A Brain-Friendly Guide

Head First SQL

Draft Version

Load important concepts directly into your brain

Avoid embarrassing mistakes

Learn what matters, when it matters

A learner's companion to database programming using SQL

Master out of this world concepts

Bend your mind around dozens of puzzles and exercises

O'REILLY®

Lynn Beighley

Head First SQL

by Lynn Beighley

Copyright © 2007 O'Reilly Media, Inc. All rights reserved.

Printed in the United States of America.

Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

O'Reilly Media books may be purchased for educational, business, or sales promotional use. Online editions are also available for most titles (*safari.oreilly.com*). For more information, contact our corporate/institutional sales department: (800) 998-9938 or *corporate@oreilly.com*.

Printing History:

August 2007: First Edition.

The O'Reilly logo is a registered trademark of O'Reilly Media, Inc. The *Head First* series designations, *Head First SQL*, and related trade dress are trademarks of O'Reilly Media, Inc.

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and O'Reilly Media, Inc., was aware of a trademark claim, the designations have been printed in caps or initial caps.

While every precaution has been taken in the preparation of this book, the publisher and the authors assume no responsibility for errors or omissions, or for damages resulting from the use of the information contained herein.

ISBN-10: 0-596-52684-9

ISBN-13: 978-0-596-52684-9

[M]

Draft Version

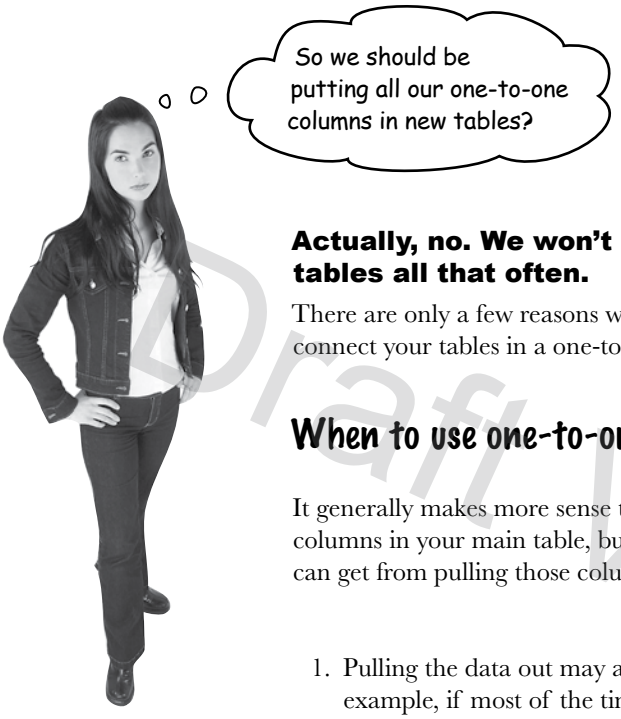
7 multi-table database design

Outgrowing your table

Draft

Sometimes your single table isn't big enough anymore. Your need for data has grown, and that **one table** you've been using just **isn't cutting it**. Your `SELECTs` are getting ***messy*** and ***harder to write***. You've gone as far as you can go. It's a big world out there, and sometimes you need **more than one table** to contain your data, control it, and ultimately, be the master of your own database.

Patterns of data: when to use one-to-one tables



Actually, no. We won't use one-to-one tables all that often.

There are only a few reasons why you might connect your tables in a one-to-one relationship.

When to use one-to-one tables

It generally makes more sense to leave those rare one-to-one columns in your main table, but there are a few advantages you can get from pulling those columns out at times:

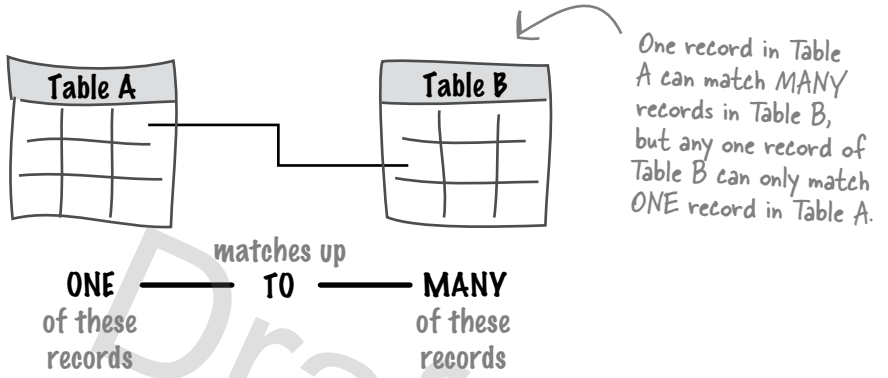
1. Pulling the data out may allow you to write faster queries. For example, if most of the time you needed to query the SSN and not much else, you could query just the smaller table.
2. If you have a column containing values you don't yet know, you can isolate it and avoid NULL values in your main table.
3. You may wish to make some of your data less accessible. Isolating it can allow you to restrict access to it. For example, if you have a table of employees, you might want to keep their salary information out of the main table.

One-to-One: a single table, or (sometimes) two tables related with primary and foreign keys.

Draft Version

Patterns of data: one-to-many

One-to-many means that a record in Table A can have **many** matching records in Table B, but a record in Table B can only match **one** record in Table A.



One-to-Many: split the data into two tables related with primary and foreign keys.

Draft Version

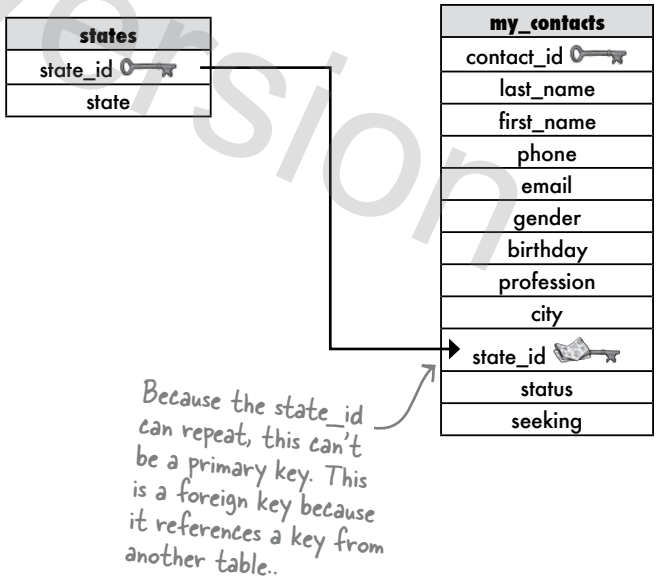
The `state` column in `my_contacts` is a good example of a one-to-many relationship. Each person has only one state in the state column for his address, but more than one person in `my_contacts` may live in any given state.

In this example, we've moved the `state` column to a new child table, and changed the `state` column in the parent table to a foreign key, the `state_id` column. Since it's a one-to-many relationship, we can use the `state_id` in both tables to allow us to connect them.

The connecting line has a ***black arrow*** at the end to show that we're linking **one** thing **to many** things.

Each row in the `states` table can have many matching rows in `my_contacts`, but each row in `my_contacts` has only one matching row in the `states` table.

For example, the `state_id` for California may show up more than once in `my_contacts`, but each person in `my_contacts` will only have one `state_id`.

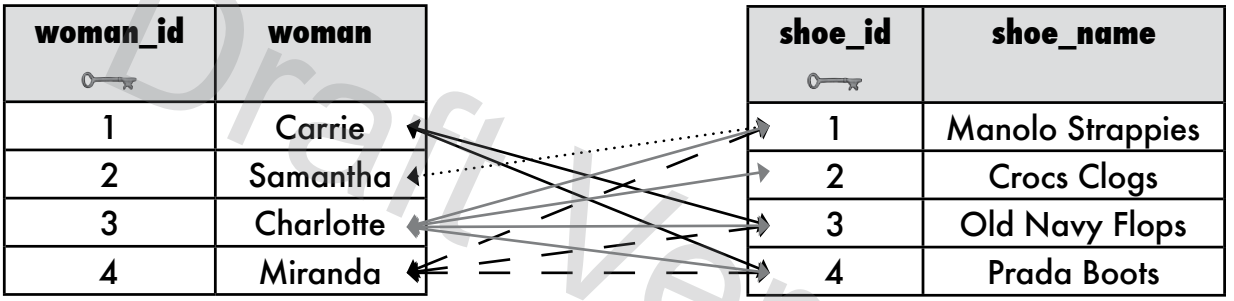


Patterns of data: getting to many-to-many

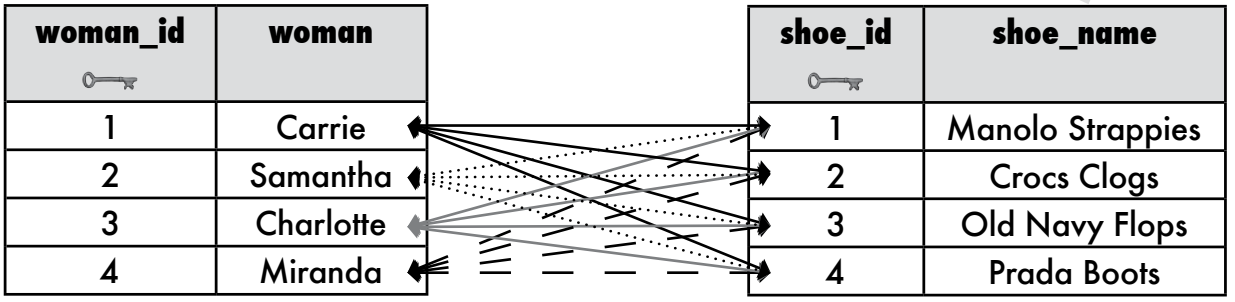
Many women own **many** pairs of shoes. If we created a table containing women and another table containing shoes to keep track of them all, we'd need to link many records to many records since more than one woman can own a particular make of shoe.

Suppose Carrie and Miranda buy both the Old Navy Flops and Prada boots, and Samantha and Miranda both have the Manolo Strappies, and Charlotte has one of each. Here's how the links between the `women` and `shoes` tables would look.

Draft Version



Imagine they loved the shoes so much, the women all bought a pair of the shoes they didn't already own. Here's how the links from women to each shoe names would look then.



Can you say: "Duplicate records"? How can we fix the tables without putting more than one value in a column and winding up like Greg did with his queries for Regis?