Gifted data retrieval

Is it really better to give than retrieve? When it comes to databases, chances are you'll need to **retrieve your data** as often than you'll need to insert it. That's where this chapter comes in: you'll meet the powerful **SELECT** statement and learn how to **gain access to that important information** you've been putting in your tables. You'll even learn how to use **WHERE**, **AND**, and **OR** to selectively get to your data and even avoid displaying the data that you *don't* need.

Date or no date?

Greg's finished adding all the sticky notes into his `my_contacts` table. Now he's ready to relax. He's got two tickets to a concert, and he wants to ask one of his contacts, a girl from San Francisco, out on a date.

He needs to find her email address, so he uses the **SELECT** statement from Chapter 1 to view his table.

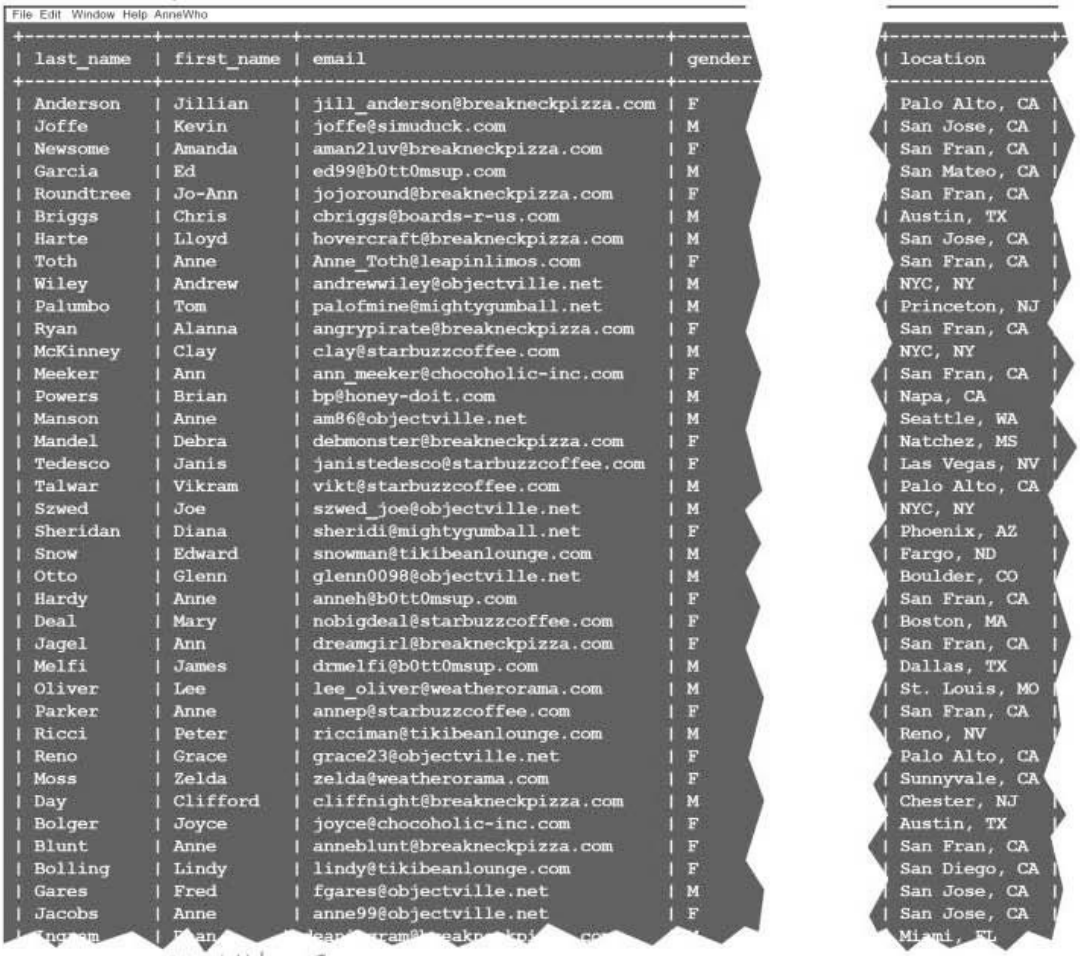
```
SELECT * from my_contacts;
```

Her details are in Greg's table... somewhere.

BE Greg

Your job is to play Greg. Search through the first part of the my_contacts table on the next page looking for Anne from San Fran.

The my_contacts table has quite a few columns. These are just the first few.



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
[illegible]	[illegible]	[illegible]	[illegible]	Miami, FL

This isn't the end of the table! Greg had a LOT of sticky notes.

BE Greg Solutions

Your job was to play Greg, searching through the first part of the my_contacts table looking for Anne from San Fran.

You had to find all the San Fran Annes, and write down their first and last names, and their email addresses.

Toth, Anne: Anne_Toth@leapinlimos.com

Hardy, Anne: anneh@b0tt0msup.com

Parker, Anne: annep@starbuzzcoffee.com

Blunt, Anne: anneblunt@breakneckpizza.com

Here are all the Annes and their email addresses.

Greg's looking for Anne with an 'e'. If you found any Ann entries, you should ignore those.

Making contact

That took **far too much time** and was **extremely tedious**. There is also the very real possibility that Greg might miss some of the matching Annes, including the one he's looking for.

Now that Greg's got all their email addresses, he emails the Annes and discovers…

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

I'm involved with a wonderful guy called Tim [illegible] at the moment. We met at a frat party.

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

I've been looking for a cowpoke like you! Pick me up at five, and we'll rustle up some grub.

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

I'm not the Anne you're looking for, but I'm sure she's a sweet girl. If things don't work out, drop me a line.

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

Of course I remember you! I just wish you had contacted me sooner. I've made plans with my ex-boyfriend who wants to get back together.

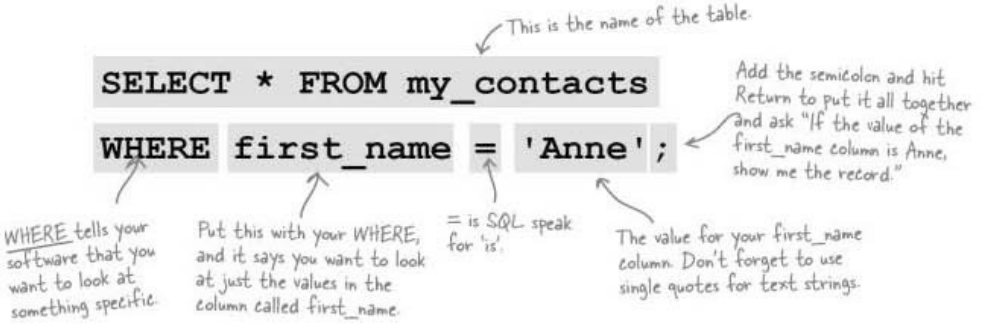
BRAIN POWER

Can you think of a way we could write a SQL query to display only those records that have a first_name of "Anne"?

A better SELECT

Here's a SELECT statement that would have helped Greg find Anne a whole lot sooner than painstakingly reading through the entire huge table looking for Annes. In the statement, we use a ***WHERE clause that gives the RDBMS something specific to search for***. It narrows down the results for us and **only returns the rows that match the condition.**

The equal sign in the WHERE clause is used to test whether each value in the column `first_name` equals, or matches, the text `'Anne'`. If it does, everything in the row is returned. If not, the row is not returned.



The console below shows you the rows that have been returned by this query, where the first name equals Anne.

```
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)
```

These are the results from our SELECT statement.

Wait a second, you're not going to sneak that * past me. What exactly does it do?

What the * is that?

That star is telling the RDBMS to give you back the values from **all** of the columns in your table.

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

When you see SELECT *, think of it like asking your SQL software to SELECT ALL THE COLUMNS.

Use the star to select all of the columns in your table.

there are no Dumb Questions

Q: What if I don't want to select all the columns? Can I use something else instead of the star?

A: Indeed you can. The star selects everything, but in a few pages you'll learn how to just select some of the columns, making your results easier to interpret.

Q: Is this star the same thing as an asterisk?

A: Yes, it's the same character on your keyboard, located above the 8 key. Hit SHIFT at the same time as the 8 to type one. This is the same for Mac and PC users.

But, although it's exactly the same character as asterisk, in SQL lingo, it's always referred to as *star*. This is a good thing, as saying "`SELECT asterisk from ...`" is not as easy as saying "`SELECT star from ...`"

Q: Are there other characters that have special meanings like the star does?

A: SQL does have other special, or reserved, characters. You'll see more of these later in the book. But the star is the only one you need to know about for right now. It's the only one used in the SELECT part of an SQL statement.

The Head First Lounge is adding mixed fruit drinks to its menu. Using what you learned in Chapter 1, create the table on this page and insert the data shown.

This table is part of a database called **drinks**. It contains the table **easy_drinks** with the recipes for a number of beverages that have only two ingredients.

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

amount1 and amount2 are in ounces.

Answer on page 117.

Before you start, do some planning.

Choose your data types carefully, and don't forget about NULL. Then ***check your code on page 117****.*

Sharpen your pencil

Don't worry about any characters in the queries you haven't seen yet. Just type them in as you see them for now, then see if they run.

NAME THAT DRINK

Use the **easy_drinks** table you just created and try out these queries on your machine. Write down which drinks are returned as the result of each query.

```
SELECT * FROM easy_drinks WHERE main = 'Sprite';
```

Which drink(s)? .....

```
SELECT * FROM easy_drinks WHERE main = soda;
```

Which drink(s)? .....

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

Which drink(s)? .....

```
SELECT * FROM easy_drinks WHERE second = "orange juice";
```

Which drink(s)? .....

```
SELECT * FROM easy_drinks WHERE amount1 < 1.5;
```

Which drink(s)? .....

```
SELECT * FROM easy_drinks WHERE amount2 < '1';
```

Which drink(s)? .....

```
SELECT * FROM easy_drinks WHERE main > 'soda';
```

Which drink(s)? .....

```
SELECT * FROM easy_drinks WHERE amount1 = '1.5';
```

Which drink(s)? .....

Yes, you're exactly right.

One of these queries won't work. The rest of them work, but the results of some aren't what you might expect.

For bonus points, write down here which query doesn't work...

.....

... and which ones worked that you didn't expect to.

.....

.....

.....

.....

Sharpen your pencil Solution

NAME THAT DRINK

You tried out these queries on your **easy_drinks** table and wrote down which drinks are returned as the result of each query.

```
SELECT * FROM easy_drinks WHERE main = 'Sprite';
```

Which drink(s)? Lime Fizz

Notice the single quotes.

```
SELECT * FROM easy_drinks WHERE main = soda;
```

Which drink(s)? Error

Hmm. Looks like this is the query that wouldn't run.

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

This one works. It's a DEC variable, so you don't use quotes at all.

Which drink(s)? Hot Gold, Indian Summer

```
SELECT * FROM easy_drinks WHERE second = "orange juice";
```

Which drink(s)? No results

```
SELECT * FROM easy_drinks WHERE amount1 < 1.5;
```

Which drink(s)? Oh My Gosh

```
SELECT * FROM easy_drinks WHERE amount2 < '1';
```

Which drink(s)? Blue Moon, Lime Fizz, Lone Tree

```
SELECT * FROM easy_drinks WHERE main > 'soda';
```

Another correctly formed WHERE clause.

Which drink(s)? Blackthorn, Lime Fizz

```
SELECT * FROM easy_drinks WHERE amount1 = '1.5';
```

Which drink(s)? Blackthorn, Blue Moon, Lime Fizz, Lone Tree, Greyhound, Bull Frog

For bonus points, write down here which query doesn't work...

This is the WHERE clause that won't work. You need quotes around that VARCHAR.

```
WHERE main = soda;
```

... and which ones worked that you didn't expect to?

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

This query works and doesn't cause an error, but returns no results. SQL expects you to use single quotes, as you did when you inserted the value.

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

But this one works, even though it shouldn't because DEC variables don't need quotes.

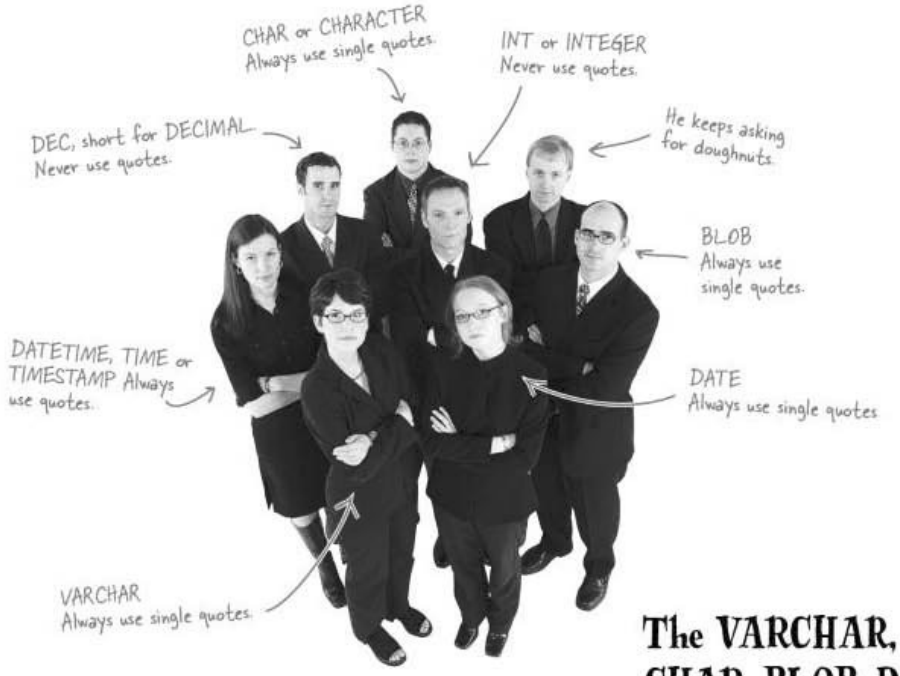
And so does this one!

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

These last two queries will work because most SQL RDBMSes give you a little latitude. They will ignore the quotes and treat your DEC and INT values as numbers, even though the quotes indicate they are text values. The queries are NOT CORRECT, but your RDBMS is forgiving.

How to query your data types

To write valid WHERE clauses, you need to make sure each of the data types you include is formatted properly. Here are the conventions for each of the major data types:



WE ♡ single quotes	No quotes for us
CHAR	DEC
VARCHAR	INT
DATE	
DATETIME, TIME, or TIMESTAMP	
BLOB	

The VARCHAR, CHAR, BLOB, DATE, and TIME data types need single quotes. The numeric types, DEC and INT, do not.

More punctuation problems

Greg picked up a few more contacts the other night. He's trying to add one to his table:


```
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');
```

But his program doesn't seem to be responding. He types a few semicolons, trying to get the query to end. No luck.

```
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');
'>
'> ;
'> ;
'>
```

Every time he hits Return, he sees this prompt: '>

What do you think is going on here?

Hmm, look at that single quote that keeps appearing before the prompt. I bet there's something wrong with the quotes in our INSERT statement...

Unmatched single quotes

Exactly! When Greg tried to add the record, the SQL program was expecting an even number of single quotes, one before and one after each VARCHAR, CHAR, and DATE value. The town name, **Grover's Mill**, confused matters because it added an extra apostrophe. The SQL RDBMS is still waiting for one more closing single quote.

You can get back control of your console.

End the statement by typing a single quote and a semicolon. This gives the RDBMS the extra single quote it's expecting.

You'll get an error when you type in the other quote and semicolon, and you'll have to enter your **INSERT** again from scratch.

You'll get an error after you do this, but at least you'll be able to try again.

```
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
>
```

Typing a single quote and semicolon ends the broken INSERT statement.

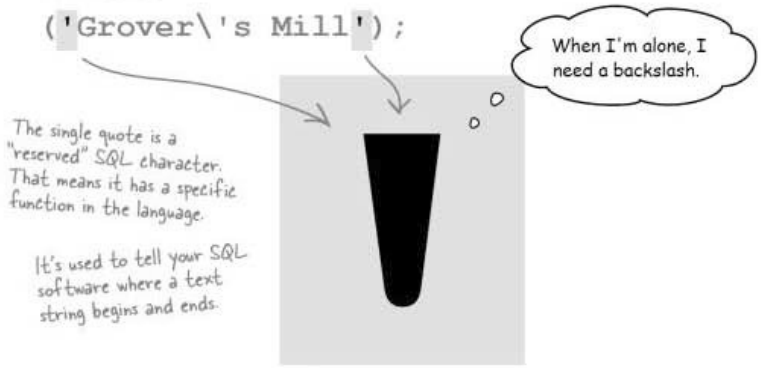
This error gives you a pretty clear idea of what's wrong. It quotes part of your query, beginning with the extra single quote.

Even though the record isn't inserted, that last > shows that at least the SQL program is responsive again.

Single quotes are special characters

When you're trying to insert a VARCHAR, CHAR, or BLOB containing an apostrophe, you must indicate to your RDBMS that it isn't meant to end the text, ***but is part of the text*** and needs to be included in the row. One way to do this is to **add a backslash** in front of the single quote.

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



there are no Dumb Questions

Q: Isn't this the same thing as an apostrophe?

A: It's exactly the same thing as an apostrophe. SQL assigns it a very specific meaning, however. It's used to tell the SQL software that the data in between two of them is text data.

Q: What data types need them?

A: The text data types. Text data simply means that the data is a VARCHAR, CHAR, BLOB, or TIMEDATE column. Anything that isn't a number.

Q: Do DEC and INT columns need them?

A: No. Numeric columns have no spaces, so it's easy to tell when the number ends and the next word in the statement begins.

Q: So, it's only used for text columns?

A: Yes. Only trouble is, text columns have spaces. This causes problems when your data contains apostrophes. SQL doesn't know how to tell the difference between an apostrophe within the column, and one that tells it when the column begins or ends.

Q: Couldn't we make it easy to tell them apart by using a double quote instead of a single quote?

A: No. Don't use double quotes in case you use SQL statements with a programming language (like PHP) later. You use " in the programming language to say "this is where the SQL statement is"; that way, single quotes are recognized as being part of that statement and not part of the programming language.

INSERT data with single quotes in it

You need to tell your SQL software that your quote isn't there to begin or end a text string, but that it's *part of* the text string.

Handle quotes with a backslash

You can do this (and fix your `INSERT` statement at the same time) by adding a backslash character in front of the single quote in your text string:

Telling SQL that a single quote is part of a text string by putting a backslash in front of it is called "escaping" it.

```
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');
```

Handle quotes with an extra single quote

Another way to "escape" the quote is to put an extra single quote in front of it.

Or you can "escape" a single quote with an extra single quote in front of it.

```
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');
```

What other characters might cause similar problems?

If you have data in your table with quotes, you might actually have to search for it with a WHERE clause at some point. To SELECT data containing single quotes in your WHERE clause, you need to escape your single quote, just like you did when you inserted it.

Rewrite the code below using the different methods of escaping the single quote.

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

1. .....

.....

.....

2. .....

.....

.....

Which method do you prefer?

If you have data in your table with quotes, you might actually have to search for it with a WHERE clause at some point. To SELECT data containing single quotes in your WHERE clause, you need to escape your single quote, just like you did when you inserted it.

Rewrite the code below using the different methods of escaping the single quote.

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

1

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

Method 1, the backslash.

2

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

Method 2, the extra single quote.

SELECT specific data

Now you've mastered how to SELECT all the data types with quotes, and how to SELECT data where the data contains quotes.

You need to know how to only SELECT the columns you wish to see.

What we need here is more precision. Let's try narrowing our results some. Narrowing our results means getting fewer columns in our output. We select only the columns we want to see.

Try this at home

Before you try this SELECT query, sketch how you think the table of results will look. (If you need to look at the easy_drinks table, you can find it on page 59.)

We've replaced the * with these column names.

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

Try this at home

Before you try this SELECT query, sketch how you think the table of results will look.

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

The old way

```
SELECT * FROM easy_drinks;
```

Here we get all the columns, and our results are too wide for our terminal window. They wrap to the next line and the display is a mess.

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 orange
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 specific columns to limit results

By specifying which columns we want returned by our query, we can choose only the column values we need. Just as you use a `WHERE` clause to limit the number of rows, you can use column selection to limit the number of columns. It's about letting SQL do the heavy lifting for you.

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

...but we can narrow our results by selecting only the columns we want to see show up in the results.

```
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 specific columns for faster results

This is a good programming practice to follow, but it has other benefits. As your tables get larger, it speeds up retrieval of your results. You'll also see more speed when you eventually use SQL with another programming language, such as PHP.

Sharpen your pencil

Many ways to get a Kiss on the Lips

Remember our easy_drinks table? This SELECT statement will result in a Kiss on the Lips:

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

Finish the other four SELECT statements on the next page to get a Kiss also.

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

```
SELECT .....
WHERE  .....

SELECT .....
WHERE  .....

SELECT .....
WHERE  .....

SELECT .....
WHERE  .....
```

Now write three SELECT statements that will give you a Bull Frog.

1. .....
   .....

2. .....
   .....

3. .....
   .....

Sharpen your pencil Solution

Finish the other four SELECT statements to get a Kiss also.

```
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';
```

This is one you'll seldom use, but it does give you the result you want. You might use something like this when you want to make sure your drink_name column doesn't have a typo.

Now write three SELECT statements that will give you a Bull Frog.

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';
```

BULLET POINTS

- Use single quotes in your WHERE clause when when selecting from text fields.
- Don't use single quotes when selecting from numeric fields.
- Use the * in your SELECT when you want to select all of the columns.
- If you've entered your query and your RDBMS doesn't finish processing it, check for a missing single quote.
- When you can, select specific columns in your table, rather than using SELECT *.

there are no Dumb Questions

Q: What if I need all the columns from my table returned by a query? Should I actually be naming them in the SELECT rather than using the *?

A: If you need them all, then by all means use the *. It's only when you don't need them all that you should try not to use it.

Q: I tried to copy and paste a query from the Internet, and I kept getting errors when I tried to use it. Am I doing something wrong?

A: Queries pasted from web browsers sometimes contain invisible characters that look like spaces but mean something different to SQL. Pasting them into a text editor is one way to see and remove these "gremlin" characters. Your best bet is to paste it into a text editor first and take a close look at it.

Q: So I should paste it into something like Microsoft Word?

A: No, Word isn't a good choice, since it does nothing to show you the invisible formatting that might be in the text. Try Notepad (PC) or TextEdit in plain-text mode (Mac).

Q: About escaping the apostrophe, is there any reason to use one method over the other?

A: Not really. We tend to use the backslash method only because we find that it's easier to spot where that extra apostrophe is when things go wrong in a query. For example, this is easier to process visually:

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

Than this:

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

Other than that, there's really no reason to favor one method over the other. Both methods allow you to enter apostrophes into your text columns.

Doughnut ask what your table can do for you...

To find the best glazed doughnut in the table, you need to do at least two SELECT statements. The first one will select rows with the correct doughnut type. The second will select rows with doughnuts with a rating of 10.

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

Imagine that this table contains 10,000 records.

1 One way is to search for the doughnut type:

You need to SELECT rating to search through the highest scores, and location because that gives you the name of the winner.

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

All of the results will be the correct type of doughnut

First query results, but imagine hundreds more.

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

Ask what you can do for your doughnut

2 Or you need to search for that high rating:

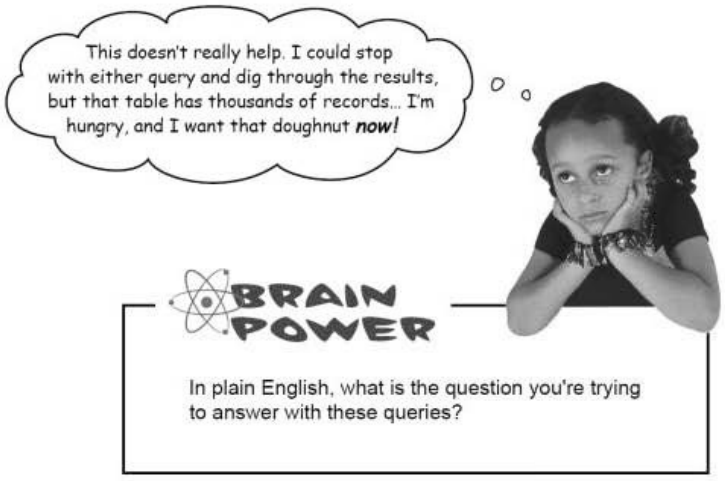
```
SELECT location, type FROM doughnut_ratings
WHERE
rating = 10;
```

You need to look through all the types, and then location will give you the name of your winner.

All of the results will be the highest rated.

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

Second query results, again, picture hundreds of these.



BRAIN POWER

In plain English, what is the question you're trying to answer with these queries?

Combining your queries

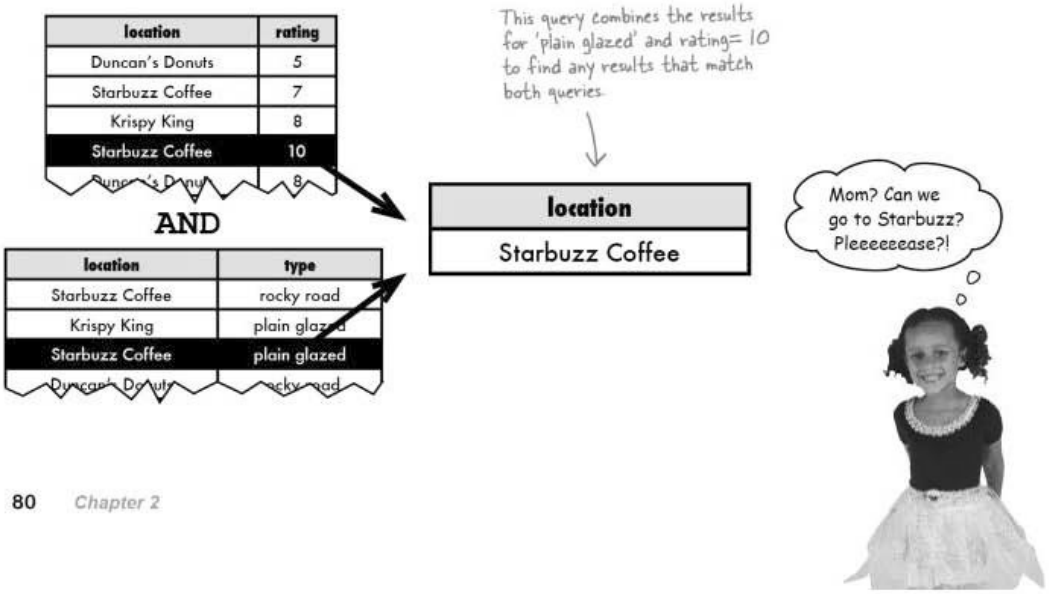
We can handle the two things we're searching for, 'plain glazed' for the type and 10 for the rating into a single query using the keyword AND. The results we get from the query must satisfy both conditions.

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

Now all we need to SELECT is the location.

Use the word AND to combine your two WHERE clauses.

Here's the result of the AND query. Even if we received more than one row as a result of our query, you would know that all locations have glazed doughnuts with a rating of 10, so you could go to any of them. Or all of them.



Exercise

So I could have found Anne using AND?

Using the my_contacts table, write some queries for Greg. SELECT only the columns you really need to give you your answer. Pay attention to single quotes.

Write a query to find the email addresses of all computer programmers.

Write a query to find the name and location of anyone with your birthdate.

Write a query to find the name and email of any single people who live in your town. For extra points, only pick those of the gender you'd want to date.

Write the query Greg could have used to find all the Annes from San Francisco.

Using the my_contacts table, write some queries for Greg. SELECT only the columns you really need to give you your answer. Pay attention to single quotes.

Write a query to find the email addresses of all computer programmers.

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

We want the email column.

The profession we want is computer programmer.

Write a query to find the name and location of anyone with your birthdate.

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

This should be your birthdate in quotes.

Write a query to find the name and email of any single people who live in your town. For extra points, only pick those of the gender you'd want to date.

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

Your town here.

The gender you wish to date here.

Write the query Greg could have used to find all the Annes from San Francisco.

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

Looking back at the table, Greg seems to have shortened San Francisco to San Fran. Hope he was consistent.

Finding numeric values

Let's say you want to find all the drinks in the `easy_drinks` table that contain more than an ounce of soda in *a single query*. Here's the hard way to find the results. You can use two queries:

We just want the names of the drinks.

Soda drinks with 1.5 ounces of soda.

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

```
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)
```

Soda drinks with 2 ounces of soda.

```
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 anc 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

Once is enough

But it's a waste of time to use two queries, and you might miss drinks with amounts like 1.75 or 3 ounces. Instead, you can use a **greater than** sign:

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

The GREATER THAN symbol will give you all the drinks that contain more than 1 ounce of soda.

```
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)
```

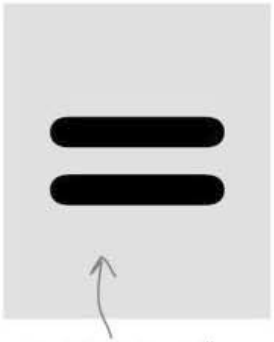
BRAIN POWER

Why can't you combine the first two queries with an additional AND?

~~Smooth~~ Comparison Operators

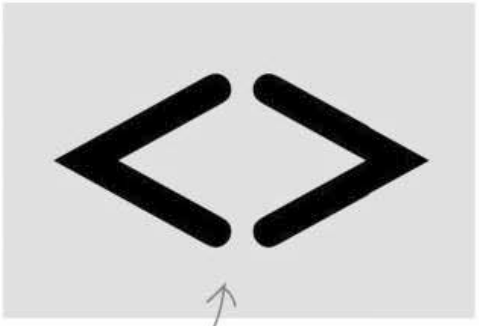
So far, we've only used the **equal** sign in our `WHERE` clause. You just saw the **greater than** symbol, >. What that does is compare one value against another. Here are the rest of the comparison operators:

The equal sign looks for exact matches. This does us no good when we want to find out if something is less than or greater than something else.



The EQUAL sign we all know and love.

This confusing sign is **not equal**. It returns precisely the opposite results of the equal sign. Two values are either equal, or they are not equal.



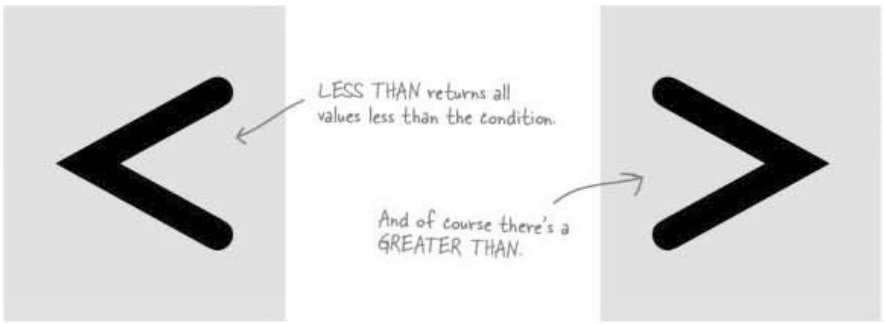
This one means NOT EQUAL. It returns all the records that don't match the condition.

BRAIN BARBELL

Have you noticed that every `WHERE` clause so far always has a column name on the left. Would it work if the column name was on the right?

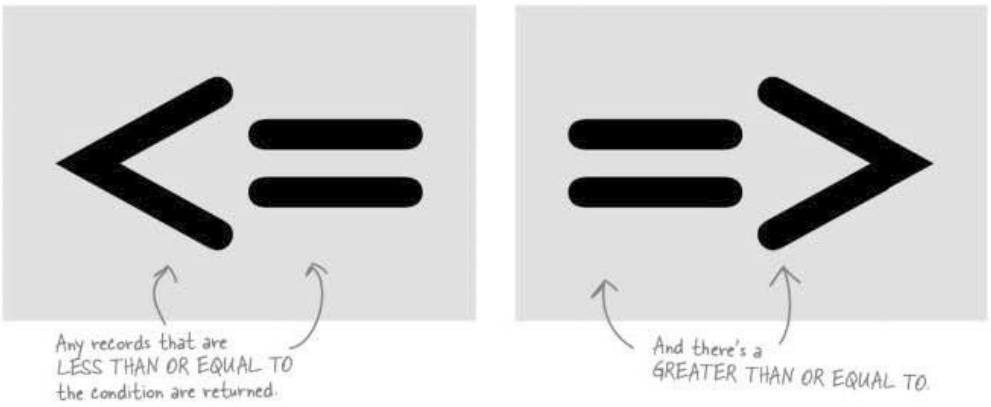
The **less than** sign looks at the values in the column on the left and compares them to the value on the right. If the column value is less than the value on the right, that row is returned.

The **greater than** sign is the reverse of the less than. It looks at the values in the column and compares them to the value on the right. If the column value is greater than the value on the right, that row is returned.



The only difference with the **less than or equal to** sign is that column values equal to the condition value are also returned.

Same thing with this **greater than or equal to sign**. If the column value matches or is greater than the condition value, the row is returned.



Finding numeric data with Comparison Operators

The Head First Lounge has a table with the cost and nutritional information about their drinks. They want to feature higher priced, lower calorie drinks to increase profits.

They're using comparison operators to find the drinks that are priced at least $3.50 and have less than 50 calories in the `drink_info` table.

The total carbohydrate grams in each drink.

The calories in each drink.

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;
```

This says: "find drinks that cost £3.50 or more" This includes drinks that cost exactly £3.50.

This says: "find drinks with calories less than 50".

This query only returns drinks where **both** of these conditions are met because of the AND combining the two results. The drinks that are returned are: Oh My Gosh, Lone Tree, and Soda and It.

Sharpen your pencil

Your turn to do some mixing. Write queries that will return the following information. Also write down what the result of each query is:

The cost of each drink with ice that is yellow and has more than 33 calories.

.....
.....
.....
.....
.....
.....

Result: .....

The name and color of each drink which does not contain more than 4 grams of carbs and uses ice.

.....
.....
.....
.....
.....
.....

Result: .....

The cost of each drink whose calorie count is 80 or more.

.....
.....
.....
.....
.....
.....

Result: .....

Drinks called Greyhound and Kiss on the Lips, along with each one's color and whether ice is used to mix the drink.

.....
.....
.....
.....
.....
.....

Result: .....

Sharpen your pencil Solution

Your turn to do some mixing. Write queries that will return the following information. Also write down what the result of each query is:

The cost of each drink with ice that is yellow and has more than 33 calories.

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

Result: £4.00

The name and color of each drink which does not contain more than 4 grams of carbs and uses ice.

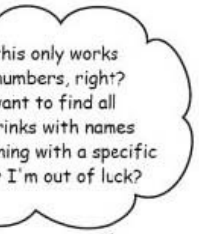
```
SELECT drink_name, color FROM drink_info
WHERE
carbs <= 4
AND
ice = 'Y';
```

Result: Blue Moon, blue

The cost of each drink whose calorie count is 80 or more.

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

Result: £5.50, £3.20, £2.60



Drinks called Greyhound and Kiss on the lips, along with each one's color and whether ice is used to mix the drink.

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

This one's tricky. You had to look through the table and find some column you could use to get those drinks and just those drinks.

Result: Kiss on the Lips, purple, Y
Greyhound, yellow, Y

Text data roping with Comparison Operators

Comparing text data works in a similar way with your text columns like CHAR and VARCHAR. The comparison operators evaluate everything **alphabetically**. So, say you want to select all the drinks that begin with an 'L', here's a query that will select all the drinks that match that criteria.

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';
```

This query returns drinks whose first letter is L or later, but whose first letters come earlier in the alphabet than M.

Don't worry about the order of your results for now.

In a later chapter we'll show you ways to sort your results alphabetically.

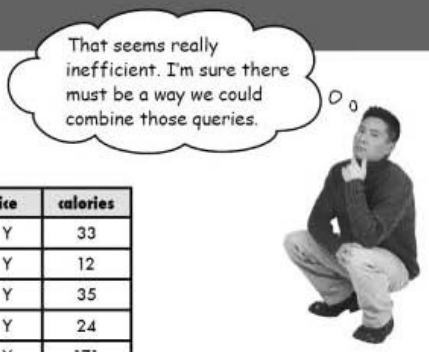
Selecting your ingredients

One of the bartenders has been asked to mix a cocktail that has cherry juice in it. The bartender could use two queries to find the cocktails:

Each query checks the two ingredients columns.

```
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)
```



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

To be OR not to be

You can combine those two queries using OR. This condition returns records when ***any*** of the conditions are met. So, instead of two the two separate queries, you can combine them with OR like this:

```
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)
```

Sharpen your pencil

Cross out the unnecessary parts of the two SELECTs below and add an OR to turn it into a single SELECT statement.

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


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

Use your new selection skills to rewrite your new SELECT.

.....

.....

Sharpen your pencil Solution

Cross out the unnecessary parts of the two SELECTs below and add an OR to turn it into a single SELECT statement.

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

We need to get rid of that semicolon so the statement doesn't end yet.

With this OR we get drink_names with main ingredients of orange juice OR apple juice.

We can simply cross out this line, we've already got this covered by the first part of the query (now joined by our OR.

Use your new selection skills to rewrite your new SELECT.

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

Here's the final query.

Don't get your ANDs and ORs confused!

When you want **ALL** of your conditions to be true, use **AND**.

When you want **ANY** of your conditions to be true, use **OR**.

Still confused? Turn the page.

AND

OR

there are no Dumb Questions

Q: Can you use more than one AND or OR in the same WHERE clause?

A: You certainly can. You can combine as many as you like. You can also use both AND and OR together in the same clause.

The difference between AND and OR

In the queries below you'll see examples of all the possible combinations of two conditions with AND and OR between them.

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                          RESULTS

WHERE location = 'Krispy King' AND rating = 10;            plain glazed

WHERE location = 'Krispy King' OR rating = 10;             plain glazed

WHERE location = 'Krispy King' AND rating = 3;             no results

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

WHERE location = 'Snappy Bagel' AND rating = 10;           no results

WHERE location = 'Snappy Bagel' OR rating = 10;            plain glazed

WHERE location = 'Snappy Bagel' AND rating = 3;            no results

WHERE location = 'Snappy Bagel' OR  rating = 3;            no results
```

Yes, there is a match

Yes

No matches

No match

BE the Conditional

Below, you'll find a series of WHERE clauses with ANDs and ORs. Become one with these clauses and determine whether or not they will produce results.

```
SELECT type FROM doughnut_ratings
```

Did you get a result?

```
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;
```

.....

To improve your karma, note down why two of your results are a bit different than all the rest.

BE the Conditional Solution

Below, you'll find a series of WHERE clauses with ANDs and ORs. Become one with these clauses and determine whether or not they will produce results.

```
SELECT type FROM doughnut_ratings
```

Did you get a result?

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

```
WHERE location  = 'Krispy King' AND  rating = 3;
```
no result

```
WHERE location  = 'Snappy Bagel' AND  rating >= 6;
```
no result

```
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

To improve your karma, note down why two of your results are a bit different than all the rest.

Two queries return NULL.

Those NULL values may cause you problems in future queries. It's better to enter some sort of value than leave a NULL value in a column ***because NULLs can't be directly selected from a table.***

Use IS NULL to find NULLs

I tried selecting NULL values directly, but it didn't work. How do I find the NULLs in my tables?

drink_info

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

You can't select a NULL value directly.

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

Won't work because nothing can be equal to NULL. It's an undefined value.

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

This won't work because NULL isn't the same thing as zero.

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

And this won't work either, because NULL isn't a text string.

But you can select it using keywords.

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

The only way to directly select a NULL value is to use the keywords IS NULL.

Keywords are not text strings, so they don't have quotes.

there are no Dumb Questions

Q: You say you can't "directly select" NULL without using IS NULL. Does that mean you can indirectly select it?

A: Right. If you wanted to get to the value in that column, you could use a WHERE clause on one of the other columns. For example, your result will be NULL if you use this query:

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

Q: What would my result from that query actually look like?

A: It would look exactly like this:

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

Meanwhile, back at Greg's place...

Greg's been trying to find all the people in California cities in his `my_contacts` table. Here's part of the query he's been working on:


```
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...
```

He knows he's entered SF at least these two ways. And what about typos?

Saving time with a single keyword: LIKE

There are simply too many cities and variations, and possible typos. Using all those ORs is going to take Greg a very long time. Luckily, there's a timesaving keyword—LIKE—that, used with a wildcard, looks for part of a text string and returns any matches.

Greg can use LIKE like this:

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

Place a percent sign inside the single quotes. This tells your software you're looking for all values in the location column that end with CA.

The call of the Wild(card)

LIKE teams up with two wildcard characters. Wildcards are stand-ins for the characters that are actually there. Rather like a joker in a card game, a wildcard is equal to any character in a string.

The call of the wild(card).

Wildcards are stand-in characters.

BRAIN BARBELL

Have you seen any other wildcards earlier in this chapter?

That's more LIKE it

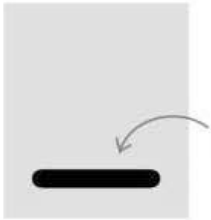
LIKE likes to play with wildcards. The first is the percent sign, %, which can stand in for any number of unknown characters.

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

The percent is a stand-in for any number of unknown characters.

Results in names with any number of characters before the "im", like Ephraim, Slim, and Tim.

The second wildcard character that LIKE likes to hang out with is the underscore, _ which stands for just one unknown character.



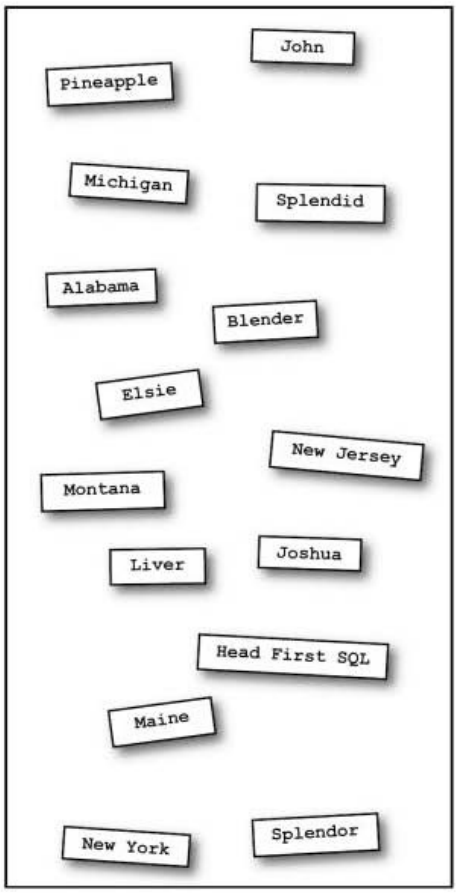
```
SELECT first_name FROM my_contacts
WHERE first_name LIKE '_im';
```

The underscore is a stand-in for just one unknown character.

Results in names with just one character before the "im", like Jim, Kim, and Tim.

Magnet Matching

A bunch of WHERE clauses with LIKE are all scrambled up on the fridge. Can you match up the clauses with their appropriate results? Some may have multiple answers. Write your own LIKE statements with wild cards for any results that are left hanging around.




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

```
WHERE cow_name LIKE '__sie';
```

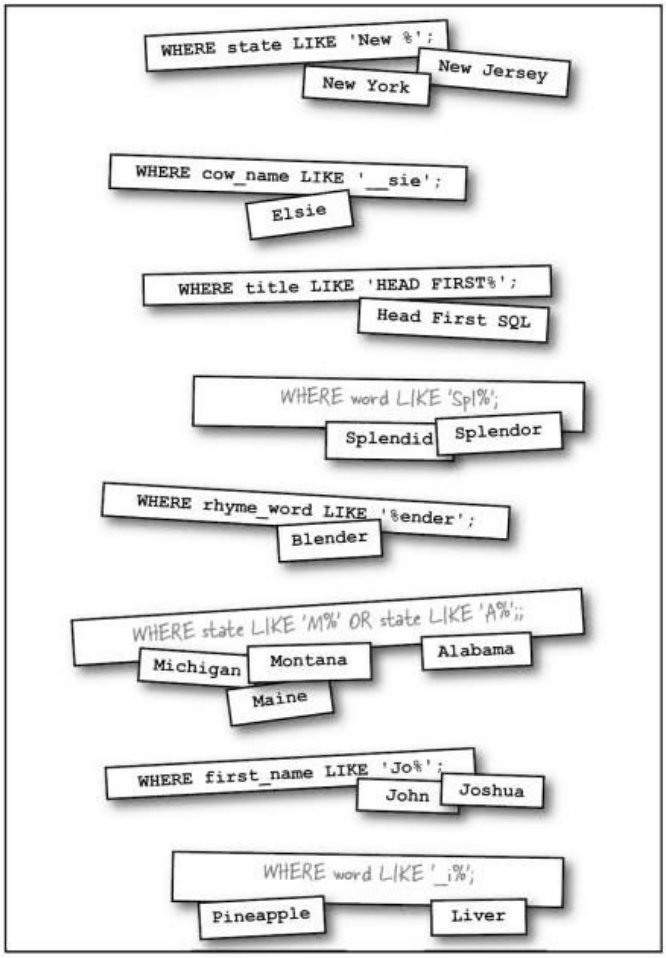
```
WHERE title LIKE 'HEAD FIRST%';
```

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

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

Magnet Matching Solutions

A bunch of WHERE clauses with LIKE are all scrambled up on the fridge. Can you match up the clauses with their appropriate results? Some may have multiple answers. Write your own LIKE statements with wild cards for any results that are left hanging around.



Selecting ranges using AND and comparison operators

The people at the Head First Lounge are trying to pinpoint drinks with a certain range of calories. How will they query the data to find the names of drinks that fall into the range of calories between, and including, 30 and 60?

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;
```

The results will include drinks with calories equal to 30, if there are any, as well as the drinks with 60 calories, as well as drinks with calorie counts in between.

Just BETWEEN us... there's a better way

We can use the BETWEEN keyword instead. Not only is it shorter than the previous query, but it gives you the same results. Notice that the endpoint (30 and 60) are also included. BETWEEN is equivalent to using the <= and >= symbols, but not the < and > symbols.

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

This will give you exactly the same results as the query on the previous page, but look how much quicker it is to type!

This includes the drinks with 30 and 60 calories.

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

Sharpen your pencil

Rewrite the query on the previous page to SELECT all the names of drinks that have more than 60 calories and less than 30.

.....

.....

.....

Try using BETWEEN on text columns. Write a query that will SELECT the names of drinks that begin with the letters G through O.

.....

.....

.....

What do you think the results of this query will be?

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

.....

.....

.....

Sharpen your pencil Solution

Rewrite the query on the previous page to SELECT all the names of drinks that have more than 60 calories and less than 30.

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

This gives us drink names with calories greater than 60.

and these are the ones with calories less than 30.

Try using BETWEEN on text columns. Write a query that will SELECT the names of drinks that begin with the letters G through O.

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

We'll get drink names that begin with G and O, and all the letters in between.

What do you think the results of this query will be?

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

Order matters, so you won't get any results from this query.

We're looking for values that are between 60 and 30. There are no values in between 60 and 30, because 60 comes after 30 numerically. The smaller number must always be first for the **BETWEEN** to be interpreted the way you expect.

After the dates, you are either IN...

Greg's friend Amanda has been using Greg's contacts to meet guys. She's gone on quite a few dates, and has started to keep a "little black book" table with her impressions of her dates.

She's named her table `black_book`. She wants to get a list of the good dates, so she uses her positive ratings.

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

You need a line for each positive score.

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

These are the positive ratings.

Instead of using all those ORs, we can simplify it with the keyword `IN`. Use IN with a set of values in parentheses. When the value in the column matches one of the values in the set, the row or specified colums are returned.

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

Using the keyword IN tells your RDBMS that a set of values is coming up.

This is the set of positive ratings.

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

... or you are NOT IN

Of course, Amanda wants to know who got the bad ratings so that if they call she can be washing her hair or otherwise engaged.

To find the names of those she didn't rate highly, we're going to add the keyword `NOT` to our `IN` statement. `NOT` gives you the opposite results, anything that doesn't match the set.


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

Using the keywords *NOT IN* tells your software that the results aren't in the set of terms.

The results of the *NOT IN* query are the people who didn't get positive ratings and won't get a second date, either.

```
File Edit Window Help BadDates
> SELECT date_name FROM black_book
WHERE

rating NOT IN ('innovative', 'fabulous',
'delightful', 'pretty good');

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

Why might you sometimes choose to use `NOT IN` rather than `IN`?

More NOT

You can use NOT with BETWEEN and LIKE just as you can with IN. The important thing to keep in mind is that **NOT goes right after WHERE** in your statement. Here are some examples.

```
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%';
```

When you use NOT with AND or OR, it goes right after the AND or OR.

there are no Dumb Questions

Q: Wait, you just said that NOT goes after WHERE. What about when you use NOT IN?

A: That's an exception. And even moving the NOT after WHERE will work. These two statements will give you exactly the same results:

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

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

Q: Would it work with <> the "not equal to" comparison operator?

A: You could, but it's a double negative. It would make much more sense to just use an equal sign. These two queries return the same results:

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

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

Q: How would it work with NULL?

A: Just like you might guess it would. To get all :he values that aren't NULL from a column, you could use this:

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

But this will also work:

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

Q: What about with AND and OR?

A: If you wanted to use it in and AND or OR clause, it would go right after that word, like this:

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

Rewrite each of the following WHERE clauses so they are as simple as possible. You can use AND, OR, NOT, BETWEEN, LIKE, IN, IS NULL, and the comparison operators to help you. Refer back to the tables used in this chapter.

```
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 = 'peach nectar'
OR main = '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%';
```

.....

.....

.....

Rewrite each of the following WHERE clauses so they are as simple as possible. You can use AND, OR, NOT, BETWEEN, LIKE, IN, IS NULL, and the comparison operators to help you. Refer back to the tables used in this chapter.

```
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 = 'peach nectar'
OR main = 'soda';
```

```
SELECT drink_name FROM easy_drinks
WHERE main BETWEEN 'P' AND 'S';
```

This will only work because we don't have any other main ingredients that satisfy the condition. If our table had pomegranate juice, this wouldn't work.

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

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

We never have negative calories, so we're safe with the greater than sign.

```
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 'B';
```

Your SQL Toolbox

You've got Chapter 2 under your belt and now you've added operators to your tool box. For a complete list of tooltips in the book, see Appendix iii.

`SELECT *`

Use this to select all the columns in a table.

`Escape with ' and \`

Escape out apostrophes in your text data with an extra apostrophe or backslash in front of it.

`= <> < > <= >=`

You've got a whole bunch of equality and inequality operators at your disposal. .

`IS NULL`

Use this to create a condition to test for that pesky *NULL* value.

`AND and OR`

With *AND* and *OR*, you can combine your conditional statements in your WHERE clauses for more precision.

`NOT`

NOT lets you negate your results and get the opposite values.

`BETWEEN`

Lets you select ranges of values.

`LIKE with % and _`

Use *LIKE* with the wildcards to search through parts of text strings.

Your new tools: operators!

Exercise Solution from page 59.

Greg wants to create a table of mixed drinks that bartenders can query for recipes for his speed-dating events. Using what you learned in Chapter 1, create the table on this page and insert the data shown.

This table is part of a database called **drinks**. It contains the table **easy_drinks** with the recipes for a number of beverages that have only two ingredients.

```
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));
```

It's a good idea to give yourself a few extra characters in case you ever need to enter a name that's longer than the existing ones.

```
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');
```

Don't forget: numeric data types don't need quotes!

Each drink's set of values is in parentheses.

And between each drink is a comma.