

6 advanced SELECT

✧ *Seeing your data with new eyes* ✧

And then I was able to see just the enemy planes using a CASE statement! Kapow!



It's time to add a little finesse to your toolbox. You already know how to SELECT data and use WHERE clauses. But sometimes you need more **precision** than SELECT and WHERE provide. In this chapter, you'll learn about how to **order and group** your data, as well as how to **perform math operations** on your results.

Dataville Video is reorganizing

The owner of Dataville Video has a badly organized store. In his current system, movies can end up on different shelves depending on which employee is shelving them. He's ordered new shelves, and he thinks it's great time to finally label each of his movie categories.



In the current system, true and false values are used for types of movies. This makes figuring out how to categorize difficult. For example, if a movie has both T for comedy and T for scifi, where should it be shelved?

```
CREATE TABLE movie_table (  
  movie_id INT UNSIGNED AUTO_INCREMENT,  
  title VARCHAR(50),  
  rating VARCHAR(5),  
  drama CHAR(1) DEFAULT 'F',  
  comedy CHAR(1) DEFAULT 'F',  
  action CHAR(1) DEFAULT 'F',  
  gore CHAR(1) DEFAULT 'F',  
  scifi CHAR(1) DEFAULT 'F',  
  for_kids CHAR(1) DEFAULT 'F',  
  cartoon CHAR(1) DEFAULT 'F',  
  purchased DATE);
```

organize our movies. we can use the categories:

- Action & Adventure
- Drama
- Comedy
- Family
- Horror
- SciFi & Fantasy
- Misc

I'll leave it to you to figure out how to make our current table work with these new categories.

Let's do lunch,

Your boss

"T" and "F" are short for True and False.

movie_table

This is when the store acquired a copy.

movie_id	title	rating	drama	comedy	action	gore	scifi	for_kids	cartoon	purchased
1	Monsters, Inc.	G	F	T	F	F	F	T	T	3-6-2002
2	The Godfather	R	F	F	T	T	F	F	F	2-5-2001
3	Gone with the Wind	G	T	F	F	F	F	F	F	11-20-1999
4	American Pie	R	F	T	F	F	F	F	F	4-19-2003
5	Nightmare on Elm Street	R	F	F	T	T	F	F	F	4-19-2003
6	Casablanca	PG	T	F	F	F	F	F	F	2-5-2001

All these columns exist so that we can answer customer questions about the content of an individual movie.

Insert the table information from page

```
CREATE TABLE movie_table (  
  movie_id INT UNSIGNED AUTO_INCREMENT,  
  title VARCHAR(50),  
  rating VARCHAR(5),  
  drama CHAR(1) DEFAULT 'F',  
  comedy CHAR(1) DEFAULT 'F',  
  action CHAR(1) DEFAULT 'F',  
  gore CHAR(1) DEFAULT 'F',  
  scifi CHAR(1) DEFAULT 'F',  
  for_kids CHAR(1) DEFAULT 'F',  
  cartoon CHAR(1) DEFAULT 'F',  
  purchased DATE,  
  PRIMARY KEY(movie_id)  
);
```

Insert all of this information into the movie_table. Do not worry about the movie_id for the second table, just auto increment this field to the next value i.e. 7, 8, 9, etc. When you have completed inserting your information, please make a backup of your file.

for true and false. movie_table This is when the store acquired a copy.

movie_id	title	rating	drama	comedy	action	gore	scifi	for_kids	cartoon	purchased
1	Monsters, Inc.	G	F	T	F	F	F	T	T	3-6-2002
2	The Godfather	R	F	F	T	T	F	F	F	2-5-2001
3	Gone with the Wind	G	T	F	F	F	F	F	F	11-20-1999
4	American Pie	R	F	T	F	F	F	F	F	4-19-2003
5	Nightmare on Elm Street	R	F	F	T	T	F	F	F	4-19-2003
6	Casablanca	PG	T	F	F	F	F	F	F	2-5-2001

title	rating	drama	comedy	action	gore	scifi	for_kids	cartoon	purchased
Big Adventure	G	F	F	F	F	F	T	F	3-6-2002
Greg: The Untold Story	PG	F	F	T	F	F	F	F	2-5-2001
Mad Clowns	R	F	F	F	T	F	F	F	11-20-1999
Paraskavedekatriaphobia	R	T	T	T	F	T	F	F	4-19-2003
Rat named Darcy, A	G	F	F	F	F	F	T	F	4-19-2003
End of the Line	R	T	F	F	T	T	F	T	2-5-2001
Shiny Things, The	PG	T	F	F	F	F	F	F	3-6-2002
Take it Back	R	F	T	F	F	F	F	F	2-5-2001
Shark Bait	G	F	F	F	F	F	T	F	11-20-1999
Angry Pirate	PG	F	T	F	F	F	F	T	4-19-2003
Potentially Habitable Planet	PG	F	T	F	F	T	F	F	2-5-2001

All these columns exist so that we can answer customer questions about the content of an individual movie.

Problems with our current table

Here's a rundown of the problems Dataville Video has with the current table.

When movies are returned, we don't know where they belong.

If we have T values for a number of the columns in the table, there's no clear way to know where that movie needs to be shelved. Movies should always be associated with a **single category**.

People aren't clear what the movie is about.

Our customers get confused when they spot a gory cover in the comedy section. Currently none of our T/F values take precedence over any others when movies are shelved.

Adding True and False data is time-consuming, and mistakes often happen.

Every time a new movie comes in, it has to be inserted with all those T/F columns. And the more of those that get entered, the more errors that crop up. Sometimes a column that should have been T is accidentally entered as F, and vice versa. A category column would help us double-check our T/F columns, and eventually we might be able to get rid of those T/Fs altogether.

What we need here is a category column to speed up shelving, help customers figure out what type of movie it is they're renting, and limit errors in our data.



How would you reorganize the current columns into new **categories**? Are there any films that might fit into more than one of the new categories?

Matching up existing data

You know how to ALTER your table to add in the new category column, but adding in the actual categories is a bit trickier. Luckily, the data that's already in the table can help us figure out the category for each movie, without us actually having to watch each one.

Let's rewrite the relationships in simple sentences:

ALTER TABLE movie_table
ADD COLUMN category VARCHAR(10)
AFTER rating;
see page 200

- If this column is 'T': **drama** we set the category column to 'drama'
- If this column is 'T': **comedy** we set the category column to 'comedy'
- If this column is 'T': **action** we set the category column to 'action'
- If this column is 'T': **gore** we set the category column to 'horror'
- If this column is 'T': **scifi** we set the category column to 'scifi'
- If this column is 'T': **for_kids** we set the category column to 'family'
- If this column is 'T': **cartoon** and this column is 'G': **rating** we set the category column to 'family'
- If this column is 'T': **cartoon** and this column is NOT 'G': **rating** we set the category column to 'misc'

Not all cartoons are for kids. The rating column helps you determine if a film is in the family category or not, depending on whether it's true or false. If the rating is G, we can call it 'family'; if not, we'll call it 'misc'.

Populating the new column

Now we can translate those sentences into SQL UPDATE statements:

```
UPDATE movie_table SET category = 'drama' where drama = 'T';
UPDATE movie_table SET category = 'comedy' where comedy = 'T';
UPDATE movie_table SET category = 'action' where action = 'T';
UPDATE movie_table SET category = 'horror' where gore = 'T';
UPDATE movie_table SET category = 'scifi' where scifi = 'T';
UPDATE movie_table SET category = 'family' where for_kids = 'T';
UPDATE movie_table SET category = 'family' where cartoon = 'T' AND rating = 'G';
UPDATE movie_table SET category = 'misc' where cartoon = 'T' AND rating <> 'G';
```

Rating is not equal to 'G'.



Sharpen your pencil

Fill in the category value for these movies.

movie_table

title	rating	drama	comedy	action	gore	scifi	for_kids	cartoon	category
Big Adventure	G	F	F	F	F	F	T	F	
Greg: The Untold Story	PG	F	F	T	F	F	F	F	
Mad Clowns	R	F	F	F	T	F	F	F	
Paraskavedekatriaphobia	R	T	T	T	F	T	F	F	
Rat named Darcy, A	G	F	F	F	F	F	T	F	
End of the Line	R	T	F	F	T	T	F	T	
Shiny Things, The	PG	T	F	F	F	F	F	F	
Take it Back	R	F	T	F	F	F	F	F	
Shark Bait	G	F	F	F	F	F	T	F	
Angry Pirate	PG	F	T	F	F	F	F	T	
Potentially Habitable Planet	PG	F	T	F	F	T	F	F	

Does the order in which we evaluate each of the T/F columns matter?



Sharpen your pencil Solution

Fill in the category value for these movies.

movie_table

title	rating	drama	comedy	action	gore	scifi	for_kids	cartoon	category
Big Adventure	G	F	F	F	F	F	T	F	family
Greg: The Untold Story	PG	F	F	T	F	F	F	F	action
Mad Clowns	R	F	F	F	T	F	F	F	horror
Paraskavedekatriaphobia	R	T	T	T	F	T	F	F	?
Rat named Darcy, A	G	F	F	F	F	F	T	F	family
End of the Line	R	T	F	F	T	T	F	T	?
Shiny Things, The	PG	T	F	F	F	F	F	F	drama
Take it Back	R	F	T	F	F	F	F	F	comedy
Shark Bait	G	F	F	F	F	F	T	F	family
Angry Pirate	PG	F	T	F	F	F	F	T	?
Potentially Habitable Planet	PG	F	T	F	F	T	F	F	?

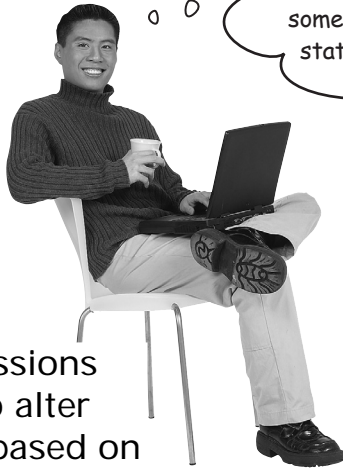
The question marks mean a column was changed by more than one UPDATE. This value will change depending on the order the UPDATES were executed.

Does the order in which we evaluate each of the T/F columns matter? Yes, it does matter.

The order does matter

For example, if we go through the columns in order 'Paraskavedekatriaphobia' would end up being classified as scifi, even though it might be more of a comedy. We don't know if it should be considered comedy, action, drama, cartoon, or scifi. Since it's unclear where it belongs, it might best be placed in the misc category.

Order matters.
Two UPDATE
statements may
change the same
column's value.



That seems fine for a small table, but what if you had hundreds of columns? Is there some way we could combine all those UPDATE statements into one big one?

Well, you could write one big UPDATE statement, but there's a better way.

The CASE expression combines all the UPDATE statements by checking an existing column's value against a condition. If it meets the condition, the new column is filled with a specified value.

It even allows you to tell your RDBMS *what to do if any records don't meet the conditions*

Here's its basic syntax:

UPDATE my_table

SET new_column =

CASE

→ WHEN column1 = somevalue1

→ THEN newvalue1

→ WHEN column2 = somevalue2

→ THEN newvalue2

ELSE newvalue3

END ;

The value in the column you specify here will be changed to the appropriate value below.

This begins the CASE expression.

WHEN this condition is met...

THEN set the value of new_column to this value.

WHEN a different condition is met...

THEN set the value of new_column to this different value.

This ends the CASE expression and the entire UPDATE statement (because it's followed by a semicolon).

Anything that doesn't match either of the conditions gets this value instead.

The indenting doesn't do anything to the expression; it just makes it easier to track what's going on when you look at the code.

CASE expressions allow you to alter the output based on a logical condition, as it applies to an evaluation of specific columns.

CHAPTER 6

COLUMN-BASED LOGIC

KEYWORDS INTRODUCED: **CASE**, **WHEN**, **THEN**, **ELSE**, **END**



The main topic of this chapter is something called the CASE expression. As indicated by the title of this chapter, CASE expressions are a form of column-based logic. That term is meant to indicate that these expressions apply logic to columns rather than rows. CASE expressions are also sometimes referred to as *conditional logic*. Basically, CASE expressions allow you to alter the output you present to a user based on a logical condition, as it applies to an evaluation of specific columns or data elements.

As a beginning SQL developer, you should know that the CASE expression is a relatively advanced topic. You can get by without ever using CASE expressions and still write some useful queries. But your knowledge of this topic is the type of thing that can really set you apart. In fact, after you've gone through the entire book, this is one of the topics you may want to review again, to get you thinking about some of the interesting things that can be accomplished with this technique.

IF-THEN-ELSE Logic

Let's turn to some honest-to-goodness logic next. Up until now, you've learned how to select columns from a single table, apply some calculations and functions, and add a sort. However, you have not yet done anything all that logical.

The `CASE` expression in SQL enables you to apply a traditional `IF-THEN-ELSE` type of logic to a single expression in a `SELECT` statement. The term *IF-THEN-ELSE* refers to a commonly used logical construct employed by procedural programming languages. In general terms, this type of logic looks like:

```
IF some condition is true
THEN do this
ELSE do that
```

A `CASE` expression is a construct that can appear in a number of places in a `SELECT` statement. In this chapter, we're going to focus on `CASE` expressions that appear within the *columnlist* immediately following the `SELECT` keyword. A `SELECT` statement that includes both columns and a `CASE` expression might look like this:

```
SELECT
column1,
column2,
CaseExpression
FROM table
```

The Simple Format

There are two general formats for the `CASE` expression, generally referred to as *simple* and *searched*. The simple format is:

```
SELECT
CASE ColumnOrExpression
WHEN value1 THEN result1
WHEN value2 THEN result2
(repeat WHEN-THEN any number of times)
[ELSE DefaultResult]
END
```

As can be seen, the `CASE` expression utilizes a number of keywords other than `CASE`: `WHEN`, `THEN`, `ELSE`, and `END`. These additional keywords are needed to fully define the logic of the `CASE` expression. The `WHEN` and `THEN` keywords define a condition that is evaluated. If the value after the `WHEN` is true, then the result after `THEN` is utilized. The `WHEN` and `THEN` keywords can be repeated any number of times. When there is a `WHEN`, there must also be a corresponding `THEN`. The `ELSE` keyword is used to define a default value to be used if none of the `WHEN-THEN` conditions is true. As indicated by the brackets, the `ELSE`

keyword is not required. However, it is generally a good idea to include the `ELSE` keyword in every `CASE` expression, so as to explicitly state a default value.

Let's look at a specific example, using this `Products` table:

ProductID	CategoryCode	ProductDescription
1	F	Apple
2	F	Orange
3	S	Mustard
4	V	Carrot

A `SELECT` with a `CASE` expression against data in this table might look like:

```
SELECT
CASE CategoryCode
WHEN 'F' THEN 'Fruit'
WHEN 'V' THEN 'Vegetable'
ELSE 'Other'
END AS 'Category',
ProductDescription AS 'Description'
FROM Products
```

The `AS` are alias for column names. It has not been discussed

and produces this output:

Category	Description
Fruit	Apple
Fruit	Orange
Other	Mustard
Vegetable	Carrot

Let's look at the previous `SELECT` statement line by line. The first line contains the `SELECT` keyword. The second line, with the `CASE` keyword, tells you that the `CategoryCode` column is to be analyzed. The third line introduces the first `WHEN-THEN` condition. This line says that if the `CategoryCode` column equals `F`, then the value to display should be "Fruit". The next line says that if it's `V`, then display "Vegetable". The `ELSE` line provides a default value of "Other" to use if the `CategoryCode` is not `F` or `V`. The `END` line terminates the `CASE` statement and also includes an `AS` keyword to provide a column alias for the `CASE` expression.

The next line with `ProductDescription` is merely another column and has nothing to do with the `CASE` expression.

The `CASE` expression is very useful for translating cryptic values into meaningful descriptions. In this example, the `CategoryCode` column in the `Products` table contains only a single character code to indicate the type of product. It's using `F` to denote Fruit, `V` for Vegetables, `S` for Spices, and so on. The `CASE` clause allows you to specify the translation.

The Searched Format

The general format for the searched `CASE` expression is:

```
CASE
WHEN condition1 THEN result1
WHEN condition2 THEN result2
repeat WHEN-THEN any number of times)
[ELSE DefaultResult]
END
```

The equivalent of the above `SELECT` statement using this second format is:

```
SELECT
CASE
WHEN CategoryCode = 'F' THEN 'Fruit'
WHEN CategoryCode = 'V' THEN 'Vegetable'
ELSE 'Other'
END AS 'Category',
ProductDescription AS 'Description'
FROM Products
```

The data retrieved is identical to the first format. Notice the subtle differences. In the simple format, the column name to be evaluated is placed after the `CASE` keyword, and the expression following the `WHEN` is a simple literal value. In the searched format, a column name to be evaluated is not put next to the `CASE` keyword. Instead, this format allows for a more complex conditional expression following the `WHEN` keyword.

For the previous example, either format of the `CASE` clause can be used, and it will produce the same result. Let's now look at another example for which only the searched format will yield the desired result.

This next example will be taken from this data:

ProductID	Fruit	Vegetable	Spice	ProductDescription
1	X			Apple
2	X			Orange
3			X	Mustard
4		X		Carrot

In this situation, the database contains multiple columns to indicate whether the product is a fruit, vegetable, or spice. A CASE expression to create the same output for this data is:

```
SELECT
CASE
WHEN Fruit = 'X' THEN 'Fruit'
WHEN Vegetable = 'X' THEN 'Vegetable'
ELSE 'Other'
END AS 'Category',
ProductDescription AS 'Description'
FROM Products
```

Once again, the result is:

Category	Description
Fruit	Apple
Fruit	Orange
Other	Mustard
Vegetable	Carrot

Since the data now uses three separate columns to indicate if the product is a fruit, vegetable, or spice, you need to use the searched format of the CASE clause in order to apply the needed logic. The simple format only works with an analysis of a single column.

Due to the inherent complexity of IF-THEN-ELSE logic, the CASE expression is one of the more challenging topics in this book. In this chapter, we have focused on using CASE expressions in the *SELECT columnlist*. However, CASE expressions can also be utilized in other SQL clauses, such as the ORDER BY clause, and other clauses not yet discussed, such as the WHERE and HAVING clauses.

Let's give just one example of additional uses of the CASE expression. Although we have not yet talked about the WHERE clause, let's imagine that we know something about it. As will be explained in Chapter 7, the WHERE clause allows you to apply selection criteria to the rows that will be presented to the user. A typical expression might be something like:

```
WHERE ProductDescription = 'White Glove'
```

This is a very specific directive. You only want to see rows where the product is a white glove. The value of the CASE expression is that it allows you to apply conditional logic to the value you're looking for, perhaps based on the value of some other column. For example, you may have another column, named `ProductType`, which gives more information about products. Using a CASE expression, you can select products that are white gloves if the `ProductType` equals X, or products that are socks if the `ProductType` equals Y. In essence, you can substitute a CASE expression for the value 'White Glove' in the WHERE clause in order to describe some more complex logic.

Looking Ahead

CASE expressions can be utilized to provide a logical evaluation for a column or expression in a `SELECT columnlist`. There are two basic formats for the expression: the simple and the searched. A typical use is to provide translations for data items with cryptic values. Finally, although this chapter is titled "Column-Based Logic," CASE expressions can be used in places other than the columns in a `SELECT columnlist`. They can be used anywhere where you would like to specify conditional logic for a specific column or data element.

UPDATE with a CASE expression

Let's see the CASE expression in action on our `movie_table`.

```
UPDATE movie_table
SET category =
CASE
  WHEN drama = 'T' THEN 'drama'
  WHEN comedy = 'T' THEN 'comedy'
  WHEN action = 'T' THEN 'action'
  WHEN gore = 'T' THEN 'horror'
  WHEN scifi = 'T' THEN 'scifi'
  WHEN for_kids = 'T' THEN 'family'
  WHEN cartoon = 'T' THEN 'family'
  ELSE 'misc'
END;
```

This is the same as saying `UPDATE movie_table SET category = 'drama' WHERE drama = 'T'`—but with a whole lot less typing!

Everything that doesn't match the conditions in the lines above is given a category value of 'misc'.

The values that were unknown when we used `UPDATE` on its own to populate the new column now have category values.

But notice how we also have new values for 'Angry Pirate' and 'End of the Line'.

movie_table

title	rating	drama	comedy	action	gore	scifi	for_kids	cartoon	category
Big Adventure	PG	F	F	F	F	F	F	T	family
Greg: The Untold Story	PG	F	F	T	F	F	F	F	action
Mad Clowns	R	F	F	F	T	F	F	F	horror
Paraskavedekatriaphobia	R	T	T	T	F	T	F	F	drama
Rat named Darcy, A	G	F	F	F	F	F	T	F	family
End of the Line	R	T	F	F	T	T	F	T	drama
Shiny Things, The	PG	T	F	F	F	F	F	F	drama
Take it Back	R	F	T	F	F	F	F	F	comedy
Shark Bait	G	F	F	F	F	F	T	F	family
Angry Pirate	PG	F	T	F	F	F	F	T	comedy
Potentially Habitable Planet	PG	F	T	F	F	T	F	F	comedy

As each movie title's T/F values are run through the CASE statement, the RDBMS is looking for the first 'T' to set the category for each film.

Here's what happens when 'Big Adventure' runs through the code:

```

UPDATE movie_table
SET category =
CASE
  WHEN drama = 'T' THEN 'drama'
  WHEN comedy = 'T' THEN 'comedy'
  WHEN action = 'T' THEN 'action'
  WHEN gore = 'T' THEN 'horror'
  WHEN scifi = 'T' THEN 'scifi'
  WHEN for_kids = 'T' THEN 'family'
  WHEN cartoon = 'T' THEN 'family'
  ELSE 'misc'
END;

```

FALSE: no category yet
 FALSE: no category yet
 FALSE: no category yet
 FALSE: no category yet
 FALSE: no category yet
 FALSE: no category yet
 TRUE: category set to 'family', and we skip to the END and exit the code.

Let's do one with multiple matches. Again, we're looking for the first 'T' value here to set the category.

Here's what happens when 'Paraskavedekatriaphobia' runs through the code:

```

UPDATE movie_table
SET category =
CASE
  WHEN drama = 'T' THEN 'drama'
  WHEN comedy = 'T' THEN 'comedy'
  WHEN action = 'T' THEN 'action'
  WHEN gore = 'T' THEN 'horror'
  WHEN scifi = 'T' THEN 'scifi'
  WHEN for_kids = 'T' THEN 'family'
  WHEN cartoon = 'T' THEN 'family'
  ELSE 'misc'
END;

```

TRUE: category set to drama; we skip to the END and exit the code. All our other T values are ignored.

Looks like we have a problem

We may have a problem. 'Great Adventure' is an R-rated cartoon. Somehow it ended up categorized as 'family'.

MESSAGE

Date Today Time 13.41

To The Boss

WHILE YOU WERE OUT

Really angry customer

Telephoned	<input checked="" type="checkbox"/>	Please call	<input checked="" type="checkbox"/>
Called to see you	<input type="checkbox"/>	Will call again	<input type="checkbox"/>
Wants to see you	<input type="checkbox"/>	Returned your call	<input type="checkbox"/>

MESSAGE Some lady called to complain that her little kid Nathan ended up watching a cartoon with a lot of profanity, and now he keeps chasing around his sister and calling her a %#!@

Taken By Me URGENT

☒



Change the CASE expression so that cartoons get put in the 'misc' category, not 'family'. If a cartoon has a G rating, put it in the family category.

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....



How might we use the R rating to keep this sort of thing from happening in the future?



Sharpen your pencil Solution

Change the CASE expression to test for the conditions that set a cartoon to 'misc' instead of 'family'. If a cartoon has a G rating, put it in the family category.

```
UPDATE movie_table
SET category =
CASE
  WHEN drama = 'T' THEN 'drama'
  WHEN comedy = 'T' THEN 'comedy'
  WHEN action = 'T' THEN 'action'
  WHEN gore = 'T' THEN 'horror'
  WHEN scifi = 'T' THEN 'scifi'
  WHEN for_kids = 'T' THEN 'family'
  WHEN cartoon = 'T' AND rating = 'G' THEN 'family'
  ELSE 'misc'
END;
```

Your condition can have multiple parts: add an AND to your WHEN to test for whether the film is a cartoon AND it's rated 'G'. If it is, then it gets a category of 'family'.

there are no Dumb Questions

Q: Do I have to use the ELSE?

A: It's optional. You can simply leave that line out if you don't need it, but it's nice to have to update the value of your column when nothing else fits. It's better to have some sort of value than NULL, for example.

Q: What happens if I leave off the ELSE but none of the WHEN conditions match?

A: No values will be changed in the column you are updating.

Q: What if I want to only use the CASE expression on some columns but not others? For example, if I wanted to do a CASE where my category = 'misc'. Can I use a WHERE?

A: Yes, you can add a WHERE clause after the END keyword. The CASE will only apply to those columns that match the WHERE.

Q: Can I use a CASE expression with anything other than UPDATE statements?

A: Yes. You can use a CASE expression with SELECT, INSERT, DELETE, and, as you've seen, UPDATE.

CASE CONSTRUCTION

Your boss, always a bit wishy-washy, has decided to change things up a bit. Read his email and write a single SQL statement that will accomplish what he wants.

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

It turns out that the new categories are causing customers to have a tough time finding movies. Write a statement that gets rid of the new R-rated categories you just created.

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

Finally, delete all those T/F columns we don't need anymore.

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

To: Dataville Video Staff
 From: The Boss
 Subject: New sections mean new categories!

My happy video family,

I've decided to create some new sections. I'm thinking that R-rated movies should be shelved in a different section than G and PG. Let's just create 5 new categories:

horror-r
 action-r
 drama-r
 comedy-r
 scifi-r

And if there are any G-rated movies in the misc section, move 'em to Family.

Thanks. That'll be great,
 Your boss

CASE CONSTRUCTION SOLUTION

Your boss, always a bit wishy-washy, has decided to change things up a bit. Read his email and write a single SQL statement that will accomplish what he wants.

```
UPDATE movie_table
SET category =
CASE
  WHEN drama = 'T' AND rating = 'R' THEN 'drama-r'
  WHEN comedy = 'T' AND rating = 'R' THEN 'comedy-r'
  WHEN action = 'T' AND rating = 'R' THEN 'action-r'
  WHEN gore = 'T' AND rating = 'R' THEN 'horror-r'
  WHEN scifi = 'T' AND rating = 'R' THEN 'scifi-r'
  WHEN category = 'misc' AND rating = 'G' THEN 'family'
END;
```

It turns out that the new categories are causing customers to have a tough time finding movies. Write a statement that gets rid of the new R-rated categories you just created.

```
UPDATE movie_table
SET category =
CASE
  WHEN category = 'drama-r' THEN 'drama'
  WHEN category = 'comedy-r' THEN 'comedy'
  WHEN category = 'action-r' THEN 'action'
  WHEN category = 'horror-r' THEN 'horror'
  WHEN category = 'scifi-r' THEN 'scifi'
END;
```

Finally, delete all those T/F columns we don't need anymore.

```
ALTER TABLE movie_table
DROP COLUMN drama,
DROP COLUMN comedy,
DROP COLUMN action,
DROP COLUMN gore,
DROP COLUMN scifi,
DROP COLUMN for_kids,
DROP COLUMN cartoon;
```

To: Dataville Video Staff
 From: The Boss
 Subject: New sections mean new categories!

My happy video family,

I've decided to create some new sections. I'm thinking that R-rated movies should be shelved in a different section than G and PG. Let's just create 5 new categories:

horror-r
 action-r
 drama-r
 comedy-r
 scifi-r

And if there are any G-rated movies in the misc section, move 'em to Family.

Thanks. That'll be great,
 Your boss

Tables can get messy

When a movie arrives at the store, it gets added to our table and becomes the newest row in our table. There's no order to the movies in our movie table. And now that it's time to reshelve our movies, we have a bit of a problem. We know that each of the new shelves holds 20 movies, and every one of the more than 3,000 movies has to have a sticker on it indicating its category. ***We need to select the movies in each category, in alphabetical order within its category.***

We know how to query the database to find all of the movies in each category, but we need them listed alphabetically within their categories somehow.

movie_table

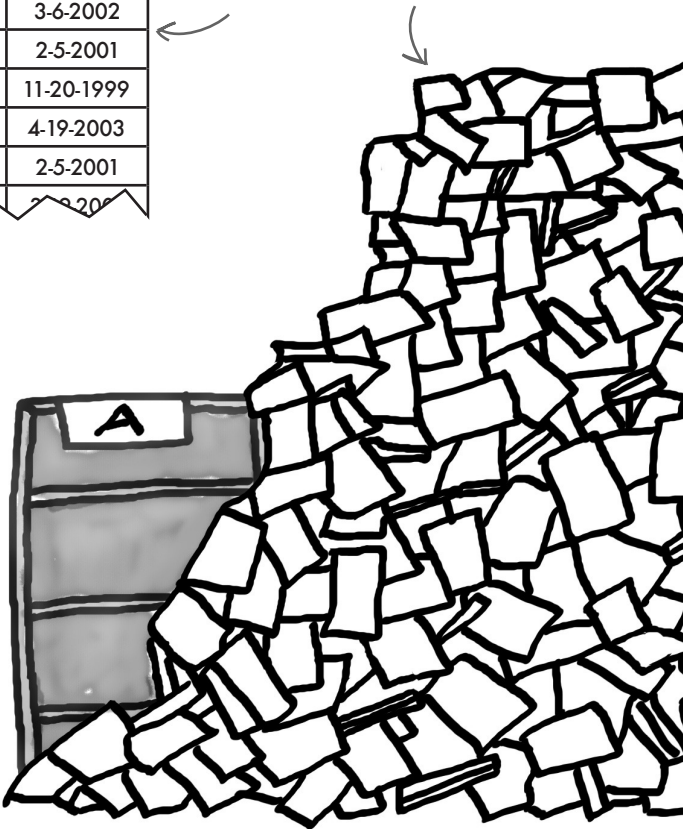
movie_id	title	rating	category	purchased
83	Big Adventure	G	family	3-6-2002
84	Greg: The Untold Story	PG	action	2-5-2001
85	Mad Clowns	R	horror	11-20-1999
86	Paraskavedekatriaphobia	R	action	4-19-2003
87	Rat named Darcy, A	G	family	4-19-2003
88	End of the Line	R	misc	2-5-2001
89	Shiny Things, The	PG	drama	3-6-2002
90	Take it Back	R	comedy	2-5-2001
91	Shark Bait	G	misc	11-20-1999
92	Angry Pirate	PG	misc	4-19-2003
93	Potentially Habitable Planet	PG	scifi	2-5-2001
94	Cosmic Will	R	horror	2-5-2001

These are just a few of the more than 3,000 movies Dataville Video has in stock.



BRAIN BARBELL

How would you organize this data alphabetically using a SQL statement?




We need a way to organize the data we SELECT

Each one of the more than 3,000 movies has to have a sticker on it indicating its category. Then it has to be shelved in alphabetical order.

We need a master list of the movies in alphabetical order by title for each category. So far, we know how to SELECT. We can easily select movies by category, and we can even select movies by first letter of the title and by category.

But to organize our big list of movies means that we would need to write at least 182 SELECT statements: Here are a just a few of them:

```
SELECT title, category FROM movie_table WHERE title LIKE 'A%' AND category = 'family';
SELECT title, category FROM movie_table WHERE title LIKE 'B%' AND category = 'family';
SELECT title, category FROM movie_table WHERE title LIKE 'C%' AND category = 'family';
SELECT title, category FROM movie_table WHERE title LIKE 'D%' AND category = 'family';
SELECT title, category FROM movie_table WHERE title LIKE 'E%' AND category = 'family';
SELECT title, category FROM movie_table WHERE title LIKE 'F%' AND category = 'family';
SELECT title, category FROM movie_table WHERE title LIKE 'G%' AND category = 'family';
```



We need to know the title so we can dig in the pile to find it, and the category so we can sticker and shelve it.

This is the letter of the alphabet that the movie titles should begin with.

And this is the category we're looking for.

It's 182 queries because we have 7 categories and 26 letters of the alphabet. This number doesn't include movies that have a number at the beginning of their titles (like '101 Dalmatians' or '2001: A Space Odyssey').



Where do you think titles that begin with a number or a non-letter character—like an exclamation point—will appear in the list?

Sharpen your pencil

We still have to manually alphabetize the titles within their category list using the letters that follow the initial 'A' to decide the order.

Take a closer look at some of the output from just one of our 182 (or more) queries. Try alphabetizing the list of movie titles by hand.

 `SELECT title, category FROM movie_table WHERE title LIKE 'A%' AND category = 'family';`

 A few of our query results

title	category
Airplanes and Helicopters	family
Are You Paying Attention?	family
Acting Up	family
Are You My Mother?	family
Andy Sighs	family
After the Clowns Leave	family
Art for Kids	family
Animal Adventure	family
Animal Crackerz	family
Another March of the Penguins	family
Anyone Can Grow Up	family
Aaargh!	family
Aardvarks Gone Wild	family
Alaska: Land of Salmon	family
Angels	family
Ann Eats Worms	family
Awesome Adventure	family
Annoying Adults	family
Alex Needs a Bath	family
Aaargh! 2	family



Sharpen your pencil Solution

We still have to manually alphabetize the titles within their category list using the letters that follow the initial 'A' to decide the order.

Take a closer look at some of the output from just one of our 182 (or more) queries. Try alphabetizing the list of movie titles by hand.

```
SELECT title, category FROM movie_table WHERE title LIKE 'A%' AND category = 'family';
```

title	category
Aaargh!	family
Aaargh! 2	family
Aardvarks Gone Wild	family
Acting Up	family
After the Clowns Leave	family
Airplanes and Helicopters	family
Alaska: Land of Salmon	family
Alex Needs a Bath	family
Andy Sighs	family
Angels	family
Animal Adventure	family
Animal Crackerz	family
Ann Eats Worms	family
Annoying Adults	family
Another March of the Penguins	family
Anyone Can Grow Up	family
Are You My Mother?	family
Are You Paying Attention?	family
Art for Kids	family
Awesome Adventure	family

How long did these 20 movies take you to order?

Can you imagine how long it would take to order 3,000 or more movies in this way?

The titles starting 'Are You...' come towards the end of the order since the letter following the initial 'A' is an 'r', but then we had to look at the seventh letter into the title before we could work out where each movie should be shelved.

Try a little **ORDER BY**

You say you need to order your query? Well, it just so happens that you can tell SQL to **SELECT** something and **ORDER** the data it returns **BY** another column from the table.

No surprises in this part. It's exactly the same as the **SELECT** query we just tried.

```
SELECT title, category
FROM movie_table
WHERE
title LIKE 'A%'
AND
category = 'family'
ORDER BY title;
```

Here's the new bit. Just like it sounds, it tells the program to return the data in alphabetical order by title.

Seriously. Are you telling me this is the only way we can alphabetize our results? There's **NO WAY** I'm doing that for every letter of the alphabet.



Sharpen your pencil

You're right. What can we take out of the query above to make it much more powerful?

.....

.....

.....

.....

STOP! Do this exercise before turning the page.

ORDER a single column

If our query uses `ORDER BY title`, we don't need to search for titles that start with a particular letter anymore because the query returns the data listed in alphabetical order by title.

All we need to do is take out the `title LIKE` part, and `ORDER BY title` will do the rest.



Sharpen your pencil Solution

What can we take out of the query above to make it much more powerful?

```
SELECT title, category
FROM movie_table
WHERE
title LIKE 'A%'
AND
category = 'family'
ORDER BY title;
```

use drama instead of family

```
SELECT title, category
FROM movie_table
WHERE
category = 'family'
ORDER BY title;
```

This time we'll get the entire list of movies in the family category.

Even better, this list will include movies that begin with numbers in the title. They'll be first in the list.

This isn't the end of the results; we don't have room to show them all here. They continue all the way through Z titles.

ORDER BY allows you to alphabetically order any column.

Notice that the first few titles begin with a number.

title	category
1 Crazy Alien	family
10 Big Bugs	family
101 Alsations	family
13th Birthday Magic	family
2 + 2 is 5	family
3001 Ways to Fall	family
5th Grade Girls are Evil	family
7 Year Twitch	family
8 Arms are Better than 2	family
Aaargh!	family
Aaargh! 2	family
Aardvarks Gone Wild	family
Acting Up	family
After the Clowns Leave	family
Airplanes and Helicopters	family
Alaska: Land of Salmon	family
Alex Needs a Bath	family
Andy Sighs	family
Angels	family
Animal Adventure	family
Animal Crackerz	family
Ann Eats Worms	family
Annoying Adults	family
Another March of the Penguins	family
Anyone Can Grow Up	family
Are You My Mother?	family
Are You Paying Attention	family
Art for Kids	family
Awesome Adventure	family



Exercise

Create a simple table with a single CHAR(1) column called 'test_chars'.

Insert the numbers, letters (both upper- and lowercase), and non-alphabet characters shown below in this column, each in a separate row. Insert a space and leave one row NULL.

Try your new ORDER BY query on the column and fill in the blanks in the SQL's *Rules of Order* book shown below.

0123ABCDabcd!@#\$%^&*() - _
+=[]{};:'"\|`~.,<>/?

SQL's Rules of Order

When you've run your ORDER BY query, fill in the blanks using the order the characters appear in your results to help you.

Non-alphabet characters show up numbers.

Numbers show up text characters.

NULL values show up numbers.

NULL values show up alphabet characters.

Uppercase characters show up lowercase characters.

"A 1" will show up "A1".

SQL's Rules of Order

When you've run your ORDER BY query, put these characters in the order they appear in the results.

+ = ! (& ~ "
* @ ? ' ←

Remember how to insert a single quote? They're tricky.



Create a simple table with a single CHAR(1) column called 'test_chars'.

Insert the numbers, letters (both upper- and lowercase), and non-alphabet characters shown below in this column, each in a separate row. **Insert a space and leave one row NULL.**

Try your new ORDER BY query on the column and fill in the blanks in the 'SQL's Rules of Order' book shown below.

! " # \$ % & ' () * + , - . / 0 1 2 3 : ; < = >
? @ A B C D [\] ^ _ ` a b c d { | } ~

The order that the characters may have shown up in your results. Note the space at the beginning. Your order may be a bit different depending on your RDBMS. The point here is to know that there IS an order, and what the order is for your RDBMS.

SQL's Rules of Order

When you've run your ORDER BY query, fill in the blanks using the order the characters appear in your results to help you.

Non-alphabet characters show up
...before and after... numbers.

Numbers show up before text characters.

NULL values show up before numbers.

NULL values show up before alphabet characters.

Uppercase characters show up before lowercase characters.

"A 1" will show up before "A1".

SQL's Rules of Order

When you've run your ORDER BY query, put these characters in the order they appear in the results.

+ = ! (& ~ "
* @ ? '

! " # \$ % ' (* + = ? @ ~

```
SELECT *
from test_chars
order by single;
```

ORDER with two columns

Seems like everything is under control. We can alphabetize our movies, and we can create alphabetical lists for each category.

Unfortunately, your boss has something else for you to do...

To: Dataville Video Staff
From: The Boss
Subject: Out with the old (movies)

Hey,

I think we need to get rid of some of the movies we've had for the longest time. Can you come in this weekend and give me a list of movies in each category by order of purchase date?

That would be great,
Your boss

Fortunately, you can order multiple columns in the same statement.

```
SELECT title, category, purchased  
FROM movie_table  
ORDER BY category, purchased;
```

We want to make sure the purchased date shows up in the results.

This will be the first column ordered. We'll get a list of every movie in the store, ordered by category.

And this will be the second column ordered, AFTER the category column has been ordered.



BRAIN BARBELL

Will the oldest movies show up first or last in each category? And what do you think will happen if two movies are in the same category with the same purchase date? Which will show up first?

ORDER with multiple columns

You're not restricted to sorting by just two columns. You can sort by as many columns as you need to get at the data you want.

Take a look at this ORDER BY with three columns. Here's what's going on, and how the table gets sorted.

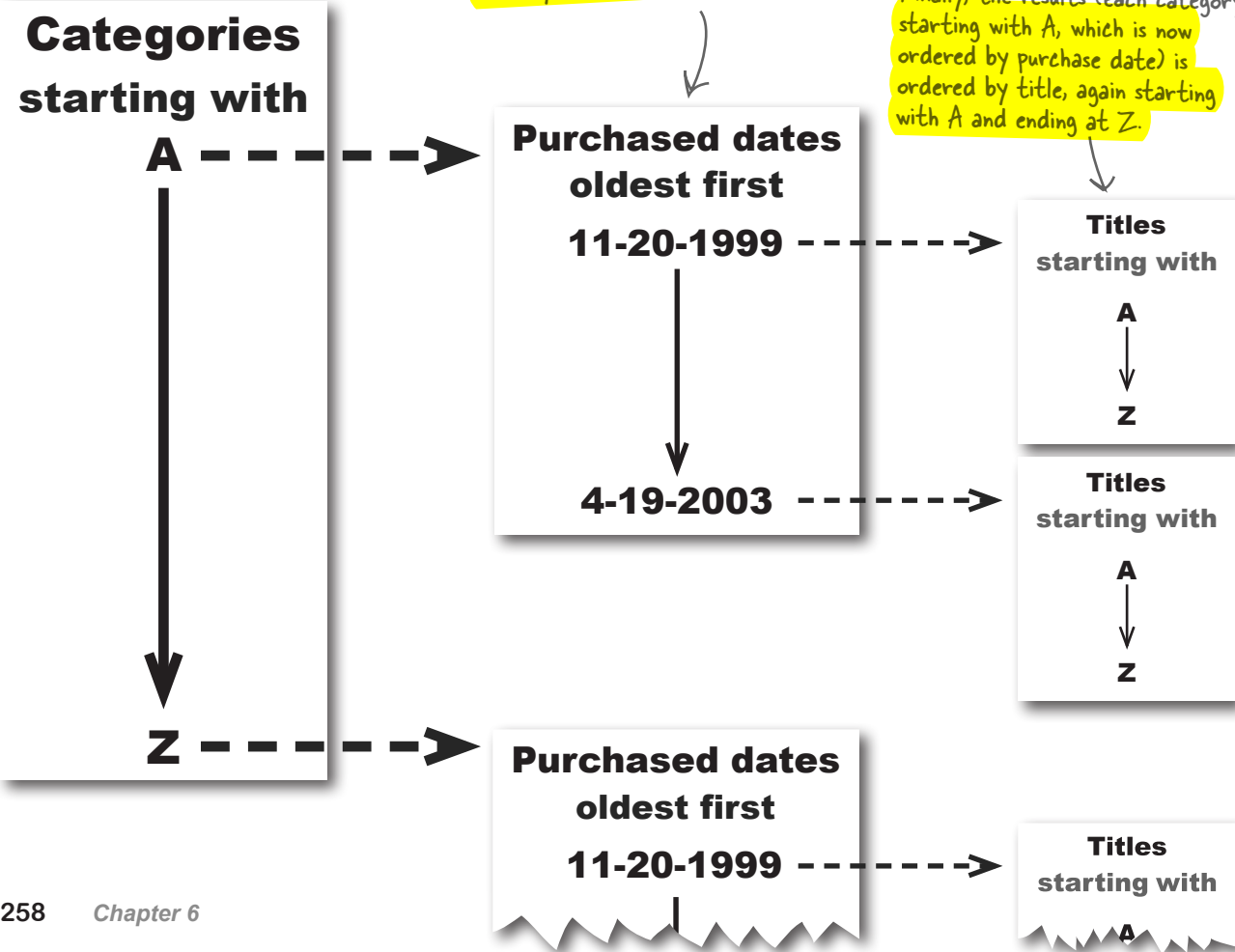
You can sort by as many columns as you need.

```
SELECT * FROM movie_table
ORDER BY category, purchased, title;
```

First your results are ordered by category, since that was the first column listed after your ORDER BY. The results are listed A through Z.

Next, the results (each category in the table starting with A since that's how the categories are ordered now) is sorted by date, with the oldest date first. Dates are always sorted by year, then by month, then by day.

Finally, the results (each category, starting with A, which is now ordered by purchase date) is ordered by title, again starting with A and ending at Z.



An orderly movie_table

Let's see what this SELECT statement actually returns when we run it on our original movie table.

Our original movie_table
There's no real order here; movies just show up in the order in which the records were inserted into the table.

movie_id	title	rating	category	purchased
83	Bobby's Adventure	G	family	3-6-2002
84	Greg: The Untold Story	PG	action	2-5-2001
85	Mad Clowns	R	horror	11-20-1999
86	Paraskavedekatriaphobia	R	action	4-19-2003
87	Rat named Darcy, A	G	family	4-19-2003
88	End of the Line	R	misc	2-5-2001
89	Shiny Things, The	PG	drama	3-6-2002
90	Take it Back	R	comedy	2-5-2001
91	Shark Bait	G	misc	11-20-1999
92	Angry Pirate	PG	misc	4-19-2003
93	Potentially Habitable Planet	PG	scifi	2-5-2001

and the ordered results from our query:

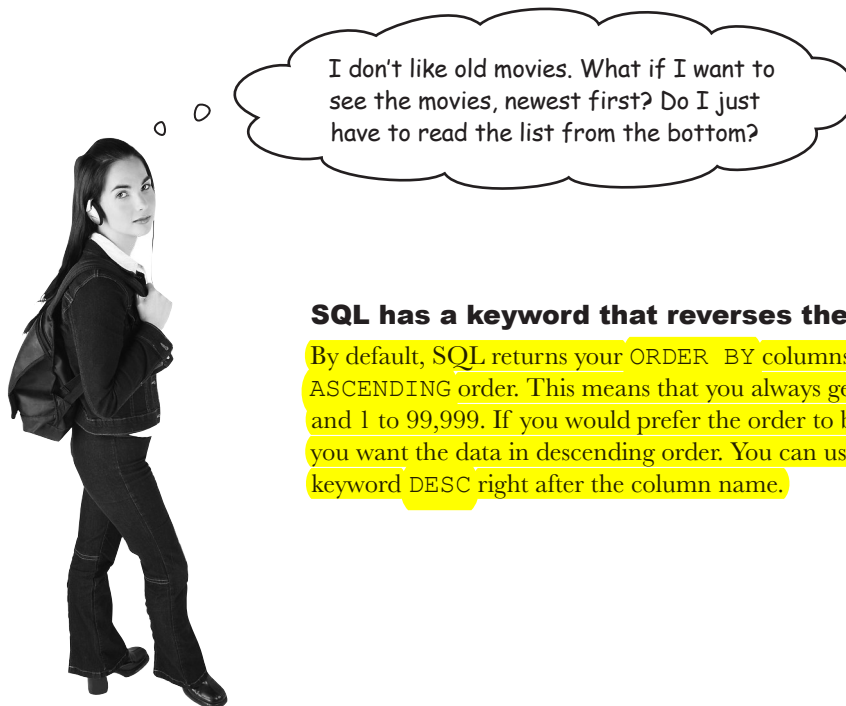
Third column that was ordered

First column that was ordered

Second column that was ordered

movie_id	title	rating	category	purchased
84	Greg: The Untold Story	PG	action	2-5-2001
86	Paraskavedekatriaphobia	R	action	4-19-2003
90	Take it Back	R	comedy	2-5-2001
89	Shiny Things, The	PG	drama	3-6-2002
83	Bobby's Adventure	G	family	3-6-2002
87	Rat named Darcy, A	G	family	4-19-2003
85	Mad Clowns	R	horror	11-20-1999
91	Shark Bait	G	misc	11-20-1999
88	End of the Line	R	misc	2-5-2001
93	Potentially Habitable Planet	PG	scifi	2-5-2001

```
SELECT * FROM movie_table
ORDER BY category, purchased, title;
```



I don't like old movies. What if I want to see the movies, newest first? Do I just have to read the list from the bottom?

SQL has a keyword that reverses the order.

By default, SQL returns your **ORDER BY** columns in **ASCENDING** order. This means that you always get A to Z and 1 to 99,999. If you would prefer the order to be reversed, you want the data in descending order. You can use the keyword **DESC** right after the column name.

there are no Dumb Questions

Q: I thought that **DESC** was used to get the **DESCRIPTION** of a table. Are you sure this works to change the **ORDER**?

A: Yes. It's all about context. When you use it in front of a table name—for example, **DESC movie_table**;—you'll get a description of the table. In that case, it's short for **DESCRIBE**.

When you use it in an **ORDER** clause, it stands for **DESCENDING** and that's how it will order the results.

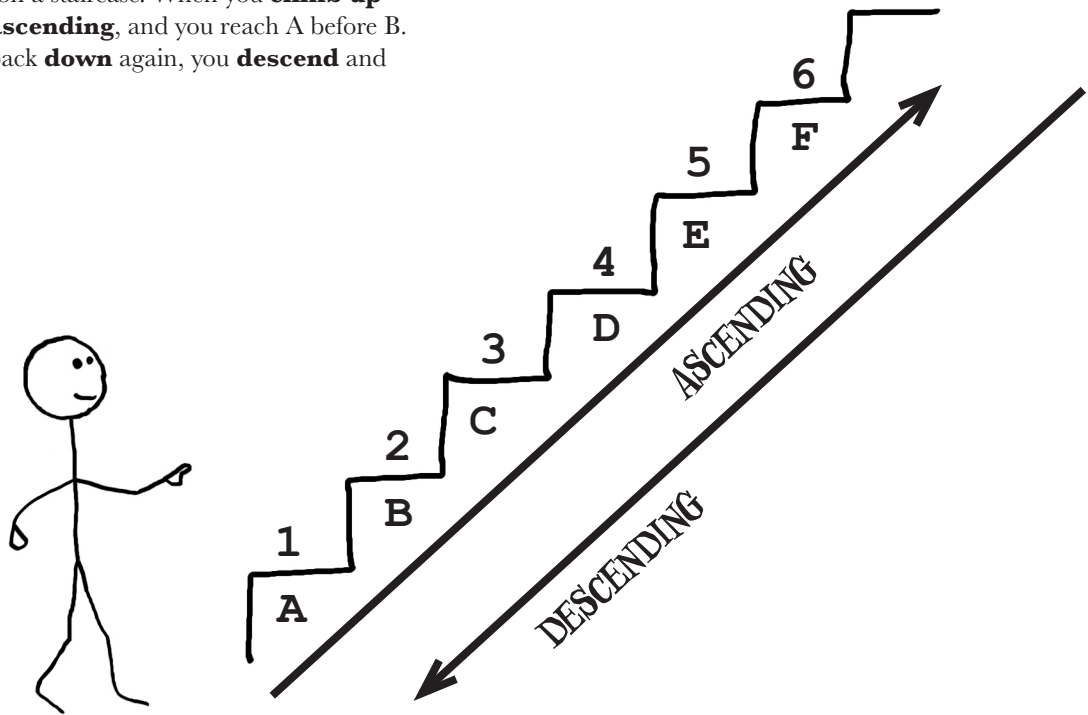
Q: Can I use the whole words **DESCRIBE** and **DESCENDING** in my query to avoid confusion?

A: You can use **DESCRIBE**, but **DESCENDING** won't work.

Use the keyword DESC after your column name in **ORDER BY clauses to reverse the order of your results.**

Reverse the ORDER with DESC

Picture your data on a staircase. When you **climb up** the stairs, you're **ascending**, and you reach A before B. When you come back **down** again, you **descend** and reach Z before A.



This query gives us a list of movies ordered by the purchase date, with the **newest** ones first. For each date, the movies purchased on that date are listed in alphabetical order.

```
SELECT title, purchased
FROM movie_table
ORDER BY purchased DESC, title ASC;
```

If we want to order our data from Z to A or from 9 to 1, we have to use the DESC keyword.

We can put ASC there, but it's not necessary. Just remember that ASC is the default order.

To: Dataville Video Staff
From: The Boss
Subject: Freebies all round!

Hey,

The store is looking great! You've got all those movies stacked in the right places, and, thanks to those fancy ORDER BY clauses in your SQL, everybody can find exactly what they're looking for.

To reward you for all of your hard work, I'm throwing a little pizza party at my house tonight. Show up at 6ish.

Don't forget to bring those reports!

Your boss

P.S. Don't wear anything too nice, I've got these bookshelves I've been itching to reorganize...