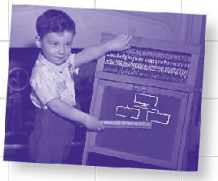
A Brain-Friendly Guide

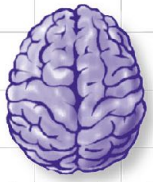
Head First SQL

Help Greg improve his data relationships

Stop misplacing your primary and foreign keys



Finally be able to explain what's normal



Load important SQL query concepts directly into your brain

Avoid embarrassing ALTER scenarios

Put your SQL knowledge to the test with dozens of exercises

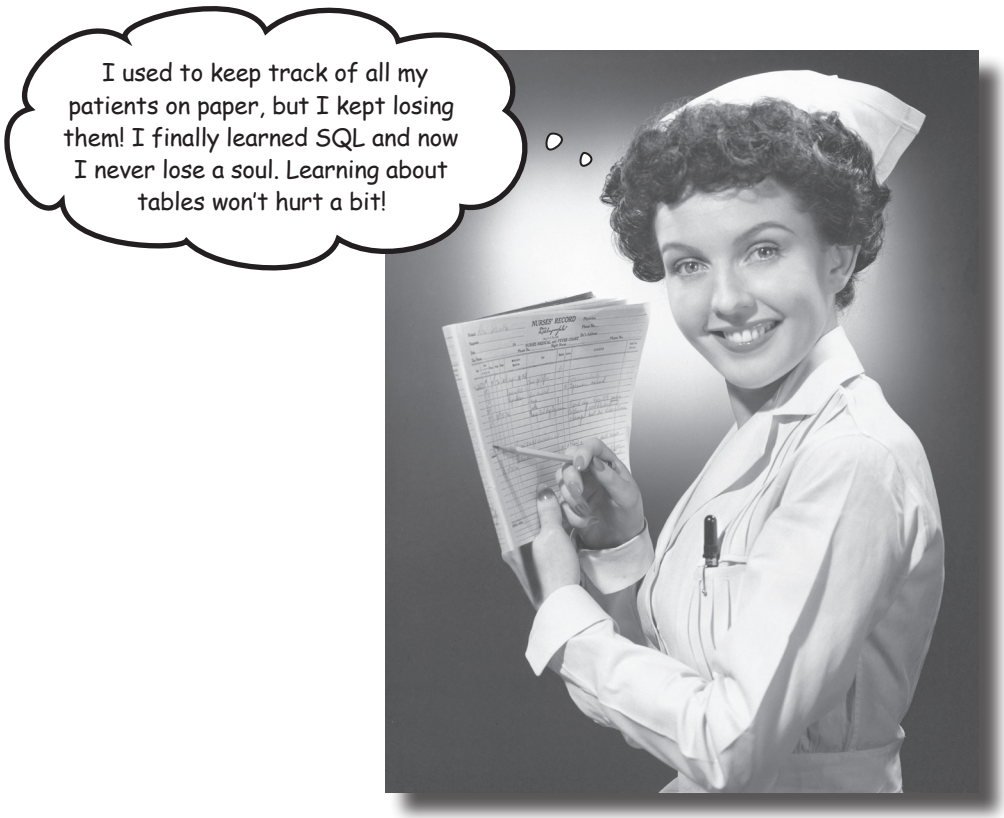
O'REILLY®

Lynn Beighley

Table of Contents

1 data and tables

A place for everything

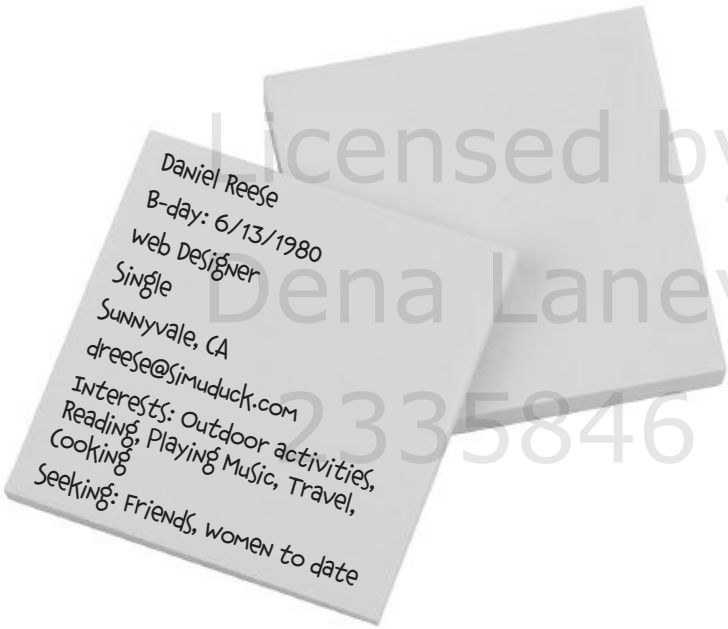


Don't you just hate losing things? Whether it's your car keys, that 25% off coupon for Urban Outfitters, or your application's data, there's nothing worse than not being able to **keep up with what you need**... when you need it. And when it comes to your applications, there's no better place to store your important information than in a **table**. So turn the page, come on in, and take a walk through the world of **relational databases**.

Chapter 1. data and tables. Head First SQL, ISBN: 9780596526849

Defining your data

Greg knows many lonely single people. He likes keeping track of what his friends are up to, and enjoys introducing them to each other. He has lots of information about them scrawled on sticky notes like this:



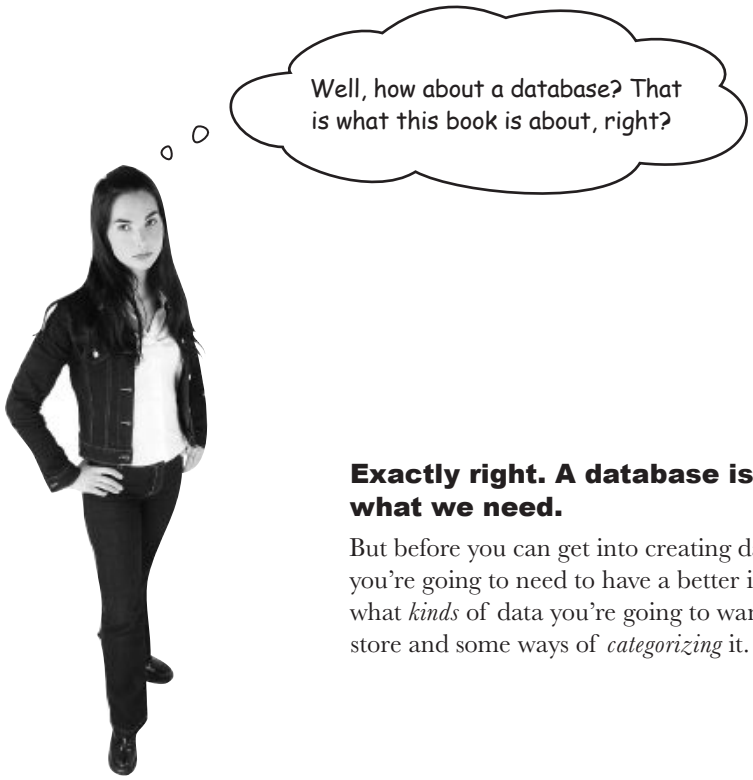
Greg's been using his system for a very long time. Last week he expanded his connections to include people who are seeking new jobs, so his listings are growing quickly. Very quickly…

Just a few of Greg's notes

BRAIN POWER

Is there a better way to organize this information?
What would you do?

Chapter 1. data and tables. Head First SQL, ISBN: 9780596526849
Prepared for denalaney@gmail.com, Dena Laney



Exactly right. A database is just what we need.

But before you can get into creating databases, you're going to need to have a better idea of what *kinds* of data you're going to want to store and some ways of *categorizing* it.

Sharpen your pencil

Here are some of Greg's notes. Look for similar information that Greg's collected about each person. Give each common bit of data a label that describes the category of information it is, then write those labels in the space below.

Ann Branson
B-day: 7/1/1962
Software Engineer

Single, but involved

Mountain View, CA
annie@boards-r-us.com

Interests: Collecting books, Beermaking, Equestrian

Seeking: New Job

Jamie Hamilton
B-day: 9/10/1964
System Administrator

Single

Sunnyvale, CA
dontbother@breakneckpizza.net

Interests: Hiking, writing,

Seeking: Friends, women to date

Seeking

Alan Soukup
B-day: 7/1/1966
Aeronautical Engineer

Married

San Antonio, TX
soukup@breakneckpizza.net

Interests: RPG, programming

Seeking: Nothing

Angelina Mendoza
B-day: 8/19/1979
Unix Sysadmin

Married

San Francisco, CA
angelina@starbuzzcoffee.com

Interests: Acting, Dancing

Seeking: New job

.....

.....

.....

.....

.....

.....

.....

.....

.....

Sharpen your pencil Solution

Here are some of Greg's notes. Look for similar information that Greg's collected about each person. Give each common bit of data a label that describes the category of information it is, then write those labels in the space below.

Now that we've created these categories, we can use them to organize our data.

First Name

Ann Branson
B-day: 7/1/1962
Software Engineer

Status

Single, but involved

Mountain View, CA
annie@boards-r-us.com

Interests: Collecting books, Beermaking, Equestrian

Seeking: New Job

Last Name

Jamie Hamilton
B-day: 9/10/1964
System Administrator

Single

Sunnyvale, CA
dontbother@breakneckpizza.net

Interests: Hiking, writing,

Seeking: Friends, women to date

Alan Soukup
B-day: 7/1/1966
Aeronautical Engineer

Profession

Married

Location

San Antonio, TX
Soukup@breakneckpizza.net

Email

Interests: RPG, programming

Seeking: Nothing

We've split names into first name and last name. This will help you sort the data later.

First Name .....

Last Name .....

Birthday .....

Profession .....

Status .....

Location .....

Email .....

Interests .....

Seeking .....

Angelina Mendoza
B-day: 8/19/1979
Unix Sysadmin

Married

San Francisco, CA
angelina@starbuzzcoffee.com

Interests: Acting, Dancing

Seeking: New job

Greg already gave some information the category names "B-day", "Interests" and "Seeking" on his stickies.

Look at your data in categories

Let's look at your data in a different way. If you cut each note into pieces, then spread the pieces out horizontally you'd get something that looked like this:

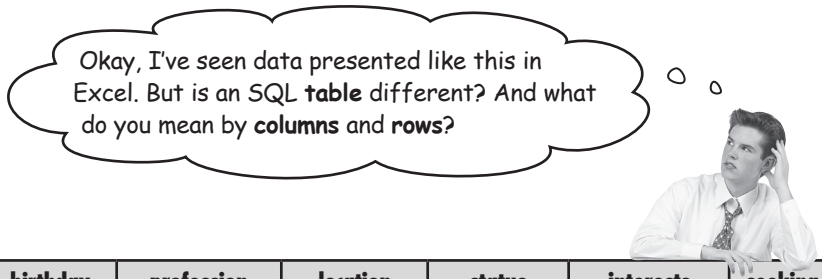
Angelina | Mendoza | 8/19/1979 | Unix SySadmin | Married | San Francisco, CA | angelina@starbuzzcoffee.com | cting, Dancing | New job

Then if you cut up another sticky note with the categories you just noticed, and put the pieces above their corresponding information, you'd have something that looks a lot like this:

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

Angelina | Mendoza | 8/19/1979 | Unix SySadmin | Married | San Francisco, CA | angelina@starbuzzcoffee.com | Acting, Dancing | New job

Here's that same information nicely displayed in a **TABLE** in **columns** and **rows.**



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

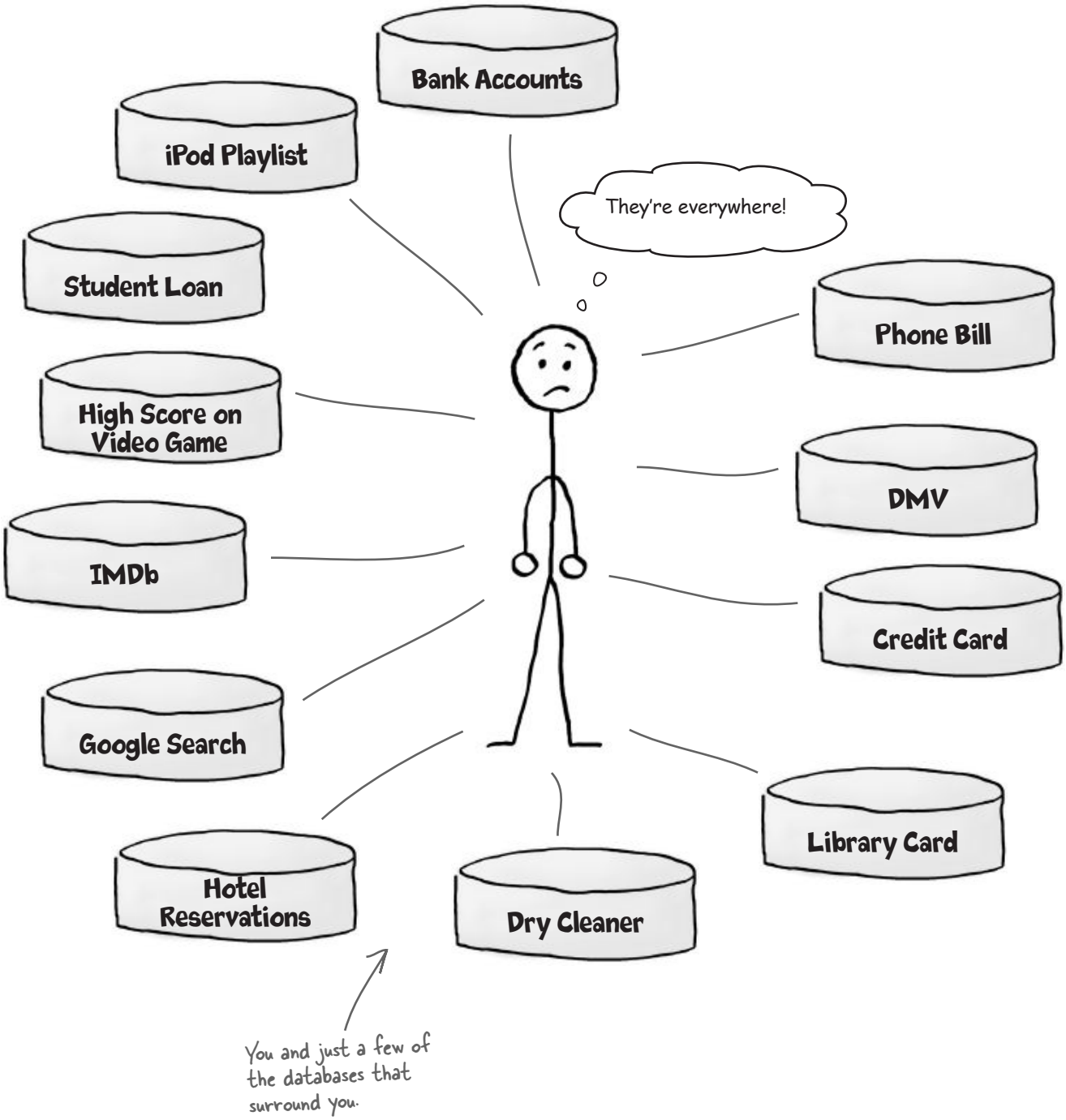
What's in a database?

Before we get into the details of what tables, rows, and columns are, let's step back and look at the bigger picture. The first SQL structure you need to know about is the container that holds all your tables known as a ***database***.

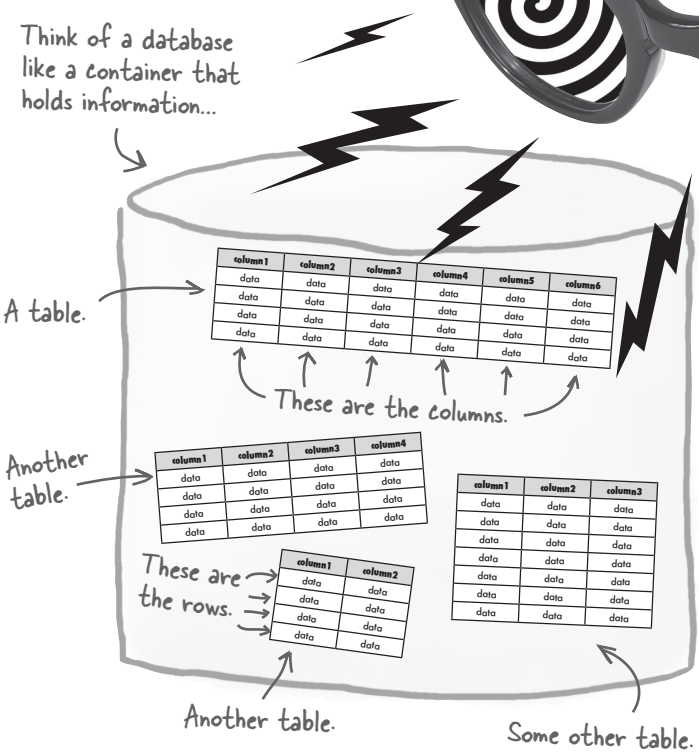
> A **database** is a container that holds tables and other SQL structures related to those tables.

Every time you search online, go shopping, call information, use your TiVo, make a reservation, get a speeding ticket, or buy groceries, a database is being asked for information, otherwise known as being ***queried***.

In diagrams and flow charts, databases are depicted as cylinders. So when you see this, think database.



Anatomy of a Database



A database contains tables.

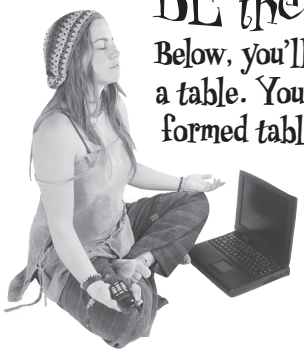
A **table** is the structure inside your database that contains data, organized in **columns** and **rows**.

Remember those categories you came up with? Each category becomes a **column** in your table. These values might be in the same column: Single, Married, Divorced.

A table **row** contains all the information about one object in your table. In Greg's new table, a row would be all the data about one person. Here's an example of some of the data that might be in one row: John, Jackson, single, writer, jj@boards-r-us.com.

> The information inside the database is organized into **tables**.

BE the table



Below, you'll find some sticky notes and a table. Your job is to be the partially formed table and fill in the empty bits to create inner peace. After you've done the exercise, turn the page to see if you've become one with the table.

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

Use one of the fields as a title that gives the table a meaningful name.

shop				
			9	
		4/25	5	
				not enough jelly

BE the table Solution

Your job was to be the partially formed table and fill in the empty bits to increase inner peace.

You should have been able to work out what the table's title could be from the stickies.

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

Don't worry if your answers for the column names don't match ours exactly.

Databases contain connected data

All of the tables in a database should be ***connected*** in some way. For example, here are the tables that might be in a database holding information about doughnuts:

Here's a database with three tables in it. The database is called 'my_snacks'.

Database and table names are not usually capitalized.

my_snacks

Table containing information about jelly doughnuts.

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

Table containing information about glazed doughnuts.

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

Table containing information about snacks that aren't doughnuts.

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

Tables Up Close

A **column** is a piece of data stored by your table. A **row** is a single set of columns that describe attributes of a single thing. Columns and rows together make up a table.

Here's an example of what an address book table containing your personal information might look like. You'll often see the word **field** used instead of **column**. They mean the same thing. Also, **row** and **record** are often used interchangeably.

These are the columns.

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

These are the rows.

Put the columns and rows together and you've got yourself a table.

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

So we have enough data from my stickies to turn them into a table?

Exactly. You can identify categories for the type of data you're collecting for each person.

Your categories then become your columns. Each sticky note becomes a row. You can take all that information from your stickies and turn it into a table.

Categories from page 7

First Name | Last Name | Birthday | Profession | Status | Location | | Interests | Seeking

Angelina | Mendoza | 8/19/1979 | Unix SySadmin | Married | San Francisco, CA | angelina@starbuzzcoffee.com | Acting, Dancing | New job

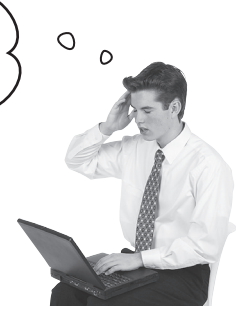
Data from a single sticky laid out to form a row.

Now you know that the categories are called columns.

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.net	9-10-1966	System Administrator	Sunnyvale, CA	Single	Hiking, Writing	Friends, Women to date
Soukup	Alan	soukup@breakneckpizza.net	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

...and that each sticky's data can be placed on a single row called a record.

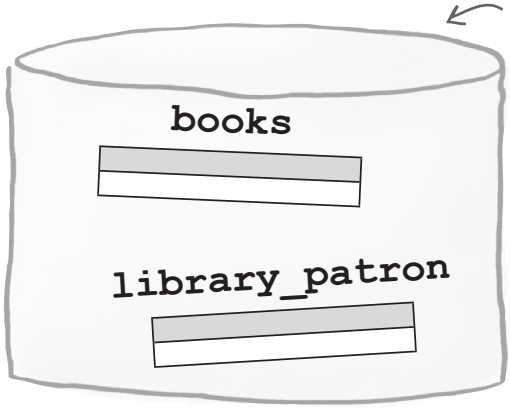
Finally. Okay so how do I create my table?



Consider the databases and tables below. Think about what categories of data you might find in each. Come up with some likely columns for each table.

library_db

Database for a library



books: .....

library_patron: .....

bank_db

Database for a bank

customer_info: .....

bank_account: .....

onlinestore_db

Database for an online store

product_info: .....

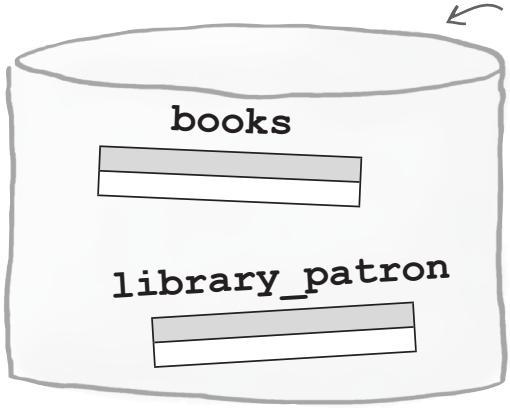
shopping_cart: .....

Consider the databases and tables below. Think about what categories of data you might find in each. Come up with some likely columns for each table.

library_db

Database for a library

Don't worry if your answers for the column names don't match ours exactly.



books: title, author, cost, scan_code

library_patron: first_name, last_name, address

bank_db

Database for a bank

customer_info: first_name, last_name, address, account_number, ssn

bank_account: balance, deposits, withdrawals

onlinestore_db

Database for an online store

product_info: name, size, cost

shopping_cart: total_charge, customer_id

Take command!

Start up your SQL relational database management system (RDBMS) and open a command-line window or graphical environment that allows you to communicate with your RDBMS. Here's our terminal window after we start MySQL.

```
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.

>
```

This angle bracket is the command prompt. You'll be typing your commands right after it.

First you're going to need to create a database to hold all your tables.

Spaces aren't allowed in the names of databases and tables in SQL, so an underscore can be used instead.

1 Type in the line of code below to create your database called **gregs_list**.

```
CREATE DATABASE gregs_list;
```

Your command must end with a semicolon.

CREATE DATABASE is the command.

The name of the database is gregs_list

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

This is feedback from the RDBMS, letting you know your query executed successfully.

Watch it!

Did you read the intro?

We're using MySQL to command our databases, so commands in your Database Management System (DBMS) might look a little different. See Appendix II for instructions on installing MySQL on your server. And, don't forget, you can follow along with many of the examples in the book at http://www.headfirstlabs.com/sql_hands_on/

2 Now you need to tell your RDBMS to actually *use* the database you just created:

```
USE gregs_list;
```

Now everything we do will happen inside the gregs_list database!

```
File Edit Window Help USEful
> USE gregs_list;
Database changed
```

there are no Dumb Questions

Q: Why do I need to create a database if I only have one table?

A: The SQL language requires all tables to be inside of databases. There are sound reasons behind this. One of the features of SQL is the ability to control access to your tables by multiple users. Being able to grant or deny access to an entire database is sometimes simpler than having to control the permissions on each one of multiple tables.

Q: I noticed that we used all uppercase for the CREATE DATABASE command. Is that necessary?

A: Some systems do require certain keywords to be capitalized, but SQL is case insensitive. That means it's not necessary to capitalize commands, but it's considered a good programming practice in SQL. Look at the command we just typed,

```
CREATE DATABASE
gregs_list;
```

The capitalization makes it easy to tell the command (`CREATE DATABASE`) from the name of the database (`gregs_list`).

Q: Is there anything I should know about naming my databases, tables, and columns?

A: It's generally a good idea to create descriptive names. Sometimes this results in you needing to use more than one word in a name. You can't use spaces in your names, so the underscore lets you create more descriptive names. Here are variations you might see used:

```
gregs_list
gregslist
Gregslist
gregsList
```

Generally it's best to avoid capitalizing your names to avoid confusion since SQL is case insensitive..

Q: What if I prefer to use "gregsList" with no underscore?

A: Go right ahead. The important thing is to be consistent. If you use `gregsList` as the database name with no underscore and the second word capitalized, then you should stick to that naming convention throughout all your tables in this database, for example naming your table `myContacts`, to be consistent.

Q: Shouldn't the database be called greg's_list? Why leave out the apostrophe?

A: The apostrophe is reserved for a different use in SQL. There are ways you could include one, but it's far easier to omit it.

Q: I also noticed a semicolon at the end of the CREATE DATABASE command. Why did we need that?

A: The semicolon is there to indicate that the command has ended.

Capitalization and underscores help you program in SQL (even though SQL doesn't need them!)

Setting the table: the CREATE TABLE statement

Let's see all this in action with the doughnut data. Say you were having trouble remembering what type of doughnuts a snack in your list was just from its name, you might ***create a table*** to save having to remember them instead. Below is a single command to type into your console window. When you've typed it, you can press RETURN to tell your SQL RDBMS to carry out the command.

doughnut_list

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

Here's the SQL command to create the table— notice the caps.

Your table's name should be lowercase and have an underscore in place of any spaces.

Just hit return to start a new line in your command to make it easier to read what's what.

```
CREATE TABLE doughnut_list
(
  doughnut_name VARCHAR(10),
  doughnut_type VARCHAR(8)
);
```

The opening parenthesis opens the list of columns to create.

The comma separates the columns being created.

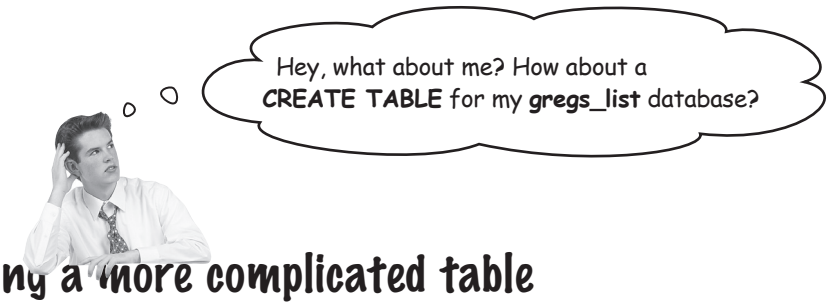
The name of the first column in the table.

The name of the second column.

The closing parenthesis closes the list of columns.

The semicolon tells the SQL RDBMS that it's reached the end of the command.

This is a DATA TYPE. It stands for VARiable CHARacter and is used to hold information that's stored as text. The (8) means that the text it holds can be up to 8 characters long.



Creating a more complicated table

Remember the columns for Greg's table? We've jotted them down on a sticky note. You'll need those to write your **CREATE TABLE** command.

You'll be using the CREATE TABLE command to go from this...

...to this

last_name	first_name	email	birthday	profession	location	status	interests	seeking

BRAIN POWER

In which two ways do the column names on the sticky note differ from those in the table above? Why are they significant?

Look how easy it is to write SQL

You've seen that to create a table you categorize your data into columns. Then you come up with the right data type and length for each column. After you estimate how long each column needs to be, writing the code is straightforward.

Sharpen your pencil

The code to the left is our CREATE TABLE statement for Greg's new database. Try to guess what each line of the CREATE TABLE command is doing. Also include an example of the data that will go in each column.

```
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)
);
```


Sharpen your pencil Solution

Here's what each line of the CREATE TABLE command is doing, and some example data for each column type.

`CREATE TABLE my_contacts`	Creates a table named 'my_contacts'	
`(`	Opens the list of columns to add	
`last_name VARCHAR(30),`	Adds a column named 'last_name' that can hold up to 30 characters	'Anderson'
`first_name VARCHAR(20),`	Adds a column named 'first_name' that can hold up to 20 characters	'Jillian'
`email VARCHAR(50),`	Adds a column named 'email' that can hold up to 50 characters	'jill_anderson@breakneckpizza.net'
`birthday DATE,`	Adds a column named 'birthday' that can hold a date value	'1980-09-05'
`profession VARCHAR(50),`	Adds a column named 'profession' that can hold up to 50 characters	'Technical Writer'
`location VARCHAR(50),`	Adds a column named 'location' that can hold up to 50 characters	'Palo Alto, CA'
`status VARCHAR(20),`	Adds a column named 'status' that can hold up to 20 characters	'Single'
`interests VARCHAR(100),`	Adds a column named 'interests' that can hold up to 100 characters	'Kayaking, Reptiles'
`seeking VARCHAR(100)`	Adds a column named 'seeking' that can hold up to 100 characters	'Relationship, Friends'
`);`	Closes the list of columns to add, and the semicolon ends the command	

Create the my_contacts table, finally

Now you know exactly what each line is doing, you can type in the **CREATE TABLE** command. You can enter it one line at a time, copying the code at the top of this page.

Or you can enter it all as one really long single line:

```
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));
```

Trust us, this really is the command, it's just written out r-e-a-l-l-y small so it fits on the page!

Whichever way you choose to enter it, before you hit return after the semicolon, make sure you haven't missed any characters:
`last_name VARCHAR(3)` is a very different column than `lastname VARCHAR(30)`!

Your table is ready

```
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)
```

Did you notice how hitting return after the semicolon ended the command and told your SQL RDBMS to process it?

So I'll always store everything in either **VARCHAR** or **DATE** data types?

Actually, you'll need a few more data types for other kinds of data, like numbers.

Suppose we added a price column to our doughnut table. We wouldn't want to store that as a VARCHAR. Values stored as VARCHARs are interpreted as text, and you won't be able to perform mathematical operations on them But there are more data types you haven't met yet…

BRAIN POWER

Before going further, come up with other types of data that need a data type other than **VARCHAR** or **DATE**.

Take a meeting with some data types

These are a few of the most useful data types. It's their job to store your data for you without mucking it up. You've already met VARCHAR and DATE, but say hello to these.

CHAR or CHARACTER. He's rigid and prefers his data to be a set length.

INT or INTEGER thinks numbers should be whole, but he's not afraid of negative numbers.

DEC, short for DECIMAL. He'll give you all the decimal places you ask for, until he's full.

We don't know who he is, he just wandered in.

Call him BLOB. He likes large gobs of text data.

She goes by either DATETIME or TIMESTAMP depending on the SQL RDBMS. She keeps track of the date and time. She's also got a fraternal twin, TIME, who doesn't care what the date is.

DATE keeps track of your dates. She doesn't care about the time, though.

VARCHAR holds text data of up to 255 characters in length. She's flexible and can adapt to the length of your data.

Watch it!

These data type names may not work with your SQL RDBMS!

Unfortunately, there are no universally accepted names for various data types. Your particular SQL RDBMS might use different names for one or more of these types. Check your documentation to find the correct names for your RDBMS.

Which Data Type?

Determine which data type makes the most sense for each column. While you're at it, fill in the other missing info.

These two numbers show how many total digits the database should expect, and how many after the decimal.

Column Name	Description	Example	Best Choice of Data Type
price	The cost of an item for sale	5678.39	DEC(6,2)
zip_code			
atomic_weight	Atomic weight of an element with up to 6 decimal places		
comments	Large block of text, more than 255 characters	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	How many of this item in stock		
tax_rate		3.755	
book_title		Head First SQL	
gender	One character, either M or F		CHAR(1)
phone_number	Ten digits, no punctuation	2105552367	
state	Two-character abbreviation for a state	TX, CA	
anniversary		11/22/2006	DATE
games_won			INT
meeting_time		10:30 a.m. 4/12/2020	

there are no Dumb Questions

Q: Why not just use BLOB for all of my text values?

A: It's a waste of space. A `VARCHAR` or `CHAR` takes up a specific amount of space, no more than 256 characters. But a `BLOB` takes up much more storage space. As your database grows, you run the risk of running out of space on your hard drive. You also can't run certain important string operations on `BLOB`s that you can on `VARCHAR`s and `CHAR`s (you'll learn about these later).

Q: Why do I need these numeric types like INT and DEC?

A: It all comes down to database storage and efficiency. Choosing the best matching data type for each column in your table will reduce the size of table and make operations on your data faster.

Q: Is this it? Are these all the types?

A: No, but these are the most important ones. Data types also differ slightly by RDBMS, so you'll need to consult your particular documentation for more information. We recommend *SQL in a Nutshell* (O'Reilly) as a particularly good reference book that spells out the differences between RDBMSs.

Chapter 1. data and tables. Head First SQL, ISBN: 9780596526849

WHICH DATA TYPE?

Determine which data type makes the most sense for each column. While you're at it, fill in the other missing info.

A zip code may not always be 10 characters long, so we use VARCHAR to save space in the database. You might also have used CHAR here and assumed a specific length.

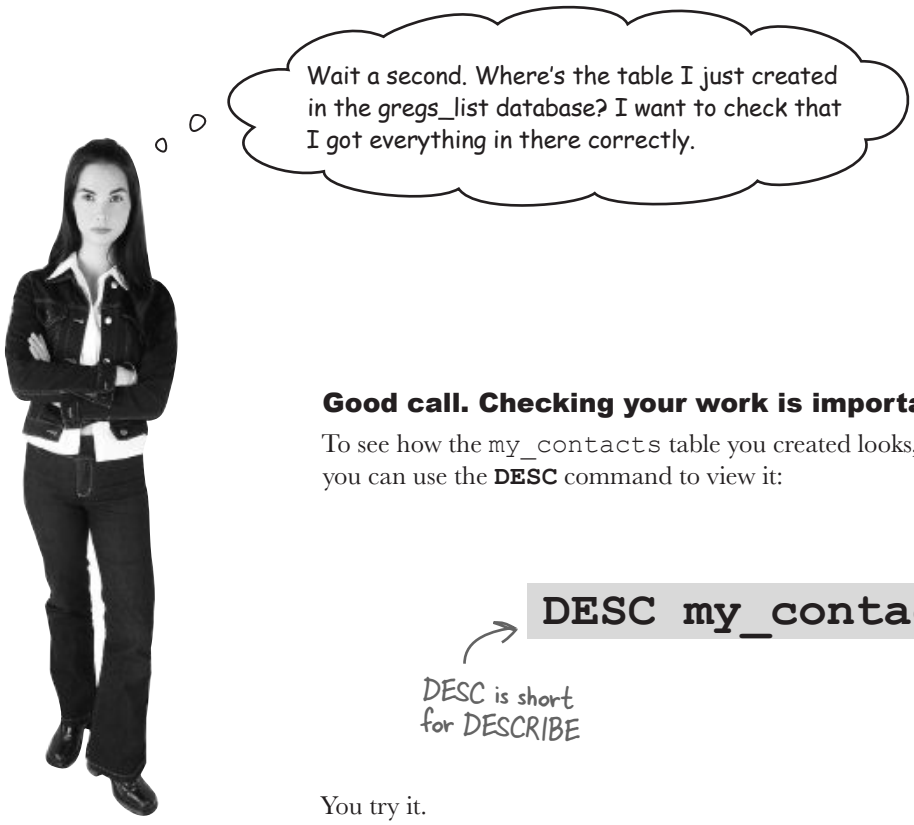
Column Name	Description	Example	Best Choice of Data Type
price	The cost of an item for sale	5678.39	DEC(6,2)
zip_code	Five to 10 characters	90210-0010	VARCHAR(10)
atomic_weight	Atomic weight of an element with up to 6 decimal places	4.002602	DEC(10, 6)
comments	Large block of text, more than 255 characters	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	How many of this item in stock	239	INT
tax_rate	A percentage	3.755	DEC(6, 3)
book_title	A text string	Head First SQL	VARCHAR(50)
gender	One character, either M or F	M	CHAR(1)
phone_number	Ten digits, no punctuation	2105552367	CHAR(10)
state	Two character abbreviation for a state	TX, CA	CHAR(2)
anniversary	Month, day, year	11/22/2006	DATE
games_won	An integer representing number of games won	15	INT
meeting_time	A time and day	10:30 a.m. 4/12/2020	DATETIME

A phone number will always be exactly this length. And we treat it like a text string because we don't need to do any mathematical operations on it, even though it's a number.

TIMESTAMP is usually used to capture the current time. DATETIME is best used to store a future event.

BULLET POINTS

- Break your data up in categories before you create your table. Pay special attention to the type of data for each column.
- Use the **CREATE DATABASE** statement to create the database which will hold all of your tables.
- Use the **USE DATABASE** statement to get inside your database to create your table.
- All tables are created with a **CREATE TABLE** statement, containing column names and their corresponding data types.
- Some of the most common datatypes are **CHAR**, **VARCHAR**, **BLOB**, **INT**, **DEC**, **DATE**, and **DATETIME**. Each has different rules for what goes inside.



Good call. Checking your work is important.

To see how the `my_contacts` table you created looks, you can use the **DESC** command to view it:

```
DESC my_contacts;
```

DESC is short for DESCRIBE

You try it.

```
File Edit Window Help DescTidy
> DESC my_contacts;
```

Your table, DESCribed

When you've entered the DESC command. You'll see something that looks similar to this:

Don't worry about these right now; we'll get to them shortly.

```
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)
```

BRAIN POWER

What do you think? What sorts of problems could adding a new column create?

SQL Magnets

The code to create the database and table with the new gender column is all scrambled up on the fridge. Can you reconstruct the code snippets to make it work? Some of the parentheses and semicolons fell on the floor and they were too small to pick up, so feel free to add as many of those as you need!

```
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)
```

When you finish, try typing the new CREATE TABLE code into your SQL console to add the new gender column!

SQL Magnets Solution

Your job was to reconstruct the code snippets to make the code that would create the database and table with the new gender column.

gregs_list already exists.

Here's the code reconstructed. Check your answer against it, then keep reading...

```
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)
);
```

You can't recreate an existing table or database!

Did you try entering the new CREATE TABLE statement? If you did, you'll already know that the solution to the exercise won't help you add the new column.

If you did enter it into your console, you probably saw something like this:

```
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
```

The new column for gender.

Uh oh. That statement gives you an error message. Looks like the table wasn't created.

there are no Dumb Questions

Q: About that SQL Magnets exercise, why did I get an error?

A: You can't create a table that already exists. And once you create a database, you don't need to create it again. Other possible errors include you forgetting the semicolon. Also, check to see if you typoed any of the SQL keywords.

Q: Why isn't there a comma after "seeking VARCHAR(100)" like all the other columns have?

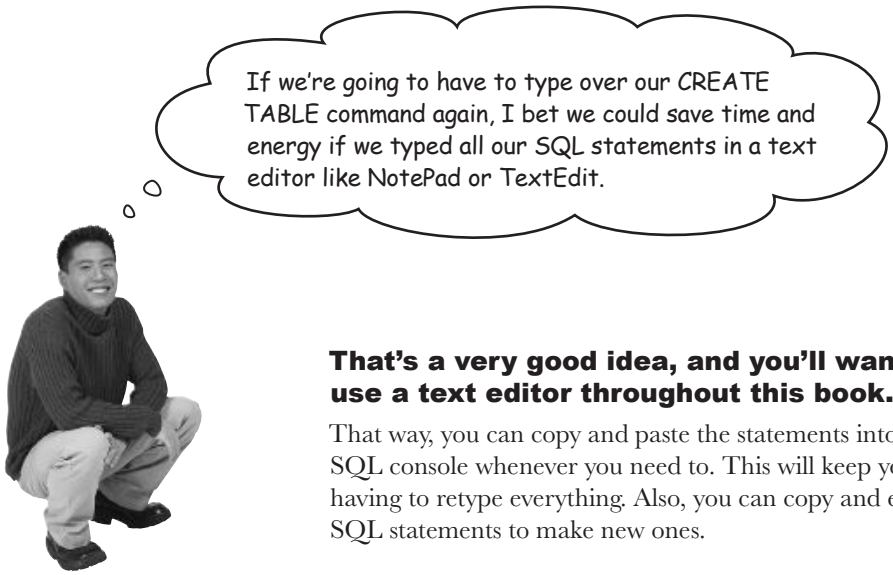
A: The column 'seeking" is the last of them before we reach the closing parenthesis. That tells the RDBMS that the end of the statement is here, so no comma is needed.

Q: So, is there a way to add the forgotten column or will I have to start over?

A: You're going to have to start over, but before you can create the table with the added gender column you have to get rid of the old one. Since there is no data in the table yet, we can simply get rid of the old one and start over.

Q: But what if I've got a table with data in it, and I need to add a column? Is there a way to do it without deleting the whole table and starting over?

A: Great question! There is a way to change your table without damaging the data in it. We'll get to that a bit later, but for now, since our table is empty, we'll get rid of the table and create a new one.



That's a very good idea, and you'll want to use a text editor throughout this book.

That way, you can copy and paste the statements into your SQL console whenever you need to. This will keep you from having to retype everything. Also, you can copy and edit old SQL statements to make new ones.

Out with the old table, in with the new

1. Getting rid of a table is much easier than creating a table. Use this simple command:

The command to delete your table...

...and the name of the table to be deleted.

```
DROP TABLE my_contacts;
```

Don't forget the semicolon.

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

`DROP TABLE` will work whether or not there is data in your table, so *use the command with extreme caution*. Once your table is dropped, it's gone, along with any data that was in it.

DROP TABLE deletes your table and any data in it!

2. Now you can enter your new **`CREATE TABLE`** statement:

```
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)
```

This time it worked.

Who am I?

A bunch of SQL keywords and data types, in full costume, are playing the party game "Who am I?" They give you a clue, and you try to guess who they are, based on what they say. Assume they always tell the truth about themselves. If they happen to say something that could be true for more than one guy, then write down all for whom that sentence applies. Fill in the blanks next to the sentence with the names of one or more attendees.

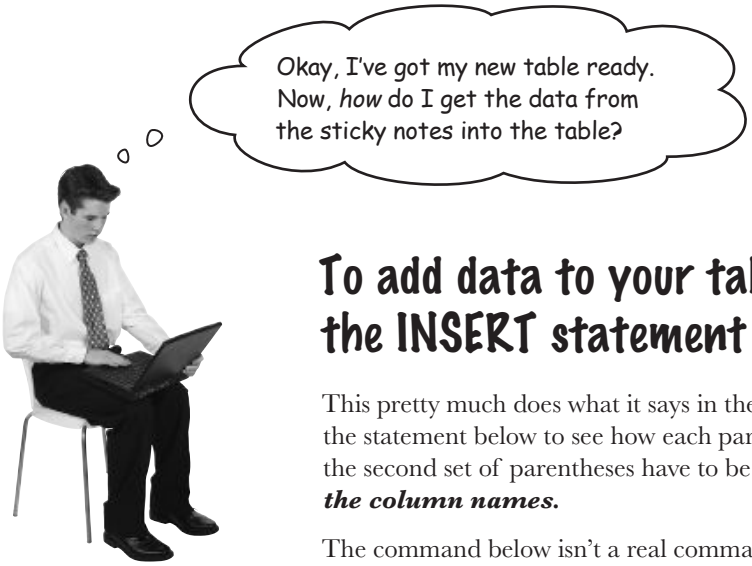
Tonight's attendees:

CREATE DATABASE, USE DATABASE, CREATE TABLE, DESC, DROP TABLE, CHAR, VARCHAR, BLOB, DATE, DATETIME, DEC, INT

	Name
I've got your number.	.....
I can dispose of your unwanted tables.	.....
T or F questions are my favorite.	.....
I keep track of your mom's birthday.	.....
I got the whole table in my hands.	.....
Numbers are cool, but I hate fractions.	.....
I like long, wordy explanations.	.....
This is the place to store everything.	.....
The table wouldn't exist without me.	.....
I know exactly when your dental appointment is next week.	.....
Accountants like me.	.....
I can give you a peek at your table format.	.....
Without us, you couldn't even create a table.	.....
	.....

Answers on page 51.

Anatomy of a Statement



To add data to your table, you'll use the INSERT statement

This pretty much does what it says in the name. Take a look at the statement below to see how each part works. The values in the second set of parentheses have to be in the ***same order as the column names.***

The command below isn't a real command, it's a template of a statement to show you the format of an `INSERT` statement.

The keywords INSERT INTO begin the statement.

The name of your table. In Greg's case, it will be my_contacts.

This next part is a list of your column names, separated by commas. You already know that Greg's list will have columns like first_name, last_name, and email.

More column names follow, no comma after the last one.

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

Another keyword. This signals that the values for the columns follow.

This next part is a list of your values, separated by commas. In Greg's case, the list will contain the information from his sticky notes.

The single quotes are correct. Use them whenever you're inserting text, even if it's a single character like 'M', or 'F'.

More values follow, no comma after the last one.

The usual semicolon ending the statement.

IMPORTANT: the values need to be in the same order as the column names.

WHO DOES WHAT?

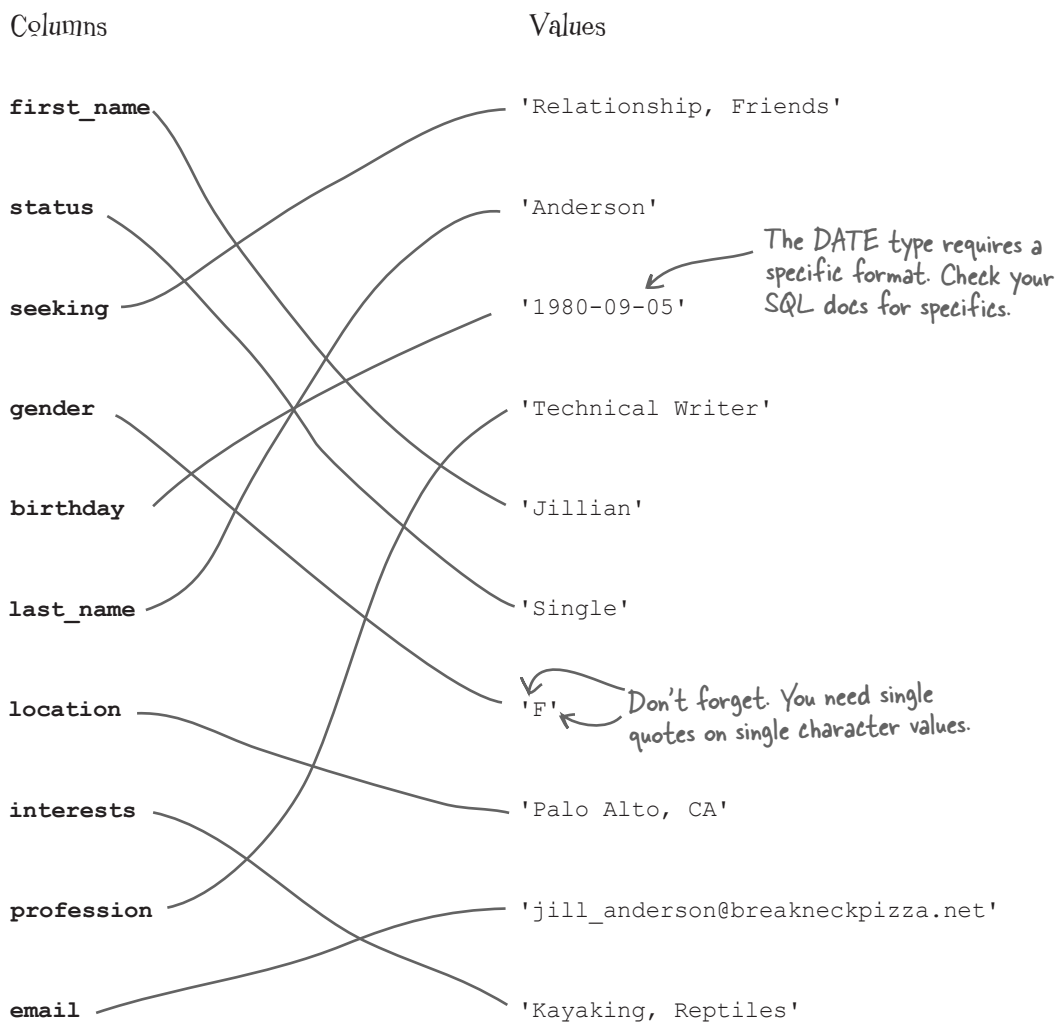
Before you can write your INSERT statement, you need to match up your column names and values.

Columns	Values
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.net'
email	'Kayaking, Reptiles'

Chapter 1. data and tables. Head First SQL, ISBN: 9780596526849

WHO DOES WHAT?

Before you can write your INSERT statement, you need to match up your column names and values.



Create the INSERT statement

Your column names are in the first set of parentheses and divided by commas.

You can hit return before the opening parenthesis to make the code easier to read in your console window.

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

Hit return after the closing columns parenthesis and another after VALUES to make the code easier to read.

The values for each column are in the second set of parentheses and are also separated by commas.

Any value that goes into a VARCHAR, CHAR, DATE, or BLOB column has single quotes around it.

Watch it!

Order matters!

*The **values** should be listed in **exactly** the **same order** as the **column names**.*

Exercise

Try this at home

This is one way to add a row to your table. Try typing it in yourself. Type it in a text editor first so if you make a mistake you won't have to retype the entire thing. Pay special attention to the single quotes and commas. Write the response you get here:

You just told me that **CHAR**, **VARCHAR**, **DATE**, and **BLOB** values have **single quotes** around them **in the INSERT** statement. So that means **numeric values** like **DEC** and **INT don't use quotes**?

Exactly right.

Here's an `INSERT` statement you might use if you had a table of doughnut purchases. Notice how, in the values, the numbers that match the dozens of donuts purchased and price columns have no quotes.

The dozens column is an INT, since you don't usually buy part of a dozen and don't need decimal places.

The price column is DEC(4,2) which means it's four digits long, with two decimal places.

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

The values inserted into the dozens and price columns don't need quotes!

Sharpen your pencil

Your SQL RDBMS will tell you when something is wrong with your statement, but will sometimes be a bit vague. Take a look at each INSERT statement below. First try to guess what's wrong with the statement, and then try typing it in to see what your RDBMS reports.

```
INSERT INTO my_contacts

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

What's wrong? .....

Your RDBMS says: .....

```
INSERT INTO my_contacts

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

What's wrong? .....

Your RDBMS says: .....

```
INSERT INTO my_contacts

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

What's wrong? .....

Your RDBMS says: .....

```
INSERT INTO my_contacts

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

What's wrong? If this one causes your RDBMS to "hang," try typing a single quote followed by a semicolon after you've entered the rest of the statement.

Your RDBMS says: .....

Sharpen your pencil Solution

Your SQL RDBMS will tell you when something is wrong with your statement, but will sometimes be a bit vague. Take a look at each INSERT statement below. First try to guess what's wrong with the statement, and then try typing it in to see what your RDBMS reports.

```
INSERT INTO my_contacts

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

What's wrong? It's missing a location value — We've got a location column in the column list, but no location in the values list, we're short one value.

Your RDBMS says: ERROR 1136 (21S01): Column count doesn't match value count at row 1

Notice that many different problems result in the same error. Watch out for typos; they can be tricky to track down.

```
INSERT INTO my_contacts

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

What's wrong? Missing email in column list — This time we have a value for all the columns, but we're missing our email column in the column list.

Your RDBMS says: 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.net',
'F', '1980-09-05', 'Technical Writer' 'Palo Alto, CA', 'Single', 'Kayaking, Reptiles',
'Relationship, Friends');
```

What's wrong? Missing comma between two values — No comma in the values list between 'Technical Writer' and 'Palo Alto, CA'

Your RDBMS says: 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.net',
'F', '1980-09-05', 'Technical Writer', 'Palo Alto, CA', 'Single', 'Kayaking, Reptiles',
'Relationship, Friends);
```

What's wrong? It's missing a single quote after the last value

Your RDBMS says: 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

Variations on an INSERT statement

There are three variations of **INSERT** statements you should know about.

1 Changing the order of columns

You can change the order of your column names, as long as the matching values for each column come in that same order!

```
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.net', '1980-09-05', 'Technical
Writer', 'Palo Alto, CA', 'Single', 'Relationship, Friends');
```

Notice the order of the column names? Now look at the values; they're in that same order. So long as the values match the column names, the order you INSERT them in doesn't matter to you, or your SQL RDBMS!

2 Omitting column names

You can leave out the list of column names, but the values must be **all** there, and all **in the same order** that **you added the columns in**. (Double-check the order on page 37 if you're unsure.)

```
INSERT INTO my_contacts

VALUES

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

We left the column names out altogether, but if you do that, you must include ALL the values, and in the EXACT ORDER that they are in the table!

3 Leaving some columns out

You can insert a few columns and leave some out.

```
INSERT INTO my_contacts

(last_name, first_name, email)

VALUES

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

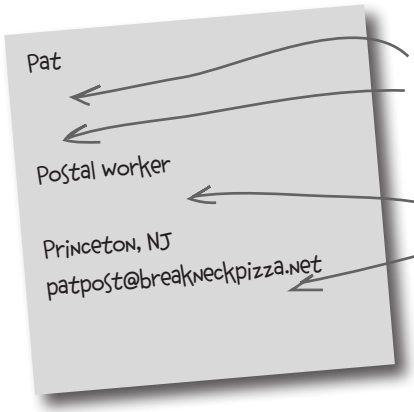
This time, we're only inserting part of our data. Since your SQL RDBMS won't know which parts, you'll need to tell it by specifying the column names and values that you are entering.

BRAIN POWER

What do you think shows up in the table in columns that you don't assign a value to?

Columns without values

Let's insert a record into the `my_contacts` database from this incomplete sticky note:



Missing last name and birthday, and we can't be sure about gender, either.

Also missing status, interests, and seeking columns.

COLUMNS:	VALUES:
last_name	?
first_name	'Pat'
email	'patpost@breakneckpizza.net'
gender	?
birthday	?
profession	'Postal Worker'
location	'Princeton, NJ'
status	?
interests	?
seeking	?

Because the sticky is missing some data, Greg will have to enter an incomplete record. But that's okay, he'll be able to add in the missing information later.

We're using the version of INSERT where we don't have to provide data for all columns because it lets us include just the columns where we know the values.

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

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

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

```
File Edit Window Help LotsOfNULLs
> SELECT * FROM my_contacts;
+-----------+------------+-----------------+-----------+-----------+-----------+-----------+-----------+
| last_name | first_name | email           | gender    | birthday  | profession | location  | status    | interests | seeking   |
+-----------+------------+-----------------+-----------+-----------+-----------+-----------+-----------+
| NULL      | Pat        | patpost@breakneckpizza.net | NULL | NULL | Postal Worker | Princeton, NJ | NULL | NULL | NULL |
| Anderson  | Jillian    | jill_anderson@breakneckpizza.net | F | 1980-09-05 | Technical Writer | Palo Alto, CA | Single | Kayaking, Reptiles | Relationship, Friends |
+-----------+------------+-----------------+-----------+-----------+-----------+-----------+-----------+
2 rows in set (0.00 sec)
```

There's a NULL value in each column we had no value for.

Peek at your table with the SELECT statement

So you want to see what your table looks like? Well, `DESC` won't cut it anymore, because it only shows the *structure* of the table and not the information inside of it. Instead, you should use a simple `SELECT` statement so you can see what data is in your table.

We want to select all the data in our table...

... and the asterisk says to select EVERYTHING.

Our table name.

```
SELECT * FROM my_contacts;
```

Don't worry what the `SELECT` statement does for now.

We'll be looking at it in a lot more detail in chapter 2. For now, just sit back and marvel at the beauty of your table when you use the statement.

Now try it yourself. You'll have to stretch out your window to see all the results nicely laid out.

BRAIN POWER

Now you know that NULL appears in any columns with no assigned value. What do you think NULL actually *means*?

SQL Exposed

This week's interview:
Confessions of a NULL

Head First: Welcome, NULL. I have to admit I didn't expect to see you. I didn't think you actually existed. Word on the street is that you're nothing more than a zero, or nothing at all.

NULL: I can't believe you'd listen to such lies. Yes, I'm here, and I'm quite real! So you think I'm nothing, just dirt under your feet?

Head First: Easy there, calm down. It's just that you show up whenever something has no value…

NULL: Sure, better me than, say, a zero, or an empty string.

Head First: What's an empty string?

NULL: That would be if you used two single quotes with nothing inside of them as a value. It's still a text string, but of length zero. Like setting a value for first_name in the my_contacts table to ''.

Head First: So you aren't just a fancy way of saying nothing?

NULL: I told you, I'm not nothing! I'm something… I'm just a bit… undefined, is all.

Head First: So you're saying that if I compared you to a zero, or to an empty string, you wouldn't equal that?

NULL: No! I'd never equal zero. And actually, I'd never even equal another NULL. You can't compare one NULL to another. A value can **be** NULL, but it never **equals** NULL because NULL is an undefined value! Get it?

Head First: Calm down and let me get this straight. You aren't equal to zero, you aren't an empty string variable. And you aren't even equal to yourself? That makes no sense!

NULL: I know it's confusing. Just think of me this way: I'm undefined. I'm like the inside of an unopened box. Anything could be in there, so you can't compare one unopened box to another because you don't know what's going to be inside of each one. I might even be empty. You just don't know.

Head First: I've been hearing rumors that sometimes you aren't wanted. That maybe there are times where you NULLs cause problems.

NULL: I'll admit that I've shown up where I wasn't wanted before. Some columns should always have values. Like last names, for example. No point to having a NULL last name in a table.

Head First: So you wouldn't go where you weren't wanted?

NULL: Right! Just tell me, man! When you're creating your table and setting up your columns, just let me know.

Head First: You don't really look like an unopened box.

NULL: I've had enough. I've got places to go, values to be.

Controlling your inner NULL

There are certain columns in your table that should always have values. Remember the incomplete sticky note for Pat, with no last name? She (or he) isn't going to be very easy to find when you have twenty more NULL last name entries in your table. You can easily set up your table to not accept NULL values for columns.

```
CREATE TABLE my_contacts
(
  last_name VARCHAR (30) NOT NULL,
  first_name VARCHAR (20) NOT NULL
);
```

Just add the words NOT NULL right after the data type.

If you use these, you must provide a value for the column in your INSERT statement. If you don't, you'll get an error.

Sharpen your pencil

```
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)
);
```

Look at each of the columns in our my_contacts CREATE TABLE command. Which should be set to be NOT NULL? Think about columns that should never be NULL and circle them.

We've given you two to start, now finish up the rest. Primarily consider columns that you'll use later to search with or columns that are unique.

Sharpen your pencil Solution

```
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)
);
```

Look at each of the columns in our my_contacts CREATE TABLE command. Which should be set to be NOT NULL? Think about columns that should never be NULL and circle them.

We've given you two to start, now finish up the rest. Primarily consider columns that you'll use later to search with or columns that are unique.

All of the columns should be NOT NULL.

You will use ALL your columns to search with. It's important to make sure your records are complete and your table has good data in it...

...but, if you have a column that you know will need to be filled in later, you may want to allow NULL values in it.

NOT NULL appears in DESC

Here's how the `my_contacts` table would look if you set all the columns to have `NOT NULL` values.

Here's where we create our table with NOT NULL in each column.

```
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)
```

This is the table described. Notice the word NO under NULL.

Fill in the blanks with DEFAULT

If we have a column that we know is usually a specific value, we can assign it a **DEFAULT** value. The value that follows the DEFAULT keyword is automatically inserted into the table each time a row is added ***if no other value is specified***. The default value has to be of the same type of value as the column.

```
CREATE TABLE doughnut_list
(
 doughnut_name VARCHAR(10) NOT NULL,
 doughnut_type VARCHAR(8) NOT NULL,
 doughnut_cost DEC(3,2) NOT NULL DEFAULT 1.00
);
```

We want to make sure that we always have a value in this column. Not only can we make it NOT NULL, we can also assign it a DEFAULT value of $1.

The total digits allowed are 3, with 1 before and 2 after the decimal.

This will be the value inserted in the table for the doughnut_cost column when no other value is designated.

doughnut_list

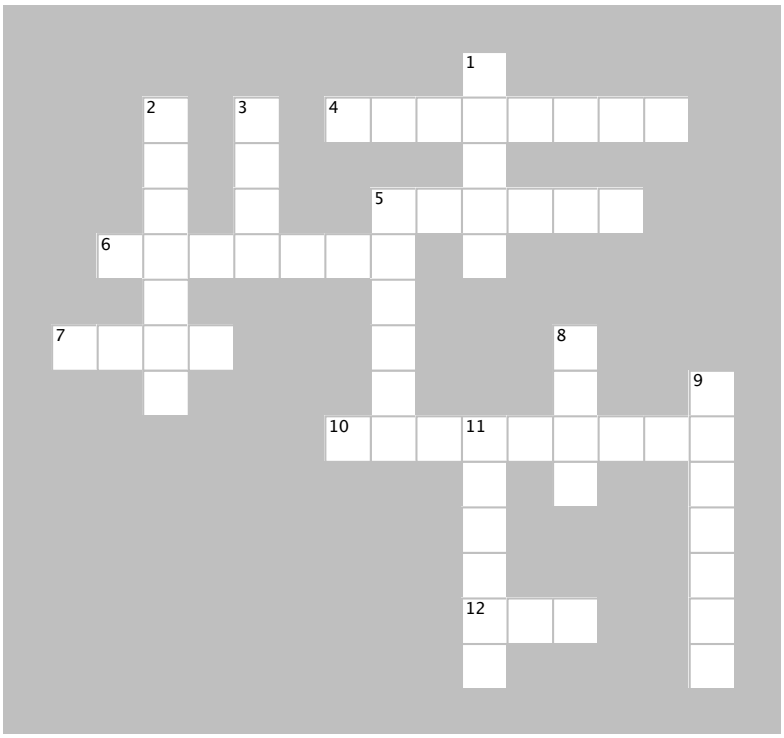
doughnut_name	doughnut_type	doughnut_cost
Blooberry	filled	2.00
Cinnamondo	ring	1.00
Rockstar	cruller	1.00
Carameller	cruller	1.00
Appleblush	filled	1.40

Here's how your table would look if you left the doughnut_cost values blank when you were inserted the records for the Cinnamondo, Rockstar, and Carameller doughnuts.

Using a DEFAULT value fills the empty columns with a specified value.

Tablecross

Take some time to sit back and give your left brain something to do. It's your standard crossword; all of the solution words are from this chapter.

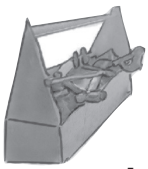


Across

4. A _____ is a container that holds tables and other SQL structures related to those tables.
5. A _____ is a piece of data stored by your table.
6. This holds text data of up to 255 characters in length.
7. You can't compare one _____ to another.
10. End every SQL statement with one of these.
12. This is a single set of columns that describe attributes of a single thing.

Down

1. This is the structure inside your database that contains data, organized in columns and rows.
2. Use this in your CREATE TABLE to specify a value for a column if no other value is assigned in an INSERT.
3. Use this keyword to see the table you just created.
5. This word can be used in front of both TABLE or DATABASE.
8. To get rid of your table use _____ TABLE.
9. This datatype thinks numbers should be whole, but he's not afraid of negative numbers.
11. To add data to your table, you'll use the _____ statement.



Your SQL Toolbox

You've got Chapter 1 under your belt, and you already know how to create databases and tables, as well as how to insert some of the most common data types into them while ensuring columns that need a value get a value.

BULLET POINTS

- If you want to see the structure of your table, use the **DESC** statement.
- The **DROP TABLE** statement can be used to throw away your table. Use it with care!
- To get your data inside your table, use one of the several varieties of **INSERT** statements.
- A **NULL** value is an undefined value. It does not equal zero or an empty value. A column with a **NULL** value **IS NULL**, but ***does not*** **EQUAL NULL**.
- Columns that are not assigned values in your **INSERT** statements are set to **NULL** by default.
- You can change a column to not accept a **NULL** value by using the keywords **NOT NULL** when you create your table.
- Using a **DEFAULT** value when you **CREATE** your table fills the column with that value if you insert a record with no value for that column.

CREATE DATABASE

Use this statement to set up the database that will hold all your tables.

USE DATABASE

Gets you inside the database to set up all your tables.

CREATE TABLE

Starts setting up your table, but you'll also need to know your **COLUMN NAMES** and **DATA TYPES**. You should have worked these out by analyzing the kind of data you'll be putting in your table.

NULL and NOT NULL

You'll also need to have an idea which columns should not accept **NULL** values to help you sort and search your data. You'll need to set the columns to **NOT NULL** when you create your table.

DEFAULT

Lets you specify a default value for a column, used if you don't supply a value for the column when you insert a record.

DROP TABLE

Lets you delete a table if you make a mistake, but you'll need to do this before you start using **INSERT** statements, which let you add the values for each column.

Who am I?

A bunch of SQL keywords and data types, in full costume, are playing the party game "Who am I?" They give you a clue and you try to guess who they are, based on what they say. Assume they always tell the truth about themselves. If they happen to say something that could be true for more than one guy, then write down all for whom that sentence applies. Fill in the blanks next to the sentence with the names of one or more attendees.

Tonight's attendees:

CREATE DATABASE, USE DATABASE, CREATE TABLE, DESC, DROP TABLE, CHAR, VARCHAR, BLOB, DATE, DATETIME, DEC, INT

	Name
I've got your number.	DEC, INT
I can dispose of your unwanted tables.	DROP TABLE
T or F questions are my favorite.	CHAR(1) ← Bonus points if you added the (1)!
I keep track of your mom's birthday.	DATE
I got the whole table in my hands.	CREATE DATABASE
Numbers are cool, but I hate fractions.	INT
I like long, wordy explanations.	BLOB
This is the place to store everything.	CREATE TABLE
The table wouldn't exist without me.	CREATE DATABASE
I know exactly when your dental appointment is next week.	DATETIME
Accountants like me.	DEC
I can give you a peek at your table format.	DESC
Without us, you couldn't even create a table.	CREATE DATABASE, USE DATABASE
	DROP TABLE

DataAndTablescross Solution