNION
SELECT salary FROM job_listings;
```

猜猜联接结果的数据类型，然后利用刚刚设计的UNION写出CREATE TABLE AS 语句。

```
CREATE TABLE my_table SELECT contact_id
FROM job_current UNION SELECT salary FROM
job_listings;
```

请用 DESC 观察表的类型，看看你的猜测是否正确。

```
DEC(12,2)
```

磨笔上阵 解答

第444页上的习题。

以下是加上子查询的 WHERE 子句，但以内联接的方式重写：

```
SELECT mc.first _ name, mc.last _ name, mc.phone, jc.title
FROM job _ current AS jc NATURAL JOIN my _ contacts AS mc
INNER JOIN job _ listings jl
ON jc.title = jl.title;
```

可使用INNER JOIN替代包含子查询的WHERE子句。

解释查询中的 INNER JOIN 部分将如何像子查询那样取得相同结果。

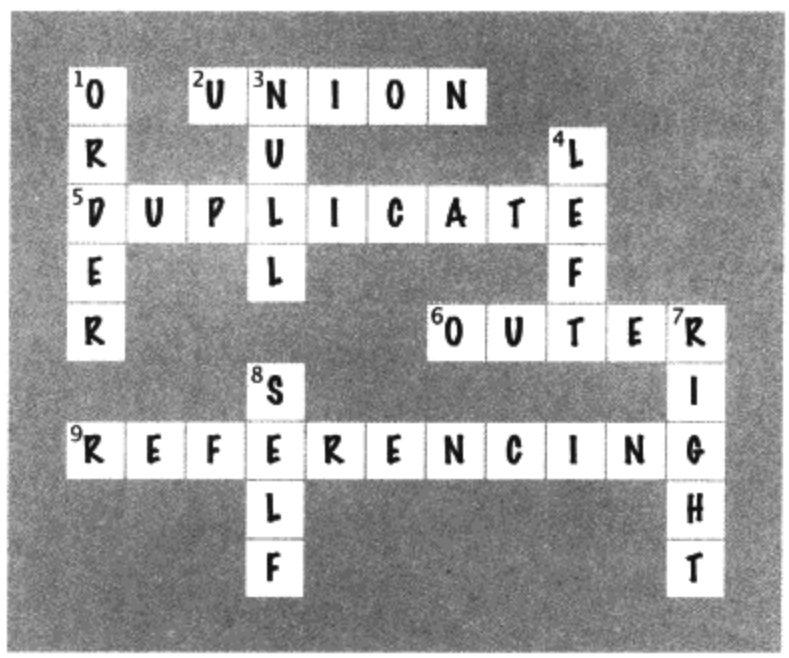
INNER JOIN只会呈现jc.title = jl.title的记录，相当于加上子查询的WHERE子句：

```
WHERE jc.title IN (SELECT title FROM job_listings);
```

你觉得哪个查询更容易理解？

这没有标准答案！但你的答案能显示出你已经开始思考未来处理数据的方式。

联接与联合填字游戏解答



11 约束、视图与事务

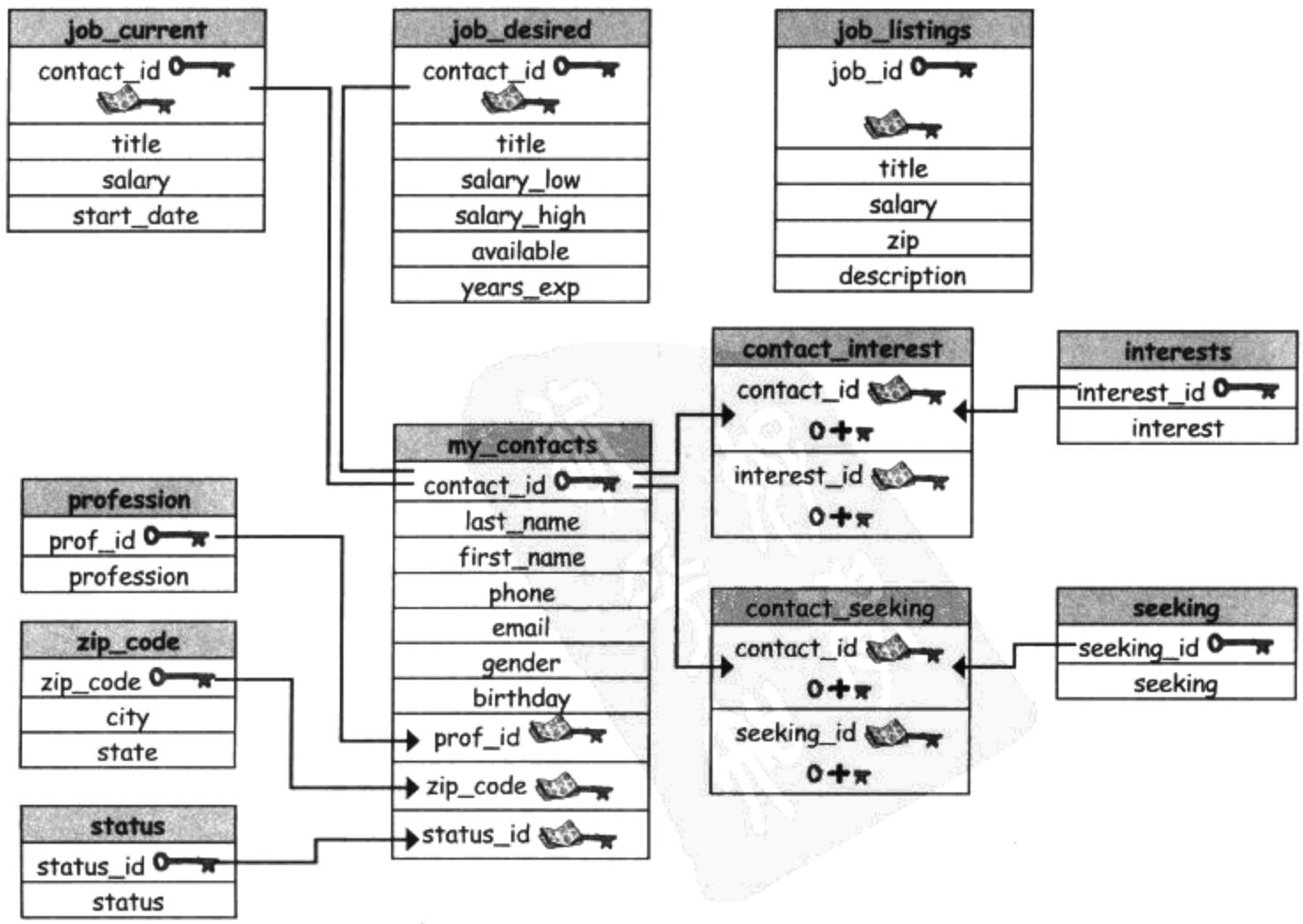
人多手杂，数据库受不了

你的数据库成长到一定规模了，出现了其他需要使用数据库的人。问题是，其他人的 SQL 技术可能不像各位这么娴熟。结果我们需要**防止其他人输入错误的数据**，需要让其他人**只看到部分数据**的技术，还需要**防止大家同时输入数据时互相踩到别人的地盘**。在本章中，我们开始保护数据，以免其他人对数据进行错误的操作。欢迎来到"数据库自卫术 Part 1"。

Greg雇用了帮手

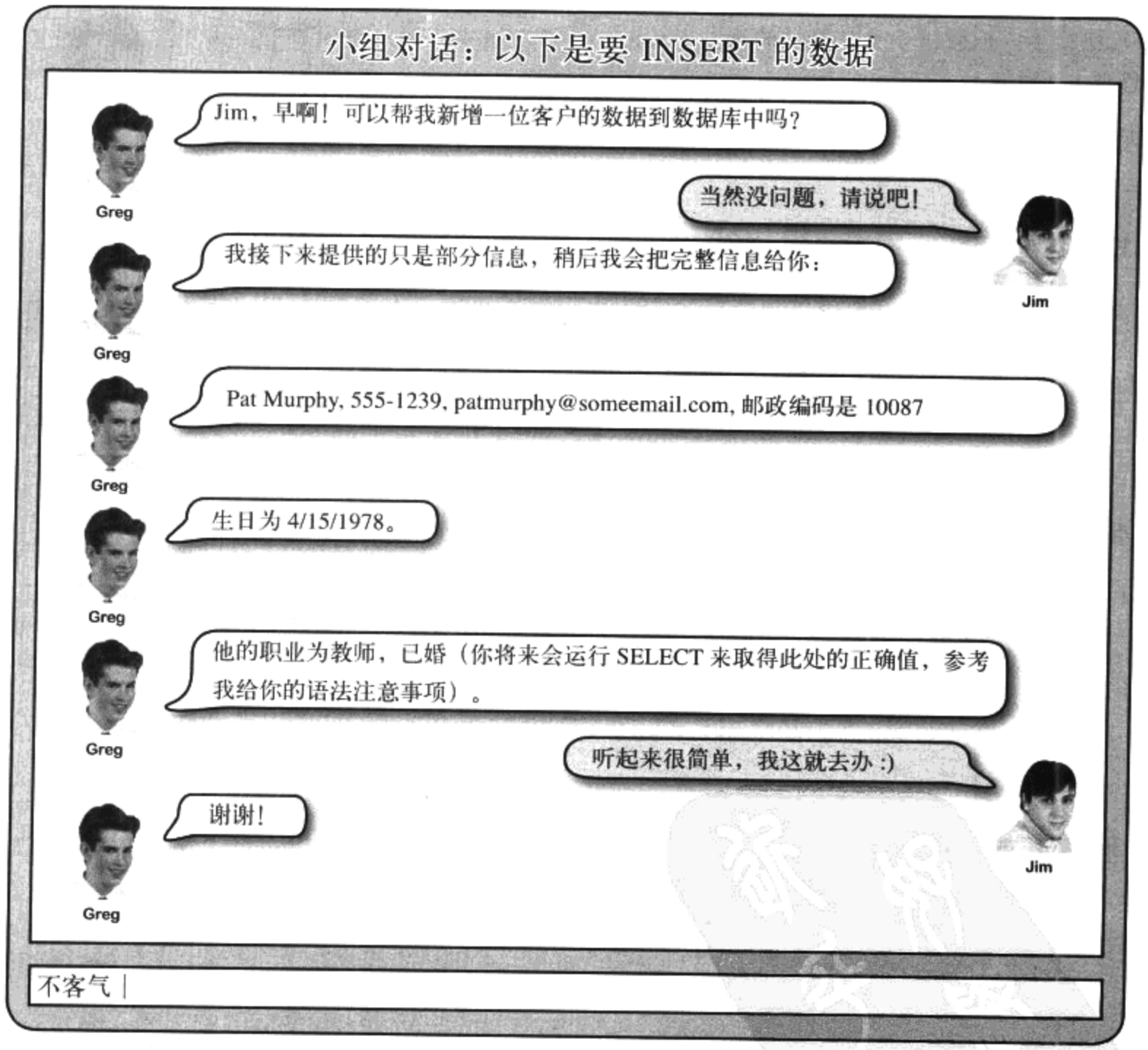
Greg 雇用了两位朋友来帮助他管理日渐成长的数据库。Jim 负责将新客户的数据输入到数据库，Frank 则负责帮求职者匹配出相符职缺。

Greg 解释数据库以及每张表的功用。



Jim的第一天：插入新客户的数据

Jim 一来到自己的办公室，Greg 的 IM 信息就出现了：



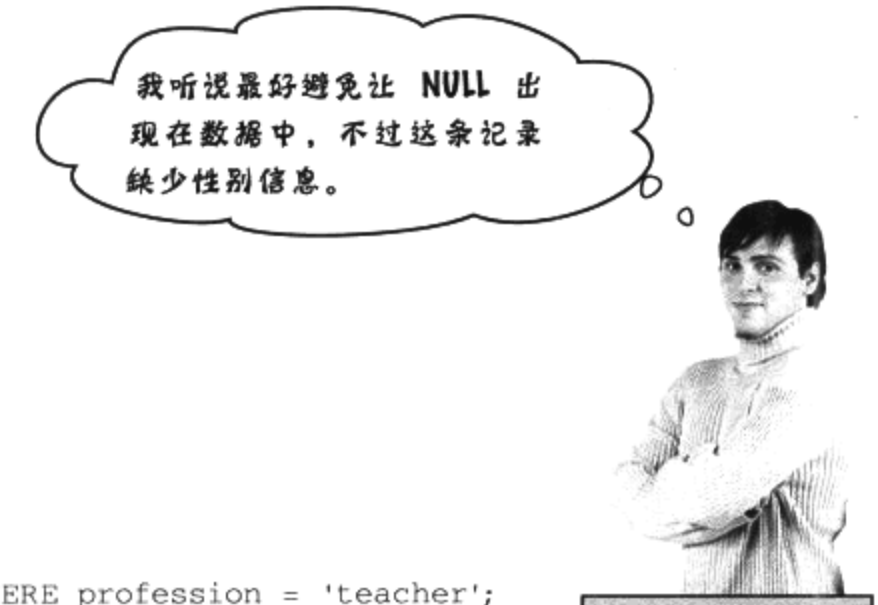
动动脑

你可以写出添加这位新客户的数据至数据库的查询吗？

Jim尽力避开NULL

在输入数据时，Jim 发现 Pat 的性别不明，Greg忘记给了。他做了一个整体性的决定：性别未知时输入“X”。

以下是 Jim 的查询：



他从 profession 表中取得 prof_id

```
SELECT prof _ id FROM profession WHERE profession = 'teacher';
```

prof_id
19

这是对应至“teacher”的职业id，Jim可以把这个值用在查询里。

他从 status 表中取得 status_id

```
SELECT status _ id FROM profession WHERE status = 'single';
```

status_id
4

这是对应至“single”的状态id。

my_contacts
contact_id
last_name
first_name
phone
email
gender
birthday
prof_id
zip_code
status_id

他插入这些值并以“X”代表性别

有AUTO_INCREMENT列时不需要放入数据值。两个单引号要求表为我们插入主键列的值。

```
INSERT INTO my _ contacts VALUES('', 'Murphy', 'Pat',
'5551239', 'patmurphy@someemail.com', 'X', 1978-04-15,
19, '10087', 4);
```

这里是Jim从上述查询取得的ID。不过，他可以改用子查询。

这里是Jim决定的插入性别的方式，他不打算猜测性别或直接输入 NULL。

三个月后

雇用 Jim 三个月后，Greg 想找出一些统计数据。他想知道 my_contacts 表中记录了几位男性、几位女性以及总人数。Greg 做了三个查询：首先计算所有男性与女性的人数，然后计算总人数。

```
SELECT COUNT(*) AS Females FROM my_contacts WHERE gender = 'F';
```

Females
5975

Greg 从 my_contacts 表中找出 5,975 条性别为 "F" 的记录。

```
SELECT COUNT(*) AS Males FROM my_contacts WHERE gender = 'M';
```

Males
6982

Greg 从 my_contacts 表中找出 6,982 条性别为 "M" 的记录。

```
SELECT COUNT(*) AS Total FROM my_contacts;
```

Total
12970

此段查询检查表里的记录总数。

Greg发现总数有问题。表中有13行完全未出现于男性或女性查询的结果中。他尝试了另一个查询：

```
SELECT gender FROM my_contacts
WHERE gender <> 'M' AND gender <> 'F';
```

gender
X
X
X
X
X
X
X
X
X
X
X
X
X

当 Greg 查找消失的记录时，他发现了性别列的值 "X"。

Jim 究竟该如何避免输入 "X" 呢？

检查约束：加入CHECK

在前面的章节里我们已经看过几种关于列的约束。**约束**（constraint）限定了可以插入列的内容，而在我们创建表时就要加入约束。NOT NULL、PRIMARY KEY、FOREIGN KEY、UNIQUE 都是稍早出现过的约束。

还有一种列约束称为 **CHECK**，下面即为其范例。假设我们有一个小猪存钱罐，我们想追踪放入存钱罐的硬币数量，硬币面额只可能是 P（penny）、N（nickel）、D（dime）、Q（quarter），均以首字母缩写表示。创建小猪存钱罐表时，即可用 CHECK 约束来限定插入 coin 列的值。

CHECK（检查）约束限定允许插入某个列的值。它与WHERE子句都使用相同的条件表达式。

```
CREATE TABLE piggy_bank
(
  id INT AUTO_INCREMENT NOT NULL PRIMARY KEY,
  coin CHAR(1) CHECK (coin IN ('P','N','D','Q'))
)
```

这个部分检查硬币值是否以其中之一为单位。

如果插入的值无法通过CHECK条件，则出现错误信息。

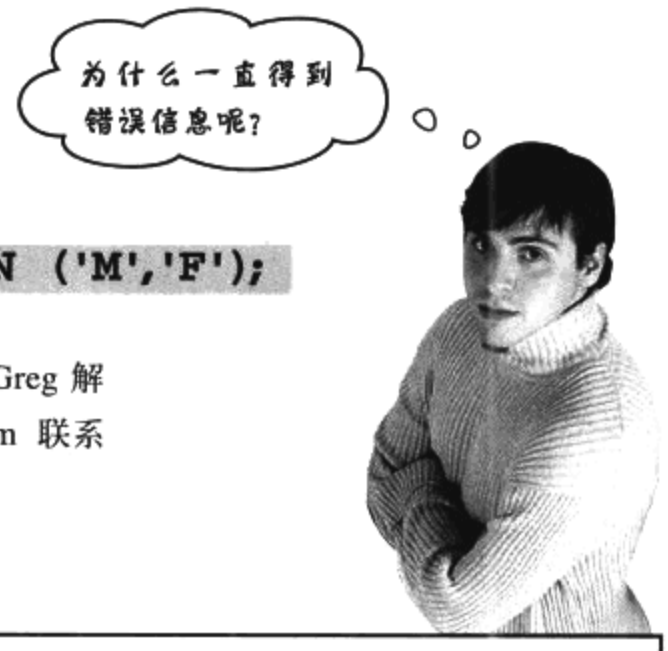
注意！

在MySQL 里，无法以 CHECK 强化数据完整性。

在 MySQL 中，虽然你可在创建表时加上检查约束，但它不会有什么帮助。MySQL 只会忽略它。

为性别列设定检查约束

如果 Greg 可以让时光倒流，他一定会创建性别列具有 CHECK 约束的 my_contacts 表。不过，他还是可以用 ALTER TABLE 亡羊补牢。




```
ALTER TABLE my _ contacts
ADD CONSTRAINT CHECK gender IN ('M','F');
```

隔天，Jim 发现在性别列不可以输入“X”了。当他询问 Greg 时，Greg 解释了新设立的约束条件，并且告诉他时光是无法倒流的，他要求 Jim 联系所有性别列为“X”的人并填入正确的性别信息。

磨笔上阵

请写出允许输入下列各列的值。

```
CREATE TABLE mystery _ table
(
  column1 INT(4) CHECK (column1 > 200),

  column2 CHAR(1) CHECK (column2 NOT IN ('x', 'y', 'z')),

  column3 VARCHAR(3) CHECK ('A' = SUBSTRING(column _ 3, 1, 1)),

  column4 VARCHAR(3) CHECK ('A' = SUBSTRING(column _ 4, 1, 1)
  AND '9' = SUBSTRING(column _ 4, 2, 1))
)
```

Column 1: .....

Column 2: .....

Column 3: .....

Column 4: .....

磨笔上阵解答

请写出允许输入下列各列的值。

```
CREATE TABLE mystery_table
(
  column1 INT(4) CHECK (column1 > 200),

  column2 CHAR(1) CHECK (column2 NOT IN ('x', 'y', 'z')),

  column3 VARCHAR(3) CHECK ('A' = SUBSTRING(column_3, 1, 1)),

  column4 VARCHAR(3) CHECK ('A' = SUBSTRING(column_4, 1, 1)
  AND '9' = SUBSTRING(column_4, 2, 1))
)
```

AND 或 OR 能结合条件表达式。

Column 1: 输入的值必须大于 200

Column 2: 只要不是字符 x、y、z，都可以输入

Column 3: 字符串的第一个字符必须为 A

Column 4: 字符串的第一个字符必须为 A，第二个字符必须是 9

问： 所以说，能用在WHERE子句中的东西都能用于CHECK?

答： 差不多。所有条件运算符——AND、OR、IN、NOT、BETWEEN等都能用于 CHECK 条件，甚至还能如上例一样结合条件运算。但是无法使用子查询。

问： 如果无法在 MySQL 里使用检查约束，该如何代替这个功能呢?

答： 真是个很难回答的问题。有些人改用触发机制(trigger)——在满足特定条件时才执行的查询。但是trigger不像 CHECK 这么简单，而且超出本书的讨论范围了。

问： 如果试着插入无法满足 CHECK 条件的值，会发什么事?

答： 你会看到错误信息而且不会插入任何记录。

问： 这样做有什么好处吗?

答： 检查约束可确保输入数据的合理性。你不会在最后发现一堆神秘的值。

Frank的工作很无聊

Frank的工作是匹配联络人与各种职缺。他发现了一些模式。例如，网站设计员的需求很多，但应征者很少；有很多技术撰稿人在找工作，但这方面的职缺却不多。

Frank 每天都在执行相同的查询，为求职的人们寻找合适的职缺。

为 Frank 设身处地

你的任务是扮演 Frank 的角色，并设计出他每天都要写的查询。一个查询是从 job_desired 表中找出所有网站设计员（web designer）并附上他们的联络信息，另一个查询则是查找技术撰稿人（technical writer）的职缺。

为Frank设身处地解答

你的任务是扮演 Frank 的角色，并设计出他每天都要写的查询。一个查询是从 job_desired 表中找出所有网站设计员（web designer）并附上他们的联络信息，另一个查询则是查找技术撰稿人（technical writer）的职缺。

```
SELECT mc.first_name, mc.last_name,
mc.phone, mc.email
FROM my_contacts mc
NATURAL JOIN job_desired jd
WHERE jd.title = 'Web Designer';
```

Greg 通常都以首字母大写的格式输入职务名。

```
SELECT title, salary, description, zip
FROM job_listings
WHERE title = 'Technical Writer';
```

两组查询都不难，可是每天都要一再重复输入查询，枯燥单调真的很容易造成失误。Frank 需要能够存储查询、每日只查看一次结果而且不需重复输入查询的方式。

他可以把查询存储在文本文件中，每天只要复制、粘贴它们就好了。有什么难的？

文件可能被覆盖或修改。

文件可能意外地被覆盖或修改。把查询存储在数据库内才是更好的方式。我们可以把查询变成**视图**（view）。

创建视图

创建视图非常简单，只需在查询中加入CREATE VIEW 语句。让我们一起为 Frank 的查询创建两个视图：

```
CREATE VIEW web _ designers AS
SELECT mc.first _ name, mc.last _ name, mc.phone, mc.email
FROM my _ contacts mc
NATURAL JOIN job _ desired jd
WHERE jd.title = 'Web Designer';
```

这部分也能改用 INNER JOIN:
ON. mc. contact_id = jd. contact_id。

```
CREATE VIEW tech _ writer _ jobs AS
SELECT title salary, description, zip
FROM job _ listings
WHERE title = 'Technical Writer';
```

动动脑

你觉得使用 VIEW 的 SQL 语句会是什么样子？

查看你的视图

以刚才创建的 web_designers 视图为例：

```
CREATE VIEW web _ designers AS
SELECT mc.first _ name, mc.last _ name, mc.phone, mc.email
FROM my _ contacts mc
NATURAL JOIN job _ desired jd
WHERE jd.title = 'Web Designer';
```

清记住，关键字AS可以省略不写。

若想查看视图的内容，可以把它想成一张表，我们一样可以使用 SELECT 选出它的内容：

```
SELECT * FROM web _ designers;
```

视图名称。

输出结果如下所示：

first_name	last_name	phone	email
John	Martinez	5559872	jm@someemail.com
Samantha	Hoffman	5556948	sammy@someemail.com
Todd	Hertz	5557888	tod@someemail.com
Fred	McDougal	5557744	fm@someemail.com
Lo	Par	55	lou@

依此类推，本表将列出所有符合"Web Designer"条件的记录。

视图的实际行动

当在查询中实际使用视图时，它的行为方式与子查询一样。下面即为前页使用视图的 SELECT：

```
SELECT * FROM web _ designers;
```

这条语句的意思是：子查询要从my_contacts里返回正在寻找网络设计员工作的联络人的first_name、last_name、phone、email等信息。

```
SELECT * FROM
(SELECT mc.first _ name, mc.last _ name, mc.phone, mc.email
FROM my _ contacts mc
NATURAL JOIN job _ desired jd
WHERE jd.title = 'Web Designer' AS web _ designers;
```

这就是我们在视图中使用的东西。

我们给子查询一个别名，以便查询把它当成一般的表。

为什么要有“AS web_designers”？为什么需要这个部分？

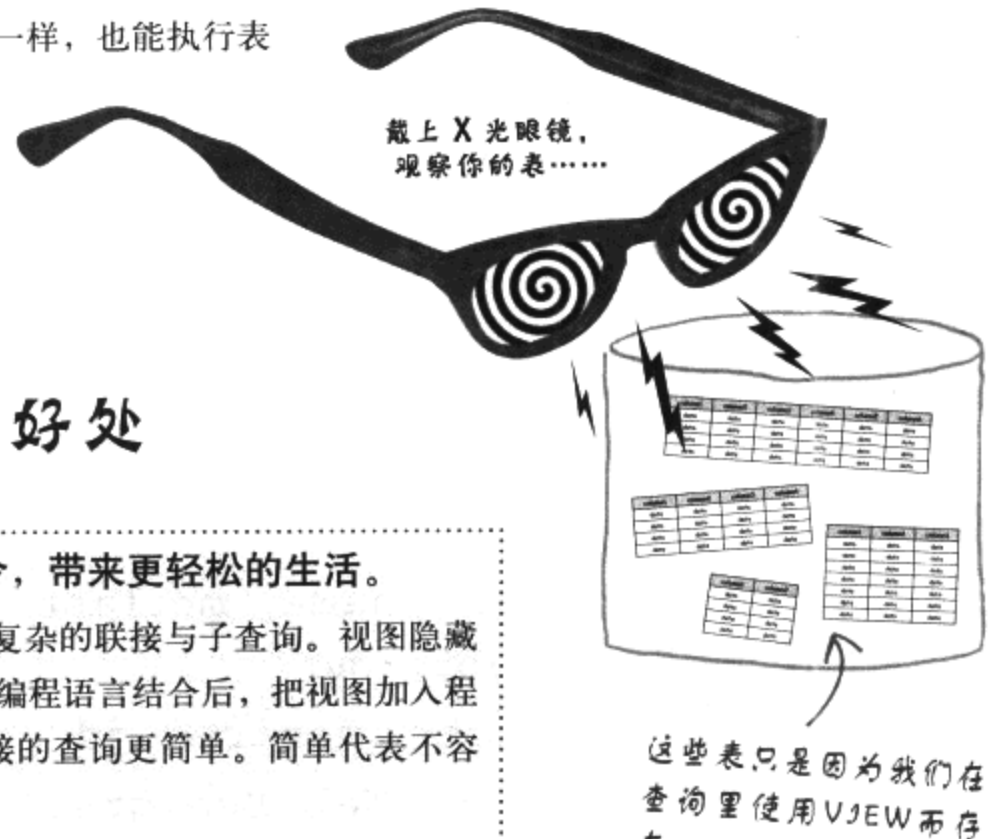
FROM 子句需要表。

当SELECT语句的结果是一个虚拟表时，若没有别名，SQL就无法取得其中的表。

何为视图

基本上视图是一个只有在查询中使用 VIEW 时才存在的表。它被视为**虚拟表**（virtual table），因为其行为和表一样，也能执行表可用的操作。

但虚拟表不会一直保存在数据库里。



为什么视图对数据库有好处

1 **视图把复杂查询简化为一个命令，带来更轻松的生活。**

如果创建了视图，就不需重复创建复杂的联接与子查询。视图隐藏了查询的复杂性。当 SQL 与 PHP 等编程语言结合后，把视图加入程序代码会比加入冗长、复杂、充满联接的查询更简单。简单代表不容易打错字，程序代码也会更容易理解。

2 **即使一直改变数据库结构，也不会破坏依赖表的应用程序。**

我们一直没有提到数据库与应用程序的结合，但总有一天，各位要把数据库的知识带到外面的世界，与其他技术一起创建应用程序。为数据创建视图，可于改变底层表结构时以视图模仿数据库的原始结构，因而无需修改使用旧结构的应用程序。

3 **创建视图可以隐藏读者无需看到的信息。**

有朝一日，假设 Greg's List 的商业规模扩大到了需要记录信用卡信息的程度。此时可创建一个视图指向信用卡拥有者的档案，但又不会透露信用卡的详细信息。你可以只允许员工看到他们需要知道的信息，同时又保护敏感信息不被外泄。

嗯，我想到一个很难的问题。我可以创建一个视图，列出每个在job_current 表里同时也在 job_desired 表里的人，还要列出他们目前的薪资，根据 salary_low 找出他们的期待薪资，并算出当前薪资与期待薪资的差距吗？换句话说，能找出让联络人感意换工作的加薪额度吗？对了，也要给我联络人的姓名、电子邮件地址和电话号码。

Frank 的要求真不少啊！但只要能用 SELECT 的查询就能转换为视图。让我们从下列问题开始，然后再把 Frank 想做的查询写成一个名为“ job_raises ”的视图。

这个查询需要哪些表？

.....

哪些表的哪些列可用于计算需要的加薪额度？

.....

该如何使用 SQL，才能于结果中实际创建名为“raise”的列？

.....

请写出 Frank 需要的查询：

.....

.....

.....

.....

.....

.....

提示：试着用两个联接结合三个表！

Frank 的要求真不少啊！但只要能用 SELECT 的查询就能转换为视图。让我们从下列问题开始，然后再把 Frank 想做的查询写成一个名为“job_raises”的视图。

这个查询需要哪些表？

job_current、job_desired、my_contacts

哪些表的哪些列可用于计算需要的加薪额度？

job_current 的 salary 列、job_desired 的 salary_low 列

该如何使用 SQL，才能于结果中实际创建名为“raise”的列？

以最低期待薪资减去当前薪资并为结果赋予别名。

请写出 Frank 需要的查询：

创建名为“job_raises”的新视图。

```
CREATE VIEW job_raises AS
SELECT mc.first_name, mc.last_name, mc.email, mc.phone, jc.contact_id, jc.salary, jd.salary_low,
jd.salary_low - jc.salary AS raise
FROM job_current jc
INNER JOIN job_desired jd
INNER JOIN my_contacts mc
WHERE jc.contact_id = jd.contact_id
AND jc.contact_id = mc.contact_id;
```

创建视图后，查询的剩余部分使用了两个INNER JOIN从三张表中取出数据。这里需要一点数学计算来算出“加薪额度”列。

把期待薪资减去当前薪资并设定结果的别名为“raise”。

这是个庞大的查询，但 Frank 只需要输入

```
SELECT * FROM job_raises;
```

就能看到他需要的信息。

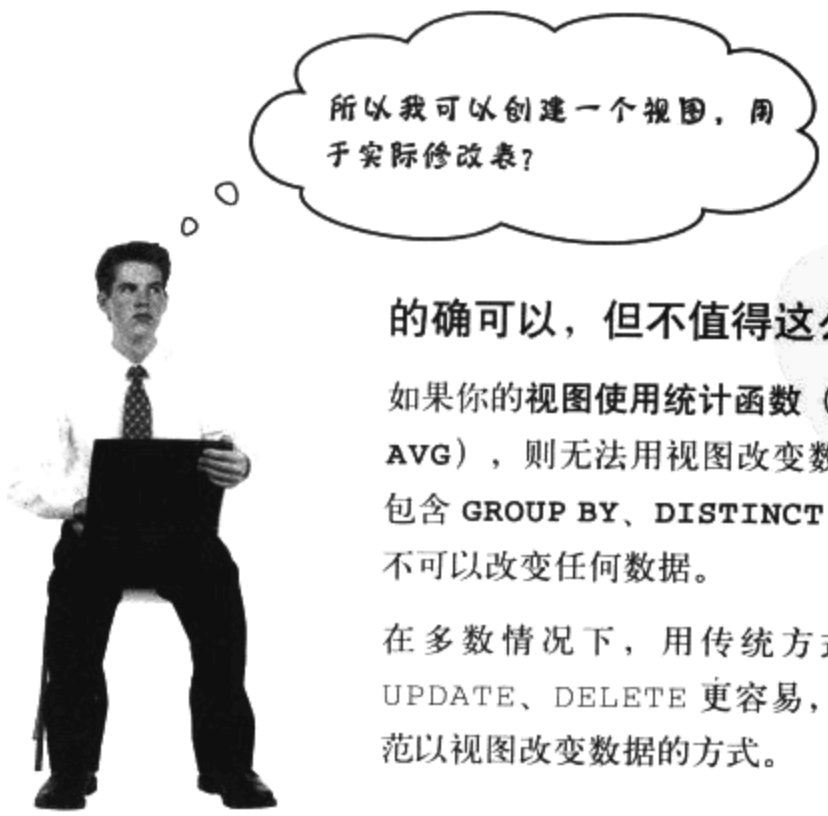
磨笔上阵

如果 Frank 用新的 job_raises 视图运行第 470 页上的查询，他该如何根据姓氏字母排序？

答案请见第 491 页

利用视图进行插入、更新与删除

视图不仅能用于SELECT，从表中选择信息，有时候，它还可以用于UPDATE、INSERT、DELETE数据。



的确可以，但不值得这么麻烦。

如果你的**视图使用统计函数**（例如 **SUM**、**COUNT**、**AVG**），则无法用视图改变数据。如果你的视图包含 **GROUP BY**、**DISTINCT**、**HAVING**，我们也不可以改变任何数据。

在多数情况下，用传统方式做 INSERT、UPDATE、DELETE 更容易，但接下来几页将示范以视图改变数据的方式。

秘密在于假装视图是真正的表

让我们从新表 piggy_bank 制造一个视图。原始表包含我们存下来的硬币。每个硬币都有自己的ID，硬币面额包括 penny（P）、neckel（N）、dime（D）、quarter（Q），并附有硬币铸造的年份。

```
CREATE TABLE piggy _ bank
(
  id INT AUTO _ INCREMENT NOT NULL PRIMARY KEY,
  coin CHAR(1) NOT NULL,
  coin _ year CHAR(4)
)
```

下表是 piggy_bank 里的现有数据：

id	coin	coin_year
1	Q	1950
2	P	1972
3	N	2005
4	Q	1999
5	Q	1981
6	D	1940
7	Q	1980
8	P	2001
9	D	1926
10	P	1999

设计一个只会呈现 quarter 的视图：

```
CREATE VIEW pb _ quarters AS
SELECT * FROM piggy _ bank
WHERE coin = 'Q';
```

动动脑

如果运行下列查询，结果表会是什么样子？

```
SELECT * FROM pb _ quarters;
```

自己动手做。使用下列查询创建 piggy_bank 表以及视图 pb_quarters、pb_dimes。

```
INSERT INTO piggy_bank VALUES ('','Q', 1950), ('','P', 1972),('','N', 2005), ('','Q',
1999),('','Q', 1981),('','D', 1940),('','Q', 1980),('','P', 2001),('','D', 1926),('','P', 1999);

CREATE VIEW pb_quarters AS SELECT * FROM piggy_bank WHERE coin = 'Q';

CREATE VIEW pb_dimes AS SELECT * FROM piggy_bank WHERE coin = 'D' WITH CHECK OPTION;
```

继续接下来的习题时，试着想出这个部分的作用。

请写下运行完上述 INSERT、DELETE、UPDATE 查询后的结果。完成习题后，画出 piggy_bank 表最终的结果。

```
INSERT INTO pb_quarters VALUES ('','Q', 1993);
```

.....

```
INSERT INTO pb_quarters VALUES ('','D', 1942);
```

.....

.....

.....

```
INSERT INTO pb_dimes VALUES ('','Q', 2005);
```

.....

.....

.....

```
DELETE FROM pb_quarters WHERE coin = 'N' OR coin = 'P' OR coin = 'D';
```

.....

.....

```
UPDATE pb_quarters  SET coin = 'Q' WHERE coin = 'P';
```

.....

.....

.....

自己动手做。使用下列查询创建 piggy_bank 表以及视图 pb_quarters、pb_dimes。

```
INSERT INTO piggy _ bank VALUES ('','Q', 1950), ('','P', 1972),('','N', 2005), ('','Q',
1999),('','Q', 1981),('','D', 1940),('','Q', 1980),('','P', 2001),('','D', 1926),('','P', 1999);

CREATE VIEW pb _ quarters AS SELECT * FROM piggy _ bank WHERE coin = 'Q';

CREATE VIEW pb _ dimes AS SELECT * FROM piggy _ bank WHERE coin = 'D' WITH CHECK OPTION;
```

继续接下来的习题时，试着想出这个部分的作用。

请写下运行完上述 INSERT、DELETE、UPDATE 查询后的结果。完成习题后，画出 piggy_bank 表最终的结果。

```
INSERT INTO pb _ quarters VALUES ('','Q', 1993);
```

上述查询将适当地运行。

```
INSERT INTO pb _ quarters VALUES ('','D', 1942);
```

上述查询插入新的值至原始表中，因为创建视图时带有 WHERE 子句，甚至可能有人以为它不会成功。

```
INSERT INTO pb _ dimes VALUES ('','Q', 2005);
```

上述查询因为 CHECK OPTION 子句会产生错误信息。虽然它是把数据输入视图的查询，但在添加数据前必须先通过 WHERE 子句的验证。

```
DELETE FROM pb _ quarters WHERE coin = 'N' OR coin = 'P' OR coin = 'D';
```

上述查询对表没有任何影响，因为它只能找到面额是 "Q" 的硬币的记录。

```
UPDATE pb _ quarters  SET coin = 'Q' WHERE coin = 'P';
```

上述查询对表没有任何影响，因为在视图 *pb_quarters* 里没有 *coin* = 'P' 的值。

原始表会变成：

id	coin	coin_year
1	Q	1950
2	P	1972
3	N	2005
4	Q	1999
5	Q	1981
6	D	1940
7	Q	1980
8	P	2001
9	D	1926
10	P	1999
11	Q	1993
12	D	1942

带有 CHECK OPTION 的视图

在视图后添加 CHECK OPTION，即要求 RDBMS 检查每个INSERT与DELETE语句——它会根据视图中的WHERE子句来检查这类查询是否符合条件。CHECK OPTION究竟如何影响INSERT与DELETE语句呢？

在前页的习题中使用CHECK OPTION时，INSERT操作的数据如果不符合pb_dimes视图中的WHERE条件，插入操作即被拒绝。如果换成UPDATE，也会出现错误信息：

```
UPDATE pb _ dimes SET coin = 'x';
```

x 不符合 pb_dimes 中的 WHERE 条件，因此不会得到更新。

CHECK OPTION 检查每个进行INSERT或DELETE的查询，它根据视图中的 WHERE 子句来判断这些查询可否执行。

可以，你的视图能够精确地反映表的内容，又能强迫 INSERT 语句服从 WHERE 子句的条件。

就以 Jim 的性别列问题为例，我们可以创建一个 my _ contacts 表的视图，让 Jim 通过视图更新数据。每次他试着在性别列填入 X 时就会出现错误信息。

使用MySQL时，可用CHECK OPTION模仿CHECK CONSTRAIN的功能。

动动脑

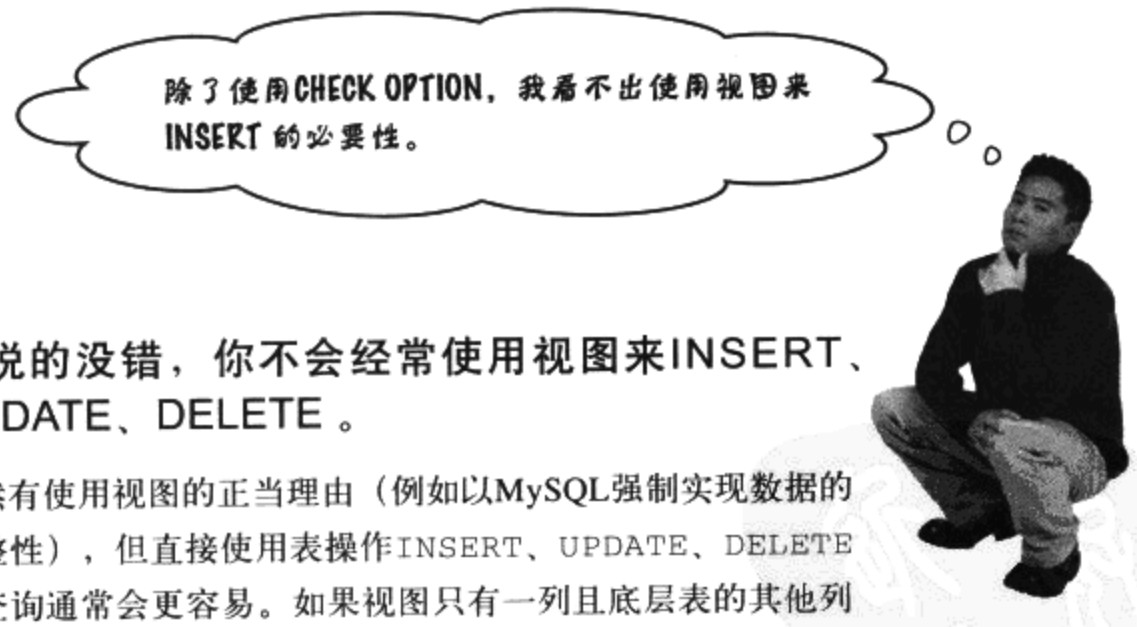
如何为 my _ contacts 创建一个视图，用于限制 Jim 只能在性别字段里填入“M”或“F”呢？

视图有可能更新，如果……

在 piggy_bank 表的例子里，我们创建的两个视图都有更新功能。**可更新视图**（updatable view）就是可以改变底层表的视图。重点在于可更新视图的内容需要包括它引用的表中所有设定为 NOT NULL 的列。如此一来，以 INSERT 对视图添加内容时，即可确定每个必须有值的列确实都填入内容了。

可更新视图包括引用表里所有为 NOT NULL 的列。

基本上，这表示 INSERT、UPDATE、DELETE 也能使用我们创建的视图。只要视图返回的列不是 NULL，就可妥当添加内容至底层表中。



你说的没错，你不会经常使用视图来INSERT、UPDATE、DELETE 。

虽然有使用视图的正当理由（例如以MySQL强制实现数据的完整性），但直接使用表操作INSERT、UPDATE、DELETE等查询通常会更容易。如果视图只有一列且底层表的其他列都可指定为 NULL 或默认值，用视图来INSERT 确实更为方便，这种情况下用视图来 INSERT 才会合理。也可为视图加上 WHERE 子句以约束 INSERT 的内容，在 MySQL 中模拟 CHECK CONSTRAINT 的效果。

也可以只更新视图，此时的 UPDATE 查询不可包含统计类型的运算符，如 SUM、COUNT、AVG，像 BETWEEN、HAVING、IN、NOT IN这类运算符均不可使用。

当视图使用完毕

不需要使用视图后，请利用 DROP VIEW 语句清理空间。语法很简单：

```
DROP VIEW pb _ dimes;
```

问： 可以查看已创建的视图吗？

答： 视图会像表一样出现在数据库中。利用 SHOW TABLES 就能看到所有视图与表。也因为视图就像表，所以使用 DESC 即可观察它的结构。

问： 如果我卸除了有视图的表，会发生什么事？

答： 看情况而定。有些 RDBMS 允许使用视图，但不会返回任何数据。MySQL 会阻止我们卸除视图，除非它有底层表，但我们却**可以**卸除与视图有关的表。有些 RDBMS 另有不同反应。各位最好直接试验会发生的情况。一般而言，最好先卸除视图，然后再卸除它依据的表。

问： 多人操作数据库时，显然CHECK CONSTRAINT与视图很有帮助。但如果有两个人同时改变同一个列，又会发生什么事呢？

答： 关于这个问题，我们应该从事务（transaction）开始讲起。不过，先看看需要提一点现金的 Mrs. Humphries 吧。

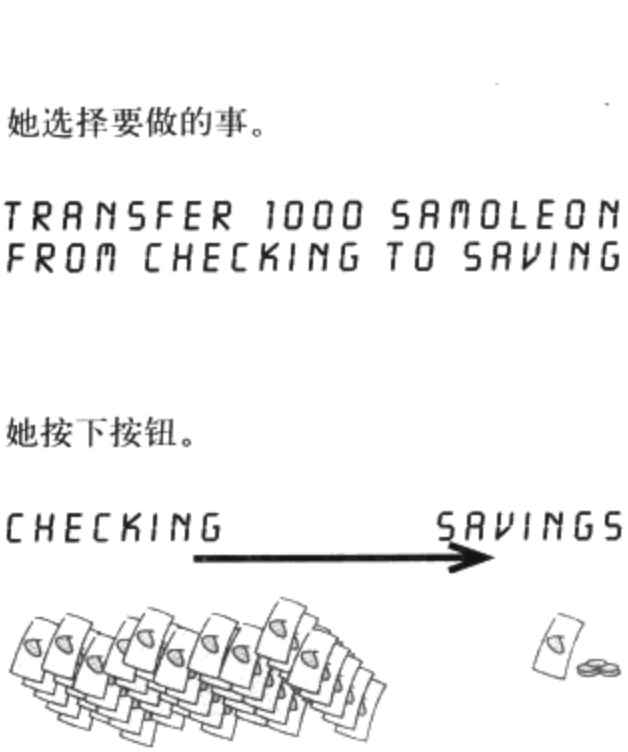
当数据库的使用者不只一人时，CHECK CONSTRAINT 与视图均有助于维护控制权。

当乖乖的数据库发生了人间惨剧

Mrs. Humphries 想从她的支票账户中转存 1,000 元至存款账户。她走向 ATM……

她检查支票账户和存款账户的余额。支票账户有1,000元，存款账户有 30 元。

她选择要做的事。

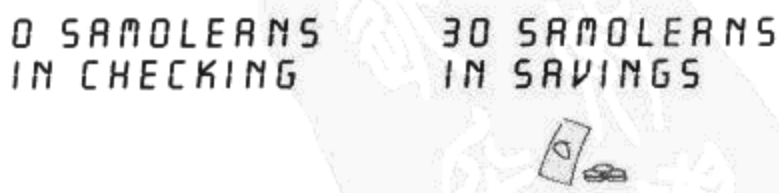


她按下按钮。

此时ATM 铃声大响，画面突然变黑。断电了。

电源恢复后。

她再次检查支票账户和存款账户的余额。



在哪儿？在哪儿？

Mrs. Humphries 的钱到哪里去了？

ATM 里发生了什么事

ATM：我是只小小鸟，飞就飞、叫就叫，自由逍遥……

ATM：啊，MRS. ETHEL P. HUMPHRIES来了。您好，MRS.ETHEL P. HUMPHRIES (ACCOUNT_ID 38221)

Mrs. Humphries：告诉我，我有多少钱。

ATM：我想一下 —(SELECT BALANCE FROM CHECKING WHERE ACCOUNT_ID = 38221;
SELECT BALANCE FROM SAVINGS WHERE ACCOUNT_ID = 38221;)
所以支票账户有 1,000 元，存款账户有 30 元。

Mrs. Humphries：从支票账户里转 1,000 元到存款账户。

ATM：MRS. HUMPHRIES，您的要求真是复杂。但我们照做：
(CHECKING_BAL > 1000。嗯，她有足够的钱可以转账。)
(从支票账户里移出 1,000元)
(INSERT BEEEP……)

这时突然断电。

ATM：

ATM：

ATM：ZZZZZZZ

ATM：（打呵欠）

ATM：啊，MRS. ETHEL P. HUMPHRIES 来了。您好，MRS. ETHEL P. HUMPHRIES (ACCOUNT_ID 38221)

Mrs. Humphries：告诉我，我有多少钱。

ATM：我想一下 ——(SELECT BALANCE FROM CHECKING WHERE ACCOUNT_ID = 38221; SELECT BALANCE FROM SAVINGS WHERE ACCOUNT_ID = 38221)
所以支票账户有 0 元，存款账户有 30 元。

ATM：噢噢！痛！你在砸我的屏幕！MRS. ETHEL P. HUMPHRIES 再见！

动动脑

该如何防止 ATM 忘记 Mrs. Humphries 的交易里的 INSERT 操作？

同时，在镇上的另一边……

ATM发生更多麻烦

John 和Mary 共用一个账户。某个周五，他们分别到不同的 ATM 取款机同时打算取出 300 元。

ATM：嗨，John，又见到你了。你以为我是造币的吗？

John：我的账户余额有多少？

ATM：我想一下——(SELECT CHECKING_BAL FROM ACCOUNTS;)

350 元。

John：给我 300 元。

ATM：无情的人，只把我当成自动取款机，拿了钱就跑得不见人影。

(CHECKING_BAL > 300。他还有足够的钱)

350 元

(从账户里移出 300 元)

(从 CHECKING_BAL 里减去 300)

50 元

John拿了钱就走了。

ATM：从来不打电话、从来不写封信……再会，绝情的John。

一切就从这里开始出现问题。

ATM：Mary！你好啊！

Mary：我的账户余额有多少？

ATM：我想一下 —— (SELECT CHECKING_BAL FROM ACCOUNTS;)

350 元。

[电话铃声响起]

Mary在皮包里摸索着电话。

Mary：给我 300 元。

ATM：遵命。

350 元

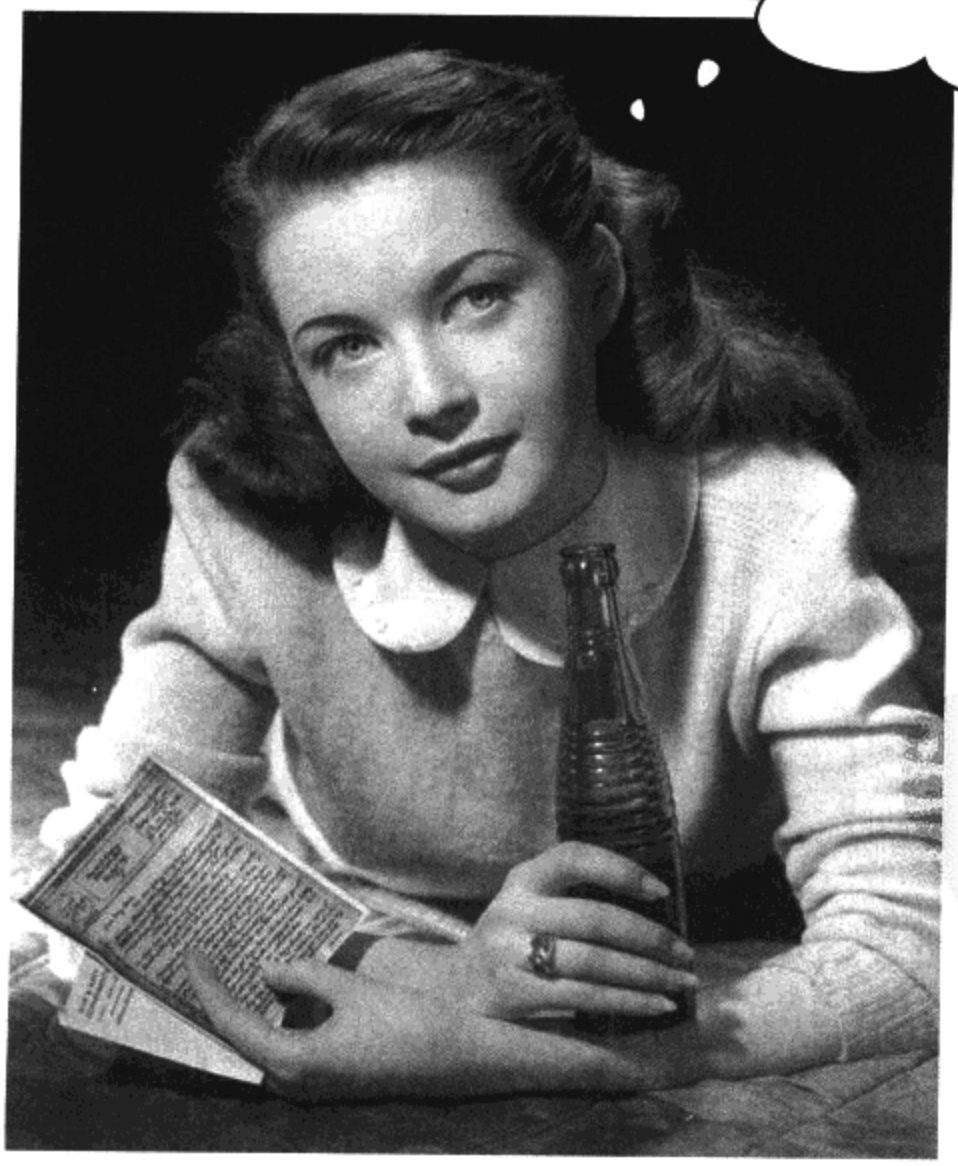
(CHECKING_BAL > 300。她还有足够的钱)

(从账户里移出 300 元)

(从 CHECKING_BAL 里减去 300)

-250 元

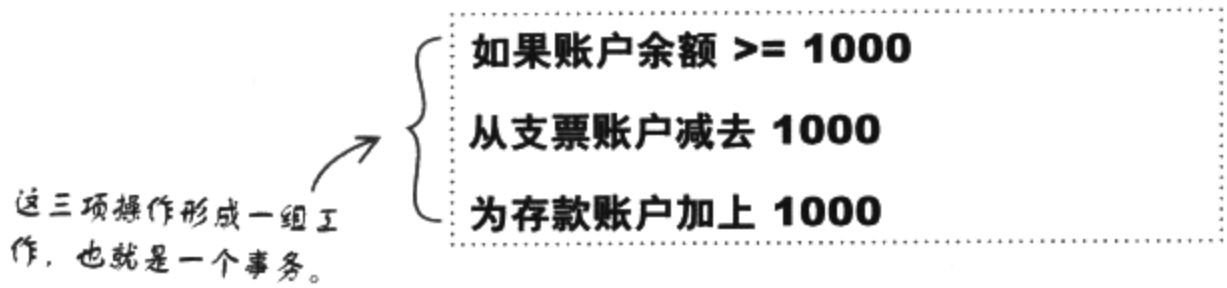
ATM：这个账户严重透支了。



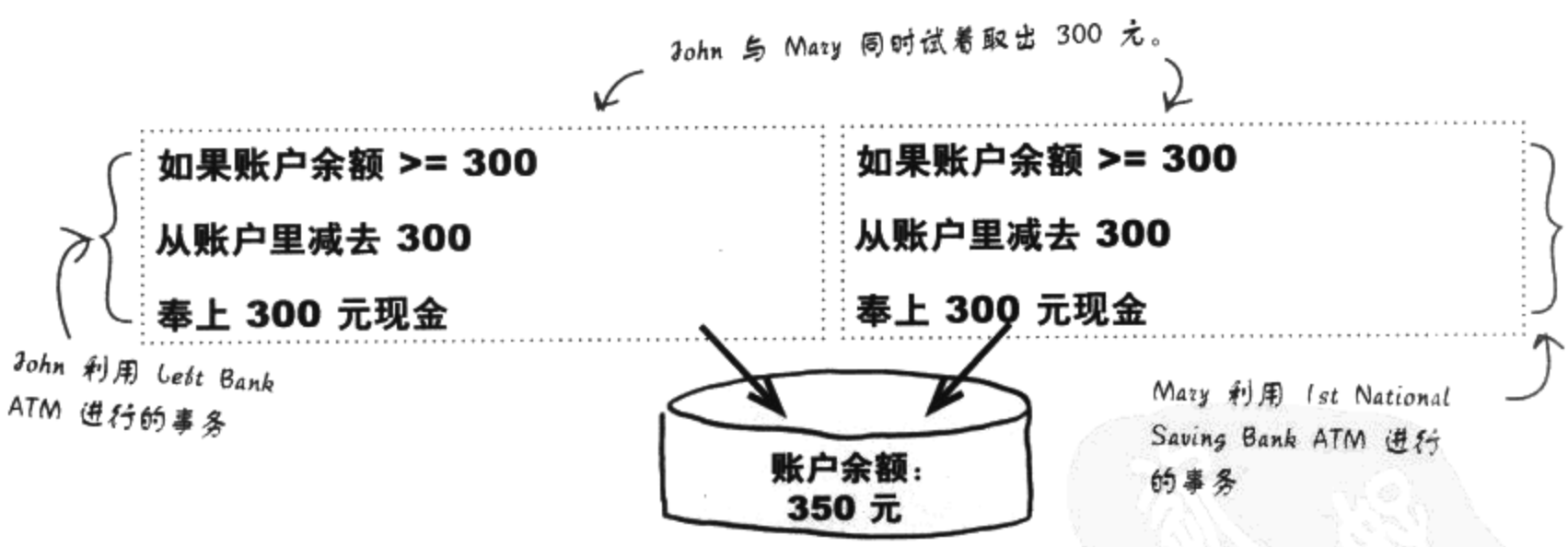
如果一系列 SQL 语句能以组的方式一起执行，而且在发生意外时SQL语句还能回到尚未执行前的状态，那该有多好啊？这可能只是我在痴人说梦吧。

并非痴人说梦，而是事务

事务（transaction）是一群可完成一组工作的SQL语句。以 Mrs. Humphries的惨剧为例，事务由转账所需的所有SQL语句构成：



John 与 Mary 同时试着**执行相同事务**：



以 John 与 Mary 为例，1st National Saving Bank ATM 不应该有权操作账户，甚至不该查询余额，应该等到 Left Bank ATM 的事务完成并释放对账户的锁定后。

在事务过程中，如果所有步骤无法不受干扰地完成，则不该完成任何单一步骤。

经典 ACID 检测

为了帮助你判断SQL步骤是否为一个事务，可以借助ACID。这个简称由四个字符组成，是判断一组SQL语句是否构成一个事务的四个原则：

ACID: ATOMICITY

原子性。事务里的每一个步骤都必须完成，否则只能都不完成。不能只执行部分事务。就是因为只有一部分事务发生，Mrs. Humphries的钱才会因为停电而凭空消失。

ACID: CONSISTENCY

一致性。事务完成后应该维持数据库的一致性。在完成两组金钱事务后，钱的数量应该符合账户余额的情况。在第一个案列中，钱应该转入存款账户；在第二个案例中，则应该换成现金。不该有钱消失了。

ACID: ISOLATION

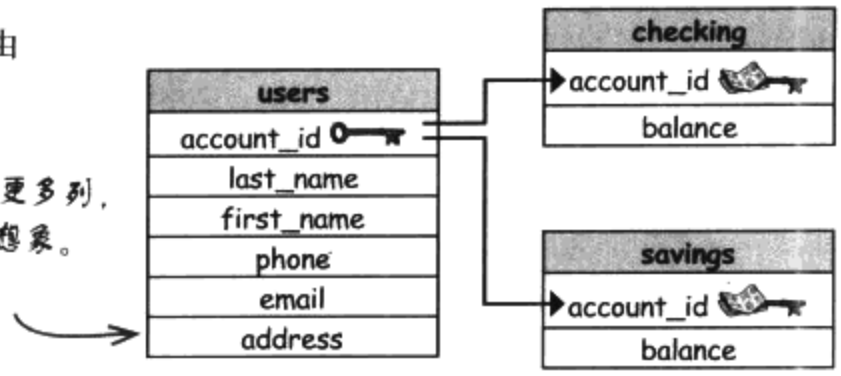
隔离性。表示每次事务都会看到具有一致性的数据库，无论其他事务有什么行动。John 与 Mary 的案例就是在这一点上出错的：Mary 的 ATM 可以看到账户余额，同时John的 ATM 正在完成事务。Mary 根本不该看到账户余额，至少也该看到“交易正在处理中”之类的信息。

ACID: DURABILITY

持久性。事务完成后，数据库需要正确地存储数据并保护数据免受断电或其他威胁的伤害。通常把事务记录存储在主数据库以外的地方。如果Mrs. Humphries的事务记录能存储在别的地方，她的1,000元或许就不会消失了。

SQL帮助你管理事务

让我们以极度简单的银行数据库为例，数据库由开户者表、支票账户表与存款账户表构成：

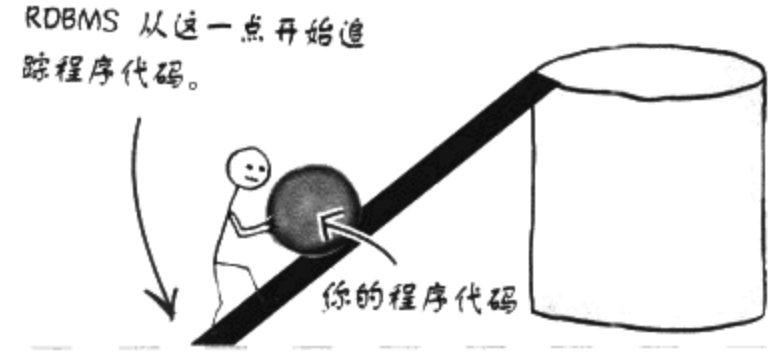


有三种 SQL 事务工具可以保障账户的安全：

```
START TRANSACTION;
```

追踪 SQL 的行为。

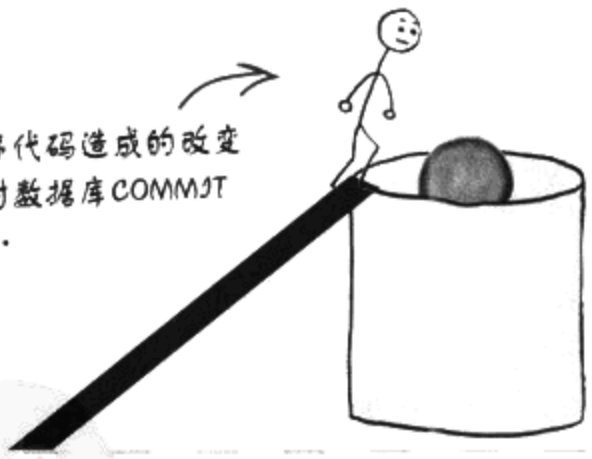
START TRANSACTION **持续追踪后续所有 SQL 语句**，直到你输入 COMMIT 或 ROLLBACK 为止。




```
COMMIT;
```

等到我们满意后再提交（COMMIT）所有程序代码造成的改变。

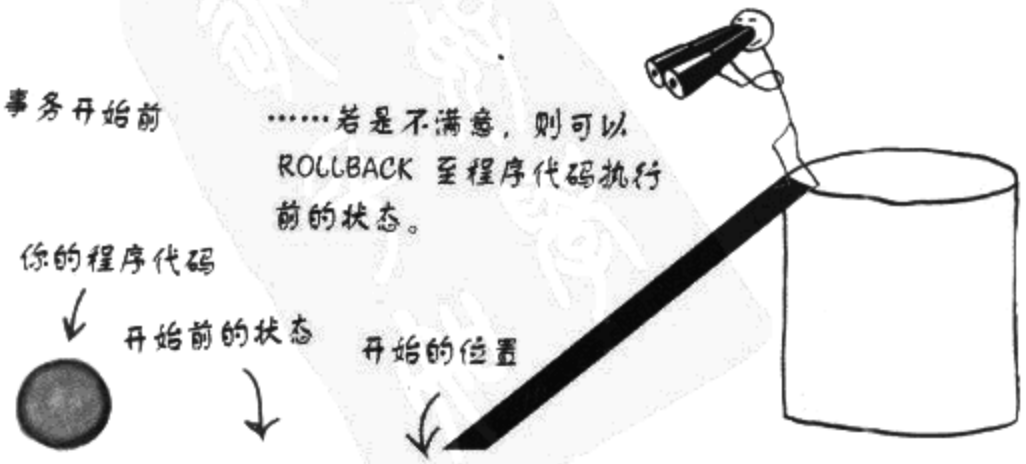
如果所有语句都已经妥当，改变后的结果似乎也不错，那么请输入 **COMMIT**，让一切改变成真。




```
ROLLBACK;
```

回滚，回到事务开始前的状态。

如果改变结果不太对劲，**ROLLBACK** 可以逆转过程，让每件事回到**START TRANSACTION** 前的状态。



在你 *COMMIT*之前数据库都不会发生任何改变

ATM 里应该发生什么事

ATM：我是只小小鸟，飞就飞、叫就叫，自由逍遥……

ATM：啊，MRS. ETHEL P. HUMPHRIES来了。您好，MRS. ETHEL P. HUMPHRIES (ACCOUNT_ID 38221)

Mrs. Humphries：告诉我，我有多少钱。

ATM：我想一下——(SELECT BALANCE FROM CHECKING WHERE ACCOUNT_ID = 38221;
SELECT BALANCE FROM SAVINGS WHERE ACCOUNT_ID = 38221;)

所以支票账户有 1,000 元，存款账户有 30 元。

Mrs. Humphries：从支票账户里转 1,000 元到存款账户里。

ATM：MRS. HUMPHRIES，您的要求真是复杂。但我们照做：
(事务开始;

SELECT BALANCE FROM CHECKING WHERE ACCOUNT_ID = 38221;)

ATM：她的支票账户里有 1,000 元，继续事务。

ATM：(UPDATE CHECKING SET BALANCE = BALANCE - 1000 WHERE ACCOUNT_ID = 38221;)

(INSERT BEEEP……)

这时突然断电。

ATM 启动备用电源：回滚事务：

ATM：

ATM：

ATM：ZZZZZZZ

ATM：（打呵欠）

ATM：啊，MRS. ETHEL P. HUMPHRIES 来了。您好，MRS. ETHEL P. HUMPHRIES (ACCOUNT_ID 38221)

Mrs. Humphries：告诉我，我有多少钱。

幸好有 ROLLBACK 机制，COMMIT 语句并未执行，所以一切都没改变。

ATM：我想一下——(SELECT BALANCE FROM CHECKING WHERE ACCOUNT_ID = 38221;
SELECT BALANCE FROM SAVINGS WHERE ACCOUNT_ID = 38221;)

所以支票账户有 1,000 元，存款账户有 30 元。

如何让事务在 MySQL 下运作

在MySQL下使用事务前，你需要先采用正确的**存储引擎**（storage engine）。存储引擎是存储所有数据库内容和结构的背后功臣。有些存储引擎允许事务，有些则不行。

回想第 4 章看到的语句：

```
SHOW CREATE TABLE my_contacts;
```

聪明的表设计

节省时间的命令

查看第183页上我们用于创建表的程序代码，还有SHOW CREATE TABLE my_contacts 语句提供的下列 SQL 代码。两者并非完全相同，但如果把下面这段代码粘贴到CREATE TABLE命令中，最后的结果会是一样的。反撇号或数据设置不需要删除，但如果删除的话，看起来会更干净。

列名和表名前后的反撇号会在我们运行SHOW CREATE TABLE命令时出现。

除非我们另行添加 SQL 程序，否则它都会假设所有数据的默认值是 NULL。

最好在创建表时指定列是否可包含 NULL。

```
CREATE TABLE `my_contacts`
(
    `last_name` varchar(30) default NULL,
    `first_Name` varchar(20) default NULL,
    `email` varchar(50) default NULL,
    `gender` char(1) default NULL,
    `birthday` date default NULL,
    `profession` varchar(50) default NULL,
    `location` varchar(50) default NULL,
    `status` varchar(20) default NULL,
    `interests` varchar(100) default NULL,
    `seeking` varchar(100) default NULL,
) ENGINE=MyISAM DEFAULT CHARSET=latin1
```

现在，我们要好好地关心存储引擎。

你不需要担心结束括号后的文字，它说明了数据如何存储，以及使用的字符集。目前用默认值就够了。

虽然我们能清理 SQL 代码（删除最后一行文字和反撇号），但不能只靠复制和粘贴来创建表。

当前位置 › 185

存储引擎必须是BDB或InnoDB，两种支持事务的引擎之一。

BDB 与 InnoDB 是两种 RDBMS 在幕后存储数据的可能方式。

这些就是存储引擎，使用其中一种可确保事务能够使用。请参考其他相关书籍，了解 MySQL 的各种存储引擎之间的差异。

对现在的目标而言，从两种引擎中选择任一种都可以。

改变存储引擎，请用这段语法：

```
ALTER TABLE your_table TYPE = InnoDB;
```

现在动手试试看

假设我们把小猪存钱罐表中的pennies都升级为quarters。

请大家利用本章稍早的piggy_bank表来动手尝试下列程序代码。第一次先使用ROLLBACK，暂时不对表造成永久性的改变：

```
START TRANSACTION;
SELECT * FROM piggy_bank;
UPDATE piggy_bank set coin = 'Q' where coin= 'P';
SELECT * FROM piggy_bank;
ROLLBACK;
SELECT * FROM piggy_bank;
```

现在你看到了改变……

可是我们又改变了心意。

……改变又不见了。

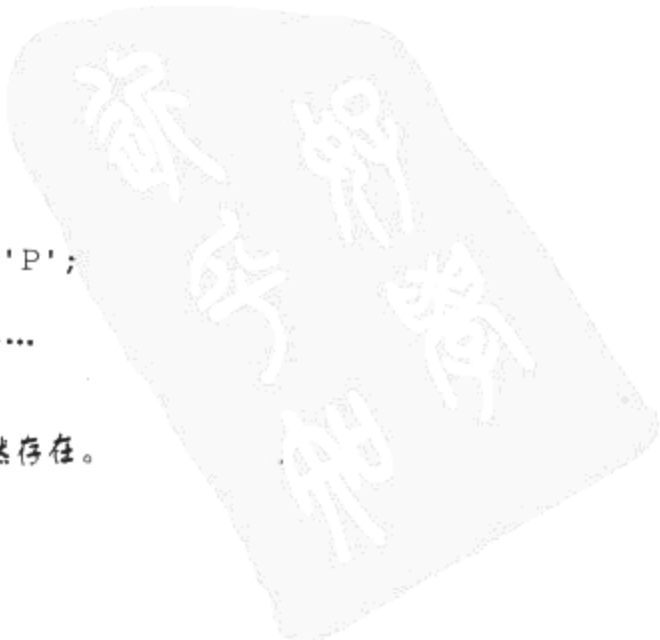
第二次改用 COMMIT，因为我们确定要改变了：

```
START TRANSACTION;
SELECT * FROM piggy_bank;
UPDATE piggy_bank set coin = 'Q' where coin= 'P';
SELECT * FROM piggy_bank;
COMMIT;
SELECT * FROM piggy_bank;
```

现在你看到了改变……

让改变成真。

……改变结果仍然存在。



磨笔上阵

填入在事务之后的piggy_bank表的内容。右表是它现在的样子：

piggy_bank

id	coin	coin_year
1	Q	1950
2	P	1972
3	N	2005
4	Q	1999

```
START TRANSACTION;
UPDATE piggy_bank set coin = 'Q' where coin = 'P'
AND coin_year < 1970;
COMMIT;
```

id	coin	coin_year
1		
2		
3		
4		

```
START TRANSACTION;
UPDATE piggy_bank set coin = 'N' where coin = 'Q';
ROLLBACK;
```

id	coin	coin_year
1		
2		
3		
4		

```
START TRANSACTION;
UPDATE piggy_bank set coin = 'Q' where coin = 'N'
AND coin_year > 1950;
ROLLBACK;
```

id	coin	coin_year
1		
2		
3		
4		

```
START TRANSACTION;
UPDATE piggy_bank set coin = 'D' where coin = 'Q'
AND coin_year > 1980;
COMMIT;
```

id	coin	coin_year
1		
2		
3		
4		

```
START TRANSACTION;
UPDATE piggy_bank set coin = 'P' where coin = 'N'
AND coin_year > 1970;
COMMIT;
```

id	coin	coin_year
1		
2		
3		
4		

答案在第492页。

问： 事务一定要以 START TRANSACTION 开始吗？COMMIT 与 ROLLBACK 可以独立运作吗？

答： 必须用START TRANSACTION告诉RDBMS“事务开始”，才能追踪事务开始的地方并知道恢复的程度。

问： 可以使用 START TRANSACTION 来测试查询吗？

答： 可以，而且也应该如此。利用这个方法，可以在数据库里实现影响数据的查询，又不会在做错事时无法挽救对表做的改变。但记得在改变完成后输入 COMMIT 或 ROLLBACK。

问： 为什么需要 COMMIT 与 ROLLBACK？

答： RDBMS 会记录事务过程中的每个操作，称为事务日志（transaction log），操作越多，日志越大。事务最好留待真正需要恢原功能时使用，以避免存储空间的浪费，也避免了 RDBMS 花费太多精力来追踪我们的每个操作。

Greg有办法完全掌控数据库里的大小行动吗？他真的能控制谁对哪一个表进行了哪些操作吗？

欲知详情，请待下章分解。

你的SQL工具包

第11章现在已经收进你的工具包了，而且你的工具包也快装满了。在这一章里，我们看到了如何为数据创建视图，也看到了事务的执行方式。如果需要本书工具的完整列表，请参考附录 3。

VIEWS

视图。使用视图把查询结果当成表。很适合简化复杂查询。

UPDATABLE VIEWS

可更新表。有些视图能用于改变它底层的实际表。这类视图必须包含底层表的所有 NOT NULL 列。

NON-UPDATABLE VIEWS

无法对底层表执行INSERT或UPDATE操作的视图。

CHECK CONSTRAINTS

检查约束。可以只让特定值插入或更新至表里。

CHECK OPTION

创建可更新视图时，使用这个关键字强迫所有插入与更新的数据都需满足视图里的WHERE条件。

TRANSACTIONS

事务。一组必须同进退的查询。如果这些查询无法不受干扰地全都执行完毕，则不承认其中的部分查询造成的改变。

START TRANSACTION

这条语句告诉 RDBMS开始事务。在COMMIT执行前改变都不具永久性。除非出现COMMIT或ROLLBACK，否则都处于事务过程中。ROLLBACK可把数据库带回START TRANSACTION前的状态。

磨笔上阵解答

第470页上的习题

如果 Frank 用新的 job_raises 视图运行第 470 页上的查询，他该如何根据姓氏字母排序？

在创建视图时，或在使用视图的SELECT语句里加上ORDER BY *last_name*。

磨笔上阵解答

第492页上的习题

填入在事务之后的piggy_bank表的内容。右表是它现在的样子：

piggy _ bank

id	coin	coin_year
1	Q	1950
2	P	1972
3	N	2005
4	Q	1999

```
START TRANSACTION;
UPDATE piggy _ bank set coin = 'Q' where coin = 'P'
AND coin _ year < 1970;
COMMIT;
```

没有匹配记录，
也不会有任何改变。

id	coin	coin_year
1	Q	1950
2	P	1972
3	N	2005
4	Q	1999

```
START TRANSACTION;
UPDATE piggy _ bank set coin = 'N' where coin = 'Q';
ROLLBACK;
```

回滚，不会改变。

id	coin	coin_year
1	Q	1950
2	P	1972
3	N	2005
4	Q	1999

```
START TRANSACTION;
UPDATE piggy _ bank set coin = 'Q' where coin = 'N'
AND coin _ year > 1950;
ROLLBACK;
```

回滚，不会改变。

id	coin	coin_year
1	Q	1950
2	P	1972
3	N	2005
4	Q	1999

```
START TRANSACTION;
UPDATE piggy _ bank set coin = 'D' where coin = 'Q'
AND coin _ year > 1980;
COMMIT;
```

这一行受到影响。

id	coin	coin_year
1	Q	1950
2	P	1972
3	N	2005
4	D	1999

```
START TRANSACTION;
UPDATE piggy _ bank set coin = 'P' where coin = 'N'
AND coin _ year > 1970;
COMMIT;
```

这一行受到影响。

id	coin	coin_year
1	Q	1950
2	P	1972
3	P	2005
4	Q	1999

12 安全性

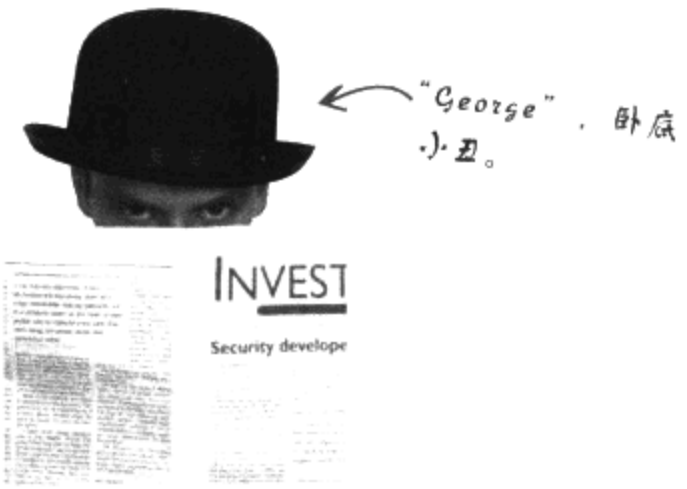
保护你的资产

为了创建数据库，大家已经花了许多时间与精力。如果数据库受到了什么伤害，你一定会崩溃吧！而且虽然让其他人**访问你的数据**有其必要性，但真的会忍不住担心有人插入或更新数据时的操作不正确，甚至发生更糟的情况：**删错了数据**。我们即将要学到如何把数据库和其中的对象变得更安全，以及如何全面控制**谁对数据进行什么操作**。

用户的问题

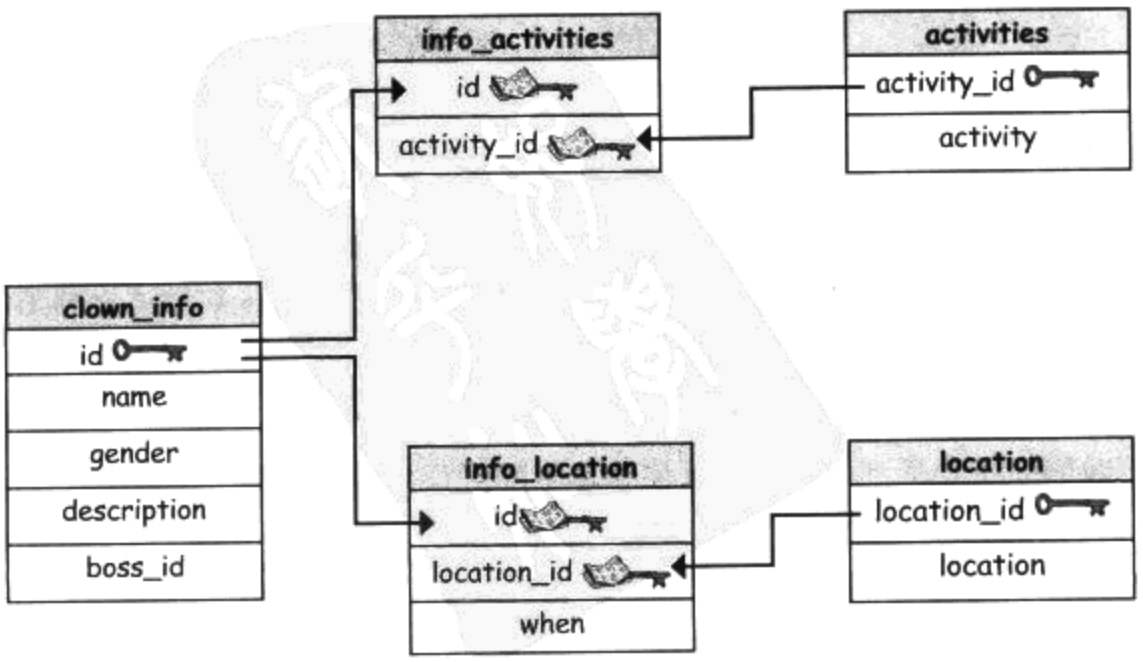
小丑的行踪已经成为非常严重的事件，Dataville市议会必须雇用一个专职小组来追踪小丑并新增数据至clown_tracking表中。

很不幸地，这个小组被伪装成正常人的小丑“George”（假名）渗透了他给数据库制造了一些问题，包括丢失数据、乱改数据以及因为故意拼错字而造成的重复记录。以下列出部分小丑追踪表中的问题：



Snuggles、Snugles、Snuggels都在clown_info表中各有一条记录。一看就知道这些记录都属于相同小丑，因为性别、外观及行为描述等都相同（只有姓名拼写的差别）。info_location表会使用clown_info表中Snuggles、Snugles、Snuggels的ID。

activities 表的内容也拼得乱七八糟。Snuggles 是 juggeler，Snugles 是 jugler，Snuggels 则是 jugular。



避免小丑追踪数据库的错误

George在其他人注意到他做的“好事”前就辞职了，只留下其他人收拾残局。从现在开始，凡是刚雇用的新员工都有SELECT的权力，可从数据库中选择数据以便识别小丑。但新员工不可以INSERT或UPDATE数据。说实话，在完成背景调查前，新员工只能选取数据，不能对数据库做其他事。

我们也需要小心一点，要求员工DELETE数据以清理George的破坏时，他们也可能会一起删除了正常有用的数据。

现在是保护小丑追踪数据库的时候了，在其他像 George 一样的小丑完全销毁数据库前，我们应该未雨绸缪。

磨笔上阵

保护小丑追踪数据库免遭破坏。在下面写出新雇员应该或不应该做的事。尽可能注明相关的表名。

新雇员应该被允许的行为：

例：从activities中选取数据

新雇员不该被允许的行为：

例：对clown_info执行DROP TABLE

磨笔上阵解答

保护小丑追踪数据库免遭破坏。在下面写出新雇员应该或不应该做的事。尽可能注明相关的表名。

新雇员应该被允许的行为：

例：从activities中选取数据

SELECT操作，可用于clown_info、info_activities、activities、info_location、location

新雇员不该被允许的行为：

例：对clown_info执行DROP TABLE

DRAP TABLE操作，不可用于clown_info、info_activities、activities、info_location、location

INSERT操作，不可用于clown_info、info_activities、activities、info_location、location

UPDATE操作，不可用于clown_info、info_activities、activities、info_location、location

ALTER操作，不可用clown_info、info_activities、activities、info_location、location

DELETE操作，不可用clown_info、info_activities、activities、info_location、location

真是个好消息，我们可以阻止 George 与他的小丑同伙破坏我们的数据了！

SQL 允许我们控制雇员对小丑追踪数据库可做与不可做的事项。但在这么做前，需要先给每个数据库用户一个**用户账号**（user account）。

保护用户账号：root

到目前为止，我们的数据库只有一位用户，而且一直都没有密码。只要有权使用数据库的终端或图形界面的人都能全权控制数据库。

默认情况下，第一位用户——**根**用户（root）具有所有数据库操控能力。这一点很重要，因为根用户必须可为其他用户创建账号。我们并不想限制根用户的权限，但他的账号应该有密码。MySQL 设定根用户密码的方式很简单：

```
SET PASSWORD FOR 'root'@'localhost' = PASSWORD('b4dc10wnZ');
```

根用户的名称就是 "root"。

"localhost" 代表安装与运行 SQL 软件的机器。

这部分是我们为根用户选择的密码。

其他 RDBMS使用的方式则各有不同，例如 Oracle 采用：

```
alter user root identified by new-password;
```

如果使用图形界面操作数据库，或许有更为简单的对话框形式来改变密码。**如何设定密码并非重点，重点在于一定要设置密码。**

请参考你使用的RDBMS的说明文档，了解保护根用户的必要信息。

问： 我还是不知道“localhost”是什么意思？能否讲得更详细一点呢？

答： localhost表示用于运行查询的计算机，就是安装了 SQL RDBMS 的同一台计算机。localhost 是默认的参数值，可选择是否将其加入语法中。

问： 不过，如果我是从另一台计算机联机使用SQL客户端的呢？

答： 你的方式通常称为远程访问。我们必须让查询知道计算机所在地，可通过 IP 地址或 localhost 以外的主机名称指定。举例来说，如果你的 SQL 软件安装在 O'Reilly 的网络中的一台叫 kumquats 的机器上，则主机名称可能会是 root@kumquats.oreilly.com。不过，它还不是货真价实的 SQL 服务器，所以无法运作。

添加新用户

SQL 如何存储用户的信息?

对于这个问题，各位心中或许都有答案了。

当然是存储在表中！SQL也以数据库存储它本身的数据，包括用户id、用户名称与密码以及各个用户对特定数据库的操作权限。

为新员工Elsie增设的用户名称。

```
CREATE USER elsie
IDENTIFIED BY 'cl3v3rp4s5w0rd';
```

这是密码。

SQL 并未指定如何管理用户。

不同的 RDBMS 创建用户的方式也不一样。请参考你的数据库软件的说明文档，找出正确的创建用户的方式。

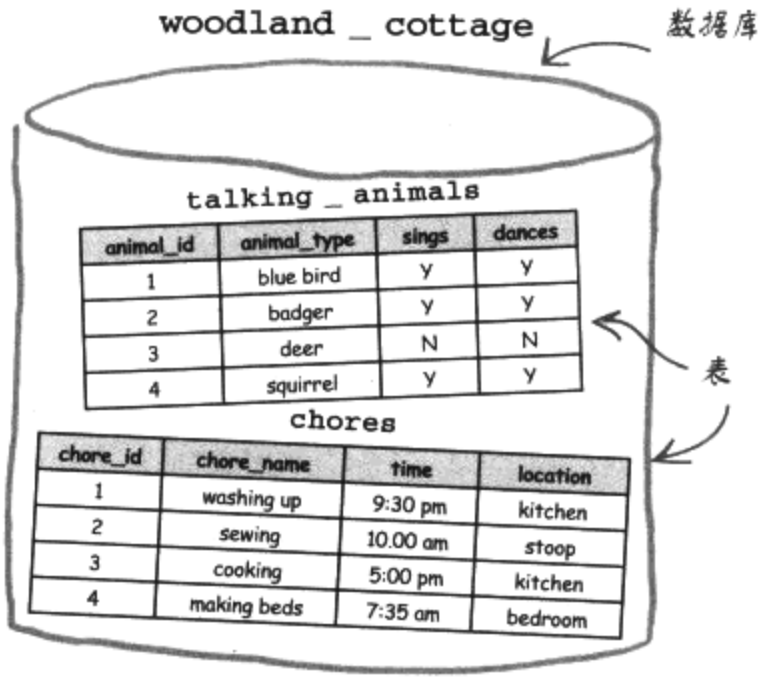
可以，但有时在新建账号时，我们并不知道需要授予何种权限。

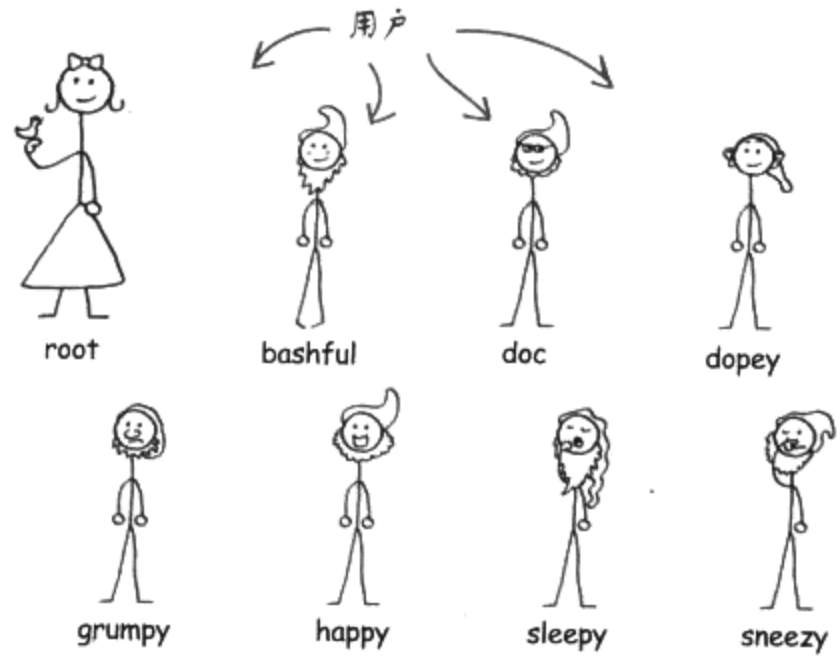
不过我们还是要明确决定用户可以访问的目标。我们每次做一件事。先创建用户，然后再授予用户需要的特殊权限。本章结束前再看这两件事如何合二为一。知道如何单独授予权限的好处，就是日后可另外因应数据库的改变而修改用户权限。

判断用户的确切需求

Elsie 的账号已经创建完毕，但她目前没有任何权限。我们必须利用 **GRANT** 语句把 clown_info 表的 SELECT 操作权限授予 Elsie。

新用户不能对数据库中的任何对象执行任何 SQL 命令，我们刚刚创建的新用户没有任何权限。GRANT 语句可以为用户授予操作数据库的特权。以下是 GRANT 的作用：





仅允许部分用户修改特定表。

只有总管 *root*才能为杂务表加入新的待办事项，也只有*root*才能执行 INSERT、UPDATE、DELETE 等任务。不过，*happy* 是表 talking_animals 的小总管，可用 ALTER 修改这张表的结构，也能执行其他操作。

特定表的数据仅允许部分用户访问。

除了 *grumpy*，每个人都可对 talking_animals 表做 SELECT 操作。反正 *grumpy* 一点也不喜欢动物。

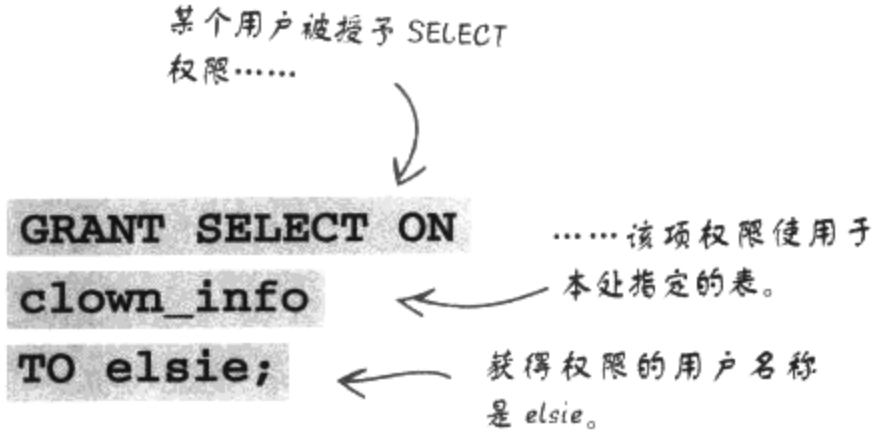
就算在表中，也可能需要权限：部分用户可看到特定列，但其他人不行。

除了 *dopey*，每个人都能看到 chores 表的说明列（说明只会让 *dopey* 越看越糊涂）。

使用 GRANT 语句可以控制用户对表和列可执行的操作。

简单的 GRANT 语句

目前，Elsie 没有做任何事的权限，虽然她能用账号与密码登录 SQL软件，但也仅止于此。Elsie 需要选取 clown_info 表中内容的权力，所以我们给予她**权限**（permission），由 GRANT... TO 语句完成：



Elsie 也需要对于其他小丑追踪表的 SELECT 权限，才能在她的 SELECT 语句中使用联接与子查询。每次授予表的权限都需另写一段 GRANT 语句：

```
GRANT SELECT ON activities TO elsie;
GRANT SELECT ON location TO elsie;
GRANT SELECT ON info_activities TO elsie;
GRANT SELECT ON info_location TO elsie;
```

现在，Elsie 的行为已经在我们的掌控下。接下来想想GRANT 语句对于第 499 页上的 woodland_cottage 数据库会有什么影响。

程序代码 **程序代码有何作用？**

1.
```
GRANT INSERT ON magic_animals
TO doc;
```
.....
.....

2.
```
GRANT DELETE ON chores
TO happy, sleepy;
```
.....
.....

3.
```
GRANT DELETE ON chores
TO happy, sleepy
WITH GRANT OPTION;
```
.....
.....

4.
```
GRANT SELECT(chore_name) ON
chores TO dopey;
```
提示：这是列名。
.....
.....

5.
```
GRANT SELECT, INSERT ON
talking_animals
TO sneezy;
```
.....
.....

6.
```
GRANT ALL ON talking_animals
TO bashful;
```
.....
.....

现在试着写出符合要求的 GRANT 语句。

7. ..... 授予 Doc SELECT表chores 内容的权限。
.....

8. ..... 授予 Sleepy DELETE表talking_animals 内容的权限，同时也允许 Sleepy 把DELETE表 talking_animals内容的权限GRANT其他人。
.....

9. ..... 把操作表chores 的所有权限授予所有用户。
.....

10. ..... 立刻就可为 Doc 设定 SELECT 权限，权限运用范围是 woodland_cottage 数据库中的所有表。
.....

现在，Elsie 的行为已经在我们的掌控下。接下来想想GRANT 语句对于第 499 页上的 woodland_cottage 数据库会有什么影响。

	程序代码	程序代码有何作用？
1.	`GRANT INSERT ON magic_animals TO doc;`	授予 Doc 插入（INSERT）内容至 *magic_animals* 表的权限。
2.	`GRANT DELETE ON chores TO happy, sleepy;`	授予 Happy 与 Sleepy 删除（DELETE）*chores* 表内容的权限。
3.	`GRANT DELETE ON chores TO happy, sleepy WITH GRANT OPTION;`	授予 Happy 与 Sleepy 删除（DELETE）*chores* 表内容的权限并可把这个权限授予其他人。
4.	`GRANT SELECT(chore_name) ON chores TO dopey;`	让 Dopey 只能从 *chores* 表中选择（SELECT）*chores_name* 列。
5.	`GRANT SELECT, INSERT ON talking_animals TO sneezy;`	授予 Sneezy 选择（SELECT）与插入（INSERT）内容至 *talking_animals* 表的权限。
6.	`GRANT ALL ON talking_animals TO bashful;`	授予 Bashful 选择（SELECT）、更新（UPDATE）、插入（INSERT）、删除（DELETE）*talking_animals* 表内容的权限。

现在试着写出符合要求的 GRANT 语句。

7.	`GRANT SELECT ON chores TO doc;`	授予 Doc SELECT表chores 内容的权限。
8.	`GRANT DELETE ON talking_animals TO sleepy WITH GRANT OPTION;`	授予 Sleepy DELETE表 talking_animals 内容的权限，同时也允许 Sleepy 把DELETE表 talking_animals 内容的权限GRANT其他人。
9.	`GRANT ALL ON chores TO bashful, doc, dopey, grumpy, happy, sleepy, sneezy;`	把操作表chores 的所有权限授予所有用户。
10.	`GRANT SELECT ON woodland_cottage.* TO doc`	立刻就可为 Doc 设定 SELECT 权限，权限运用范围是 woodland_cottage 数据库中的所有表。

GRANT的各种变化

在刚才的习题中，我们看到了 GRANT 语句的几种主要变化形式，整理如下：

1 可用同一个 GRANT 语句为多位用户设定权限。

每个提到名称的用户都会被授予相同权限。

2 WITH GRANT OPTION 让用户能把刚刚获得的权限授予其他用户。

听起来很复杂，但意思很简单：如果某人获得了 SELECT 表 chores 的权限，他可以把这个权限授予其他人，让他们也能 SELECT 表 chores。

3 指定用户可于某个表中使用的列，而不是允许用户操作整张表。

SELECT 权限也仅能限于单一列。用户看到的输出将出自指定的列。

4 一段语句可对表指定超过一种权限。

列出所有要授予用户的表操作权限并以逗号分隔每种权限。

5 GRANT ALL 把 SELECT、UPDATE、INSERT、DELETE 指定表内容的权限都授予用户了。

这只是“允许用户对某张表执行 SELECT、UPDATE、INSERT、DELETE 操作”的缩写方式。

6 使用 database_name.* 可把权限范围运用到数据库中的每张表上。

与 SELECT 语句的通配符（*）相似，代表数据库中的所有表。

撤销权限：REVOKE

假设要把授予 Elise 的 SELECT 权限收回，此时需要**REVOKE** 语句。

还记得最简单的 GRANT 语句吗？REVOKE 语句的语法几乎与它完全相同，只是把“GRANT”换成“REVOKE”，把“TO”换成“FROM”而已。

撤销 SELECT 权限。

```
REVOKE SELECT ON
clown_info
FROM elsie;
```

撤销时使用 FROM，而不是授予时用的 TO。

也可以只撤销 WITH GRANT OPTION，**但不触及权限**。在下面的范例中，*happy* 与 *sleepy* 还能对 chores 表执行 DELETE，但不再能把删除操作的权限授予其他人：

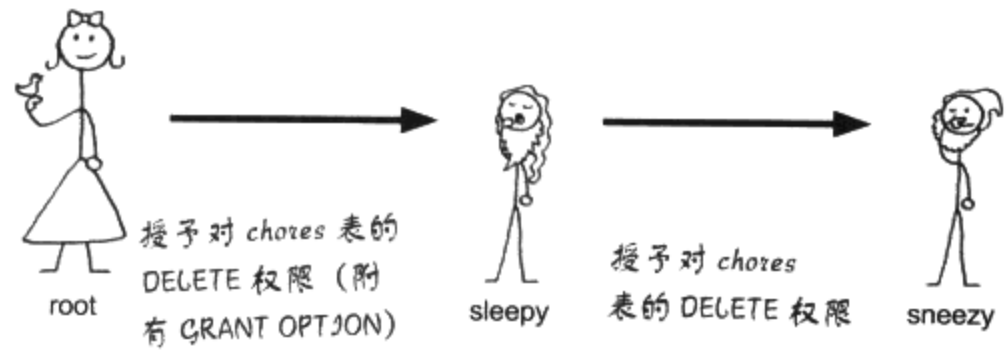
我们只移除 GRANT OPTION 权限。

```
REVOKE GRANT OPTION ON
DELETE ON chores
FROM happy, sleepy;
```

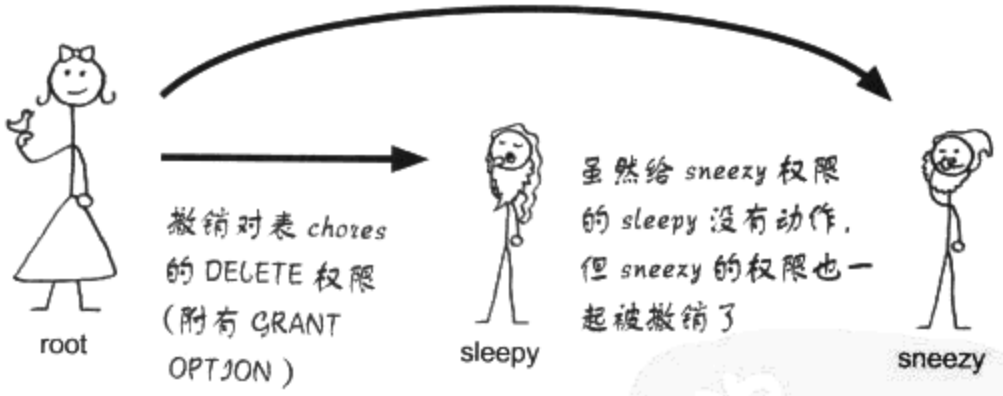
用户 happy 与 sleepy 还可以进行 DELETE，只是不能把这项权限分给别人。

撤销授权许可 (GRANT OPTION)

再来思考一下这种情况：*root*用户给了 *sleepy* 对 chores 表执行 DELETE 操作的权限并附有 GRANT OPTION，然后*sleepy* 又给了 *sneezy* 对 chores 表的 DELETE 权限。



如果*root*用户改变心意，撤销了 *sleepy* 的权限，则 *sneezy* 的该项权限也会被撤销，即使根用户只是对 *sleepy* 撤销了权限。



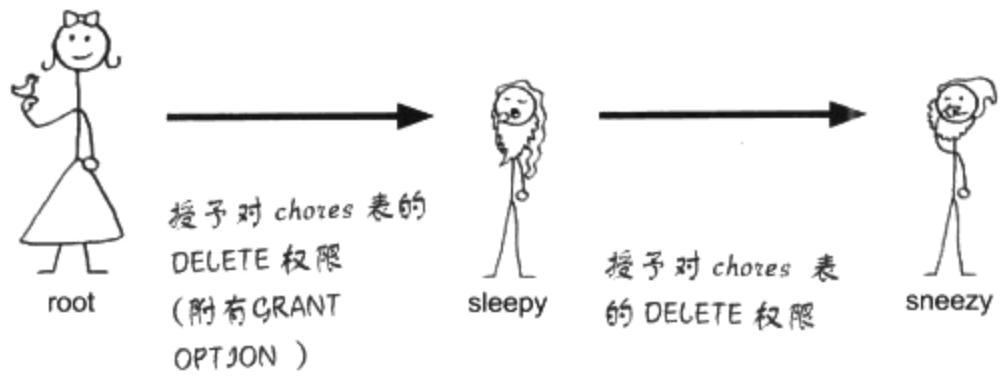
REVOKE 语句的副作用就是让 *sneezy* 也一起失去了权限。不过，有两个关键字可用于控制撤销的范围。

动动脑

我们马上就要与关键字 RESTRICT 与 CASCADE 见面了，你觉得它们分别有什么作用？

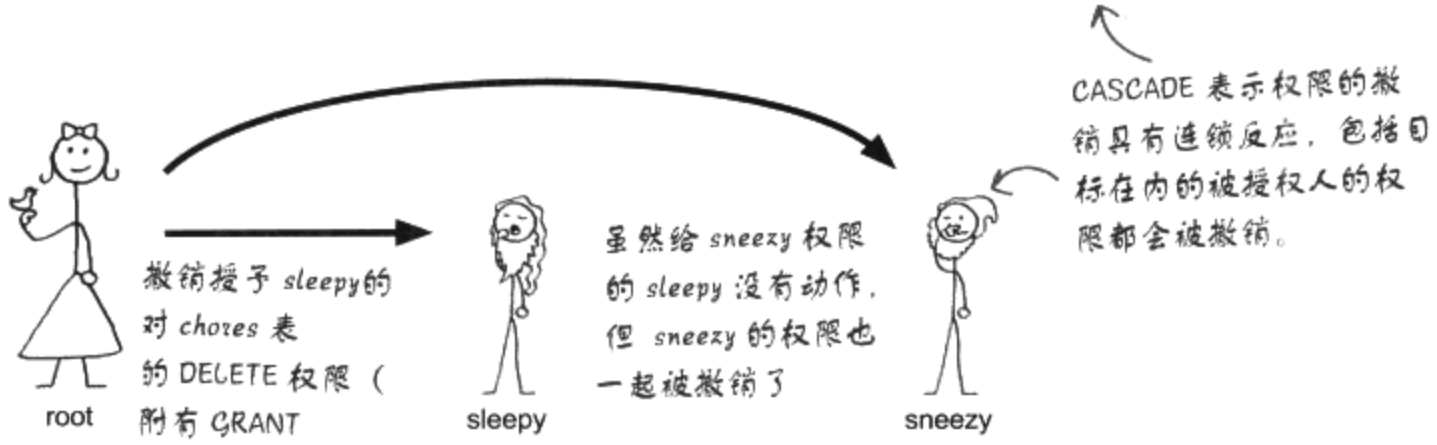
具精确度的撤销操作

有两种方式既可以撤销权限，又可以确保不影响目标以外的用户。你可以使用关键字 CASCADE 与 RESTRICT 来更精确地锁定目标用户，决定谁会失去特权，谁能保持特权。



第一种方式，使用CASCADE 移除目标用户的权限（本例为 *sleepy*）后，如果目标用户已将该权限授予他人，则连同被授予者的权限一起移除。

```
REVOKE DELETE ON chores FROM sleepy CASCADE;
```



若是被撤销权限的目标用户已把权限授予他人，则使用第二种方式，RESTRICT 可返回错误信息。

```
REVOKE DELETE ON chores FROM sleepy RESTRICT;
```

两方的权限都会被保留，*root*用户则因为权限的修改会影响 *sneezy* 而收到错误提示。

磨笔上阵

有人一直把错误的权限授予 Elsie。请写下合适的 REVOKE 语句，让 Elsie 回到安全的只有 SELECT 权限的状态。

```
GRANT SELECT, INSERT, DELETE ON locations TO elsie;
```

.....

```
GRANT ALL ON clown_info TO elsie;
```

.....

```
GRANT SELECT, INSERT ON activities TO elsie;
```

.....

```
GRANT DELETE, SELECT on info_location TO elsie
WITH GRANT OPTION;
```

.....

```
GRANT INSERT(location), DELETE ON locations TO elsie;
```

.....

磨笔上阵解答

有人一直把错误的权限授予 Elsie。请写下合适的 REVOKE 语句，让 Elsie 回到安全的只有 SELECT 权限的状态。

```
GRANT SELECT, INSERT, DELETE ON locations TO elsie;
```

```
REVOKE INSERT, UPDATE, DELETE ON locations FROM elsie;
```

```
GRANT ALL ON clown_info TO elsie;
```

```
REVOKE INSERT, UPDATE, DELETE ON clown_info FROM elsie;
```

我们还是希望 Elsie 有 SELECT 权限，所以不会一次撤销所有权限。

```
GRANT SELECT, INSERT ON activities TO elsie;
```

```
REVOKE INSERT ON activities FROM elsie;
```

另外一种做法则是撤销所有权限，然后重新授予。

```
GRANT DELETE, SELECT on info_location TO elsie
WITH GRANT OPTION;
```

```
REVOKE DELETE on info_location FROM elsie CASCADE;
```

```
GRANT INSERT(location), DELETE ON locations TO elsie;
```

```
REVOKE GRANT INSERT(location), DELETE ON locations FROM elsie;
```

看起来这里可能要用 GRANT 来确定 Elsie 还是能对 location 表进行 SELECT 操作。

而且，我们最好确认她不会把同样的权限授予其他人。

问： 指定列名的 GRANT 语句还是让我念念不忘。如果只对表中的某一列授予 INSERT 权限会发生什么事？

答： 好问题。这种情况的INSERT实际上是个无用的权限。如果只能插入某列的值，就无法真正地插入一条新记录到表中。除非在那张表中只有GRANT指定的列必须有内容。

问： 还有其他无用的权限吗？

答： 几乎所有针对列的权限都没有用，除非是GRANT语句中与SELECT有关的权限。

问： 假设我想添加一个用户，让他可以选取我所有的数据库中的所有表，有比较简单的设定方法吗？

答： 这个问题要看每个人使用的RDBMS而定，本章的很多问题也一样。在MySQL中，可以授予全局权限：

```
GRANT SELECT ON *.*
TO elsie;
```

第一个星号代表所有数据库，第二个星号表示所有表。

问： 如果未指定 REVOKE 的使用方式，CASCADE 是默认值吗？

答： CASCADE 通常是默认值，不过，还是请你先参考 RDBMS 的说明文档。

问： 如果我对用户根本没有的权限下了 REVOKE 命令，会发生什么事？

答： 会出现错误信息，告诉你一开始就根本没有 GRANT！

问： 在前面的例子中，根用户撤销了 sneezy 的某个权限，如果有两名其他用户授予 sneezy 同样权限，会发生什么事？

答： 嗯，这个问题真是棘手。有些系统在 GRANT 使用 CASCADE 时不会注意谁发出了 GRANT 语句，有些系统则会忽略。这又是一个要参考 RDBMS 的说明文档的例子。

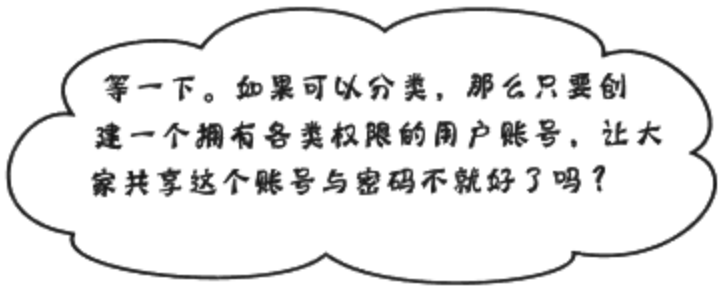
问： 除了表和列以外，还有什么可以使用 GRANT 与 REVOKE？

答： 这两个语句也能用于视图，但不包括不可更新的视图，此时，就算具有 INSERT 权根，也一样无法插入新数据。因为视图和表几乎一样，所以能单独授予视图中的某列操作权限。

如果我想让 5 名用户拥有相同权限，只要在 GRANT语句的最后面列出每位用户的账号并以逗号分隔就可以了吗？

当然可以。如果用户不多，这是最好的方式。

但随着企业成长，数据库开始有不同类型的用户。可能有10 位用户专心于数据录入，他们只需要插入与选择特定表的权限。可能有 3 位权限很高的用户，他们需要所有操作权限。另外还有更多用户只需要 SELECT。甚至还会出现连接至数据库的软件与 Web 应用程序，它们需要查询特定视图的特殊方式。



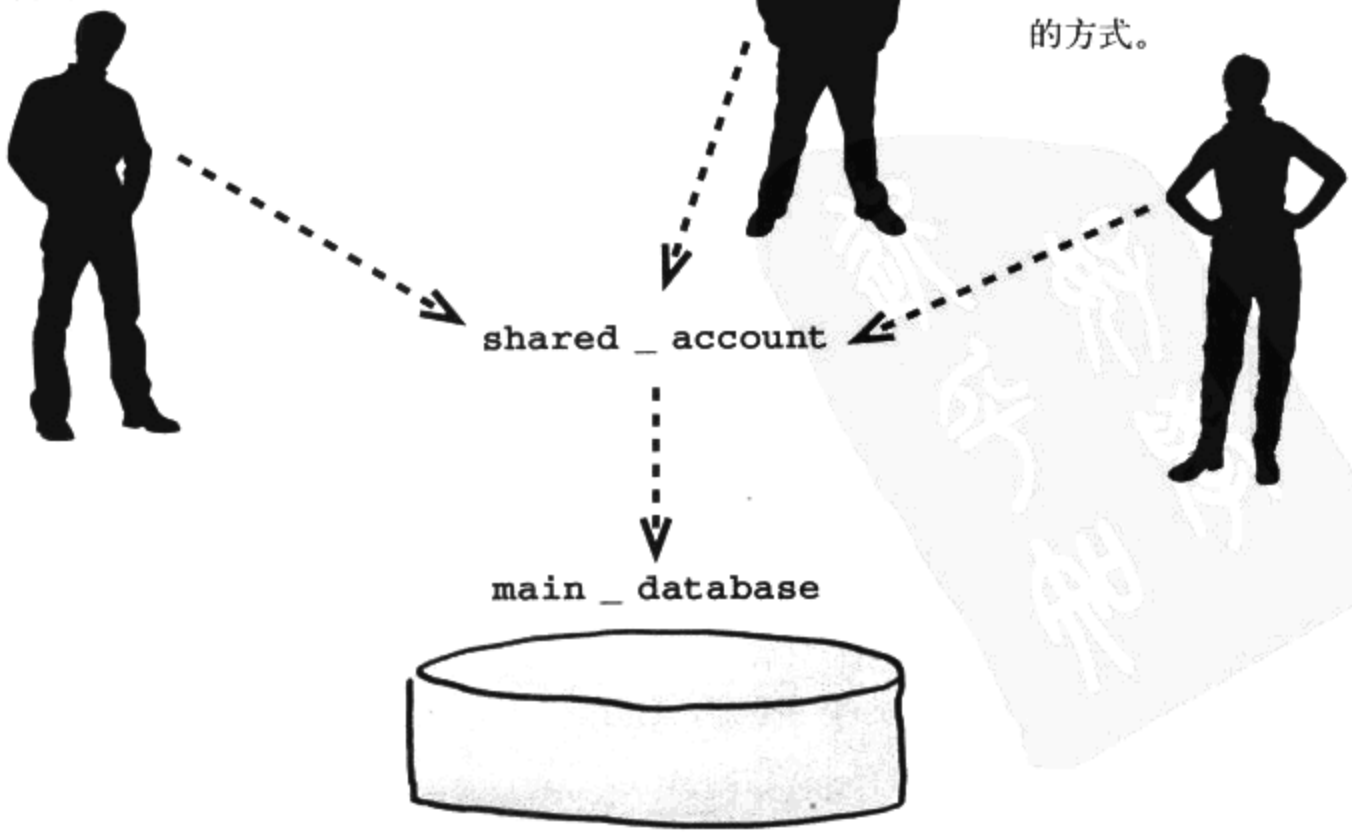
共享账号的问题

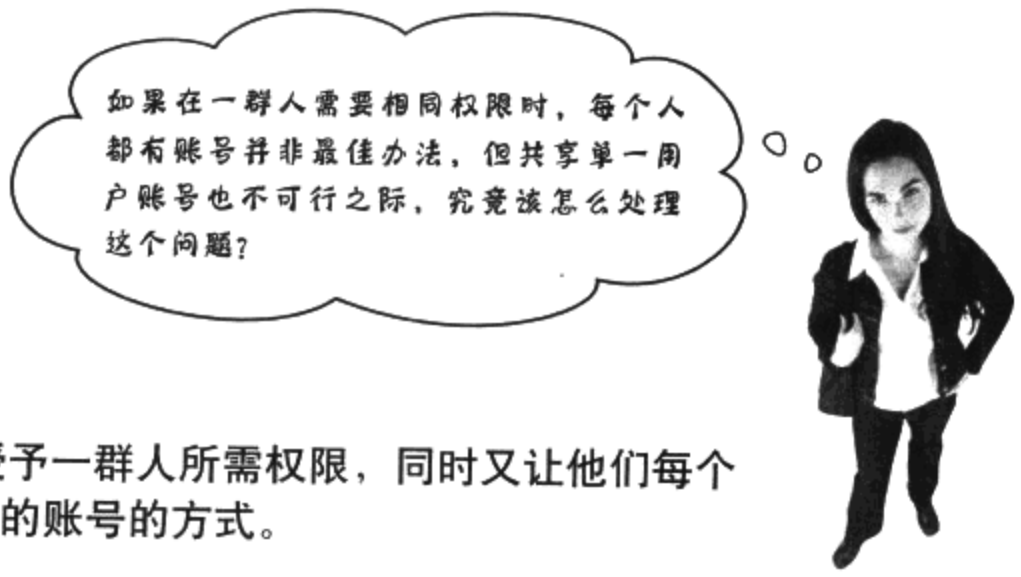
虽然有些公司的确在只有一个数据库账号的制度下运作良好，但这却非安全之道。我们举个很可能出问题的例子：

Simon 改变了密码，可是忘了告诉其他人。现在没人有办法登录数据库，除非 Simon 记得告诉大家新密码。

Randy 必须有数据库操作的完整权限才能完成他的工作。如果共享数据库账号及密码，数据库将无法拒绝其他 SQL 知识不足、较易犯错的用户。

Paula 对于设计更新查询并不在行，而且常常弄错数据。没人知道是谁一直在弄错数据，所以也没人能帮助 Paula 学习到正确的方式。





我们需要授予一群人所需权限，同时又让他们每个人都有自己的账号的方式。

此时需要**角色**（role）。角色是把特定权限汇集成组，再把组权限授予一群人的方式。角色成为一个数据库对象，可于数据库变动时依需求调整，而不需逐一指定、调整每名用户的权限。

而且设定角色很简单：

```
CREATE ROLE data_entry;
```

我们创建的角色名称

MySQL 没有角色功能

未来的 MySQL 版本或许会纳入角色功能，但现在只能用逐一指定每名用户权限的方式。

想为角色授予权限时，直接把角色当成用户就好了：

```
GRANT SELECT, INSERT ON some_table TO data_entry;
```

本例授予权限时并非授予用户，而是授予角色。

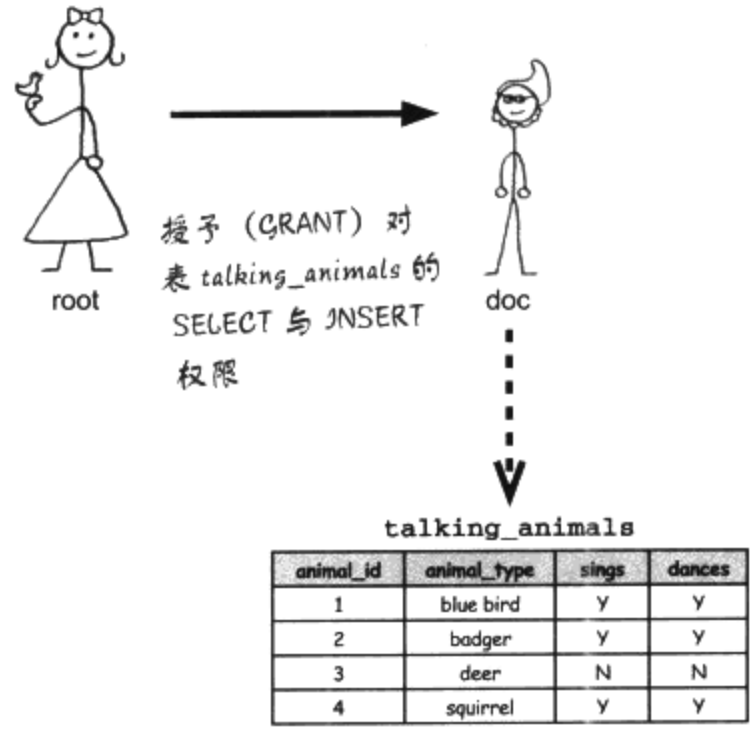
我们已经创建了角色，也已授予角色权限。现在需要指定用户的角色……

使用角色

在创建角色前，GRANT能直接把权限授予负责数据的用户，如下所示：

```
GRANT SELECT, INSERT
ON talking_animals
TO doc;
```

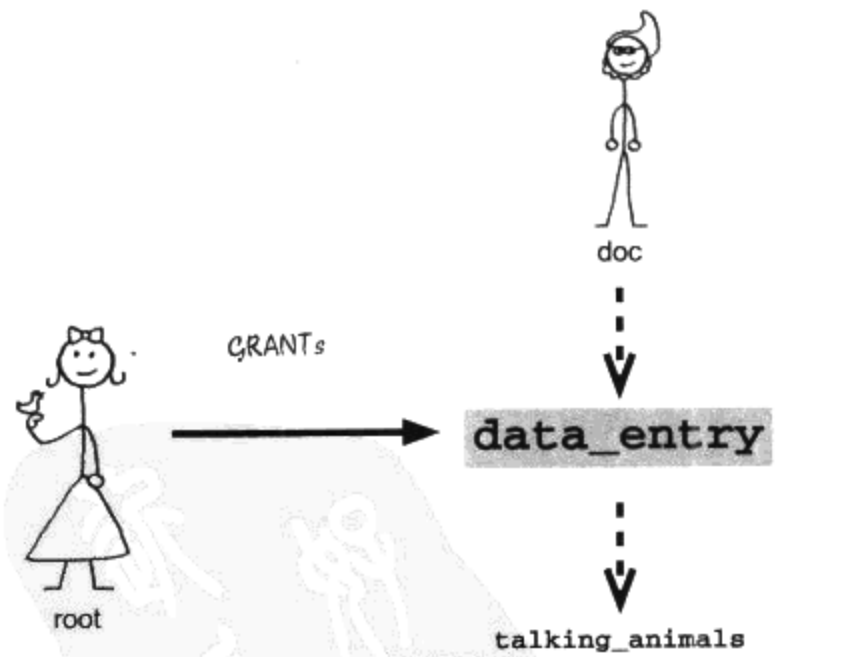
以前的方式。



现在只要把授予“哪些操作”权限的部分换成角色名称并指定给用户（如 *doc*）。我们不需提及权限或表，因为这些信息都已存储在 data _ entry 角色中：

```
GRANT data_entry TO doc;
```

角色名称替换了表名与权限



卸除角色

不再需要某个角色时，自然也没有保留它的必要，请使用 DROP 语句删除角色：

```
DROP ROLE data_entry;
```

问： **如果我想授予数据库中所有表操作权限，我需要逐一列出表吗？**

答： 不用，使用下列语法即可：

```
GRANT SELECT, INSERT, DELETE
ON gregs_list.*
TO jim;
```

你只需列出数据库名称，并使用“*”把权限指派给所有表。

问： **如果角色已赋予用户，我们仍可以卸除角色吗？**

答： 使用中的角色一样可被卸除。但卸除角色前，请注意用户是否会因此而失去必要权限。

问： **也就是说，曾经拥有某个角色的用户，一旦其角色被卸除了，他就会失去角色权限？**

答： 非常正确。就好像特别授予用户某一组权限，然后再撤销同一组权限。只不过，这次不只影响被撤销权限的单一用户，而是影响所有指定为那个角色的用户。

问： **用户可以同时身兼多角吗？**

答： 可以。但请确认角色间的权限并不冲突，否则可能自找麻烦。否定性的权限优先于授予性的权限。

撤销角色

撤销角色的运作方式与撤销权限的方式很像。请试着针对用户 doc，写下撤销 data_entry 角色的语句，**不要参考前面的内容。**

撤销角色的运作方式与撤销权限的方式很像。请试着针对用户 doc，写下撤销 data_entry 角色的语句，**不要参考前面的内容。**

```
REVOKE data_entry FROM doc;
```

加上 WITH ADMIN OPTION 的角色

就像 GRANT 语句可附加 WITH GRANT OPTION一样，角色也有提供类似功能的 WITH ADMIN OPTION。这个选择功能让具有该角色的每名用户都能把角色授予其他人。以下列语句为例：

```
GRANT data_entry TO doc WITH ADMIN OPTION;
```

WITH ADMIN OPTION 允许用户 doc 把角色 data_entry 授予其他人。

doc 现在已具有管理员（admin）权限，他可以把角色 data_entry 授予 *happy*，授予方式就和他被授予权限的语法相同：

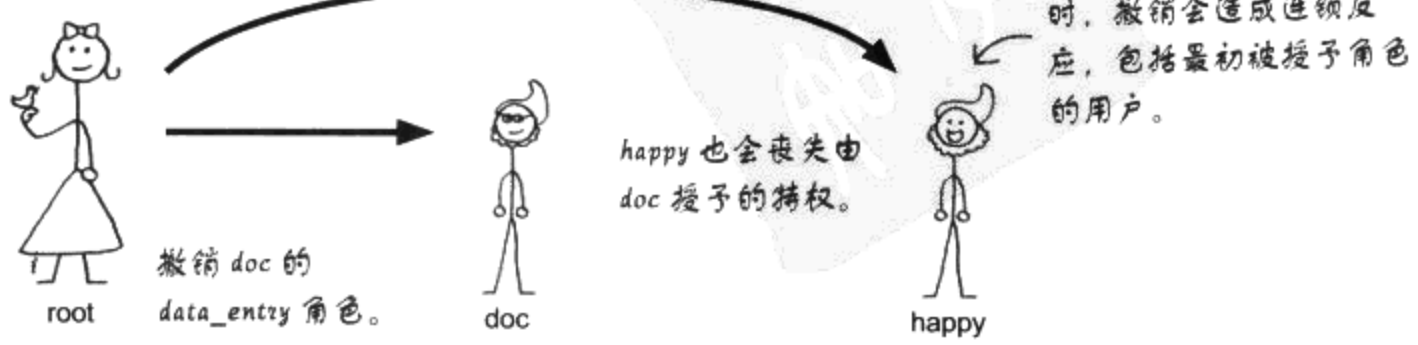
```
GRANT data_entry TO happy;
```

REVOKE 运用于角色时，仍然可以使用关键字 CASCADE 与 RESTRICT。让我们看看关键字如何运作：

撤销角色时采用 CASCADE

与 CASCADE 一起使用时，撤销（REVOKE）角色会造成连锁撤销反应，包括最初被授予角色权限的用户。

```
REVOKE data_entry FROM doc CASCADE;
```



撤销角色时采用 RESTRICT

与 RESTRICT 一起使用时，若是被撤销（REVOKE）角色的目标用户已把权限授予他人，这种方式可返回错误信息。

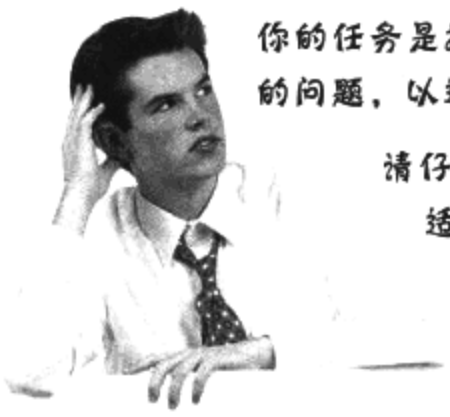
两方的权限都会先保留，*root*用户则因为权限修改会影响 *happy* 而收到错误信息。

没错，的确该为 Greg 的雇员设定更安全的数据库使用方式。

Greg 需要走过这一章的每个步骤并保护根用户的账号，想出员工的需求，并给他们正确的权限。

幸好，我们马上就要变身成 Greg……

与Greg天人合一



你的任务是扮演Greg（最后一次）并修复数据库用户的问题，以避免他的员工意外破坏数据。

请仔细阅读每个用户的工作说明，设计出合适的GRANT语句（不只一个），让每个人都能取得所需数据又不会看到不该看到的东西。

Frank：我负责为可能的职缺寻找适合的人选。我不需要输入任何数据，不过在找到合适的应征人选或某项职缺已经不再招人时，我会从职缺列表中删除该职缺。有时候，我也需要到 my _ contacts 表中找出联络信息。

Jim：我负责输入所有新数据。我很擅长录入数据，而且我现在也不会意外地在性别列填入“X”，更新数据也是我负责。我目前正在学习删除操作，但 Greg 还不让我删除数据。当然，他不知道……

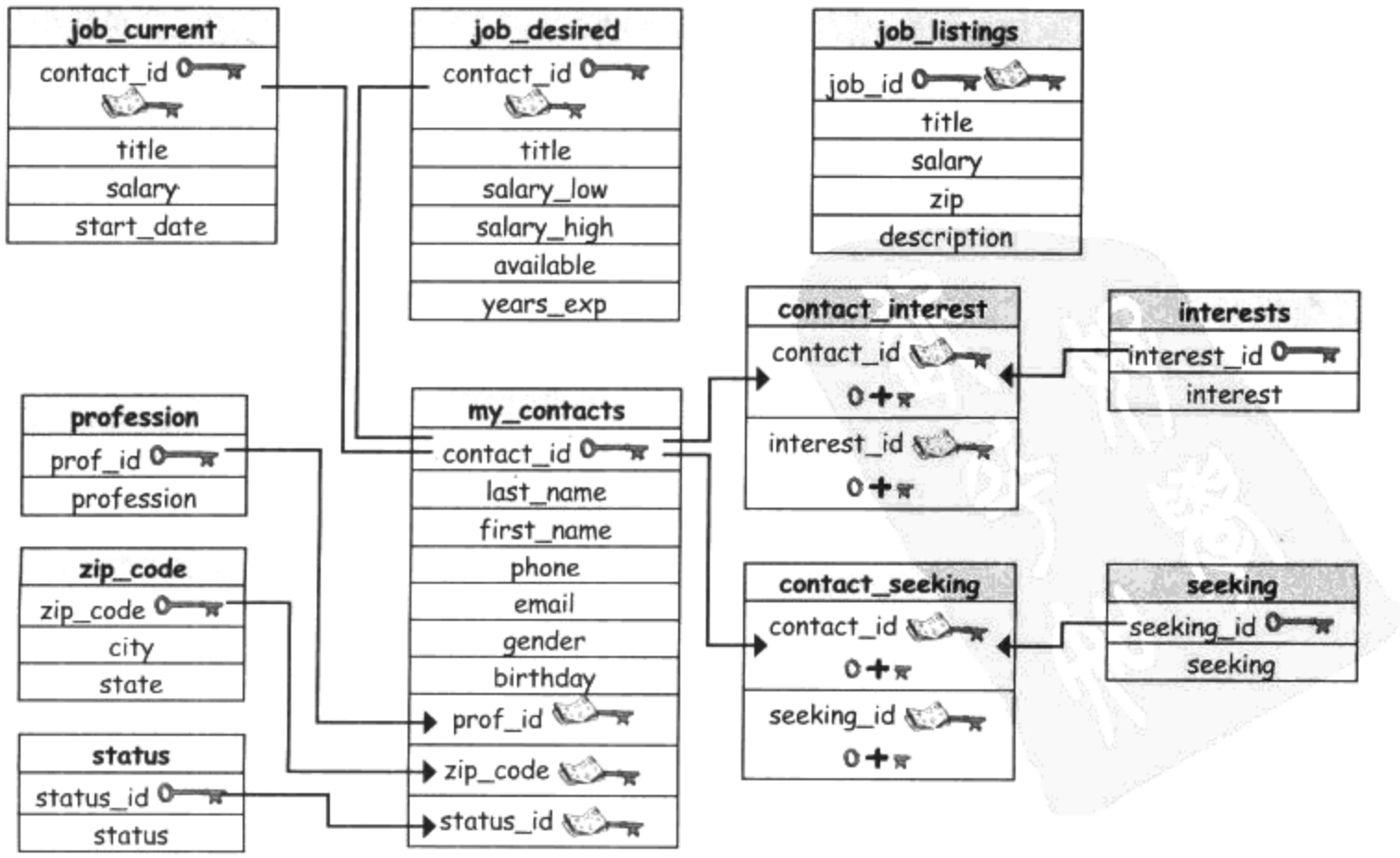
Joe：Greg 雇用我管理约会速配部分，他希望把联络人的信息传到网站上。我其实比较接近网站设计员，而非专业 SQL 人才，但我会做点简单的选取操作。我不会插入任何数据。

研究 gregs_list 数据库，并在有人破坏数据前为每个人设计 GRANT。

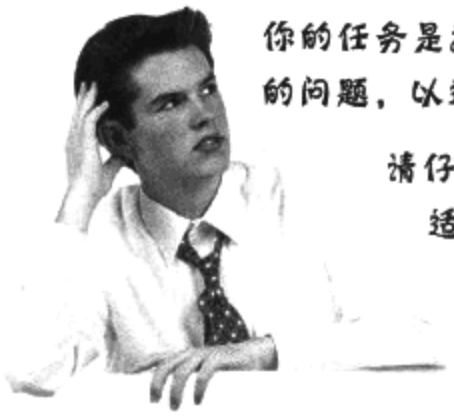
写出为当前的“根用户”加上密码的命令。

写出为每名员工创建用户账号的三条命令。

为三名员工设计 GRANT 语句，给他们合适的权限。



与Greg天人合一解答



你的任务是扮演 Greg（最后一次）并修复数据库用户的问题，以避免他的员工意外破坏数据。

请仔细阅读每个用户的工作说明，设计出合适的 GRANT 语句（不只一个），让每个人都能取得所需数据又不会看到不该看到的东西。

写出为当前的“根用户”加上密码的命令。

```
SET PASSWORD FOR root@localhost = PASSWORD( 'gr3GRulz' );
```

写出为每名员工创建用户账号的三条命令。

```
CREATE USER frank IDENTIFIED BY 'j06M4tcH';

CREATE USER jim IDENTIFIED BY 'NOmOr3Xs';

CREATE USER joe IDENTIFIED BY 's3LeCTdOOd';
```

不要担心你设计的密码与我们的不同！只要命令顺序正确，练习目的就达到了。

为三名员工设计 GRANT 语句，给他们合适的权限。

```
GRANT DELETE ON job_listings TO frank;
GRANT SELECT ON my_contacts.* TO frank;
```

Frank需要从job_listings 表中删除（DELETE）内容以及从my_contacts表寻找（SELECT）内容的权限。

```
GRANT SELECT, INSERT ON gregs_list.* TO jim;
```

Jim 需要对整个 gregs_list 数据库执行 SELECT 与 INSERT 操作。目前还要禁止他删除（DELETE）任何内容。

```
GRANT SELECT ON my_contacts.profession, zip_code, status,
contact_interest, interests, contact_seeking, seeking TO joe;
```

Joe 需要从个人表中选取（SELECT）内容，但与招聘有关的表则与他无关。

结合 CREATE USER 与 GRANT

我们可以试着把 CREATE USER 与 GRANT 结合成一条语句吗？

当然可以。其实只需要结合各位已经知道的部分。

关于 Elsie 的用户账号，我们用了两条语句：

```
CREATE USER elsie
IDENTIFIED BY 'cl3v3rp4s5w0rd';

GRANT SELECT ON
clown_info
TO elsie;
```

我们可以结合语句，省略 CREATE USER 的部分。因为用户 *elsie*必须先被创建，然后她才能获得权限，所以RDBMS会先检查用户名称是否存在，如果不存在则自动创建账号。

```
GRANT SELECT ON
clown_info
TO elsie
IDENTIFIED BY 'cl3v3rp4s5w0rd';
```

Greg's List 已经成为跨国企业了

多谢各位的帮忙，Greg 现在已经非常熟悉 SQL 的使用了，而且还能指导 Jim、Frank、Joe，他甚至还把 Greg's List 的业务扩展到本地分类广告与讨论组。

最好的消息是什么？因为Greg在Dataville的发展极为出色，全球现在有超过500个城市都在使用他的系统，Greg 一跃成为上流社会的名人了！

THE WEEKLY
INQUERYER

Greg's List 百尺竿头，更进一步

授权与讨论组

亲朋好友都说Greg 没有因为名声而改变。

Troy Armstrong

INQUERYER 编辑室

【DATAVILLE】本地青年企业家 Greg 最近发展神速。他的网络数据库系统从最初的便笺起家，转变为简易表，最后成长为包含多张表的提供约会交友、招聘等众多服务的数据库。

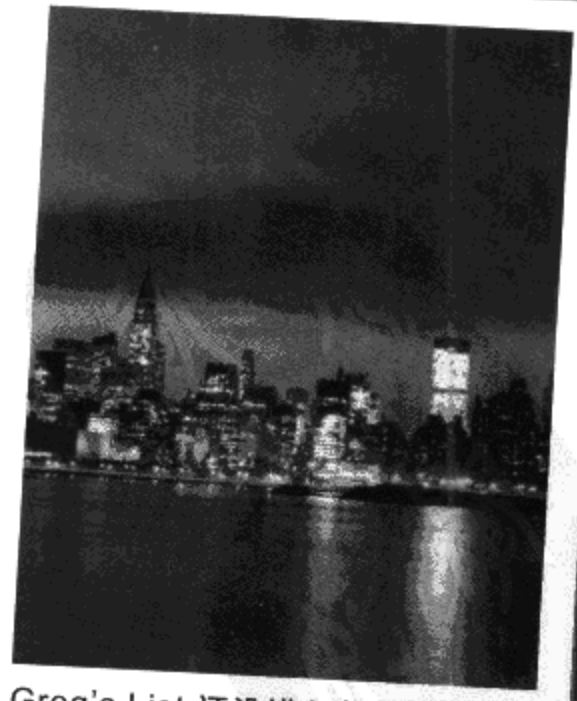
如果你也对 Greg 的事业有兴趣，请访问：

www.gregs-list.com

测试你的 SQL 技能。如果你想找人讨论内联接、事务与权限设计，最好的选择当然是我们的 SQL 讨论组：

www.headfirstlabs.com

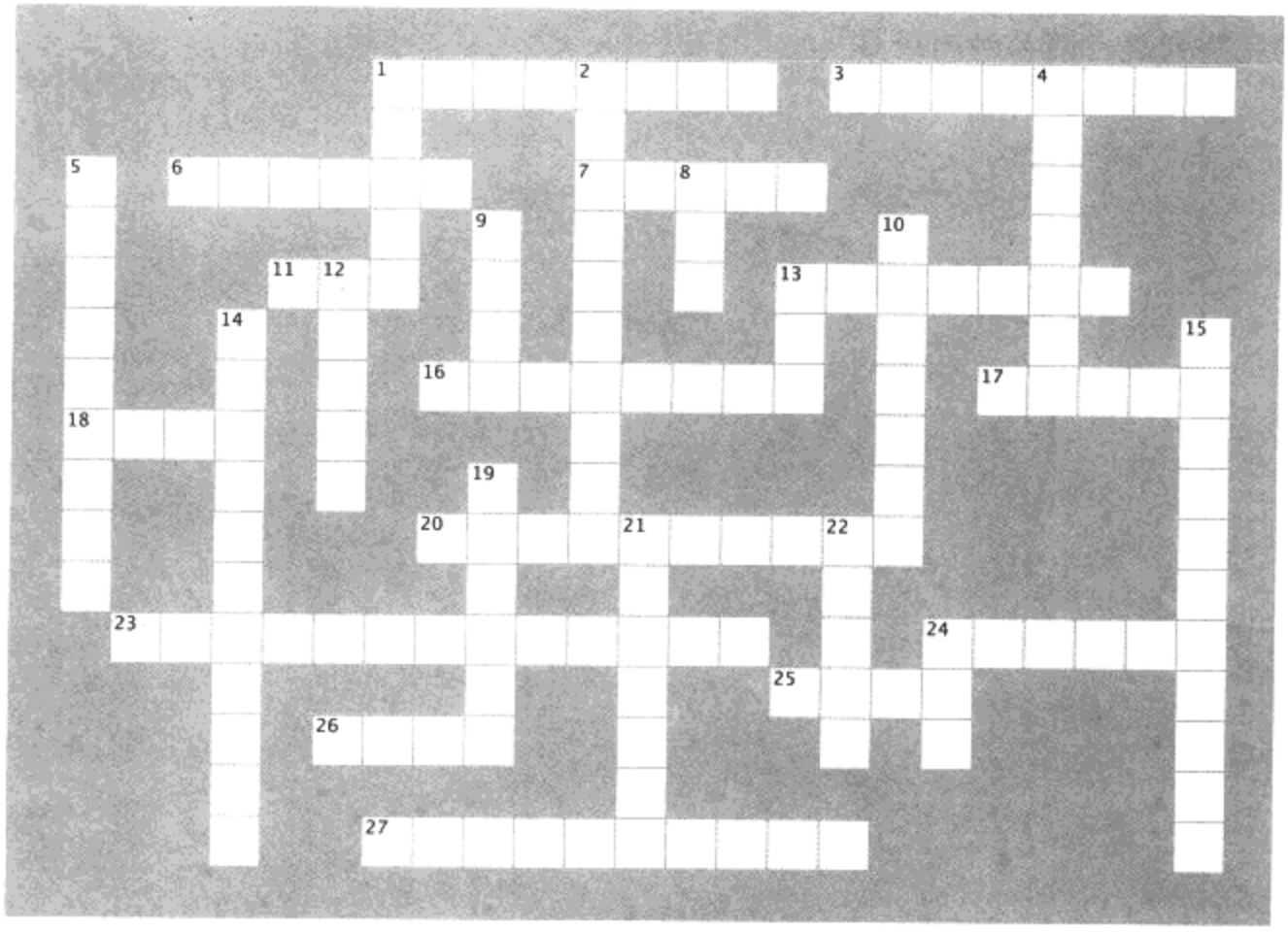
各位热爱 SQL 的朋友，最重要的一点，祝大家玩得开心！



Greg's List 还没进入你的城市吗？没关系，时机很快就到了。（城市数据分析师）

（最后一个）SQL 填字游戏

啊，人生最苦苦别离，你现在看到的是本书最后一个填字游戏。准备好，这篇告别作塞满了各种关键字与命令，开心地玩吧！



横向

1. ____ ____ 允许用户对特定表执行 SELECT、UPDATE、INSRER、DELETE 操作。

3. 值有重复时，____ 函数遇到重复的值只返回一次。

6. ____ 表没有重复的数据，因此能减少数据库的体积。

7. 授予角色时附上 WITH ____ OPTION，可允许被授予角色的用户把角色授予他人。

11. _____PASSWORD FOR 'root'@'localhost'= PASSWORD('b4dcl0wnZ')

13. 存储为 CHAR 或 VACHAR 类型的值称为 ____。

16. 如果希望撤销权限时能在影响到其他用户时收到警告，请加上关键字 ____。

17. 利用内联接，可匹配两张表的记录，但它们的 ____ 并不重要。

18. 可使用 ____-join 模拟联接两张相同表的效果。

20. 如果改变任何非键列可能造成任何其他列的改变，就是 transitive ____。

23. 如果子查询可以独立执行且不需引用外层查询，则称为 ____ 子查询。

24. 如果数据已被拆解成最小可能状态，已经不能或不该继续分解，则称为 ____。

25. 为了帮助你判断哪些 SQL 步骤可被视为事务过程，请记得这个缩写：____。

26. 一个 ____ OUTER JOIN 把左表的所有记录拿来与右表匹配。

27. ____ 子查询表示内联接依赖外联接的结果来解析。

纵向

1. 使用 ____ 语句即可控制用户对表的操作权限。

2. ____functional dependency（____ 函数依赖），表示某个非键列与其他非键列有关联。

4. 每个表只能有一个 AUTO_INCREMENT 列，且数据类型必须为 ____。

5. ____ KEY是由多列构成的有唯一性键值的主键（PRIMARY KEY）。

8. 利用 ____ 函数可找出列的最大值。

9. 把一组权限赋予一群用户的方式。

10. 这两个字能依据指定列值的字母顺序排列查询结果。

12. non-equijoin 返回并不 ____ 行。

13. 在 UPDATE 语句中加上 ____ 子句，可改变数据值。

14. self-____ 外键，也是表的主键，但作为其他用途。

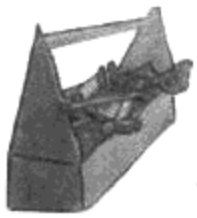
15. 在 ____ 中，如果所有步骤无法不受干扰地全部完成，则不应单独完成任何步骤。

19. 子查询必是 ____ 语句。

21. 除非联接的列在两边的表中的名称相同，____ JOIN 才会成功。

22. ____ constraint 限定了可以插入列的值。

24. 表只能通过 ALTER 语句与 ____ COLUMN 子句增加新列。



你的SQL工具包

恭喜！恭喜！各位已经完成第12章了！花一点时间复习一下我们讨论的 SQL 安全准则。如果需要本书工具的完整列表，请参考附录 3。

CREATE USER

有些 RDBMS 使用这个语句创建用户并设定其密码。

GRANT

根据授予用户的权限，精确控制用户对数据库的操作范围。

REVOKE

这个语句用于撤销用户的权限。

WITH GRANT OPTION

让用户把自己获得的权限授予其他人。

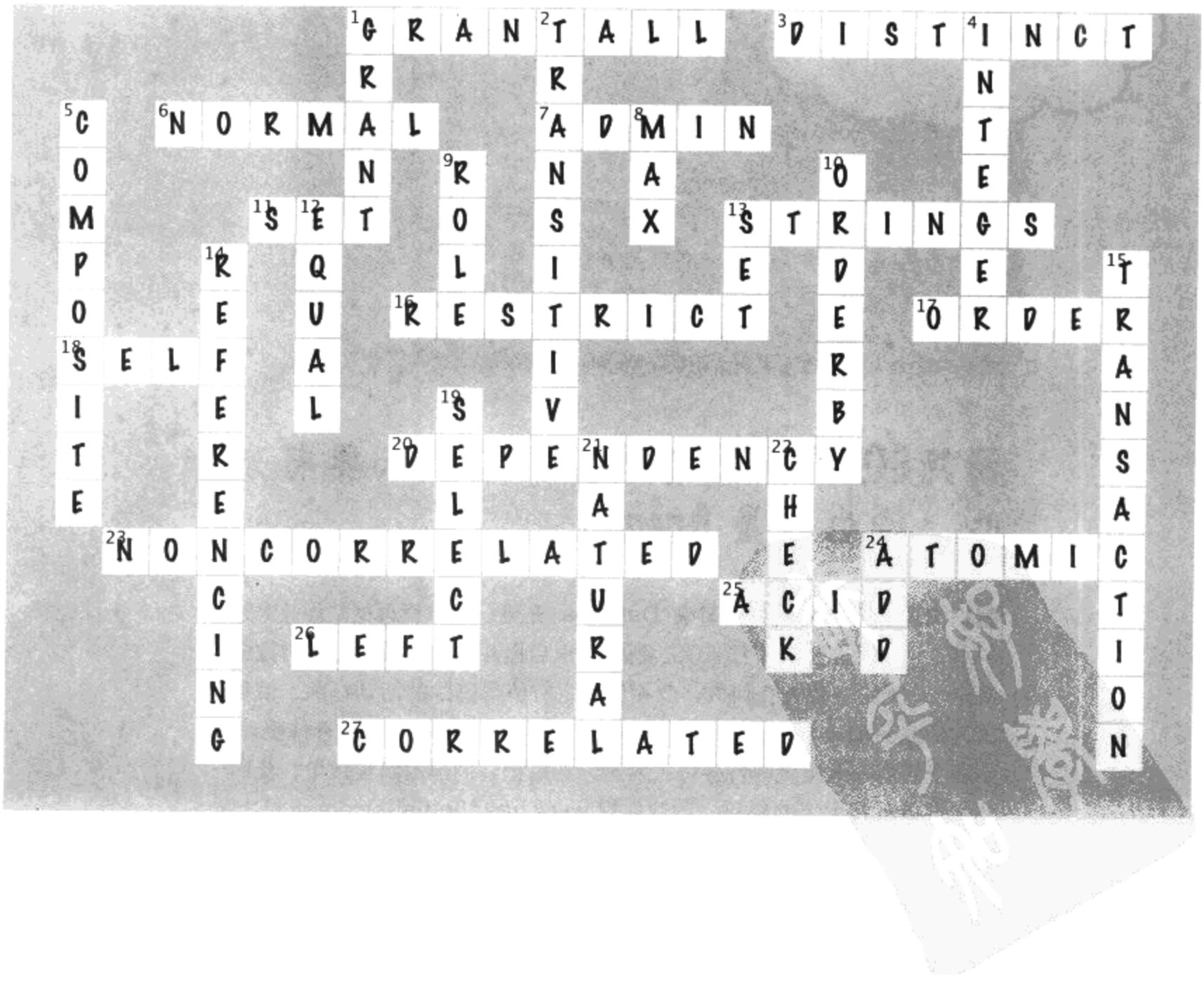
WITH ADMIN OPTION

让有角色的用户把同一个角色授予其他人。

Role

角色是指一组权限。角色能把一组特定权限一次授予多名用户。

（最后一个）SQL 填字游戏解答



Greg's List 在你的城市发展得好不好？

请把SQL应用到你的项目中，只要有心，你也会是 Greg！

我们由衷地欢迎有志之士前来 Dataville 充电。虽然说再见是如此难过的事，但重点在于大家都带着学到的知识启程并把它**应用到自己的数据库中**——无论各位人在何方，我相信一定有值得追踪的活动行踪、值得试吃的甜甜圈或其他美食，或者你自己就有一份联络人列表有待整理。封底上还有一些宝藏等待各位去发掘，然后把它们运用到实践中。我们非常期待听到大家的感想，欢迎你到 www.headfirstlabs.com 上留言，让我们知道 SQL 为你带来了哪些帮助。

附录 1

十大遗珠

尽管刚刚结束一场SQL盛宴，但总会有剩下的东西。我想各位还需要知道一些其他补充事项，就算只能简短提一下也好……忽略了就是觉得怪怪的。放下本书前，希望大家稍微看一下本章的SQL小花絮。

另外，本章结束后还有两篇附录……可能还有几篇广告……然后就真的结束了，真的！我保证！

#1. 为 RDBMS 取得图形用户界面

能够直接在控制台编写 SQL 代码固然重要，我们已经知道该怎么做了，所以可以学习更简单的创建表及观察内容的方式。

每个 RDBMS 都有相关的图形用户界面（graphical user interface，GUI），接下来简短地介绍一下可用于 MySQL 的 GUI。

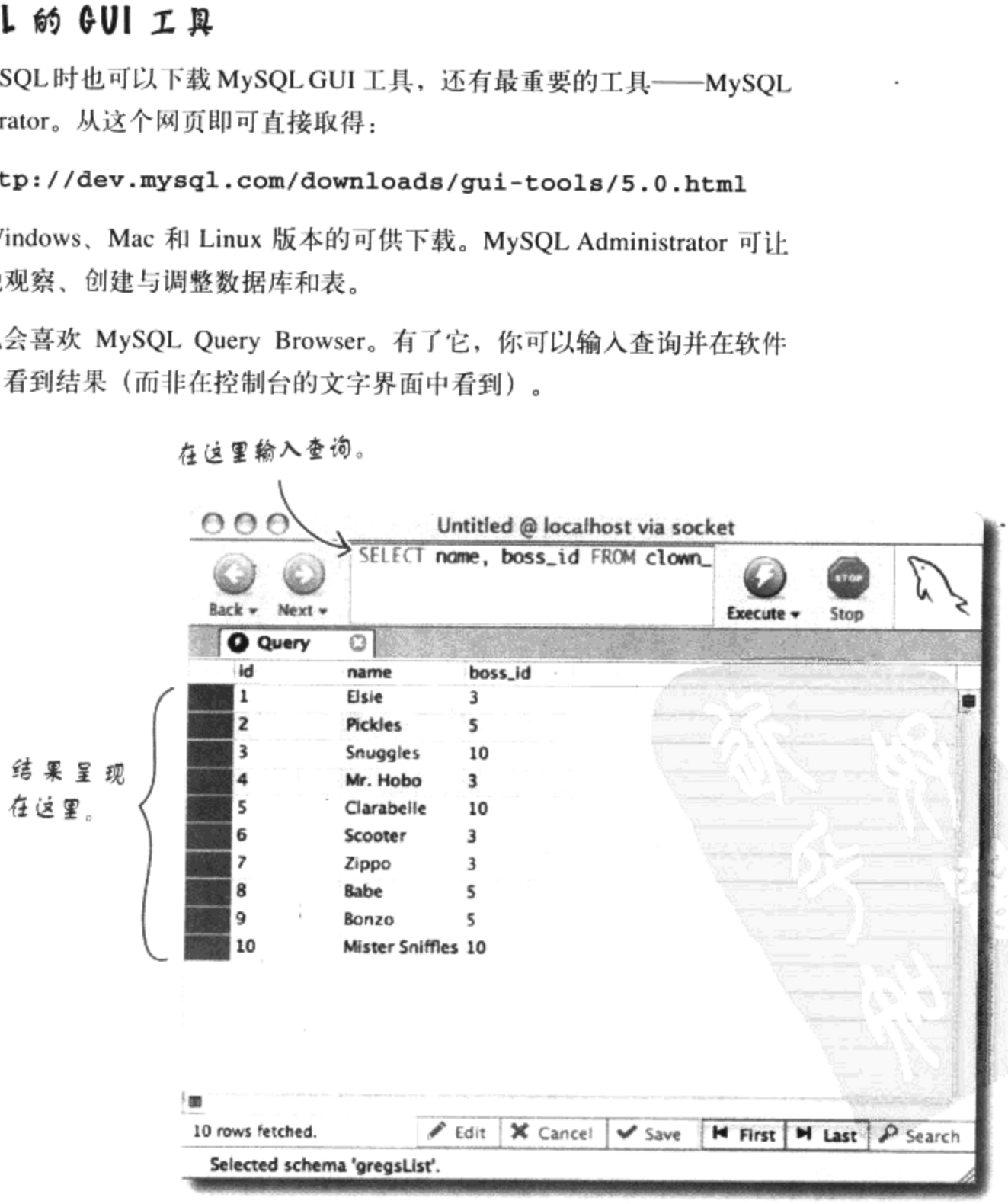
MySQL 的 GUI 工具

下载 MySQL 时也可以下载 MySQL GUI 工具，还有最重要的工具——MySQL Administrator。从这个网页即可直接取得：

```
http://dev.mysql.com/downloads/gui-tools/5.0.html
```

上面有 Windows、Mac 和 Linux 版本的可供下载。MySQL Administrator 可让你简单地观察、创建与调整数据库和表。

你或许也会喜欢 MySQL Query Browser。有了它，你可以输入查询并在软件界面卡中看到结果（而非在控制台的文字界面中看到）。

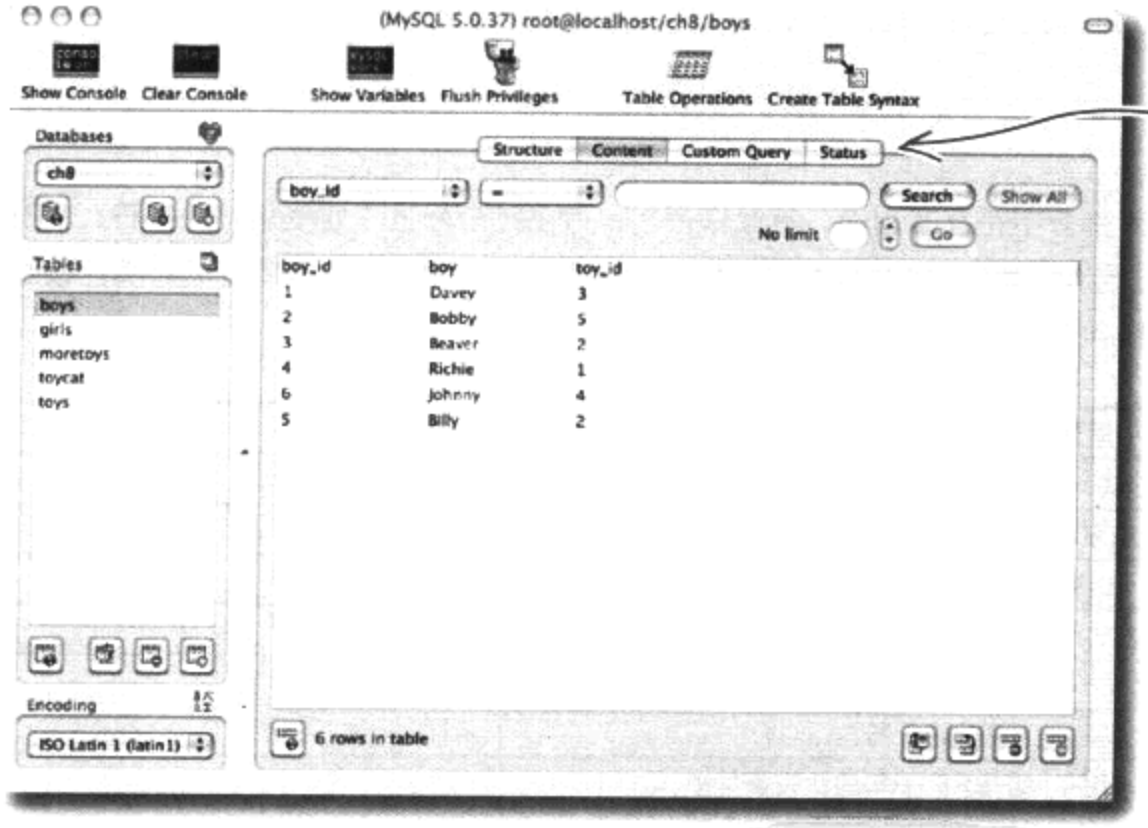


其他 GUI 工具

还有不少可供选择的其他 GUI工具，挑出最合适工具的任务就留给各位了。上网搜索，你可以轻松找到很多我们没有提到的 GUI工具。

Mac 的用户可以考虑 CocoaMySQL:

```
http://cocoamysql.sourceforge.net/
```



如果需要基于网络的解决方案，可以考虑 phpMyAdmin。当你在远程网站服务器上搭配网站管理员账号使用 MySQL 时，phpMyAdmin 运作得非常好，但它不太适合用于本地机器上。更多信息请见：

```
http://www.phpmyadmin.net/
```

还有一些常用的工具，其中有些只能用在 PC 上，你最好先仔细阅读它们的网站上的最新信息，再决定要不要下载：

```
http://www.navicat.com/
```

SQLyog 提供了免费的 Community Edition：

```
http://www.webyog.com/en/
```

#2. 保留字与特殊字符

SQL语言由许多保留字（关键字）组成。为数据库、表、列设计名称时最好不要使用保留字。就算你真的很想把新表叫做“select”，还是应该试着想出更具有描述性而且完全没用到“select”的其他名称！如果你还是坚持要用保留字，最好加上下划线并与其他词一起构成名称，以免RDBMS混淆。为了参考方便，右页列出了最好避免使用的保留字。

考虑到以后的复杂用途，SQL 还有“可能于未来的 SQL 版本里成为保留字”的非保留字列表。这里不列出来，但只要是 RDBMS 的说明文档或书籍，在其中都可找到参考信息。

特殊字符

以下列出 SQL 使用的大部分字符及其用途。就像保留字，命名时最好也不要使用特殊字符，只有下划线字符（_）例外，它是专门用于名称中的字符。一般而言，表名最好只有字母与下划线，最好也不使用数字，除非数字具有描述内容的功能。

*	从 SELECT 语句中返回表的所有列。
()	用于集合一组表达式、指定执行数学运算的顺序以及调用函数。也用于围起子查询。
;	表示 SQL 语句的结束。
,	分隔各个列表项。出现在 INSERT 语句与 IN 子句中。
.	用于引用表名以及表示浮点数。
_	在 LIKE 子句中代表一个字符的通配符。
%	在 LIKE 子句中代表多个字符的通配符。
!	感叹号等于 NOT，用于 WHERE 子句的比较条件中。
'	一对单引号，告诉 SQL 其中的内容是字符串。
"	一对双引号也有单引号的功能，不过最好使用单引号。
\	可于表内存储作为文字用的单引号字符。
+	除了表示加法，加号也能用于连接或串联两个字符串。

这两个是可用于 LIKE 的通配符。

接下来是四则运算：

+	加法	-	减法	*	在两个值中间的星号代表乘法	/	除法

还有比较运算符：

>	大于	!>	不大于	>=	大于等于
<	小于	!<	不小于	<=	小于等于
=	等于	<>	不等于	!=	不等于

&	|	^	~
书中没有提过这些字符，请查看你的 RDBMS 说明文档。			

保留字

凡是需要以单词为任何数据内容取名时，请参考这份表格，确保名称中不包含这些词。

A	ABSOLUTE ACTION ADD ADMIN AFTER AGGREGATE ALIAS ALL ALLOCATE ALTER AND ANY ARE ARRAY AS ASC ASSERTION AT AUTHORIZATION
B	BEFORE BEGIN BINARY BIT BLOB BOOLEAN BOTH BREADTH BY
C	CALL CASCADE CASCADED CASE CAST CATALOG CHAR CHARACTER CHECK CLASS CLOB CLOSE COLLATE COLLATION COLUMN COMMIT COMPLETION CONNECT CONNECTION CONSTRAINT CONSTRAINTS CONSTRUCTOR CONTINUE CORRESPONDING CREATE CROSS CUBE CURRENT CURRENT_DATE CURRENT_PATH CURRENT_ROLE CURRENT_TIME CURRENT_TIMESTAMP CURRENT_USER CURSOR CYCLE
D	DATA DATE DAY DEALLOCATE DEC DECIMAL DECLARE DEFAULT DEFERRABLE DEFERRED DELETE DEPTH DEREF DESC DESCRIBE DESCRIPTOR DESTROY DESTRUCTOR DETERMINISTIC DICTIONARY DIAGNOSTICS DISCONNECT DISTINCT DOMAIN DOUBLE DROP DYNAMIC
E	EACH ELSE END END_EXEC EQUALS ESCAPE EVERY EXCEPT EXCEPTION EXEC EXECUTE EXTERNAL
F	FALSE FETCH FIRST FLOAT FOR FOREIGN FOUND FROM FREE FULL FUNCTION
G	GENERAL GET GLOBAL GO GOTO GRANT GROUP GROUPING
H	HAVING HOST HOUR
I	IDENTITY IGNORE IMMEDIATE IN INDICATOR INITIALIZE INITIALLY INNER INOUT INPUT INSERT INT INTEGER INTERSECT INTERVAL INTO IS ISOLATION ITERATE
J	JOIN
K	KEY
L	LANGUAGE LARGE LAST LATERAL LEADING LEFT LESS LEVEL LIKE LIMIT LOCAL LOCALTIME LOCALTIMESTAMP LOCATOR
M	MAP MATCH MINUTE MODIFIES MODIFY MODULE MONTH
N	NAMES NATIONAL NATURAL NCHAR NCLOB NEW NEXT NO NONE NOT NULL NUMERIC
O	OBJECT OF OFF OLD ON ONLY OPEN OPERATION OPTION OR ORDER ORDINALITY OUT OUTER OUTPUT
P	PAD PARAMETER PARAMETERS PARTIAL PATH POSTFIX PRECISION PREFIX PREORDER PREPARE PRESERVE PRIMARY PRIOR PRIVILEGES PROCEDURE PUBLIC
Q	
R	READ READS REAL RECURSIVE REF REFERENCES REFERENCING RELATIVE RESTRICT RESULT RETURN RETURNS REVOKE RIGHT ROLE ROLLBACK ROLLUP ROUTINE ROW ROWS
S	SAVEPOINT SCHEMA SCROLL SCOPE SEARCH SECOND SECTION SELECT SEQUENCE SESSION SESSION_USER SET SETS SIZE SMALLINT SOME SPACE SPECIFIC SPECIFICTYPE SQL SQLEXCEPTION SQLSTATE SQLWARNING START STATE STATEMENT STATIC STRUCTURE SYSTEM_USER
T	TABLE TEMPORARY TERMINATE THAN THEN TIME TIMESTAMP TIMEZONE_HOUR TIMEZONE_MINUTE TO TRAILING TRANSACTION TRANSLATION TREAT TRIGGER TRUE
U	UNDER UNION UNIQUE UNKNOWN UNNEST UPDATE USAGE USER USING
V	VALUE VALUES VARCHAR VARIABLE VARYING VIEW
W	WHEN WHENEVER WHERE WITH WITHOUT WORK WRITE
X	
Y	YEAR
Z	ZONE

#3. ANY、ALL 与 SOME

有三个关键字在子查询中非常好用：ANY、ALL与SOME。它们可以与比较运算符和结果集一起使用。开始解释前，我们先回顾一下第9章讨论过的 IN 运算符：

```
SELECT name, rating FROM restaurant _ ratings
WHERE rating IN
(SELECT rating FROM restaurant _ ratings
WHERE rating > 3 AND rating < 9);
```

restaurant _ ratings

name	rating
Pizza House	3
The Shack	7
Arthur' s	9
Ribs 'n' More	5

子查询返回3与9中间的评价 —— 本例为5、7。

这段查询返回评价与括号中的子查询的结果相等的餐厅的名称。查询结果是 The Shack 与 Rib's More。

使用ALL

现在观察一下这个查询：

```
SELECT name, rating FROM restaurant _ ratings
WHERE rating > ALL
(SELECT rating FROM restaurant _ ratings
WHERE rating > 3 AND rating < 9);
```

大于加上ALL可以找出任何大于集合中最大值的值。

这一次改为查询评价高于子查询结果集的餐厅。查询结果是 Arthur's。

以下是加了 < 的查询：

```
SELECT name, rating FROM restaurant _ ratings
WHERE rating < ALL
(SELECT rating FROM restaurant _ ratings
WHERE rating > 3 AND rating < 9);
```

小于加上ALL可以找出任何小于集合中最小值的值。

上述查询会返回 Pizza House。>= 与 <= 也能与 ALL 一起使用。下面的查询会找出 The Shack 与 Authur' s 两家餐厅。我们得到高于子查询结果集的评价，还有任何等于集合中最大值的评价，也就是 7：

```
SELECT name, rating FROM restaurant _ ratings
WHERE rating >= ALL
(SELECT rating FROM restaurant _ ratings
WHERE rating > 3 AND rating < 9);
```

任何大于集合或等于集合中最高结果集的值都将匹配。

使用ANY

如果集合中的任何值符合条件，则 ANY 估算为 true。以下列查询为例：

```
SELECT name, rating FROM restaurant_ratings
WHERE rating > ANY
(SELECT rating FROM restaurant_ratings WHERE
rating > 3 AND rating < 9);
```

上例可以这么理解：选择评价高于集合（本例为 (5, 7) ）中任何一个值的行。The Shack 的评价为 7（大于 5），符合条件；评价为 9 的 Arthur's 也会被返回。

大于加上ANY可以找出任何大于集合中最小值的值。

小于加上ANY可以找出任何小于集合中最大值的值。

使用SOME

在标准 SQL 语法中，SOME 与 ANY 表示相同的意思，在 MySQL 中也是这样。请检查你喜欢用的RDBMS，确认这个关键字的使用方式。

#4. 再谈数据类型

我们谈过最常见的数据类型，这时会提到一些细节，让列能调整得更加合宜。首先看看没见过的类型，再回头深入了解几位旧识：

BOOLEAN

BOOLEAN（布尔）类型只能存储“true”、“false”或NULL，适用于任何答案只有“是”或“非”的列。RDBMS 实际上用 1 存储“true”，用 0 存储“false”。输入时可用 1 代表“true”，用 0 代表“false”。

INT

本书到处都可见到 INT 的踪影。INT 能存储的**数值范围是 0 到 4294967295**。这是只存储正整数的情况，也称为**无符号整数**（unsigned integer）。

如果想使用正负整数值，则需要采用有符号整数（signed integer），存储范围是 -2147483648 到 2147483647。下列语法可以在创建列时告诉 RDBMS 该采用有符号还是无符号整数：

```
INT(SIGNED)
```

其他整数类型

我们都知道 INT，但还有两种类型能做更好的整数存储运用 —— SMALLINT与 BIGINT。它们都指定了存储的最大数值。

可存储的最大数值根据 RDBMS 的不同而不同，以 MySQL 为例：

	有符号整数	无符号整数
SMALLINT	-32768 ~ 32767	0 ~ 65535
BIGINT	-9223372036854775808 ~ 9223372036854775807	0 ~ 18446744073709551615

MySQL 还进一步细分出下列类型：

	有符号整数	无符号整数
TINYINT	-128 ~ 127	0 ~ 255
MEDIUMINT	-8388608 ~ 8388607	0 ~ 16777215

DATE与TIME类型

回顾一下 MySQL 存储日期与时间数据类型的格式：

```
DATE          YYYY-MM-DD

DATETIME      YYYY-MM-DD HH:MM:SS

TIMESTAMP     YYYYMMDDHHMMSS

TIME          HH:MM:SS
```

some _ dates

a_date
2007-08-25 22:10:00
1925-01-01 02:05:00

选取时间类型的数据时，其实可以调整 RDBMS 返回的格式。每个 RDBMS 中负责处理时间格式的函数多少有点不一样。本书以 MySQL 的 DATE_FORMAT() 为例。

格式字符串必须加上引号。

假设有个名为 a _ date 的列：

```
SELECT DATE_FORMAT(a_date, '%M %Y') FROM some_dates;
```

%M 与 %Y 负责把要求的格式告知函数，下表即为返回结果：

a_date
August 2007
January 1925

提供时间格式字符串的完整列表会占用太多篇幅，但有了这么多丰富的格式字符串，我们就能取得实际需要的时间列，且不用看到不需看到的信息。

#5. 临时表

我们已经创建了很多表。每次创建一张表，RDBMS都存储表的结构。每次插入数据至表，也就存储了数据。表和其中的数据都会被存储。如果在终端窗口或图形界面中退出SQLsession，其中的表和数据仍然存在。除非被删除，不然数据都会存在，表则是在被卸除前都会存在。

需要临时表的可能理由

- 可用于保存中间结果 —— 例如对某列执行数学运算，但运算结果不会在现在的session 中用到，而是在下一个 session 中使用。
- 捕获某个时刻的表内容。
- 还记得我们把 Greg's List 从一个表扩展成多个表的工作吗？此时就可创建临时性的表来帮助你重新整理数据的结构，又能确保在 session 结束后临时性的表就会消失。
- 如果最后结合了 SQL 与其他编程语言，可在收集数据时创建临时表，然后把最终结果存储在永久性表中。

创建临时表

MySQL创建临时表的语法很简单，加上关键字TEMPORARY：

```
CREATE TEMPORARY TABLE my_temp_table
(
    some_id INT,
    some_data VARCHAR(50)
)
```

TEMPORARY是唯一需要加入的东西。

临时表的速记法

如下的查询可用来创建临时表：

```
CREATE TEMPORARY TABLE my_temp_table AS
SELECT * FROM my_permanent_table;
```

这一行是任何想放在 AS 后的查询。

注意！

创建临时表的语法会根据 RDBMS 的不同而不同。

请查看 RDBMS 的说明文档来了解这个功能。

#6. 转换数据类型

有时候列采用某种类型，但我们希望查询得到的结果是另一种类型。SQL的CAST()函数可以转换数据类型。

语法是：

```
CAST(your _ column, TYPE)
```

TYPE 可从下列类型中选择：

```
CHAR()
DATE
DATETIME
DECIMAL
SIGNED [INTEGER]
TIME
UNSIGNED [INTEGER]
```

可能想用CAST()的一些情况

把字符串格式的时间值转换为 DATE 类型：

```
SELECT CAST('2005-01-01' AS DATE);
```

字符串 "2005-01-01" 会转换为 DATE 格式。

把整数转换为浮点数：

```
SELECT CAST(2 AS DECIMAL);
```

整数2变为浮点数2.00。

其他可以运用 CAST() 的地方还包括列在 INSERT 语句中的值以及 SELECT 选取的列的列表。

不能使用 CAST() 的场合

* 从浮点数转换为整数。

* 从 TIME、DATE、DATETIME、CHAR 转换为 DECIMAL 或 INTEGER。

#7. 你是谁？现在几点？

有时候我们可能在RDBMS上不只拥有一个用户账号，每个账号各有不同权限与角色。如果想确认当前使用的账号，就让这个命令告诉你：

```
SELECT CURRENT_USER;
```

这个命令还会列出你使用的机器。如果RDBMS位于的机器与你使用的机器为同一台，而且你又使用根用户的账号，就会看到：

```
root@localhost
```

下列命令则可取得当前的日期与时间：

```
File Edit Window Help
> SELECT CURRENT_DATE;
+--------------+
| CURRENT_DATE |
+--------------+
| 2007-07-26   |
+--------------+
1 row in set (0.00 sec)
```

```
File Edit Window Help
> SELECT CURRENT_TIME;
+--------------+
| CURRENT_TIME |
+--------------+
| 11:26:48     |
+--------------+
1 row in set (0.00 sec)
```

```
File Edit Window Help
SELECT CURRENT_USER;
+----------------+
| CURRENT_USER   |
+----------------+
| root@localhost |
+----------------+
1 row in set (0.00 sec)
```

#8. 有用的数字函数

下面简单列出一些能够处理数字数据类型的函数。有些我们已经提过了：

数字函数	功能说明	
ABS(x)	返回 x 的绝对值	
	查询	结果
	SELECT ABS(-23);	23
ACOS(x)	返回 x 的反余弦值	
	SELECT ACOS(0);	1.5707963267949
ASIN()	返回 x 与 y 的反正弦值	
	SELECT ASIN(0.1);	0.10016742116156
ATAN(x,y)	返回 x 与 y 的反正切值	
	SELECT ATAN(-2,2);	-0.78539816339745
CEIL(x)	返回大于等于 x 的最小整数。返回值为 BIGINT	
	SELECT CEIL(1.32);	2
COS(x)	返回 x 的余弦值，以弧度计算	
	SELECT COS(1);	0.54030230586814
COT(x)	返回 x 的余切值	
	SELECT COT(12);	-1.5726734063977
EXP(x)	返回 e 的 x 次方	
	SELECT EXP(-2);	0.13533528323661
FLOOR(x)	返回小于等于 x 的最大整数	
	SELECT FLOOR(1.32);	1
FORMAT(x,y)	转换 x 为文本字符串并四舍五入至 y 指定的位数。	
	SELECT FORMAT(3452100.50,2);	3,452,100.50
LN(x)	返回 x 的自然对数	
	SELECT LN(2);	0.69314718055995
LOG(x) 与 LOG(x,y)	返回 x 的自然对数，若有两个参数，则以 x 为基数，返回 y 的对数	
	SELECT LOG(2);	0.69314718055995
	SELECT LOG(2,65536);	16

未完待续

#8. 有用的数字函数（续）

数字函数	功能说明	
MOD(x,y)	返回 x 除以 y 的余数	
	查询	结果
	SELECT MOD(249,10);	9
PI()	返回 pi	
	SELECT PI();	3.141593
POWER(x,y)	返回 x 的 y 次方值	
	SELECT POW(3,2);	9
RADIANS(x)	返回 x 从角度转换成弧度的值	
	SELECT RADIANS(45);	0.78539816339745
RAND()	返回随机浮点数	
	SELECT RAND();	0.84655920681223
ROUND(x)	返回 x 四舍五入后最接近的整数	
	SELECT ROUND(1.34);	1
	SELECT ROUND(-1.34);	-1
ROUND(x,y)	以 y 指定的小数位数对 x 四舍五入	
	SELECT ROUND(1.465, 1);	1.5
	SELECT ROUND(1.465, 0);	1
	SELECT ROUND(28.367, -1);	30
SIGN(x)	当 x 是正数时，返回 1；x 是 0 时，返回 0；x 是负数时，返回 -1	
	SELECT SIGN(-23);	-1
SIN(x)	返回 x 的正弦值	
	SELECT SIN(PI());	1.2246063538224e-16
SQRT(x)	返回 x 的平方根	
	SELECT SQRT(100);	10
TAN(x)	返回 x 的正切值	
	SELECT TAN(PI());	-1.2246063538224e-16
TRUNCATE(x,y)	返回 x 截断至 y 指定的小数位数后的值	
	SELECT TRUNCATE(8.923,1);	8.9

#9. 索引能加快速度

各位已经知道主键与外键索引了。这些索引很适合连接多张表及强化数据的完整性。也可以对列创建索引来加快查询的速度。

当 WHERE 搜索没有索引的列时，RDBMS 需要从列起始处着手，逐一读取数据。如果表很大，例如有 400 万行，查询肯定会占用很可观的时间。

对列创建索引时，RDBMS 会持有列的有关额外信息，这样可以加快搜索速度。额外信息存储在幕后的某个表中，具有特殊顺序，所以 RDBMS 搜索幕后表的速度比较快。代价则是索引会占用空间。所以要考虑创建索引的对象必须创建在常用的列上，至于不常用的列就省略吧。

下例是为列添加索引的 ALTER TABLE 命令：

```
ALTER TABLE my _ contacts
ADD INDEX (last _ name);
```

索引的理论不只这些，不过我们已大致包含了基本的概念。

#10. 给我两分钟，我给你 PHP/MySQL

结束“十大遗珠”前，我想很快地提一下 PHP 与 MySQL 的交互方式，以及它们如何协助你把数据放置到 Web 上。我只能浅尝辄止，有兴趣的读者可以另外研究这方面的相关书籍。

接下来的范例假设大家都熟悉 PHP 语言，而且也已娴熟 SQL 查询的设计。范例代码打开对 `gregs_list` 数据库的连接，并从 `my _ contacts` 表中选出所有联络人的姓名。PHP 代码接受所有查询结果并存储于数组中。最后一部分则以网页的形式呈现姓名列表。

```
<?php
$conn = mysql_connect("localhost","greg","gr3gzpAs");
if (!$conn)
  {
    die('Did not connect: ' . mysql_error());
  }
mysql_select_db("my_db", $conn);

$result = mysql_query("SELECT first_name, last_name FROM my_contacts");

while($row = mysql_fetch_array($result))
  {
    echo $row['first_name'] . " " . $row['last_name'];
    echo "<br />";
  }

mysql_close($conn);
?>
```

我们把文件存储为 `gregsnames.php`，放在 Web 服务器上。

仔细观察每一行

```
<?php
```

第一行告诉 Web 服务器：接下来是 PHP 代码。

```
$conn = mysql _ connect("localhost","greg","gr3gzpAs");
```

与 gregs _ list 连接前，必须让 Web 服务器知道 RDBMS 的位置、我们的数据库账号与密码，然后以这些信息创建连接字符串并把字符串命名为 $conn。PHP 函数 mysql _ connect() 取用前述信息后，开始尝试与 RDBMS 通信。

```
if (!$conn)
  {
    die('Did not connect: ' . mysql _ error());
  }
```

如果通信不成功，PHP 会送回 RDBMS 连接失败的原因，然后 PHP 会暂停处理过程。

```
mysql _ select _ db("gregs _ list", $conn);
```

对 RDBMS 的连接成功了。现在必须指示 PHP 使用某个数据库，本例使用我们的最爱：gregs _ list。

```
$result = mysql _ query("SELECT first _ name, last _ name FROM my _ contacts");
```

现在已经建立了RDBMS的连接，也选好了要操作的数据库，但还没有查询。我们设计一个查询并使用 mysql _ query()函数传递查询给 RDBMS。所有返回的查询结果行都会保存在名为 $result 的数组中。

```
while($row = mysql _ fetch _ array($result))
  {
```

现在使用 PHP从 $result 中取出所有数据行并以网页的形式呈现。这部分需要 while 循环，每次处理一行，直到所有记录处理完毕为止。

```
    echo $row['first _ name'] . " " . $row['last _ name'];
    echo "<br />";
  }
```

接下来的两个 echo（PHP 语句）负责把每笔姓名记录编排成网页上的一行文字。
（HTML 标签）可以制造分行的效果。

```
lose($conn);
```

所有姓名输出完成后，应该关闭 RDBMS 连接，就像退出终端窗口那样。

```
?>
```

最后， PHP 脚本到此结束。

附录 2　安装 MySQL

自己动手试

各位学到这么多SQL技巧，如果没有施展的场所，岂非英雄无用武之地？本篇附录说明了安装MySQL的方式，让大家有地方自我琢磨技艺。

开始了，冲吧！

因为只有讲 SQL 的书却没有可以动手实践 SQL 的地方很无聊，所以本章就要介绍 Windows 版与 Mac OS X 版的 MySQL 安装方式。

注意事项：本章内容适用于Windows 2000、Windows XP、Windows Server 2003及其他32位的Windows操作系统。Mac的部分则适用于Mac OS X 10.3.x或更新的版本。

我们会引领大家下载并安装 MySQL。目前，MySQL RDBMS服务器免费版的正式名称是“MySQL Community Server”。

安装说明与疑难排除

接下来会列出在 Windows 和 Mac OS X 系统上安装 MySQL 的步骤，但我们并不打算取代 MySQL 网站上的详细说明，而且希望大家都能上网看看 MySQL 的说明。若需更详细的安装说明与故障解决，请去这里：

可取得 5.0 或更新版本。

```
http://dev.mysql.com/doc/refman/5.0/en/windows-installation.html
```

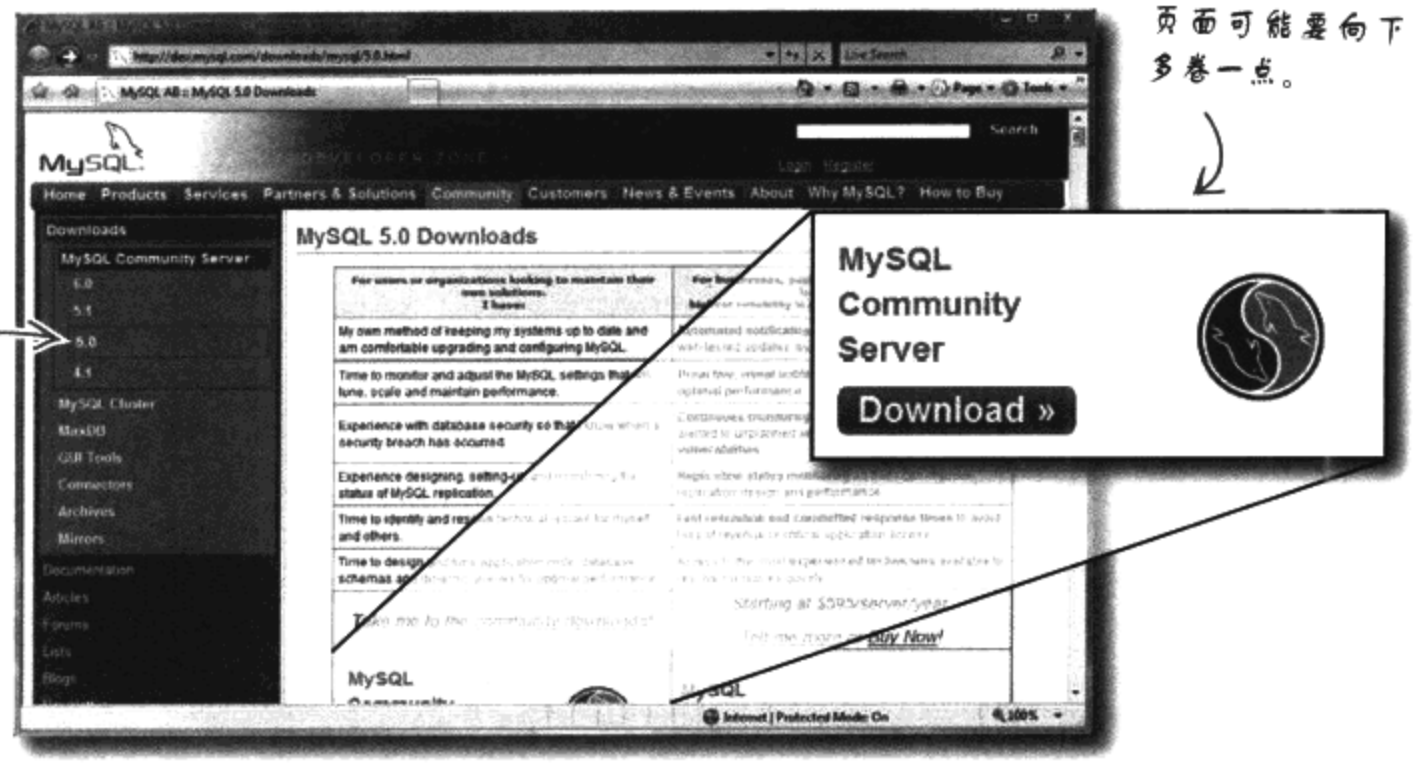
大家或许也会想安装我们在第 526~527 页上提到的 MySQL Query Browser。把查询输入到这个软件中，即可在软件界面上看到查询结果，而不需忍受宽度有限的文本界面。

在 Windows 上安装 MySQL

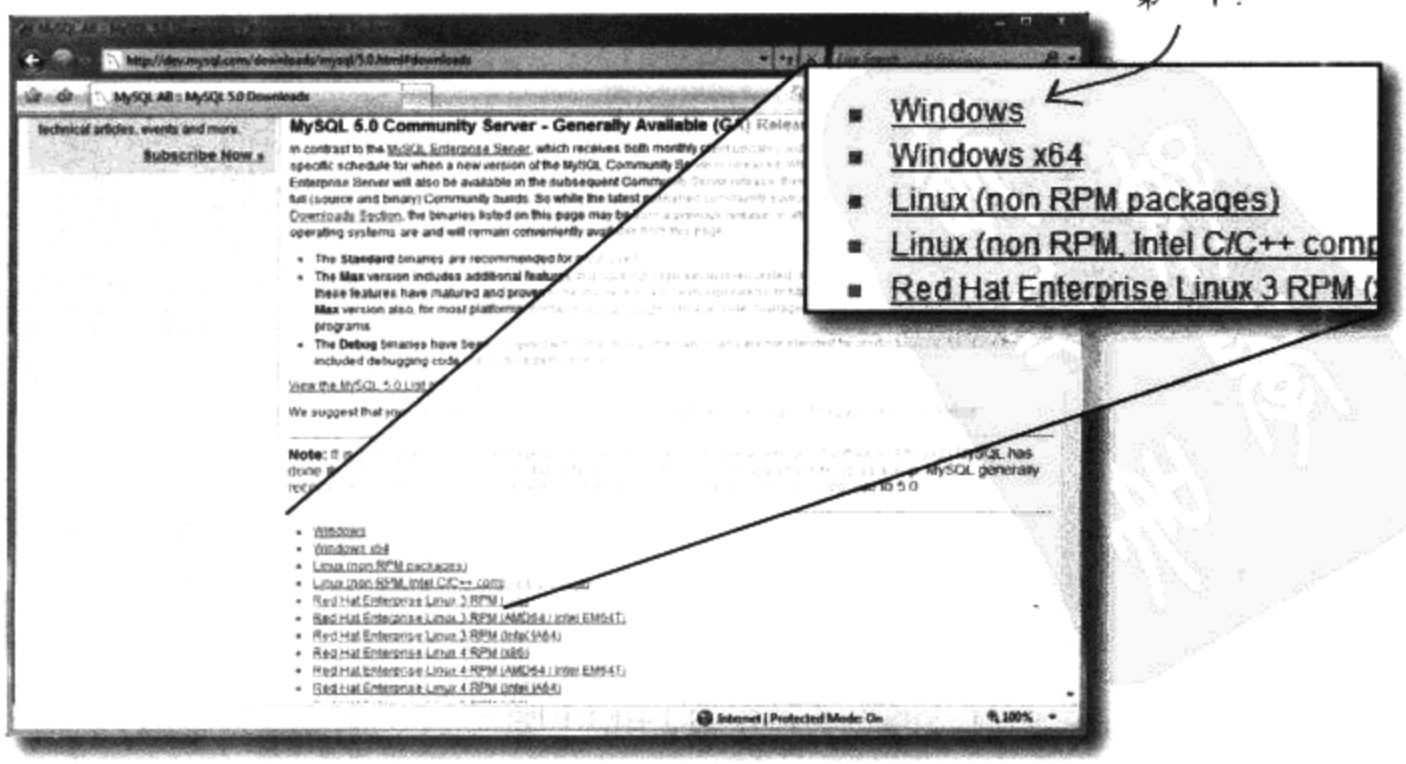
❶ 请访问：

```
http://dev.mysql.com/downloads/mysql/5.0.html
```

并点击“MySQL Community Server”的下载按钮。

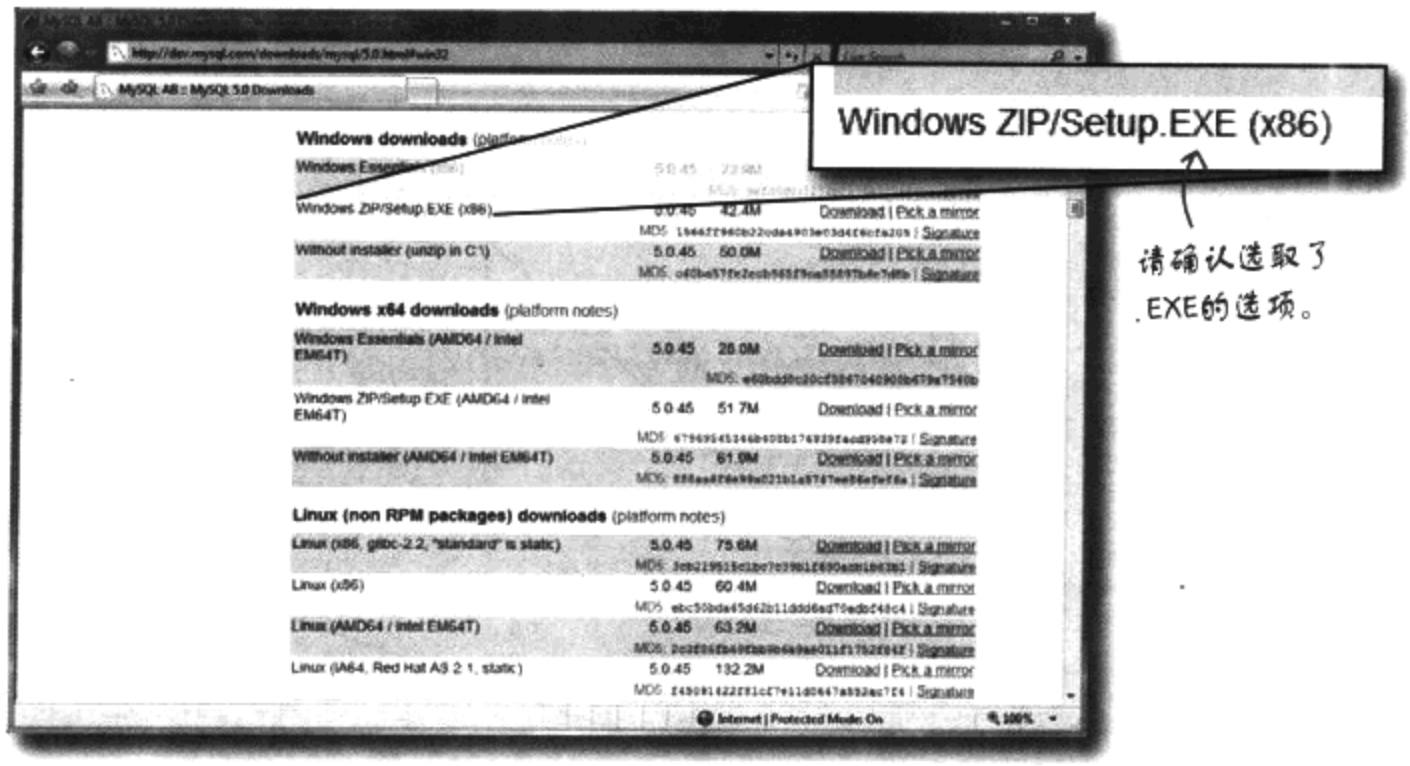


❷ 选择列表中的 Windows。



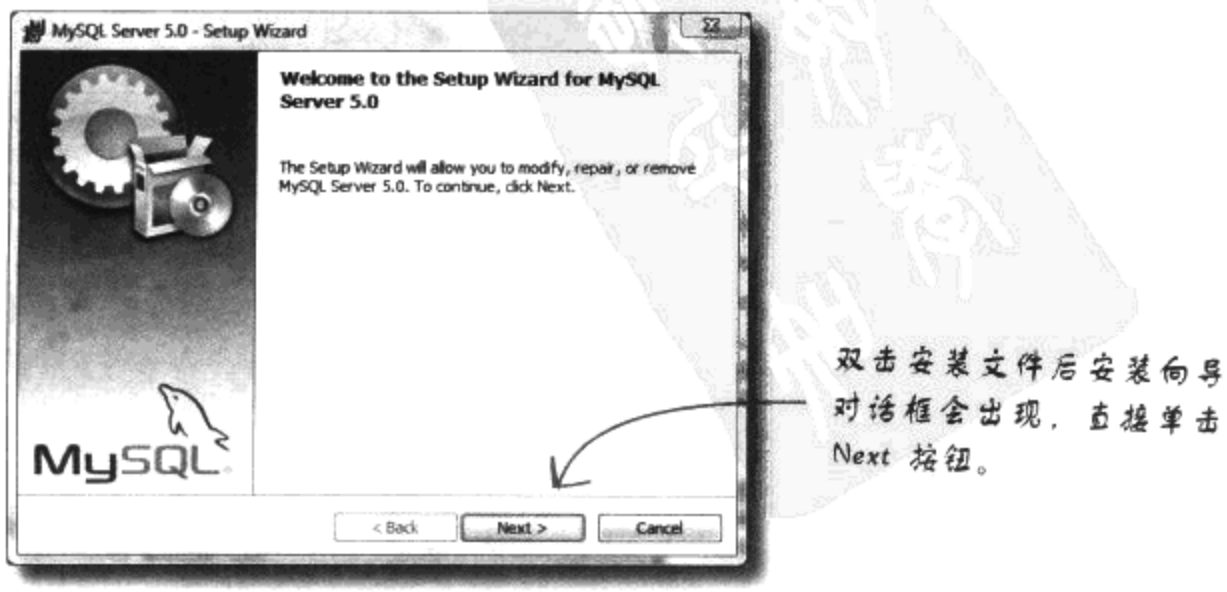
下载安装程序

❸ 在 Windows downloads 列出的选择中，我们建议使用 `Windows ZIP/Setup.EXE`，因为其中包含了大幅简化过程的安装程序。接着点击 Pick a Mirror。



❹ 画面上列出可供选择的下载地址，请选择最靠近你的位置。

❺ 下载结束后双击文件开始安装程序。接下来，随着 Setup Wizard（安装向导）一起走过安装过程。单击 Next 按钮。



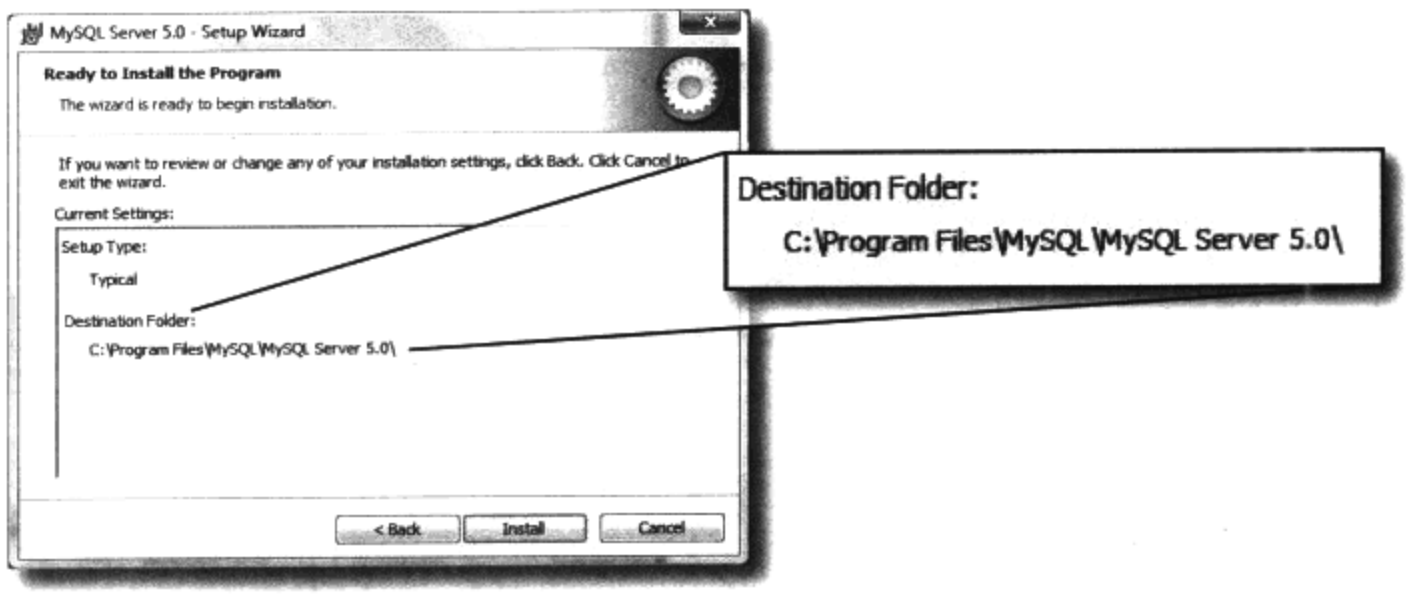
选择安装目录

❻ 接下来的画面要求我们选择安装方式，共有Typical、Complete、Custom可选。就本书的需求而言，请选择 Typical。

可以改变 MySQL 的安装目录，但我们先采用默认目录就好了：

```
C:\Program Files\MySQL\MySQL Server 5.0
```

单击 Next 按钮。



单击“Install”，大功告成！

❼ 然后你会看到“Reday to Install”的对话框，框内附有安装目录的位置。若接受安装目录，单击“Install”按钮；否则，请单击 Back，修改目录后再次回到此对话框。

最后，按下Install按钮。

在 Mac OS X 上安装 MySQL

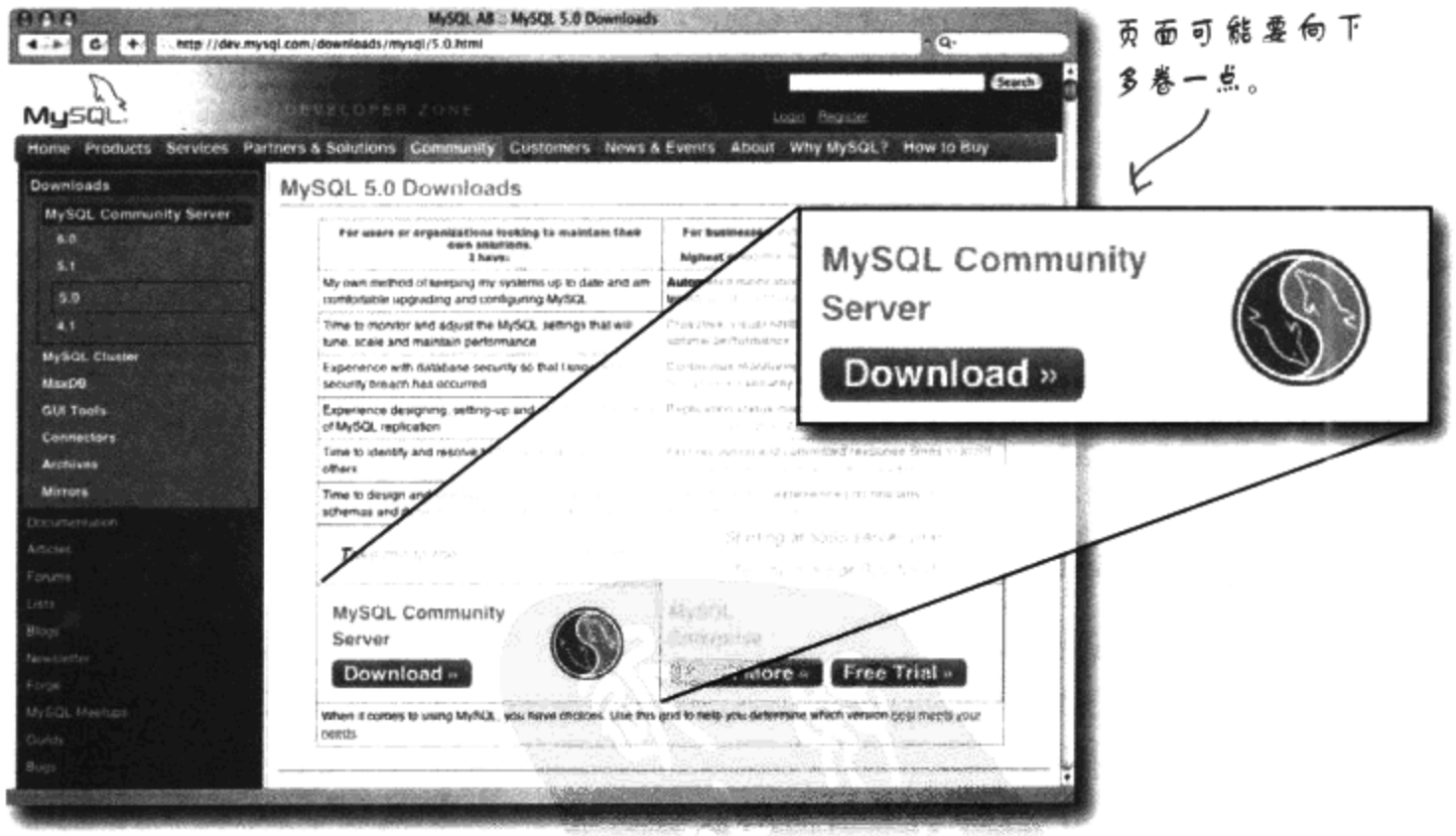
如果运行在 Mac OS X 服务器上，MySQL 应该已经安装好了。

不过在开始任何工作前，还是先检查你是否安装了任何版本。请至 **Applications/Server/MySQL Manager** 确认。

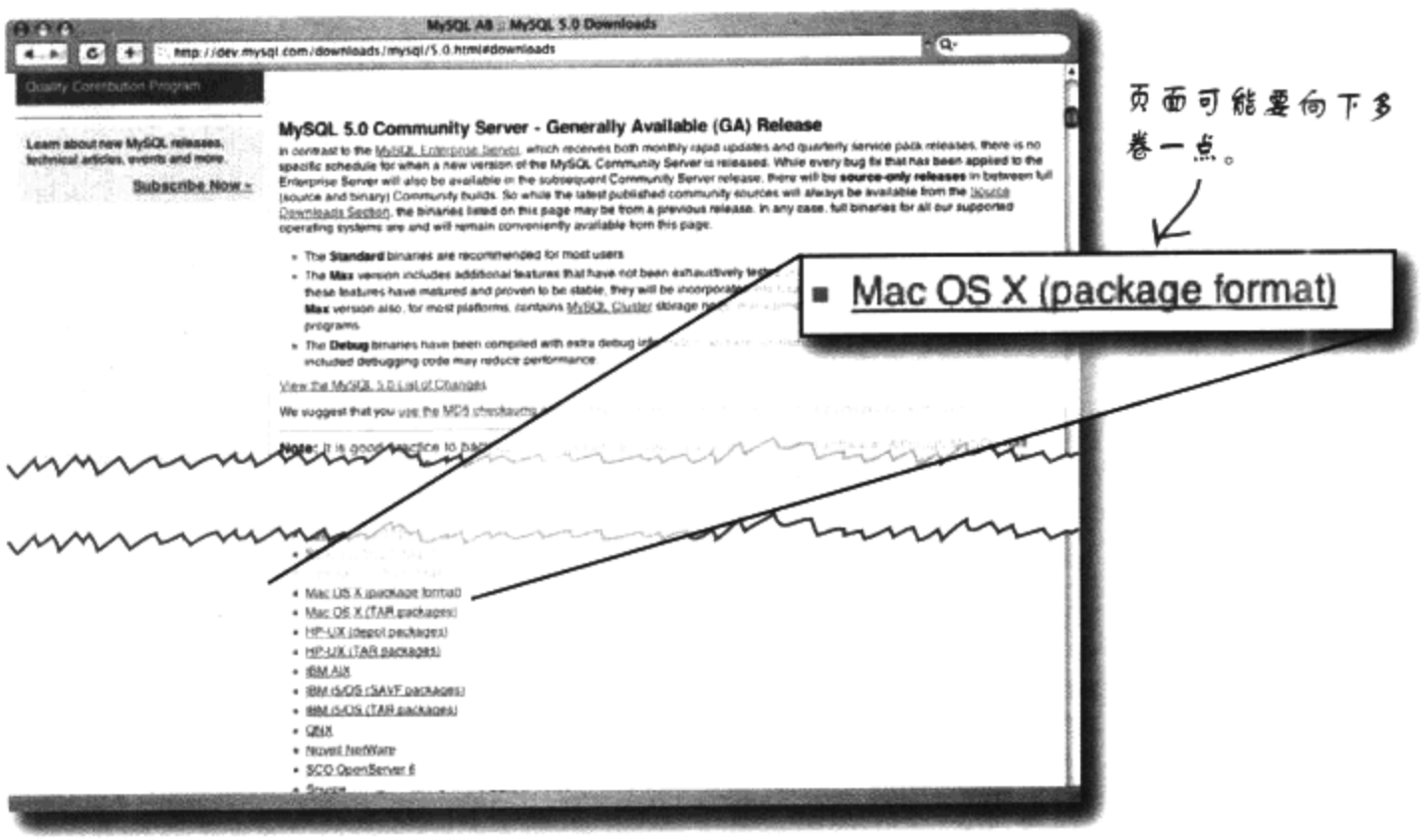
❶ 请访问：

http://dev.mysql.com/downloads/mysql/5.0.html

并单击“MySQL Community Server”的下载按钮。



❷ 选择列表中的 Mac OS X(package format)。



❸ 选择适合的 Mac OS X 版本包，接着单击 Pick a Mirror。

❹ 画面上列出可供选择的下载地址，请选择最靠近你的位置。

❺ 文件下载结束后双击文件开始安装程序。如果系统中已安装了MySQL，请参考在线说明文档，然后使用第 526 ～527 页上的 Query Browser来访问你已安装的版本。

但如果你赶时间，也有使用 Terminal 的快速检查方法。

现在可以打开 Terminal 窗口并输入：

```
shell> cd /usr/local/mysql
shell> sudo ./bin/mysqld_safe
```

（若需输入密码，请记得输入）

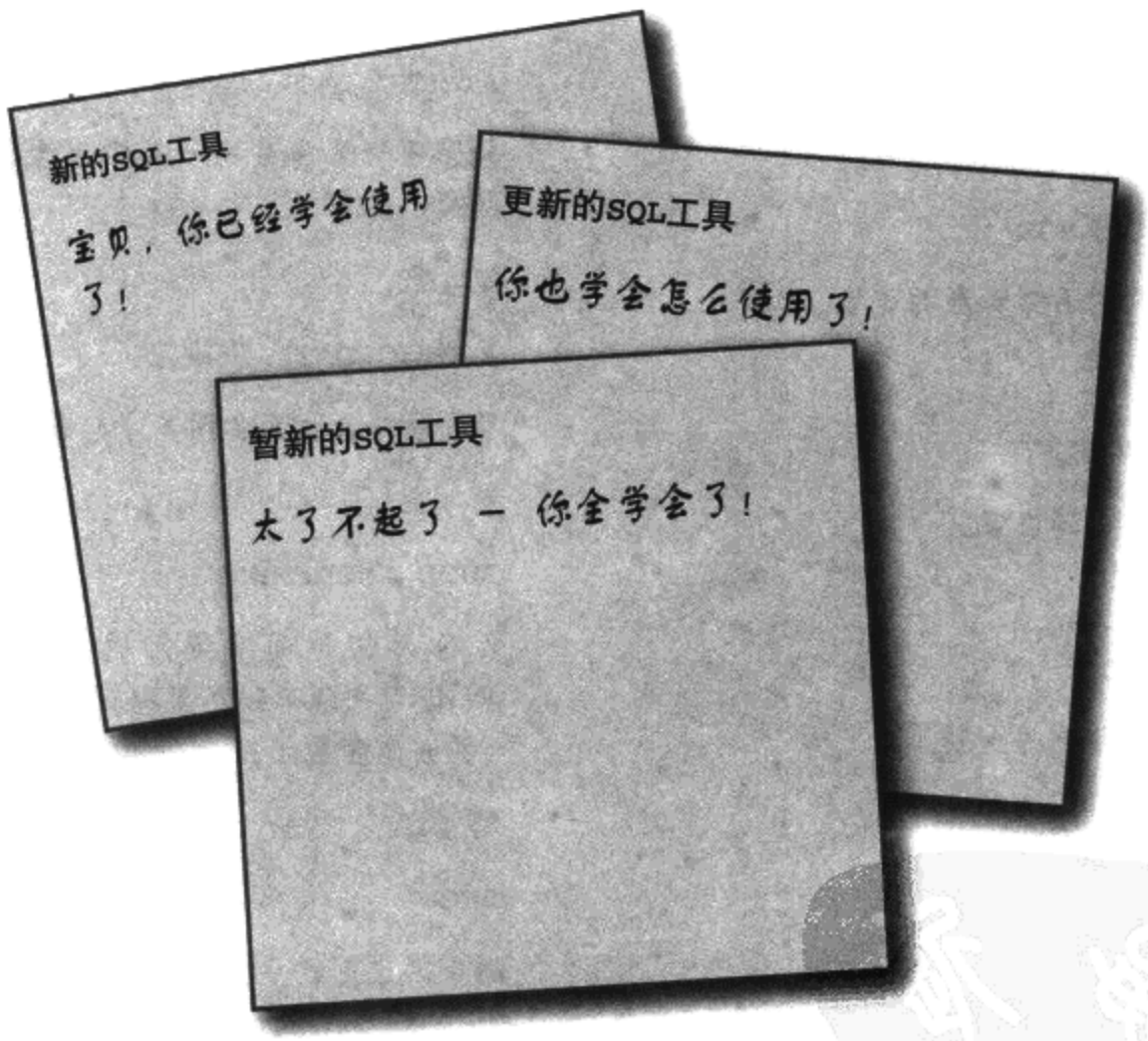
（按下 Ctrl+Z）

```
shell> bg
```

（按下 Ctrl+D或输入 exit 以退出 shell 窗口）

附录 3 SQL 工具总整理

崭新的SQL工具包



所有 SQL 工具初次齐聚一堂，仅限今夜哦！（**开玩笑啦！**）这篇附录收集了所有我们提到的 SQL 工具，花一点时间浏览一下，感受一下那份**成就感——你已经完全学会这些工具了哦！**

C

CHECK CONSTRAINTS

检查约束。可以只让特定值插入或更新至表里。

第 11 章

CHECK OPTION

创建可更新视图时，使用这个关键字强迫所有插入与更新的数据都需满足视图里的 WHERE 条件。

第 11 章

COMMA JOIN

与CROSS JOIN几乎相同，只不过以逗号取代关键字 CROSS JOIN。

第 8 章

Composite key

组合键。由多个列构成的主键，这些列需列成唯一的键值。

第 7 章

COUNT

我们不需看到记录，就能知道有多少条记录符合SELECT查询。COUNT 只返回一个整数值。

第 6 章

CREATE TABLE

开始设置你的表，但还需要知道COLUMN NAMES和 DATA TYPES —— 可通过分析要存入表的数据种类而得知。

第 1 章

CREATE TABLE AS

使用本命令从任何 SELECT 语句的查询结果创建表。

第 10 章

CREATE USER

有些 RDBMS 使用这个语句创建用户并设定其密码。

第 12 章

CROSS JOIN

交叉联接。返回一张表的每一行与另一张表的每一行所有可能的搭配结果。其他常见名称还有*Cartesian Join* 与 *No Join*。

第 8 章

D

DELETE

这是删除表中记录的工具。它和WHERE子句一起使用，可精确地瞄准要删除的行。

第 3 章

DISTINCT

不同的值只会返回一次，返回的结果中没有重复值。

第 6 章

DROP TABLE

用于删除出错的表，但最好在使用任何INSERT语句向表中插入数据前删除表。

第 1 章

E

EQUIJOIN 与 NON-EQUIJOIN

相等联接与不等联接。两得的为内联接的一种。相等联接返回相等的行，不等联接则返回不相等的行。

第 8 章

使用 ' 与 \转义

字符串中的单引号前应该加上另一个单引号或反斜线来把它转换成直接量。

第 2 章

EXCEPT

使用这个关键字返回出现在第一个查询中但不在第二个查询中的值。

第 10 章

F

FIRST NORMAL FORM (1NF)

第一范式。每个数据行的需包含原子性数据值，而且每个数据行的需具有唯一的识别方法。

第 4 章

Foreign Key

外键。引用其他表的主键的列。

第 7 章

G

GRANT

根据授予用户的权限，精确控制用户对数据库的操作范围。

第 12 章

GROUP BY

根据共用列把行分成多个组。

第 6 章

I

INNER JOIN

内联接。任何使用条件结合来自两张表的记录的联接方式。

第 8 章

Inner query

内层查询。在查询内的查询，也称为 *subquery*。

第 9 章

INTERSECT

使用这个关键字返回同时存在于第一个与第二个查询中的值。

第 10 章

IS NULL

可用于创建检查麻烦的 NULL 值的条件。

第 2 章

L

LEFT OUTER JOIN

左外联接。LEFT OUTER JOIN 接受左表中的所有记录，并从右表比对出相符记录。

第 10 章

LIKE搭配%与_

使用LIKE搭配通配符，可搜索部分文本字符串。

第 2 章

LIMIT

可以明确指定返回记录的数量，以及从哪一条记录开始返回。

第 6 章

M

Many-to-Many

多对多关系。两个通过 *junction table* 连接的表，让一张表中的多行记录能与另一张表中的多行记录相关联，反之亦然。

第 7 章

MAX 与 MIN

MAX 返回列中的最大值；MIN 返回列中的最小值。

第 6 章

N

NATURAL JOIN

自然联接。不使用“ON”子句的内联接。只有在联接的两张表中有同名列时才能顺利运作。

第 8 章

Noncorrelated Subquery

非关联子查询。一个独立而且不引用*outer query*的任何部分的*subquery*。

第 9 章

NON-UPDATABLE VIEWS

无法对底层表执行INSERT或UPDATE操作的视图。

第 11 章

NOT

NOT反转查询结果，取得相反的值。

第 2 章

NULL 与 NOT NULL

你也需要知道哪些列不应该接受NULL值，才能帮助你整理与搜索数据。当你创建表时需要设置列为NOT NULL。

第 1 章

O

One-to-Many

一对多关系。一张表中的一行记录可能与另一张表中的多行记录相关联，但后一张表中的任一行记录只会与前一张表中的一行记录相关联。

第 7 章

One-to-One

一对一关系。父表中的一行记录只与子表中的一行记录相关联。

第 7 章

ORDER BY

根据指定的列，按字母顺序排列查询结果。

第 6 章

Outer Query

外层查询。包含*inner query / subquery*的查询。

第 9 章

P

PRIMARY KEY

主键。一个或一组能识别出唯一数据行的列。

第 4 章

RIGHT OUTER JOIN

右外联接。RIGHT OUTER JOIN 接受右表中的所有记录，并从左表比对出相符记录。

第 10 章

S

Schema

数据库模式。描述数据库中的数据、其他相关对象以及这些对象相互连接的方式。

第 7 章

Second Normal Form (2NF)

第二范式。表必须先符合 1NF，同时不可包含部分函数依赖，才算满足 2NF。

第 7 章

SELECT *

用于选择表中的所有列。

第 2 章

SELF-JOIN

自联接。SELF-JOIN能用一张表做出联接两张完全相同表的效果。

第 10 章

SELF-REFERENCING FOREIGN KEY

自引用外键。这种外键就是同一张表的主键，作为其他用途。

第 10 章

SET

这个关键字属于 UPDATE 语句，可用于改变现有列的值。

第 3 章

SHOW CREATE TABLE

使用这个命令来呈现创建现有表的正确语法。

第 4 章

String functions

字符串函数。这些函数可修改字符串列的内容副本并以查询结果的形式返回。同时，原始数据不会改变。

第 5 章

Subquery

子查询，被另一个查询包围的查询，也称为*inner query*。

第 9 章

SUM

把数值列中的数据加总。

第 6 章

T

Third Normal Form (3NF)

第三范式。表必须先符合2NF，同时不可包含传递函数依赖。

第 7 章

Transitive functional dependency

传递函数依赖。指任何非键列依赖于另一个非键列。

第 7 章

U

UNION与UNION ALL

UNION（联合）根据SELECT指定的列合并两个或多个查询的结果为一张表。UNION默认为隐藏重复的值，UNION ALL则可包含重复的值。

第 10 章

UPDATABLE VIEWS

可更新表。有些视图能用于改变它底层的实际表。这类视图必须包含底层表的所有NOT NULL列。

第 11 章

UPDATE

这条语句以新值更新现有的一列或多列，它也可以使用WHERE子句。

第 3 章

USE DATABASE

带我们进入数据库以设置需要的表。

第 1 章

V

VIEWS

视图。使用视图把查询结果当成表。很适合简化复杂查询。

第 11 章

W

WITH GRANT OPTION

让用户把自己获得的权限再授予其他人。

第 12 章