# **COMP-1701 - Transferring Data to Databases**

## **Data Science & Machine Learning (DSML) - RRC Polytech**

Module D.2 - Database Build – ALTER TABLE: Parsing String into 2 Columns

Now that you've learn how to **CREATE TABLE** with a **COLUMN**, let's learn how to interact with data, converting our column into 2 fields.

From the GUI, click on your ***FL*\_*STUDENTID*\_boxstore.sql** database and open your previous working SQL script, as well as your ***FL\_STUDENTID*\_cheatsheet.sql**

**ALTER TABLE syntax**

An ALTER TABLE allows you to ADD/DROP/MODIFY table attributes or table column attributes.

-- ALTER syntax  
ALTER TABLE table\_name   
 ADD COLUMN column3 datatype(size) nullable  
, ADD COLUMN column4 datatype(size) nullable  
...  
, ADD COLUMN columnN datatype(size) nullable;  
  
ALTER TABLE table\_name   
 MODIFY COLUMN columnN datatype(size) nullable;  
  
ALTER TABLE table\_name   
 DROP COLUMN columnN;

-- can be done all together:  
ALTER TABLE table\_name   
 ADD COLUMN column3 datatype(size) nullable  
 MODIFY COLUMN columnN datatype(size) nullable;  
 DROP COLUMN columnN;

explained:

* must update at least 1 item at the table level; and/or
* ADD/DROP/MODIFY at least 1 COLUMN
* for **ADD**/**MODIFY** **COLUMN**, specify: **column\_name**, **datatype**, **size**, **nullable**
* **DEFAULT** is another attribute that can be added after the nullable field, like: **DEFAULT ''** where '' (2 single quotes, means an empty string, does not mean NULL), making the ADD COLUMN line to:  
    
  **ADD COLUMN column\_name datatype(size) nullable DEFAULT ''**
* for **DROP** **COLUMN**, just specify the **column\_name**

Regarding the **nullable** attribute, *during build phase*, best to **allow NULLs**, even if the column requires a value. When adding columns to an existing table with existing data(rows), ALTER only adds columns to the table structure, it has no row values yet, so forcing **NOT NULL** could cause issues. Default is allowing NULL and the database may override a NOT NULL to allow NULLs or error out.

BOXSTORE: Alter the table, to create a first\_name and last\_name columns, why?

In our current design, we have a full\_name column, and should you need to select, filter, order, or update the rows by their First Name or their Last Name, this becomes harder to do. In larger datasets, this can inhibit performance, so we need to split these up, which generally makes the fields more accessible.

When first building tables, it is in your best interest to default the secondary columns to allow NULLs, if its too rigid, more complex datasets may not allow rows to be inserted. Later once you’ve imported the data, you could alter the columns to be NOT NULL

-- alter people table, to add first\_name   
-- last\_name columns  
--   
ALTER TABLE people  
 ADD COLUMN first\_name VARCHAR(40) NULL  
, ADD COLUMN last\_name VARCHAR(60) NULL;   
  
-- view your people table structure mods  
DESCRIBE people;  
  
-- list instructor and your name rows  
SELECT \* FROM people WHERE p\_id<=2;

**UPDATE / SET syntax**

Data within tables get updated, so there is a SQL command for that.

UPDATE table\_name   
SET column1=value1  
 , column2=value2  
 , ...  
 , columnN=valueN  
WHERE *condition;*

explained:

* must at least update 1 column in the table
* if updating 2 columns, requires a comma to delimit
* SET is where you specify the column you want to update and the value you want to update it to
* ***condition*** is an expression that will evaluate to **TRUE**, to show the rows, such as:
* p\_id=1 will only update the result(s) where p\_id equals 1
* prior to running any update, build your SELECT with current column values and column values they will be changed to
* you also build your reversion query as well, to restore the results

Update the first 2 records of our people table

Here we will manually update the first 2 records of our people table, this will require 2 update statements, but first let's look at the 2 records:

SELECT p\_id, full\_name, first\_name, last\_name  
FROM people  
WHERE p\_id <= 2;

...with first\_name and last\_name being NULL to start out, we can safely add values to those columns.

-- reset for rerun  
UPDATE people  
SET first\_name=NULL  
 , last\_name =NULL  
WHERE p\_id IN (1,2);

-- value updates for p\_id 1 and 2  
UPDATE people  
SET first\_name='InstructorFirstName'  
 , last\_name ='InstructorLastName'  
WHERE p\_id=1;

UPDATE people  
SET first\_name='YourFirstName'  
 , last\_name ='YourLastName'  
WHERE p\_id=2;

Now what about the rest of the names, not logical that we have to go thru each one individually like this, is it?

**STRING FUNCTIONS**

When it comes to parsing data, splitting data up, SQL has several functions that can help you address this:

<https://www.w3schools.com/sql/sql_ref_mysql.asp>

When planning to do some parsing/splitting of fields, in this case the ***full\_name*** column's data structure, sometimes you may want to build yourself a quick ruler, and mark the spots you are looking to parse from, so let's look at the 'FirstName LastName' values within the full\_name field:

-- 0 1 2 ruler tens  
-- 012345678901234567890 ruler ones  
-- 'FirstName LastName'  
-- 1 ^?^ ^  
-- |^|= space position (SP)  
-- ? ? ? = LENGTH - SP  
-- SP-1 SP+1

INSTR(*stringvalue*, *stringtofind*) = NUMBER

This function will determine the numeric position of the *stringtofind* within the *stringvalue*.

Since we have already addressed the first 2 records, we will just run it on the last batch of records (thus p\_id>=3).

-- determine the position of the space  
--   
SELECT full\_name  
 , INSTR(full\_name,' ') AS pos  
 , INSTR(full\_name,' ')-1 AS first\_name\_end\_pos  
 , INSTR(full\_name,' ')+1 AS last\_name\_beg\_pos  
FROM people  
WHERE p\_id>=3;

...ok, so now we need to find a function, that will grab the *stringvalue* up to the space (and not including the space) to scope out the first\_name to start...

MID(*string*, *start*, *length*) = STRING

This function will take any string and parse out from the starting character number, up to a certain number of characters.

-- get first name, starts at position 1 and ends  
-- at space position minus 1 (aka: length of  
-- first name)  
--   
SELECT full\_name  
 , INSTR(full\_name,' ')-1 AS first\_name\_end\_pos  
 , MID(full\_name,1,INSTR(full\_name,' ')-1) AS first\_name  
FROM people  
WHERE p\_id>=3;

...phase 1 complete, we have the first\_name. Now we reuse our **INSTR** function of the space position, add 1 to it, to get to the first letter of the last\_name...

-- the last name's first letter position is the   
-- space's location plus 1  
--   
SELECT full\_name  
 , MID(full\_name, 1, INSTR(full\_name,' ')-1) AS first\_name  
 , INSTR(full\_name,' ')+1 as last\_name\_beg  
FROM people  
WHERE p\_id>=3;

...now we have the location of the first letter of the last\_name, now we just grab to the end...again using the MID function:

-- MID function, starting at space plus 1, using   
-- the column's max size as string length  
--   
SELECT full\_name  
 , MID(full\_name, 1, INSTR(full\_name,' ')-1 ) AS first\_name  
 , MID(full\_name, INSTR(full\_name,' ')+1 ,100) AS last\_name  
FROM people  
WHERE p\_id>=3;

...while you can do the math's to get the exact length of the remainder of the string, it isn't necessary, the 100 is just the maximum size of the full\_name column (ie: full\_name varchar(100) ), and it won't grab anything beyond the end of the varchar string, because there is nothing beyond the last character of it. 100 could be replaced with:

LENGTH(full\_name)-INSTR(full\_name,' ')

...to get the length of the last\_name, then the query could change to:

-- added length less space pos for last name   
-- length  
--   
SELECT full\_name  
 , MID(full\_name, 1,INSTR(full\_name,' ')-1) AS first\_name  
 , MID(  
 full\_name  
 , INSTR(full\_name,' ')+1  
 , LENGTH(full\_name)-INSTR(full\_name,' ')  
 ) AS last\_name  
FROM people  
WHERE p\_id>=3;

So now you have the column values solved for, now it's time to update both fields in the database. However, you could test by overwriting the first 2 that you already updated

-- updates first\_name and last\_name by splitting  
-- up full\_name column  
--   
UPDATE people  
SET first\_name = MID(full\_name, 1, INSTR(full\_name,' ')-1)  
 , last\_name = MID(  
 full\_name  
 , INSTR(full\_name,' ')+1  
 , LENGTH(full\_name)-INSTR(full\_name,' ')  
 )  
WHERE 1=1;  
  
-- ie: first 2 rows: p\_id<=2;  
-- ie: after row 2 to the end p\_id>=3;  
   
-- always verify your changes:  
SELECT \* FROM people;

The other thing here, is you should **always be prepared to revert** your changes. For complex updates, you may need an entire table backup. When working with a running script, you can prepare it, so you can always start over. In this particular case, it is very easy to do, you would just set first\_name and last\_name to equal NULL, as in:

-- updates first\_name and last\_name by splitting  
-- up full\_name column  
--   
UPDATE people  
SET first\_name = NULL  
 , last\_name = NULL  
WHERE 1=1;  
  
-- updates first\_name and last\_name by splitting  
-- up full\_name column  
--   
UPDATE people  
SET first\_name = MID(full\_name, 1, INSTR(full\_name,' ')-1)  
 , last\_name = MID(  
 full\_name  
 , INSTR(full\_name,' ')+1  
 , LENGTH(full\_name)-INSTR(full\_name,' ')  
 )  
WHERE 1=1;  
  
-- and again, always verify your changes:  
SELECT \* FROM people;

In this course, you can do some quick checks with \* which shows all columns, though in practice you should type out all the column names, which is required for the assignments

**QUERY CODE CLEANSING**

When writing your queries, there are multiple ways to keep the code looking clean, here are some tips:  
  
Single **COLUMN** query vs. Multiple **COLUMN** query  
If you are just selecting one column from one table, this is what I refer to as a shorty, and can be done on one line:

-- short queries should be written on 1 line  
--   
SELECT first\_name FROM people;  
  
-- if you are requesting more than 1 column, then the FROM   
-- should be pushed to another line  
--  
SELECT first\_name, last\_name -- comma then space  
FROM people;

Comment lines and column delimiter commas , should both have a space following them, as per the example above, -- comments also need a space after it, so you aren’t smudging text right after it, making the code more readable.   
  
Ample line spacing for sections is recommended, so one blank line if within a related block of code, 3 blank if it is a different section.

Take care of your code, or you will be asked to re-write it.  
  
Lining up your query vertically, also helps with readability:

-- always start your code in position 1  
SELECT ...  
FROM ...  
WHERE ...  
ORDER BY ...;

-- if your line is going beyond 70 characters,  
-- you continue from the next line, and put in   
-- 4 spaces, then continue the query  
  
-- acceptable form of query, in general, however   
-- make the query more readable, as seen below  
SELECT first\_name, last\_name   
FROM people  
WHERE first\_name='Brad'  
ORDER BY last\_name;   
  
-- now using a spaced out version, so columns, tables,  
-- predicates, and sort columns, all line up at the   
-- 10 position,   
SELECT first\_name, last\_name   
FROM people  
WHERE first\_name='Brad'  
ORDER BY last\_name;  
  
-- if no ordering, your columns/table/predicate could  
-- start at position 8  
SELECT first\_name, last\_name   
FROM people  
WHERE first\_name='Brad';  
  
-- also, don’t add a -- comment line after the semi-colon  
-- just do the comment on a separate line, if selected with  
-- your query to be ran, it will return a syntax error:  
SELECT first\_name, last\_name   
FROM people  
WHERE first\_name='Brad'; -- Syntax Error if  
 -- comment ran with   
 -- query, only select  
 -- up to semicolon

When running scripts, manually, you are either selecting a block of queries or single queries at a time. So you select highlight(select) them with your mouse, then run just those queries. If you are running your entire script, every time, you will lose time. Its all sequential, so it should run from top to bottom, though there is no need to always do that.

-- ALTER syntax  
ALTER TABLE people DROP COLUMN full\_name;

-- current final table query without full\_name  
-- adding table aliases in, when designing tables  
-- we can shorten our queries with abbreviations  
-- of our standard table names, people will be p  
SELECT p.p\_id, p.first\_name, p.last\_name  
FROM people p;

## 