Week 4 cliff notes

In week 3, we were introduced to some of the basic grammar of sql, especially the syntax of the WHERE clause and predicates. This week, we are going to play around the different ways to alter the output via different column decorations (SELECT clause) and ordering of the rows (ORDER BY clause). The examples are copied from our [SQL Fiddle](http://sqlfiddle.com/#!9/60125).

Here is a summary of the “grammar” of sql that we have covered so far:

**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]

Last week we started off with selecting columns in the same order they appeared in the table. What if we want the year to be listed first, and then the title?

select year\_released, title from movies;



You can do much more than just rearrange the order the columns appear; you can rename the column name in the output, concatenate (add strings together), do math on numbers, and apply functions like truncate, convert to uppercase, substrings, sum, min, max, etc (which we will continue with next week). Let’s take it one step at a time.

**Modify the output column name**

Change year\_released to “Year Released” (doesn’t seem to work in SQL Fiddle, oddly):

select year\_released 'Year Released', title from movies;



Note how we added the text we wanted in quotes after the table’s column name. In other words, we are saying “Fetch me year\_released, but call it ‘Year Released’ when you put it on the screen.”

**Concatenation**

Let’s smash the movieid and title together into one column:

select concat(movieid,title) m\_title from movies;



I used the “concat” function to put two columns together, and then I renamed the output to “m\_title”. Different database systems use different ways of concatenating (or the colloquial “concatting”) values, but in general “concat” is widely used.

What if we want to say something fancy like “Devdas released in 2002”:

select concat(title,' released in ',year\_released) m\_title from movies;

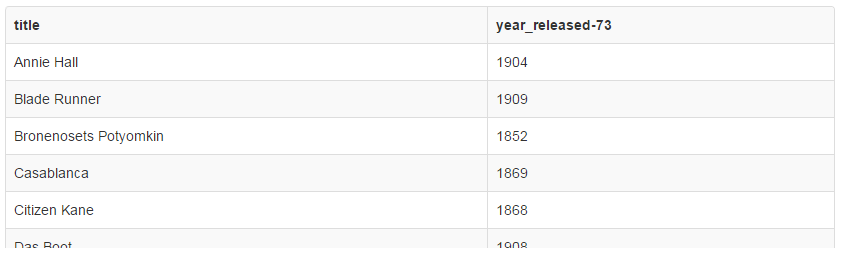


What we are doing is providing different values to be glued together. In this case, I used the title, then I stuck on a string ‘ released in ‘ (note that there are spaces around the single-quotes), and lastly I added another column to be attached, year\_released.

**Arithmetic**

Now let’s just pretend that we want all the years to be from, say, 73 years ago. To accomplish that, we want to subtract 73 from year\_released:

select title, year\_released-73 from movies;



I just took the year\_released column and used the minus sign on it. What if we use, I don’t know, say multiplication with the \* sign?

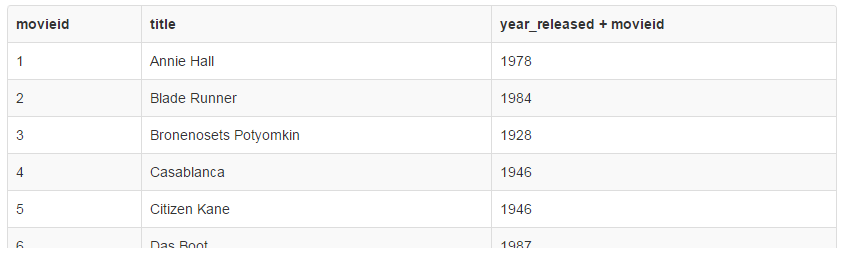
select title, year\_released\*73 from movies;



Hmm… it’s going to be a while before we see those movies.

You can take any two number fields and use any math operator on them:

select movieid, title, year\_released + movieid from movies;



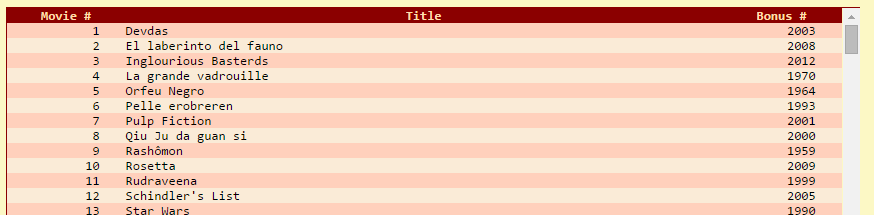
Now let’s make that year\_released + movied column heading look a little prettier - call it “Bonus #”:

select movieid, title, year\_released + movieid 'Bonus #' from movies;

|  |  |  |
| --- | --- | --- |
| **movieid** | **title** | **Bonus #** |
| 1 | Annie Hall | 1978 |
| 2 | Blade Runner | 1984 |
| 3 | Bronenosets Potyomkin | 1928 |
| 4 | Casablanca | 1946 |
| 5 | Citizen Kane | 1946 |
| 6 | Das Boot | 1987 |

Unfortunately, SQL Fiddle does not let us make the other two column names look prettier, but in most systems you can use the same syntax to give columns arbitrary names. For example, from Konagora:

select movieid 'Movie #', title 'Title', year\_released + movieid 'Bonus #' from movies;



Now that we have played around with changing the column names, let’s start working on the order that data appears in. This will be the ORDER BY clause. To start with something easy, let’s sort the output based on year\_released:

select movieid, title, year\_released from movies order by year\_released;

|  |  |  |
| --- | --- | --- |
| **movieid** | **title** | **year\_released** |
| 3 | Bronenosets Potyomkin | 1925 |
| 5 | Citizen Kane | 1941 |
| 15 | Les Visiteurs du Soir | 1942 |
| 4 | Casablanca | 1942 |

As you can see, the movieid (and title, by the way) are no longer “in order”, but instead we have sorted by year\_released. To sort “backwards”, we can use the “descending” keyword, which I typically shorten to “desc”:

select movieid, title, year\_released from movies order by year\_released desc;

|  |  |  |
| --- | --- | --- |
| **movieid** | **title** | **year\_released** |
| 21 | The Lord of the Rings | 2001 |
| 14 | Le cinquiÃ¨me Ã©lÃ©ment | 1997 |
| 8 | Goodfellas | 1990 |
| 24 | Ying hung boon sik | 1986 |

So we can tell that from these last two queries the earliest movie in the database is “Bronenosets Potyomkin” while the most recent is “The Lord of the Rings”.

Just like Excel (or most other spreadsheets), we can sort by a secondary column as well. I found five years in which more than one movie was released (a slightly more complex query, but for the curious you can see it [here](https://docs.google.com/document/d/1fZPVpAPIWCR1_NEA5fvFYbBMKP-NeS-Jmlp9fcSCuKc/edit?usp=sharing)). So combining what we learned last week with the WHERE clause, let’s select years between 1941 and 1947, but order by year\_released first, and then by title, such that within each year, the titles are alphabetized:

select title, year\_released from movies where year\_released between 1941 and 1947 order by year\_released, title

|  |  |
| --- | --- |
| **title** | **year\_released** |
| Citizen Kane | 1941 |
| Casablanca | 1942 |
| Les Visiteurs du Soir | 1942 |
| La belle et la bÃªte | 1946 |
| Notorious | 1946 |

See how I did the “order by” clauses? Just like selecting columns, the two “order by” columns are separated by a comma. What happens if we swap the “order by” columns? (You will notice that I ask a lot of “what if” questions - so do students, and my response is “try it!”)

select title, year\_released from movies where year\_released between 1941 and 1947 order by title, year\_released;

|  |  |
| --- | --- |
| **title** | **year\_released** |
| Casablanca | 1942 |
| Citizen Kane | 1941 |
| La belle et la bÃªte | 1946 |
| Les Visiteurs du Soir | 1942 |
| Notorious | 1946 |

Sorting by year\_released is rather meaningless, because in this database, there are no two movies with the same title.

Note that SQL allows you to use a shortcut by specifying the implied column number; the first column can be thought of column 1, the second as column 2, etc. So we could “order by 1” to sort by the first column. In general, this is really handy, but I advise you never use it in a real application. The reason is because the sql is horrible to maintain if you ever make fundamental changes to the table or the columns you select in the first place. Later on when we talk about subqueries and group functions, we will use the number of the column more often. But for now…. well, feel free to play around with it. :)

One very handy trick in the context of ordering is known as the “top N” kind of query. For instance, what if we want the 5 oldest movies in the database? We already know how to order the movies by year\_released (the default is from oldest to newest), but how do we “pick off” the top 5? Most database systems have some kind of keyword that will perform this for you - we are going to use the keyword LIMIT:

select title, year\_released from movies order by year\_released limit 5;

|  |  |
| --- | --- |
| **title** | **year\_released** |
| Bronenosets Potyomkin | 1925 |
| Citizen Kane | 1941 |
| Casablanca | 1942 |
| Annie Hall | 1977 |
| Blade Runner | 1982 |

Sometimes you just want to explore a brand new table, and you want to see some rows, but not necessarily all of them. LIMIT works well for that also:

select \* from credits limit 5;

|  |  |  |
| --- | --- | --- |
| **movieid** | **peopleid** | **credited\_as** |
| 1 | 75 | D |
| 1 | 229 | A |
| 1 | 470 | A |
| 1 | 719 | A |
| 2 | 969 | D |

And that wraps up specifying columns and working with the ORDER BY clause. Here is our picture of the “grammar”:

**SELECT** [either one or more columns followed by column names, the ‘\*’, or another sql]

**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]

**ORDER BY** [one or more columns to sort the output by]

Next week we will go over simple single-row functions.