Week 3 cliff notes

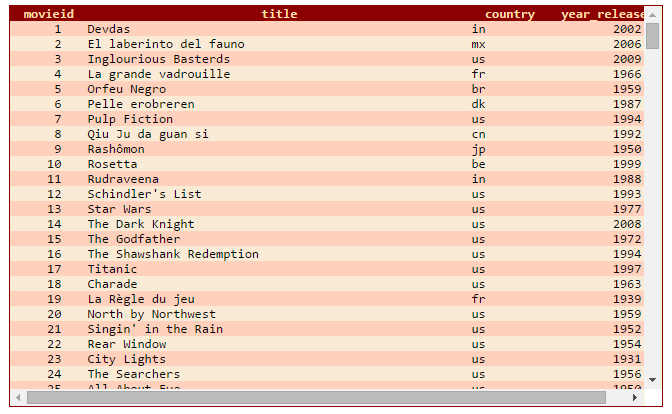
In Week 2, we got comfortable with terms like rows, columns, tables and database. I also mentioned that “relational tables”, tables that are joined together, bear the brunt of what makes SQL so complicated. Before we get there, we have to cover some basic concepts with SQL first. This week is focused entirely on selecting data out of single table. In other words, we will not be joining tables at all - that comes in a few more weeks.

SQL stands for “structured query language”, and like any good language, it has a codified grammar. Once you master the grammar, you have pretty much mastered SQL. And because it is a pain to type in all caps, I will occasionally use sql. :) Some of you really dig grammar and structure, so for you, here is a link to the [MySQL version](http://dev.mysql.com/doc/refman/5.7/en/sql-syntax.html) and the [MS SQL Server version](https://msdn.microsoft.com/en-us/library/bb510741.aspx).

So get ready to create some queries; feel free to use the [SQL Fiddle](http://sqlfiddle.com/#!9/60125) for this class, the [Konagora SQL sandbox](http://edu.konagora.com/SQLsandbox.php), or the [w3schools SQL course](http://www.w3schools.com/sql/) which has a nifty “try this” feature (note: w3schools has different tables, so the exact text here will not work). Students are expected to execute all queries and even play around with their own.

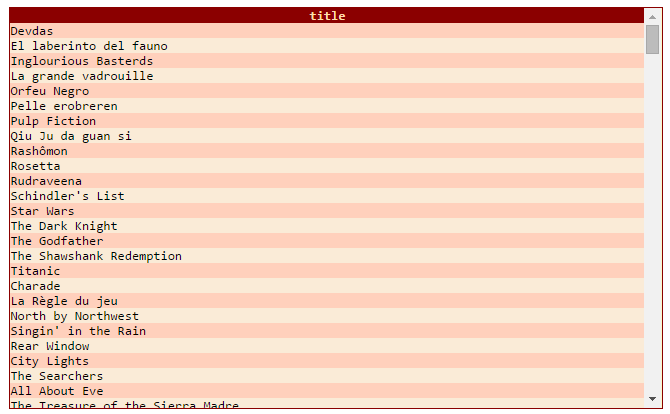
The first query most people like to do is the “get everything from some table” type. We talked about the movies table last week, so let’s “get everything” from movies:

select \* from movies;



The \* (asterisk, aka “star”) translates, literally, to all the column names. If we only want the “title” column, we would say:

select title from movies;



Note how the output looks like a table with only one column. It is very helpful to think of the output of sql as just another table; columns and rows.

So far, the queries we have used get all the rows possible. To restrict the number of rows, we use the “WHERE clause” to filter which rows we want. WHERE evaluates each and every single row to see if the row makes any of the WHERE clauses TRUE. Let’s look up a single movie that was released in the year 2001:

select \* From movies

where year\_released = 2001

|  |  |  |  |
| --- | --- | --- | --- |
| **movieid** | **title** | **country** | **year\_released** |
| 21 | The Lord of the Rings | nz | 2001 |

In other words, we compared the “year\_releasted” value in every row to “2001”, and only one row matched. Let’s look for a different movie, but by “title” this time:

select \* From movies

where title = 'Annie Hall'

|  |  |  |  |
| --- | --- | --- | --- |
| **movieid** | **title** | **country** | **year\_released** |
| 1 | Annie Hall | us | 1977 |

We have to use single-quotes around string values, otherwise we would get an error. Now let’s try a WHERE clause that selects more than one row, but still uses an equal sign (otherwise known as an equality predicate):

select \* From movies

where country = 'fr'

|  |  |  |  |
| --- | --- | --- | --- |
| **movieid** | **title** | **country** | **year\_released** |
| 11 | La belle et la bÃªte | fr | 1946 |
| 14 | Le cinquiÃ¨me Ã©lÃ©ment | fr | 1997 |
| 15 | Les Visiteurs du Soir | fr | 1942 |

Those french movies have titles which are hard to translate in SQL Fiddle. :) As we can see, every row was tested to see if country=’fr’, and the database found three rows that matched. We could also do the opposite and find rows that do not match:

select \* From movies

where country != 'fr'

|  |  |  |  |
| --- | --- | --- | --- |
| **movieid** | **title** | **country** | **year\_released** |
| 1 | Annie Hall | us | 1977 |
| 2 | Blade Runner | us | 1982 |
| 3 | Bronenosets Potyomkin | ru | 1925 |
| 4 | Casablanca | us | 1942 |

…

Record Count: 23;

And finally, we can use “range” operators like > or < to find a group of values that are greater than or less than. Let’s find movies that were released in year greater than 1990:

select \* From movies

where year\_released > 1990

|  |  |  |  |
| --- | --- | --- | --- |
| **movieid** | **title** | **country** | **year\_released** |
| 14 | Le cinquiÃ¨me Ã©lÃ©ment | fr | 1997 |
| 21 | The Lord of the Rings | nz | 2001 |

We can also combine WHERE clauses together (the technical term is “boolean logic” or “boolean arithmetic”). Just remember, it all boils down to whether or not the conditions (predicates) can be TRUE for any rows. Let’s try a couple tests with two conditions, movieid=15 year\_released=2001. First we will look for any rows that match one condition or the other:

select \* From movies

where movieid=15 or year\_released=2001

|  |  |  |  |
| --- | --- | --- | --- |
| **movieid** | **title** | **country** | **year\_released** |
| 15 | Les Visiteurs du Soir | fr | 1942 |
| 21 | The Lord of the Rings | nz | 2001 |

Note the keyword OR; that matches the question (match one condition **or** the other). Now how about matching both conditions as the same time? We expect there to be zero rows because no movie has both the movieid=15 AND released in the year 2001:

select \* From movies

where movieid=15 and year\_released=2001

Record Count: 0;

One more final step - making it slightly more complex. You can group together WHERE clauses (aka, “predicates”) with parenthesis. But keep your head screwed on tight, as it might get a little hairy.

-- no groups

select \* From movies

where country='ru' or movieid>10 and year\_released>1990

|  |  |  |  |
| --- | --- | --- | --- |
| **movieid** | **title** | **country** | **year\_released** |
| 3 | Bronenosets Potyomkin | ru | 1925 |
| 14 | Le cinquiÃ¨me Ã©lÃ©ment | fr | 1997 |
| 21 | The Lord of the Rings | nz | 2001 |

-- group the first two

select \* From movies

where (country='ru' or movieid>10) and year\_released>1990

|  |  |  |  |
| --- | --- | --- | --- |
| **movieid** | **title** | **country** | **year\_released** |
| 14 | Le cinquiÃ¨me Ã©lÃ©ment | fr | 1997 |
| 21 | The Lord of the Rings | nz | 2001 |

-- group the last two

select \* From movies

where country='ru' or (movieid>10 and year\_released>1990)

|  |  |  |  |
| --- | --- | --- | --- |
| **movieid** | **title** | **country** | **year\_released** |
| 3 | Bronenosets Potyomkin | ru | 1925 |
| 14 | Le cinquiÃ¨me Ã©lÃ©ment | fr | 1997 |
| 21 | The Lord of the Rings | nz | 2001 |

Notice that the very first test (no groupings) has the same output as the last test (group the last two). This is because of something called “precedence”; the database will always put neighboring conditions with an AND together in a group. That means that if you want OR to have a higher precedence, you have to explicitly put it in a group with AND statements.

Now for another challenge. I want you to find all the movies that if the movieid is less than 10, the country be ‘us’ and the release year between 1970 and 1980, or if the movieid is more than 10, the country be ‘cn’ or ‘de’ and the release year between 1960 and 1970. Does your head hurt, yet?

When you have something like this, the best thing to do is break it down into smaller parts. So let me say the same thing, but rearrange the way it looks

I want you to find all the movies

that

(if the movieid is less than 10,

the country be ‘us’

and the release year between 1970 and 1980)

(or if the movieid is more than 10

the country be ‘cn’ or ‘de’

and the release year between 1960 and 1970)

Let’s take that first group and break it down even further.

movied is less than 10 and country is ‘us’ and (release year greater than 1970 and release year less than 1980)

That doesn’t look so hard now. Convert that last group into a WHERE clause

where

movieid<10 and country='us' and year\_released>1970 and year\_released<1980

We can do the same thing for the second portion of the question:

where

movieid>10 and (country='cn' or country='de') and year\_released>1960 and year\_released<1970

A shortcut when you have lots of OR clauses together is to use the IN clause:

(country='cn' or country='de') means the same thing as country IN ('cn','de')

And now we can put the whole thing together - note we are going to use OR to join the two major groups (if movieid<10 OR movieid>10):

select \* from movies

where

(movieid<10 and country='us' and year\_released>1970 and year\_released<1980)

OR

(movieid>10 and (country='cn' or country='de') and year\_released>1960 and year\_released<1970)

|  |  |  |  |
| --- | --- | --- | --- |
| **movieid** | **title** | **country** | **year\_released** |
| 1 | Annie Hall | us | 1977 |
| 26 | Da Nao Tian Gong | cn | 1965 |

The magic of writing SQL is to break it down into smaller pieces. This idea will become much more important later in the semester when we talk about joining together two or more tables. “Divide and Conquer”.

Here is our picture of the “grammar”

**SELECT** \* [meaning, all columns]

**FROM** [a rowsource, like a table or a view]

**WHERE** [a list of predicates that evaluate to TRUE or FALSE for each row in the rowsource]

Next week will go over ways to manipulate the columns in the SELECT clause and introduce the ORDER BY clause.