Week 7 cliff notes

Last week we introduced group functions (aka, aggregate functions), which included the GROUP BY and HAVING clauses. Our grammar at this point is:

**SELECT** [either one or more columns and/or functions, 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]

**GROUP BY** [a list of columns around which to form buckets or groups]

**HAVING** [a list of predicates that apply to GROUP BY groups]

**ORDER BY** [a list of output columns that you would like to sort by]

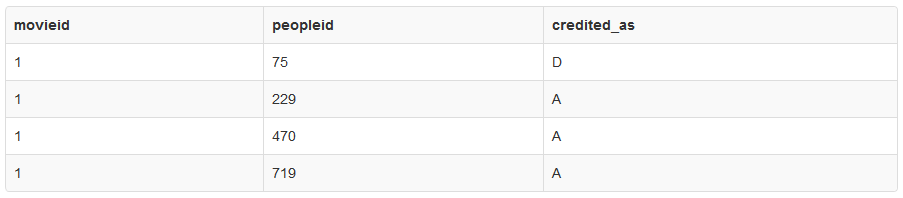
Now that we have thoroughly covered the basics of writing a query against a single table, it is time to move on to queries against multiple tables. We are not going to alter the grammar any more, but we are going to introduce complexities into the grammar we already have. We will be using our famous [SQL Fiddle](http://sqlfiddle.com/#!9/60125) for this exercise.

**JOINS**

If we go back to our analogy of a spreadsheet (that is kind of like a table), a query that acts against two different tables would be like joining together two spreadsheets. In essence, we are taking two tables and combining them to make a new rowsource. A little later, we will expand this to three tables, and then four tables.

You may recall that the CREDITS table doesn’t really tell us much at all. I mean, so what if ‘movieid’ is 1, ‘peopleid’ is 75 and ‘credited\_as’ is ‘D’ - what the heck does that translate to? For our first exercise, we are going to fill in ‘movieid’ with actual film names so we know to what movie this entry applies to. What is movieid 1?

select \* from credits where movieid = 1;



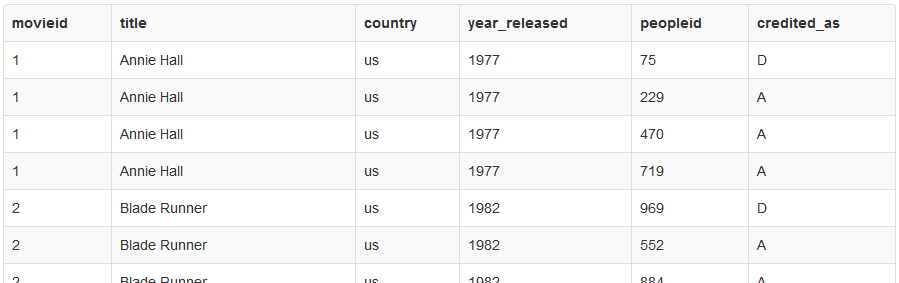
select title from movies where movieid = 1;



Our task now is to get that into one query - I want to select title, peopleid and credited\_as all on the same line.

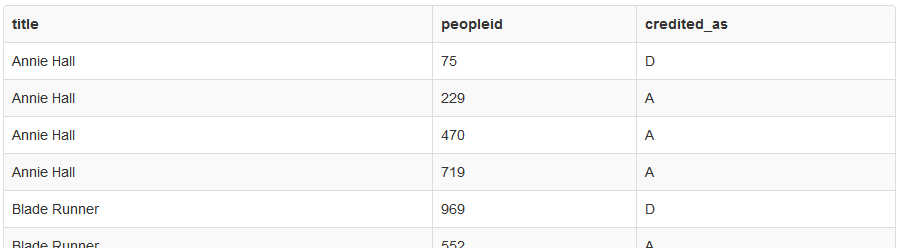
Since ‘movieid’ is a common key in both tables, we can use that information to JOIN the two tables together and create a larger, new temporary table (this is just descriptive, there isn’t really a temporary table stored anywhere). For instance, let us join these tables using movieid:

select \* from movies inner join credits using (movieid)



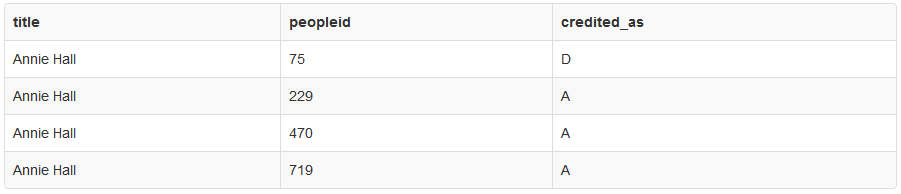
Looks exactly like a new table, right? Take special note that ‘movieid’ is only listed once. Curious? :) But now we have too much information - I don’t care about some of those columns, so let us remove them from the SELECT list:

select title, peopleid, credited\_as from movies inner join credits using (movieid);



And finally, I only want to see movieid = 1.

select title, peopleid, credited\_as from movies inner join credits using (movieid) where movieid = 1;



Are you ready for a leap of faith? What if you really really can’t wait and you just need to find out who those peopleid are? Yes, we can JOIN in yet another table. But I need a new [SQL Fiddle](http://sqlfiddle.com/#!9/3dee4) that includes the PEOPLE table (I also threw in the COUNTRIES table for a little later).

Here is our query so far, but formatted a little differently:

select title, peopleid, credited\_as

from movies

inner join credits using (movieid)

where movieid = 1;

To join in another table, we merely add another JOIN clause. But what key will be using to join on this time? What column is common between CREDITS and PEOPLE? Yes, the ‘peopleid’ column:

select title, peopleid, first\_name, surname, credited\_as

from movies

inner join credits using (movieid)

inner join people using (peopleid)

where movieid = 1;



Sweet! But let’s go back and break down what we just did. The magic sauce is understanding how the JOINs work together. When we join multiple tables together, we are always working on two tables at a time. So in this case, we first join MOVIES to CREDITS using (movieid), and then we have a virtual table with all the columns of both tables that we then join to PEOPLE.



NOTE: When I use the names “TEMP\_RS1” and “TEMP\_RS2” above, those are completely made up - again, I am just giving you a descriptive example of what is happening behind the scenes to help you better understand how JOINs work. You cannot actually select from TEMP\_RS1. :) RS is my shorthand for “rowsource”. Bonus question: What is a rowsource?

Before we put INNER JOIN to sleep, we have to go over one other alternative - what if the name of the column we wish to join on (the key) is not actually the same? It just so happens we have an excellent example of that in our current [SQL Fiddle](http://sqlfiddle.com/#!9/3dee4) (same as before).

Your task is to tell me the name of each movie and the full name of the country from which it comes.

We will still use the INNER JOIN syntax, but instead of “using (column)”, we would utilize “on table1.column = table2.column”. We know we want to JOIN two tables, MOVIES and COUNTRIES, so let’s start putting them together.

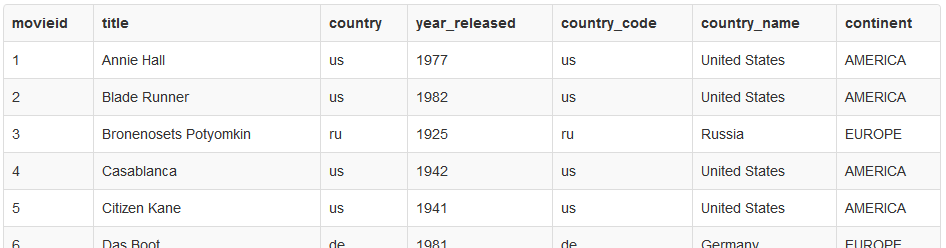


Even though there is no column that has the same name between both tables, we do see that MOVIES.country looks like COUNTRIES.country\_code. So this is our “key”.

select \*

from movies

inner join countries on movies.country = countries.country\_code;



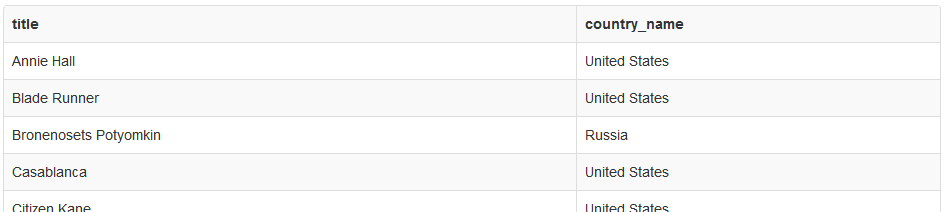


We only want the title and the country name:

select title, country\_name

from movies

inner join countries on movies.country = countries.country\_code;



From there, we could select only movies that come from countries that are not United States, we could order the result alphabetically, we could take a count to see how many come from each country…. you get the picture. Once we get the JOIN done, we already know how to do the other operations.

NOTE: The “ON” clause gets a little tricky; basically, anything you put in the WHERE clause can also go in the ON clause. We will get to more complicated cases later in the course, but for right now, let’s just stick with equality comparisons (using the ‘=’ sign).