Pivoting Data in SQL

Starting here? This lesson is part of a full-length tutorial in using SQL for Data Analysis. Check out the beginning.

Pivoting rows to columns

This lesson will teach you how to take data that is formatted for analysis and pivot it for presentation or charting. We'll take a dataset that looks like this:

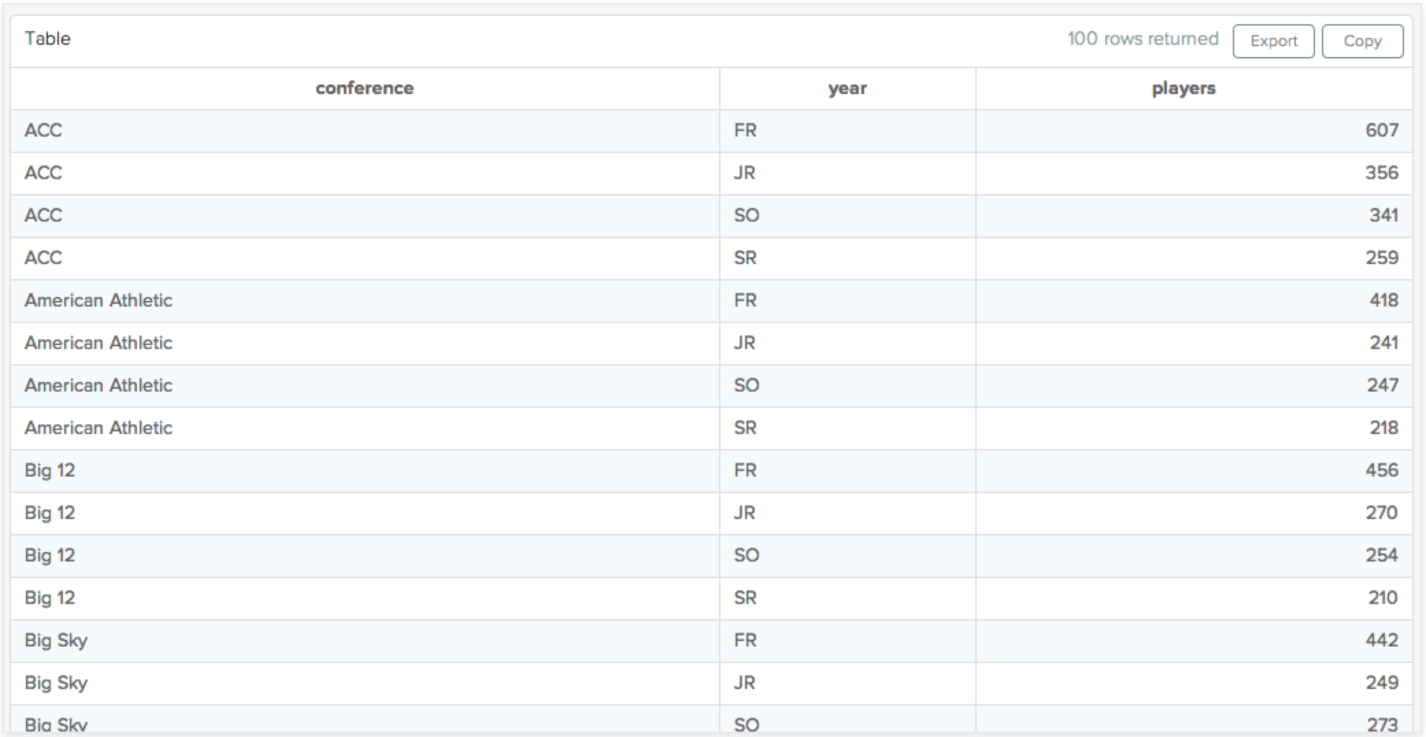


Table — 100 rows returned — Export — Copy

conference	year	players
ACC	FR	607
ACC	JR	356
ACC	SO	341
ACC	SR	259
American Athletic	FR	418
American Athletic	JR	241
American Athletic	SO	247
American Athletic	SR	218
Big 12	FR	456
Big 12	JR	270
Big 12	SO	254
Big 12	SR	210
Big Sky	FR	442
Big Sky	JR	249
Big Sky	SO	273

And make it look like this:

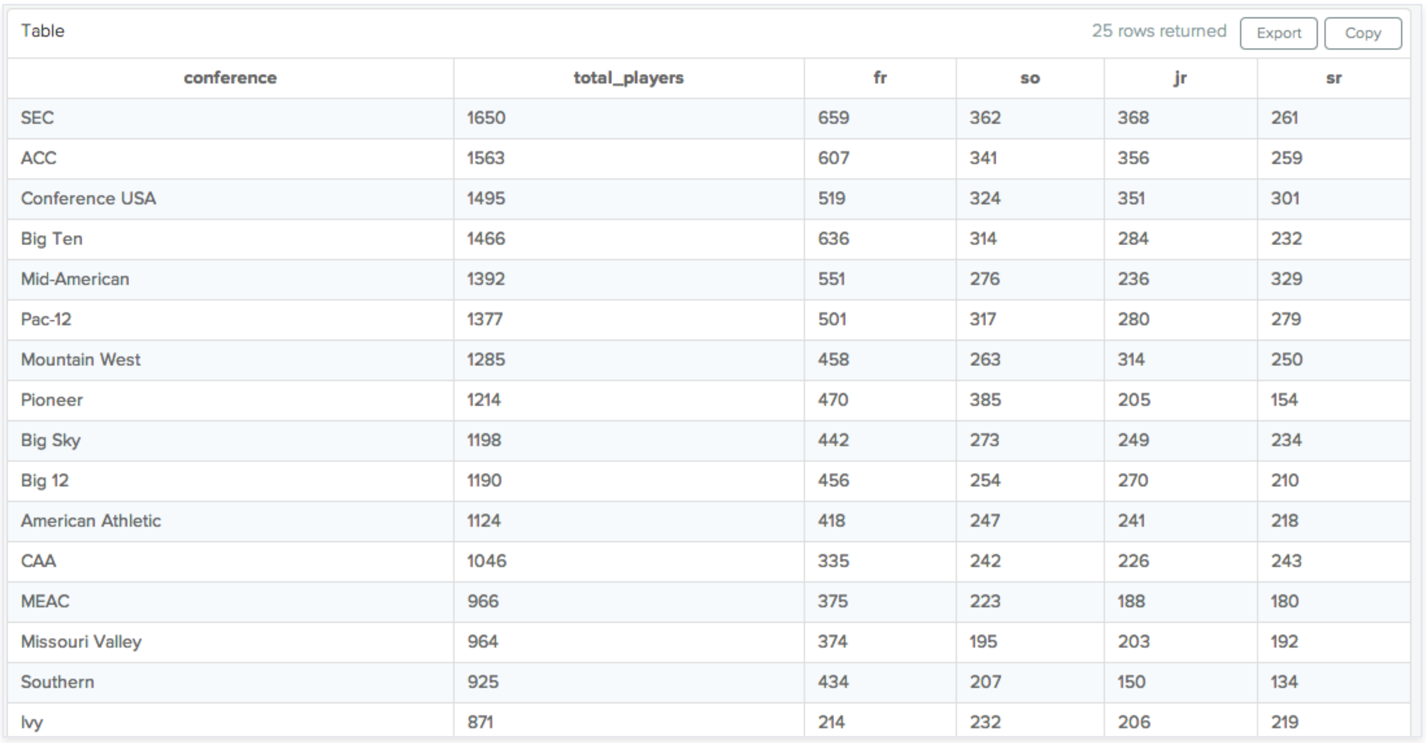


Table 25 rows returned Export Copy

conference	total_players	fr	so	jr	sr
SEC	1650	659	362	368	261
ACC	1563	607	341	356	259
Conference USA	1495	519	324	351	301
Big Ten	1466	636	314	284	232
Mid-American	1392	551	276	236	329
Pac-12	1377	501	317	280	279
Mountain West	1285	458	263	314	250
Pioneer	1214	470	385	205	154
Big Sky	1198	442	273	249	234
Big 12	1190	456	254	270	210
American Athletic	1124	418	247	241	218
CAA	1046	335	242	226	243
MEAC	966	375	223	188	180
Missouri Valley	964	374	195	203	192
Southern	925	434	207	150	134
Ivy	871	214	232	206	219

For this example, we'll use the same dataset of College Football players used in the CASE lesson. You can view the data directly here.

Let's start by aggregating the data to show the number of players of each year in each conference, similar to the first example in the inner join lesson:

```
SELECT teams.conference AS conference,
       players.year,
       COUNT(1) AS players
  FROM benn.college_football_players players
  JOIN benn.college_football_teams teams
    ON teams.school_name = players.school_name
 GROUP BY 1,2
 ORDER BY 1,2
```

View this in Mode.

In order to transform the data, we'll need to put the above query into a subquery. It can be helpful to create the subquery and select all columns from it before starting to make transformations. Re-running the query at incremental steps like this makes it easier to debug if your query doesn't run. Note that you can eliminate the `ORDER BY` clause from the subquery since we'll reorder the results in the outer query.

```
SELECT *
  FROM (
        SELECT teams.conference AS conference,
               players.year,
               COUNT(1) AS players
          FROM benn.college_football_players players
          JOIN benn.college_football_teams teams
            ON teams.school_name = players.school_name
         GROUP BY 1,2
       ) sub
```

Assuming that works as planned (results should look exactly the same as the first query), it's time to break the results out into different columns for various years. Each item in the `SELECT` statement creates a column, so you'll have to create a separate column for each year:

```
SELECT conference,
       SUM(CASE WHEN year = 'FR' THEN players ELSE NULL END) AS fr,
       SUM(CASE WHEN year = 'SO' THEN players ELSE NULL END) AS so,
       SUM(CASE WHEN year = 'JR' THEN players ELSE NULL END) AS jr,
       SUM(CASE WHEN year = 'SR' THEN players ELSE NULL END) AS sr
  FROM (
        SELECT teams.conference AS conference,
               players.year,
               COUNT(1) AS players
          FROM benn.college_football_players players
          JOIN benn.college_football_teams teams
            ON teams.school_name = players.school_name
         GROUP BY 1,2
       ) sub
 GROUP BY 1
 ORDER BY 1
```

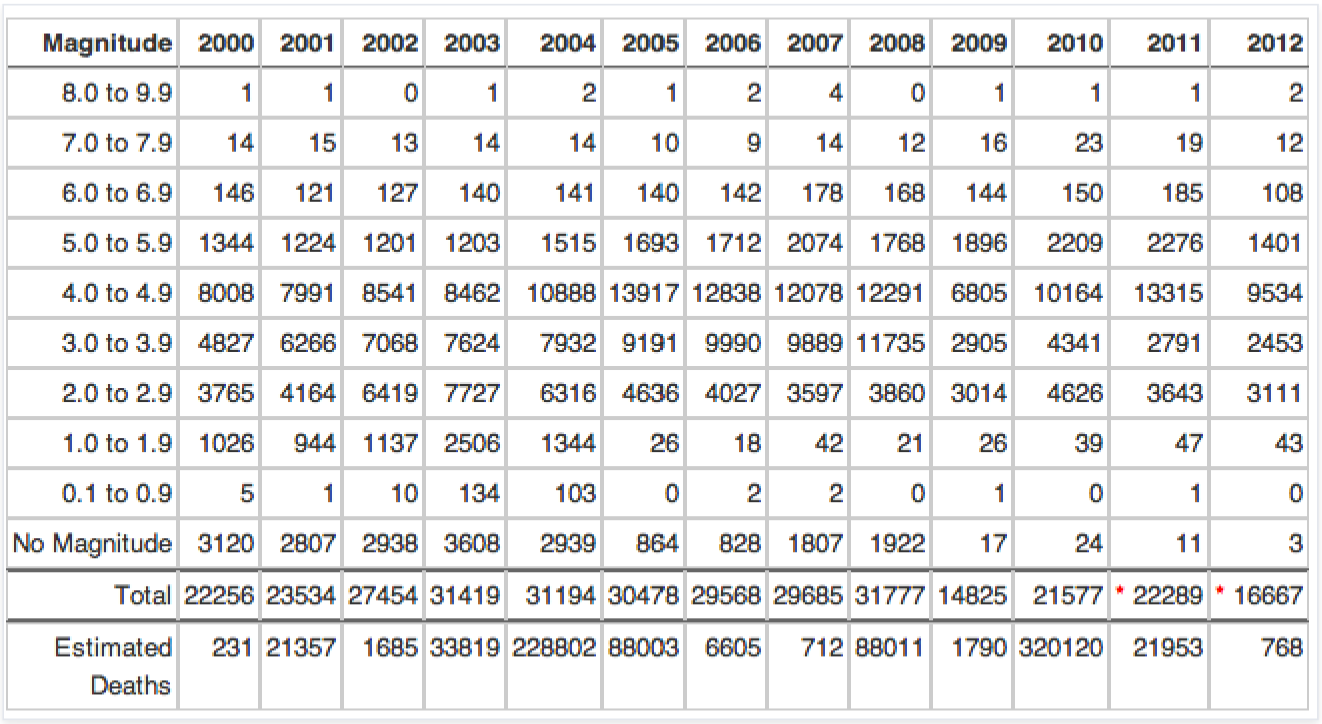
Technically, you've now accomplished the goal of this tutorial. But this could still be made a little better. You'll notice that the above query produces a list that is ordered alphabetically by Conference. It might make more sense to add a "total players" column and order by that (largest to smallest):

```
SELECT conference,
       SUM(players) AS total_players,
       SUM(CASE WHEN year = 'FR' THEN players ELSE NULL END) AS fr,
       SUM(CASE WHEN year = 'SO' THEN players ELSE NULL END) AS so,
       SUM(CASE WHEN year = 'JR' THEN players ELSE NULL END) AS jr,
       SUM(CASE WHEN year = 'SR' THEN players ELSE NULL END) AS sr
  FROM (
        SELECT teams.conference AS conference,
               players.year,
               COUNT(1) AS players
          FROM benn.college_football_players players
          JOIN benn.college_football_teams teams
            ON teams.school_name = players.school_name
         GROUP BY 1,2
       ) sub
 GROUP BY 1
 ORDER BY 2 DESC
```

And you're done! View this in Mode.

Pivoting columns to rows

A lot of data you'll find out there on the internet is formatted for consumption, not analysis. Take, for example, this table showing the number of earthquakes worldwide from 2000-2012:



Magnitude	2000	2001	2002	2003	2004	2005	2006	2007	2008	2009	2010	2011	2012
8.0 to 9.9	1	1	0	1	2	1	2	4	0	1	1	1	2
7.0 to 7.9	14	15	13	14	14	10	9	14	12	16	23	19	12
6.0 to 6.9	146	121	127	140	141	140	142	178	168	144	150	185	108
5.0 to 5.9	1344	1224	1201	1203	1515	1693	1712	2074	1768	1896	2209	2276	1401
4.0 to 4.9	8008	7991	8541	8462	10888	13917	12838	12078	12291	6805	10164	13315	9534
3.0 to 3.9	4827	6266	7068	7624	7932	9191	9990	9889	11735	2905	4341	2791	2453
2.0 to 2.9	3765	4164	6419	7727	6316	4636	4027	3597	3860	3014	4626	3643	3111
1.0 to 1.9	1026	944	1137	2506	1344	26	18	42	21	26	39	47	43
0.1 to 0.9	5	1	10	134	103	0	2	2	0	1	0	1	0
No Magnitude	3120	2807	2938	3608	2939	864	828	1807	1922	17	24	11	3
Total	22256	23534	27454	31419	31194	30478	29568	29685	31777	14825	21577	* 22289	* 16667
Estimated Deaths	231	21357	1685	33819	228802	88003	6605	712	88011	1790	320120	21953	768

In this format it's challenging to answer questions like "what's the average magnitude of an earthquake?" It would be much easier if the data were displayed in 3 columns: "magnitude", "year", and "number of earthquakes." Here's how to transform the data into that form:

First, check out this data in Mode:

```
SELECT *
  FROM tutorial.worldwide_earthquakes
```

Note: column names begin with 'year_' because Mode requires column names to begin with letters.

The first thing to do here is to create a table that lists all of the columns from the original table as rows in a new table. Unless you have a ton of columns to transform, the easiest way is often just to list them out in a subquery:

```
SELECT year
  FROM (VALUES (2000),(2001),(2002),(2003),(2004),(2005),(2006),
               (2007),(2008),(2009),(2010),(2011),(2012)) v(year)
```

Once you've got this, you can cross join it with the `worldwide_earthquakes` table to create an expanded view:

```
SELECT years.*,
       earthquakes.*
  FROM tutorial.worldwide_earthquakes earthquakes
 CROSS JOIN (
       SELECT year
         FROM (VALUES (2000),(2001),(2002),(2003),(2004),(2005),(2006),
                      (2007),(2008),(2009),(2010),(2011),(2012)) v(year)
       ) years
```

Notice that each row in the `worldwide_earthquakes` is replicated 13 times. The last thing to do is to fix this using a `CASE` statement that pulls data from the correct column in the `worldwide_earthquakes` table given the value in the `year` column:

```
SELECT years.*,
       earthquakes.magnitude,
       CASE year
         WHEN 2000 THEN year_2000
         WHEN 2001 THEN year_2001
         WHEN 2002 THEN year_2002
         WHEN 2003 THEN year_2003
         WHEN 2004 THEN year_2004
         WHEN 2005 THEN year_2005
         WHEN 2006 THEN year_2006
         WHEN 2007 THEN year_2007
         WHEN 2008 THEN year_2008
         WHEN 2009 THEN year_2009
         WHEN 2010 THEN year_2010
         WHEN 2011 THEN year_2011
         WHEN 2012 THEN year_2012
         ELSE NULL END
         AS number_of_earthquakes
  FROM tutorial.worldwide_earthquakes earthquakes
 CROSS JOIN (
       SELECT year
         FROM (VALUES (2000),(2001),(2002),(2003),(2004),(2005),(2006),
              (2007),(2008),(2009),(2010),(2011),(2012)) v(year)
     ) years
```

View the final product in Mode.

What's next?

Congrats on finishing the Advanced SQL Tutorial! Now that you've got a handle on SQL, the next step is to hone your analytical process.

We've built the SQL Analytics Training section for that very purpose. With fake datasets to mimic real-world situations, you can approach this section like on-the-job training. Check it out!