

This activity gives a basic example of using SQL to create a database (from data in XML), run both logical and numerical queries, update the database, and see how query result(s) change. It uses one already-provided data file, `NFLTeams.xml`, and SQL queries itemized below. A few questions involve composing your own SQL queries using similar syntax to the examples. Grading is based on completing the examples during the recitation and group discussion of composing commands. Please preserve your work for subsequent upload to *UBLearns*.

Update and Technical Notes: Use DB-Fiddle, <https://www.db-fiddle.com/>, in place of “SQL Fiddle”—as the latter has a usage contention issue in 2022. Directions have been updated accordingly. Be sure to change it from MySQL 5.7 to MySQL 5.6 in the drop-down tab at upper left.

The SQLizer converter works as before. Some field names are abbreviated, notably `ylpw` for “year of last playoff win,” to avoid a sporadic string-length problem. Single quotes in `UPDATE` command changed to double quotes which more PDF renderers leave alone.

1 A Look at the Data

The NFL data file is small—there are only 32 teams and we are considering only three features after the team code, name, and city/region. We can “eyeball” it before loading it into a database. It is in a table on the next page. The features are:

- Population of the team’s regional media market according to [this source](#).
- The year of the last season in which the team *won* a playoff game. When the game occurs in January or February it counts for the previous year’s season. The Bills’ last playoff win was over Miami on Dec. 30, 1995.
- The number of coaches the team has had since the 1990 season. (The XML file abbreviates this to “cs90” because I wanted to keep it on one visual page.)

For the first part of the recitation time, gather in small discussion groups to look at the data. Answer some simple and less-simple questions before you follow the directions to load the data into SQL:

1. Which teams have only 2 letters in their code?
2. Which codes come after ‘L’ in alphabetical order—i.e., after KC in the table?
3. Which team(s) haven’t won a playoff game since 2000? Or since 2010? Is this harder than the previous query?
4. How long would it take you to order the teams by last-playoff-win year? Try it using just the codes—maybe by giving each group member a block of years to look for.
5. Which teams have had the most coaches since 2010? Are the Patriots, with coach Bill Belichick, the only team that has not changed coaches since then?
6. Which team with no playoff win since 2010 has had the fewest coaching changes?

7. Looking ahead to next week, do you see a relationship between the size of the media market and lack of playoff success—that is, long playoff win “droughts”? Supposedly, small-market teams like Buffalo are expected to do worse.
8. Is the volume of coaching turnover connected to lack of playoff success? If you say yes, say which “causes ” what: do playoff absences and/or losses get coaches fired, or does a reputation for high coaching turnover scare away the best people?
9. Or does the small media market prevent from hiring the best coaches, which leads to playoff losses, which cause high turnover?

Yes, folks, this is “small data” but these are big questions—as we who lived here before Josh Allen can attest...

Code	Team Name	Region	Population	Last playoff win	C.s. 2010
ARZ	Cardinals	Arizona	4,438,000	2015	4
ATL	Falcons	Atlanta	6,462,000	2017	4
BLT	Ravens	Baltimore	4,248,000	2020	1
BUF	Bills	Buffalo	2,990,000	2021	5
CAR	Panthers	Carolina	8,152,000	2015	4
CHI	Bears	Chicago	10,606,000	2010	5
CIN	Bengals	Cincinnati	4,666,000	2021	2
CLV	Browns	Cleveland	4,794,000	2020	8
DAL	Cowboys	Dallas	8,071,000	2018	3
DEN	Broncos	Denver	4,794,000	2015	6
DET	Lions	Detroit	8,032,000	1991	6
GB	Packers	Green Bay	4,186,000	2020	3
HST	Texans	Houston	6,452,000	2019	6
IND	Colts	Indianapolis	3,266,000	2018	3
JAX	Jaguars	Jacksonville	3,660,000	2017	8
KC	Chiefs	Kansas City	2,903,000	2021	3
LAC	Chargers	Los Angeles	18,217,000	2018	4
LAR	Rams	Los Angeles	18,217,000	2021	4
LV	Raiders	Las Vegas	2,227,000	2002	8
MIA	Dolphins	Miami	5,722,000	2000	7
MIN	Vikings	Minnesota	5,073,000	2019	4
NYG	Giants	New York City	22,421,000	2011	6
NYJ	Jets	New York City	22,421,000	2010	4
NE	Patriots	New England	9,684,000	2019	1
NO	Saints	New Orleans	2,635,000	2020	4
PHI	Eagles	Philadelphia	8,688,000	2018	5
PIT	Steelers	Pittsburgh	4,396,000	2016	1
SF	49ers	San Francisco	10,645,000	2021	6
SEA	Seahawks	Seattle	4,565,000	2019	1
TB	Buccaneers	Tampa Bay	4,417,000	2021	6
TEN	Titans	Tennessee	2,667,000	2019	5
WAS	Washington Football Team	Washington DC	5,853,000	2005	4

2 Setup Directions

The resources needed are freely accessible online (alternatives are noted in a final “More Fiddling Around” section of this sheet):

- SQLizer: <https://sqlizer.io/> for conversion from XML by file upload.
- DB-Fiddle: <https://www.db-fiddle.com/> for compiling a database (called a “schema”) and executing queries, all by copy-and-paste.

The data file has a line for each of the 32 NFL teams as of 2022, giving its abbreviation, the team name, its city/region, an estimate of the population in TV households from a source noted above and in the file, the year the team last *won* a game in the NFL playoffs, and the number of head coaches the team has had since the 2010 regular season. We will run SQL on the first six questions above, checking the answers found “manually,” and try a few other queries. The same data will be used in the first part of next week’s activity to tell whether there is are (strong) correlations among playoff success, population size, and coaching turnover. We will also update the team name for Washington, DC.

1. Get the file `NFLTeams.xml` from the CSE199 repository or from the <http://www.cse.buffalo.edu/~regan/cse199/> folder.
2. Go to SQLizer and drag-and-drop the `.xml` file or use the Browse button to upload it. Make sure that MySQL is selected at left.
3. Click the big green “Convert My File” bar at the bottom. You should get options to download `NFLTeams.sql` or copy it to the Clipboard. If you do the former, open it in a plain-text editor (e.g., Notepad++ or Apple Text or WordPad or Notepad); if the latter, open your editor, paste in the text, and save it with the name `NFLTeams.sql`.
4. Go to DB Fiddle. Change MySQL 5.6 at top left to MySQL 5.7. You should see an open left-hand window and grayed-out right-hand and bottom windows saying “Please build schema.”
5. Copy and paste all the NFL SQL text (from the clipboard or `NFLTeams.sql`) into the left window, which says “Schema SQL.”
6. There is no positive feedback—things don’t happen until you paste in the queries in the next section. (But clicking the “Run” button at top without a query is OK and may verify that the SQL table code compiles.)

SQL creates optimized internal data structures for the data you entered. Those taking the CSE115-116-250 sequence will learn about the innards of such data structures, but SQL *users* don’t need to see them. It seems that the basic DB-Fiddle *interprets* your whole code on each query, rather than first *compiling* your schema.

There are some **Old Troubleshooting** tips on the last page that applied to previous experience with a utility called SQL Fiddle. The most important *non-glitch* to observe is that in response to your queries, there will be two phantom fields `NFLTeams` and `NFLTeams_Team` at left filled with the NULL value. This is because of how these names are used as prefixing labels by MySQL. We will note alternative SQL renditions at the end. The order of fields may also be different. If you wish to edit your SQL file to eliminate the prefix and/or reorder the fields manually that is OK, but the following directions leave things as-is. Generally, the output from SQLizer is verbose.

3 Try Some Queries

The first queries show the difference between `SELECT *` to display all fields versus displaying only certain fields. First let's see which teams have 2-letter rather than 3-letter abbreviations. You should be able to cut-and-paste the commands (which are all in teletype font) from this sheet into the window. Enter:

```
SELECT * FROM NFLTeams WHERE LENGTH(NFLTeams_Team_code) = 2;
```

and click the "Run" button at top. If you omit the semicolon, a single line will run, but multiple lines will not. *You should paste over your previous commands in the right-hand DB Fiddle window to avoid outputs piling up at the bottom.* Then paste and run this:

```
SELECT NFLTeams_Team_code, NFLTeams_Team_region, NFLTeams_Team_name
FROM NFLTeams
WHERE LENGTH(NFLTeams_Team_code) = 2;
```

You may need to click the full-screen arrows to the right of the blue "Copy As Markdown" button in order to be able to scroll down. Note that line-breaks do not matter in SQL and `=` not `==` is used to test for equality. Although keywords are not case-sensitive, using all-caps for them is standard. Try changing the length at the end from 2 to 3 to see the other teams. Then try:

```
SELECT NFLTeams_Team_code, NFLTeams_Team_region, NFLTeams_Team_name
FROM NFLTeams
WHERE NFLTeams_Team_code >= "L";
```

We listed `NFLTeams_Team_code` first in the selection, but did the rows come out ordered by the code? Or did they come out in order of table entry? Look at SF and SEA. To force ordering by the short code, enter:

```
SELECT NFLTeams_Team_code, NFLTeams_Team_region, NFLTeams_Team_name
FROM NFLTeams
WHERE NFLTeams_Team_code >= "L"
ORDER BY NFLTeams_Team_code;
```

The next directions are 'conceptual' where you create your own queries. These can be discussed and worked out in groups, but be sure that everyone saves the individual queries and notes about the results for upload to *UBLearns*.

1. Include the last playoff win year in the selection and order by it. (When you modify the selection, don't forget the `NFLTeams_Team` prefix and comma between selection items.) Does it come out in ascending or descending order?
2. Remove the `WHERE` clause to see all the teams. Which team follows Detroit in the walk of shame?
3. To order descending, add the keyword `desc` at the very end. What can you say about the ordering of the top six—the teams that won a playoff game last season?
4. Try ordering by the `cs2010` (coaches since 2010) field instead. Which team(s) moved majorly?
5. Find all teams with no playoff win since 2010 that have had 4 or fewer coaches since 2010.

4 Some Formulas

Now we want to add a column showing each team's playoff *drought*, that is, the number of years since the last playoff win. We can do this in the *display* by using a formula as an extra selection. Here we append the formula `2016 - NFLTeams_Team_ylpw`:

```
SELECT NFLTeams_Team_code, NFLTeams_Team_region, NFLTeams_Team_name,
       NFLTeams_Team_ylpw,
       2016 - NFLTeams_Team_ylpw
FROM NFLTeams
ORDER BY NFLTeams_Team_ylpw;
```

If we want to give the new column a descriptive name, we can use the “as” keyword. Note that the following also uses the code as a secondary ordering to break ties:

```
SELECT NFLTeams_Team_code, NFLTeams_Team_region, NFLTeams_Team_name,
       NFLTeams_Team_ylpw,
       2016 - NFLTeams_Team_ylpw AS PlayoffDrought
FROM NFLTeams
ORDER BY NFLTeams_Team_ylpw, NFLTeams_Team_code;
```

Now we want to see the *average* playoff drought is among the NFL teams. SQL has the built-in AVG function for that. Try the following (**note: this query breaks if the upper-left tab says MySQL 5.7 not 5.6**):

```
SELECT NFLTeams_Team_code, NFLTeams_Team_region, NFLTeams_Team_name,
       NFLTeams_Team_ylpw,
       AVG(NFLTeams_Team_ylpw)
FROM NFLTeams
ORDER BY NFLTeams_Team_ylpw;
```

You might expect to see a line giving the average above or underneath all the teams. Is this what you see? (Is Arizona “average”?) OK, remove the code, region, team-name, and last-playoff-win fields from the selection, remove the `ORDER BY` line at the end, and try again. Now you see just the number. If you want to see the team lines again, paste one of the previous lines above as a second SQL query. (That's when the ; separator becomes important.)

What happened with the `AVG` query is that by default it treats the whole database as a single *group*. It can be applied to each of several sub-groups in the data. A typical case is average income grouped by age. To see a similar example, first change just the argument of `AVG` to be `NFLTeams_Team_pop`. This gives you the average population of the teams' media markets. *Then* group this application by the last playoff win year:

```
SELECT NFLTeams_Team_code, NFLTeams_Team_region, NFLTeams_Team_name,
       NFLTeams_Team_ylpw,
       AVG(NFLTeams_Team_pop)
FROM NFLTeams
GROUP BY NFLTeams_Team_ylpw;
```

Averaging together the populations of teams with the same playoff drought length is more meaningful. For good measure, we can have SQL count how many teams are in each average. Add

COUNT(NFLTeams_Team_ylpw) as one more selection to display. SQL still chooses (seemingly arbitrarily) one team from each group to fill out the textual fields. The simple remedy is just removing them from that selection. A fuller way to handle this involves the operation of `join` on databases, whose complexity is beyond our purpose here.

Notice we zapped the previous `ORDER BY` directive. You can put it back, and/or order by the average instead—for the latter you could give it a name using `as` and use that name after `order-by`. Feel welcome to play around if you have time, but there is one more required section of this activity.

5 Updating Data

We now want to update Washington with its new team name. Paste the following in the **left-hand** window of DB-Fiddle. So that you see output, also paste in the third query from section 3, the one with `>= "L"` at the end.

```
UPDATE NFLTeams SET NFLTeams_Team_name = "Commanders"
WHERE NFLTeams_Team_code = "WAS";
```

Click the Run button. At the bottom of the output, you should see that the Washington team name has been updated. (Here is where DB-Fiddle may have ‘gateway’ issues—if they persist, see troubleshooting below.)

The biggest “SQL gotcha” here is forgetting to include the `WHERE` clause. Without it, you would change every team name to be “Commanders”! The upshot is a more subtle responsibility: ensuring that every record has a unique ID. Suppose we’d accidentally entered ‘WAS’ instead of ‘JAX’ for Jacksonville. Then we would rename them to be “Commanders” too and the database would begin to get corrupted...

On that cheery note, we end the in-recitation portion. Again a reminder to upload results of your individual investigations to *UBLearns*.

6 Grading Criteria

The basic 0-1-2-3 scheme will be followed, with 1 for showing up and taking part. System glitches won’t be counted against you in judging how much of the exercise you completed. In any event, this is just “a taste”—not any real training in SQL, though if this inspires you to explore it further, more power to you! We are also trying to encourage discussion. If either new problems occur or good discussion points for everyone emerge we will pause everyone and take time to fix or highlight it. In particular:

- You are welcome to try this ahead of your recitation, but—
- —we still expect everyone to partake in doing it *in* recitation, and—
- —the emphasis is not on people being able to work it all through individually but on community experience and learning.
- In particular, glitches are *to be shared*—knowing them might help others deal with a different but technically-related glitch later.
- This implies multiple ways to earn a top grade but active participation must be one of them.

Old and New Troubleshooting

Your browser viewing `NFLTeams.xml` might show a first line “This XML file does not appear to have any style information...” and then if you use auto-open after download the display may be blank. The fix is to open it with a text-editor to remove the line—or manually select and copy text not including that line and save it in a text editor. From some devices, the previously-used SQL Fiddle gave an error of a too-long `user-string`. The fix was to click “View Sample Fiddle” at top. This shows an example of 15 lines. It is not clear if this applies to DB-Fiddle.

The final `UPDATE` step of this activity used to have single quotes around strings that were rendered “curly” and pasted that way by some systems as characters SQL Fiddle could not interpret. Hence they were changed to double quotes. If they still misbehave, see if there is a paste-without-formatting option. The final fix is to manually edit them in a text editor like Notepad++, or after pasting into the DB Fiddle left-hand window, to be “normal” single or double quotes, around three entries in each of the two lines. If you can’t download the XML file to your device so that the SQLizer upload button will find it, a TA will help you get `NFLTeams.sql` or `NFLTeams.txt` from my unlinked “`/.../cse199/deepweb`” sub-folder.

SQL Fiddle (<http://sqlfiddle.com/>) worked in 2017 and works in single-use now but gives errors when multiple local users access it at the same time. It is not yet clear how much this issue applies to DB Fiddle. In such a case, try zapping everything, entering the original data again, and then pasting in these update lines to try again in one shot. This may also need removing the `IF NOT EXISTS` in the first `CREATE TABLE` line—which can also be prevented in SQLizer by unchecking “Check Table Exists” before clicking the big green Convert button.

A final potential issue might come from SQLizer now specifying that the team-code, name, and region fields use UTF8 encoding. One can try removing the three `CHARACTER SET utf8` parts. This has not been observed, however.