Databases for Actors:

A TREATISE ON THE VIRTUES OF SQL STATEMENTS FOR THOSE OF THE CREATIVE PROFESSIONS

Edmund Alyn Jones, MFA | SQL Course | June 16, 2019

LEARN A NEW LANGUAGE

I LOOOOOOOOOVE making lists. Lists of items to purchase. My favorite video games of all time. Random assortments of words that would make great band names. You name it I've listed. I once kept track of the everything I ate for several days to notice patterns in my diet. On way to collect a hold all this information is in a database. If you're like me you have an extensive collection of movies (or in my case video games) that you'd like to keep track of some how on your computer. Allow me to introduce you to the world of data-driven programming. A great tool to help you organize your treasures into wonderful collections of data. Let's explore some elements of a computer language that I find easy to use and fun to write. But first some housekeeping, let's talk about data-driven programming and what it does.

DATA-DRIVEN PROGRAMMING

In data-driven program the data within a database is the main reference point in the code (the stuff you tell a computer to do). In a data-driven program (aka software, aka application) you use data that you have collected as a means to tell the computer what to do. I know it sounds a little weird but keep reading. Looking a tiny piece of code might help.

Example:

```
SELECT tbl_species.col_species_id FROM tbl_species
```

In this example the actual data of the table and the columns therein are a part of this line of code and thus instruct the computer what to do *because* of the data not just simply manipulating data. The data 'drives' the program to take action (usually the location of some data) just like you would drive your car to a destination.

WEB-SITES THAT FOLLOW THE DATA

Some websites use databases to 'decide' what to show a user (you) on a web browser. Imagine the web browser is an actor and computer program is the script. Websites that use the 'script' from a language called HTML follow the instructions and displays what the script tells it to do. Database driven websites are connected to databases (or large, organized collections of information) and tend to be more dynamic. It's kind of like the difference between Shakespeare and improv. The Shakespeare is pretty static it hasn't changed for hundreds of years (unless we edit it of course). Improv on the other hand draws from a 'database' of rules and structures to follow but is different every time it is performed. So a database driven webpage is always 'improvising' what it will show you based on what's in its database.

HAVE YOUR DATA-LAYER AND EAT IT TOO

Data-access layer is simple a technical jargon for the program that acts as the go between the user and the user and the database. It provides simplified *access* to the data through a computer software. We explore such a data-access layer (aka DAL to all the software developers in the streets) through the magic of a programming language called SQL or Structured-Query Language.

COMMANDMENTS OF SQL

Nothing gets done without action. Verbs are paramount to telling a story. They allow the writer to tell the reader what the subject is doing. SQL is no different. It needs action words to tell the program what to do with all that glorious data we have in the world. The following are some of those action words in the programming language called SQL.

THOU SHALT INSERT

So you've created this amazing table about your favorite movies. You've got a column about for the title of the movie, genre, and maybe a synopsis. Now you have to enter the names, genre, and synopsis into your shiny new table.

How do you do this?

You *INSERT* the data with the INSERT command (or as they like to call it in SQL world the INSERT *statement*. Coupled with its trusty sidekick *INTO* the INSERT statement can put your precious data into your table.

> *For example:*
>
> Lets say you have your favorite movie ("A Goofy Movie") and ready to include it into your custom made MyMovieDatabase. Using the INSERT statement you'd write:
>
> INSERT INTO table_movie_title (col_movie_title)VALUES ('A Goofy Movie');
>
> What you are doing here is putting the title ("A Goofy Movie") into the table called "table_movie_title" where the column is called "col_movie_title.

So, just like that you've entered new data into your table. Let's see what else we can do.

THOU SHALT SELECT

Great, you've busted your tail painstakingly inserting all of your movie collection into the database. Now you want to know how many horror movies you have exactly. That's where the SELECT statement comes in handy. The SELECT statement allows you to choose which columns contain certain data you want to see in a result that you can populate in a special program called database management software. In this example you want to select the amount of horror flicks you've accumulated.

SELECT COUNT(col_genre) FROM table_genre WHERE col_genre = 'HORROR'

I know what your thinking. There are some things in this line of code that I haven't explained yet but stay with me. The main thing to notice is that the SELECT statement grabbed the number of movies (specified by the COUNT statement) from the table entitled "table_genre" (hence the FROM statement) where the genre is equal to 'HORROR'. We'll discuss the WHERE statement in more detail next.

THOU SHALT FIND OUT FROM WHERE THE DATA COMES

We introduced the WHERE statement in the last example, let's find out more about this nifty little piece of functionality. WHERE statements really mean a number of things at the same time. Let's review the above example again:

SELECT COUNT(col_genre) FROM table_genre WHERE col_genre = 'HORROR'

We could just as easily replace WHERE with 'IN THE PARTICULAR INSTANCE…'. That would take forever to write each time and would make the code much more cluttered. More or less the WHERE statement let's you get results with conditions you specify. You could, for example, find list the movies "WHERE" both the genre is horror and has 'Evil' in the title (that should be a short list… *right?)*.

THOU SHALL ORDER BY

Similar to the WHERE statement is another handy code magic spell called the ORDER BY statement. This statement is great when you want to sort data. Say you wanted to alphabetize your massive movie collection, but it hurts your brain to do it in real time. You could tell your DBMS (it means database management software; computer people love acronyms) to select all your titles and ORDER BY the first letter of the title. Simple, *yay?*

THOU SHALT 'JOIN' THE TWO INTO ONE

Now let's say you have two tables. One table houses the data for the movie titles in your home collection and the other table contains your favorite actors and all the films that they've been in for the last twenty years.

> *To establish a relationship between two tables you have to create a PRIMARY KEY in the movies table and link said PRIMARY KEY with something called a FOREIGN KEY in the actors table (you want more about KEYS consult the mighty sage Google, she knows all).*

Once the two tables are connected by KEYs we can then join the two tables together to create a new results 'table' that contains the info from both.

The statement that handles this digital witchcraft is called the INNER JOIN statement. It interconnects the identical data in each table and overlaps them into one sweet result. In this example of linking actors to movies the linking data set is the movie ID. Each film would have a unique ID number and so the actor table would list the actor and the unique ID of a movie that they starred in called, "col_movie_id". INNER JOIN uses that special ID number to link the actor to the movie and then spits out a new table that has both.

col_movie_id	**Movie_title**	**Actor_name**
3225	"Detective Pikachu"	Ryan Reynolds
2411	"Green Lantern"	Ryan Reynolds
3115	"Deadpool"	Ryan Reynolds

OUTER JOIN will show you the data from both tables that are NOT shared by both tables. In this example it would show you that you are missing from your personal library thirteen films by greatest actor of all time Alec Baldwin and that you better get on that quickly. The OUTER JOIN will also show you which movies on your list don't have an actor linked to them on your actor table.

Col_movie_id	**Movie_title**	**Actor_name**
NULL	"Drunk Parents"	Alec Baldwin
NULL	"The Cat in the Hat"	Alec Baldwin
1155	"The Hunt for Red October"	NULL

There were so many great actors in "The Hunt for Red October" and yes Alec Baldwin is one of them. Shame on you for not including this vital information in your actor database.

THOU SHALT DELETE

Now we come back to the beginning, the circle of life. The lord giveth data and the lord taketh away data. The DELETE statement will help you remove that embarrassing guilt flick from your database. Just like INSERT has its companion statement INTO, DELETE has a road dog called FROM. DELETE FROM allows you to obliterate a row of data from your table so use with caution.

BEGIN…END

This concludes the actor's guide to SQL. There is so much fun to be had in computer programming for the performing artist. I mean they don't call some computer code a script for nothing. Anyway, good luck on building that movie database, and who knows maybe you'll build a table containing all the movies YOU have been in some day. Thanks for reading.