UPDATE to Edit Multiple Rows

by Sophia Tutorial

WHAT'S COVERED

This tutorial explores using the UPDATE statement to update a set of rows in one statement rather than individually, in two parts:

1. Using UPDATE to Edit Multiple Rows
2. Using RETURNING

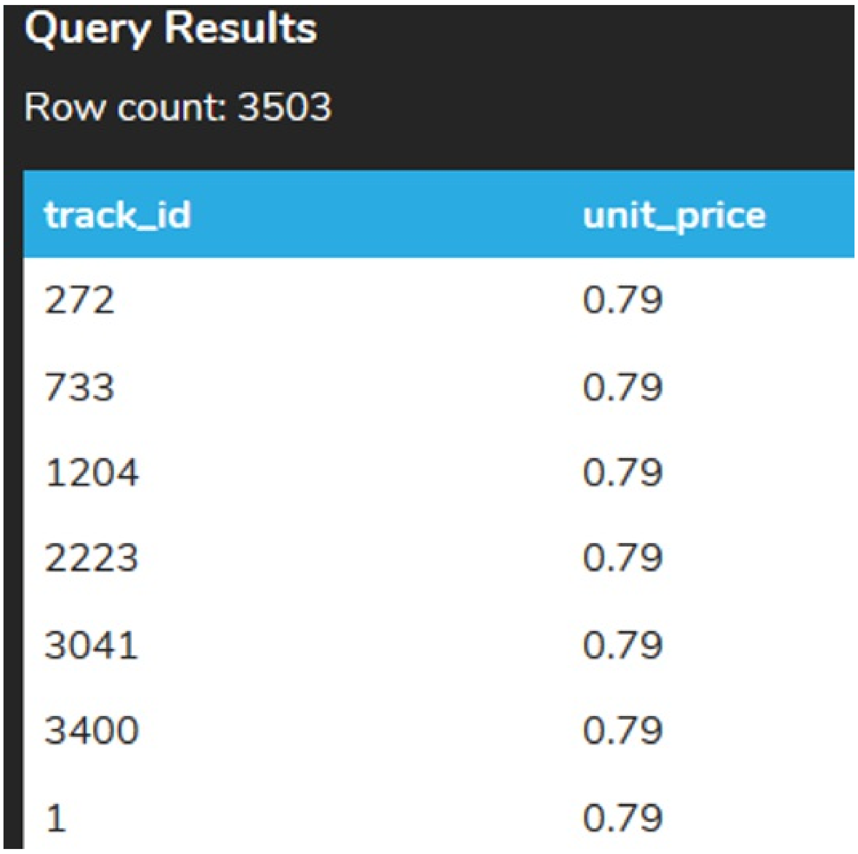
1. Using UPDATE to Edit Multiple Rows

The UPDATE statements that we have worked on so far have been focused on updating individual rows. The complexity of the UPDATE statement does increase as you have to be careful of updating rows that you did not intend to update. There are times when we may want to update the entire table. For example, if we consider that we may have a permanent sale on the tracks to give a 20% discount, we can update the entire table like this:

```
UPDATE track
SET unit_price = unit_price * .8;
```

Notice with our UPDATE statement that we have unit_price = unit_price * .8. This is taking the existing value of unit_price, multiplying it by .8 (or discounting it by 20%), and then setting that value to the unit_price. Since we do not have a WHERE clause, this will update every single row.

We will be able to see that the unit_price has been decreased from 0.99 for most tracks to 0.79:


Query Results

Row count: 3503

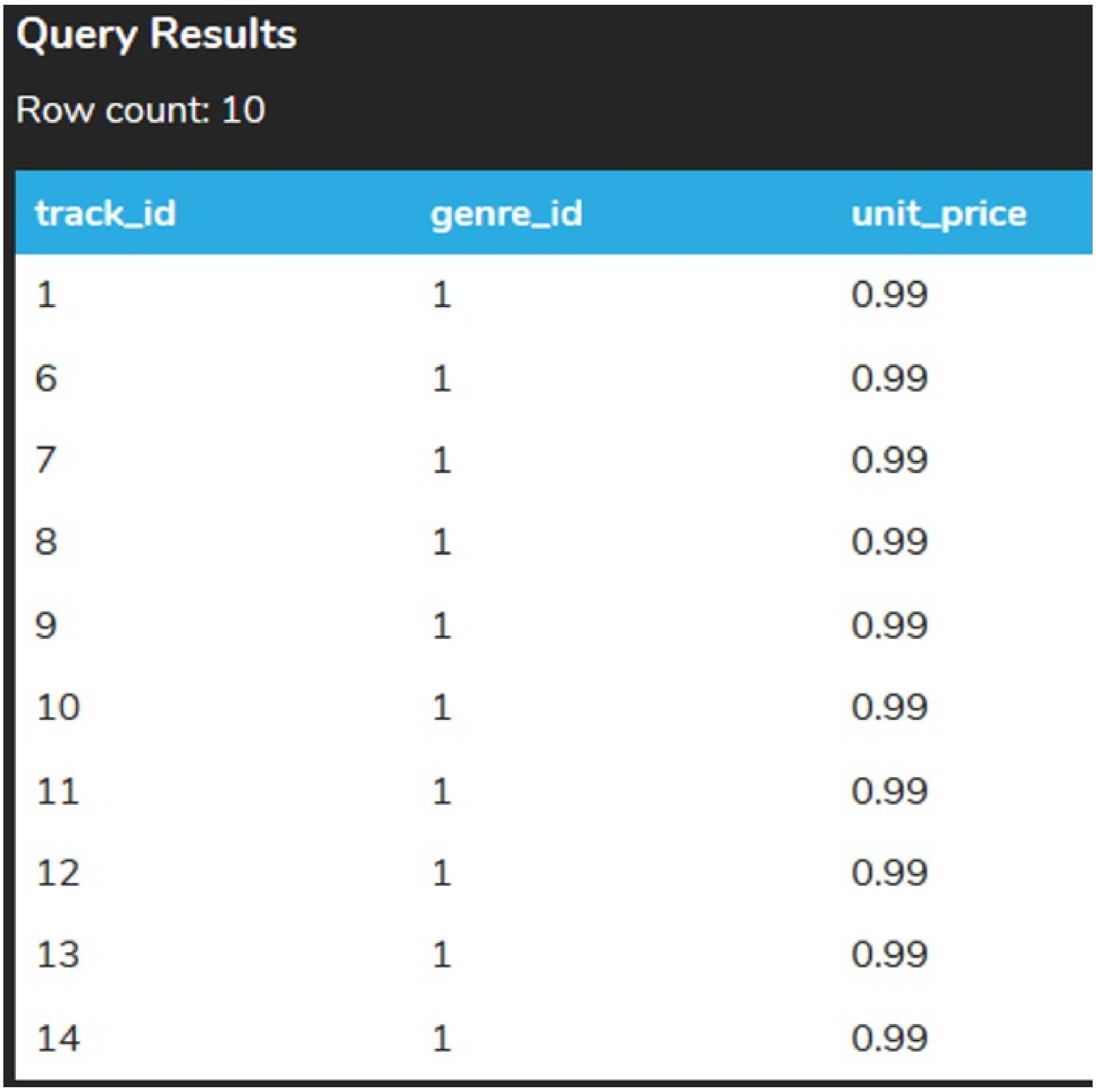
track_id	unit_price
272	0.79
733	0.79
1204	0.79
2223	0.79
3041	0.79
3400	0.79
1	0.79

It can be beneficial to first use the WHERE clause to verify the rows that you want to update before you add it to the UPDATE statement.

For example, we may want to give a discount on the tracks for a specific album. We can first start by selecting it:

```
SELECT track_id, genre_id, unit_price
FROM track
WHERE album_id = 1;
```

This returns us with 10 rows:


Query Results

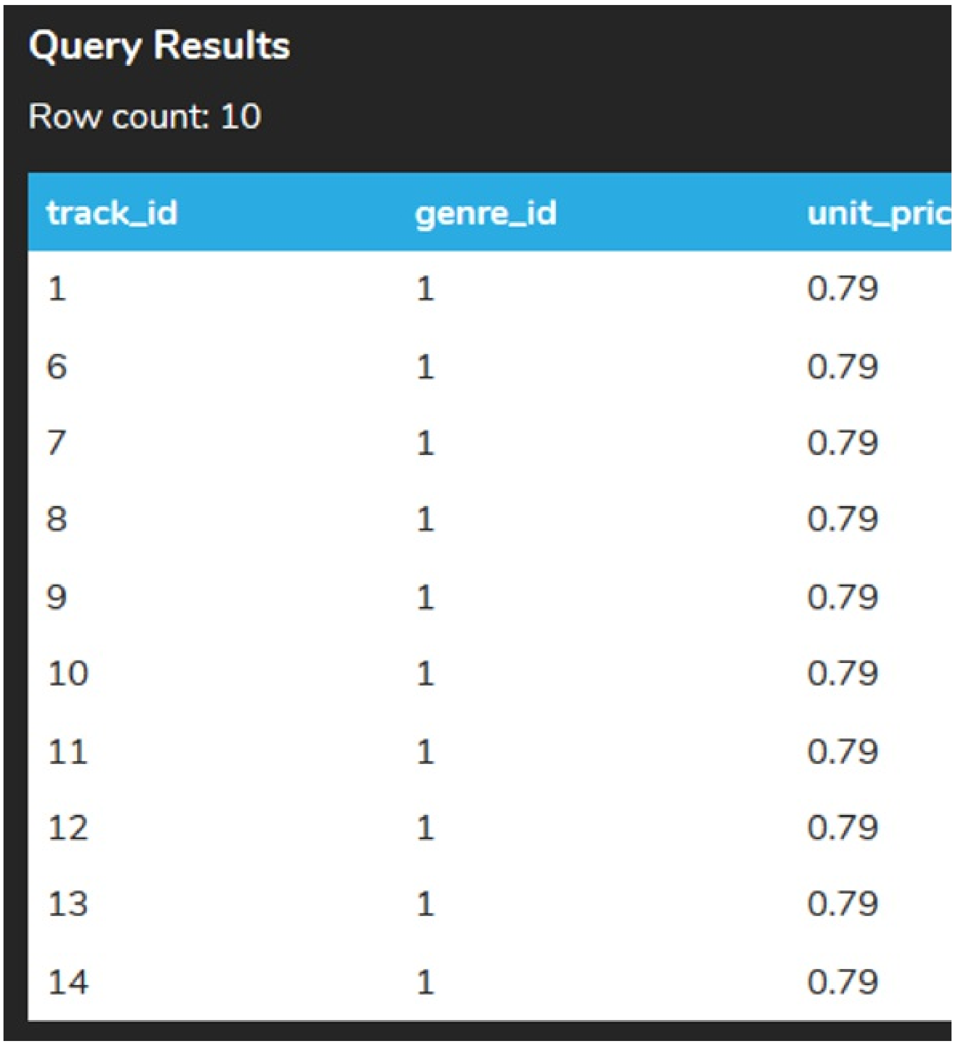
Row count: 10

track_id	genre_id	unit_price
1	1	0.99
6	1	0.99
7	1	0.99
8	1	0.99
9	1	0.99
10	1	0.99
11	1	0.99
12	1	0.99
13	1	0.99
14	1	0.99

Using the same WHERE clause, we can then apply the update:

```
UPDATE track
SET unit_price = unit_price * .8
WHERE album_id = 1;
```

If we now query the table, we should see the unit_price being updated:


Query Results

Row count: 10

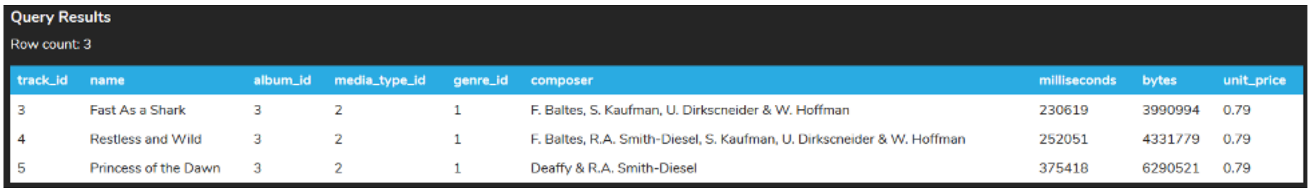
track_id	genre_id	unit_pric
1	1	0.79
6	1	0.79
7	1	0.79
8	1	0.79
9	1	0.79
10	1	0.79
11	1	0.79
12	1	0.79
13	1	0.79
14	1	0.79

2. Using RETURNING

However, it would not be feasible to check every single other row, since there are thousands of rows in the table. This is where using the RETURNING * would be useful. Let's apply the same discount to those tracks with the album_id set to 3.

```
UPDATE track
SET unit_price = unit_price * .8
WHERE album_id = 3
RETURNING *;
```

Immediately after running the statement, we can quickly see that three rows were affected by the changed data:

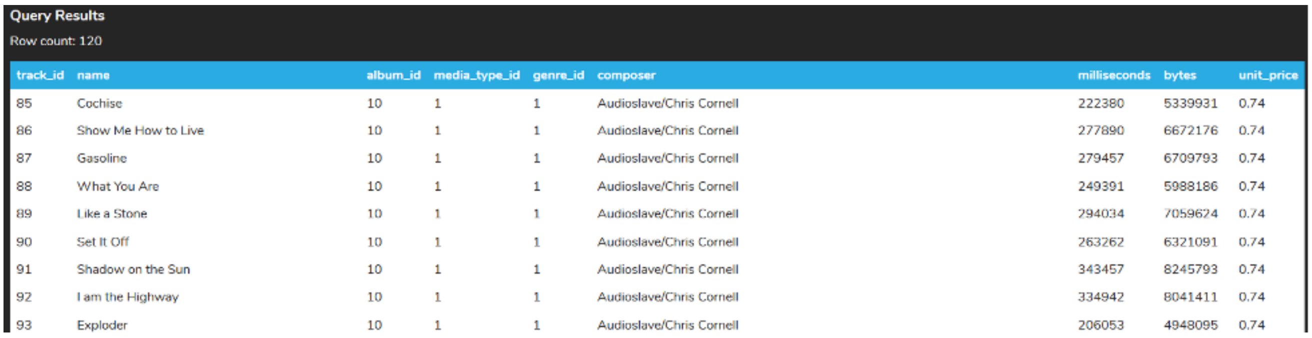

Query Results

Row count: 3

track_id	name	album_id	media_type_id	genre_id	composer	milliseconds	bytes	unit_price
3	Fast As a Shark	3	2	1	F. Baltes, S. Kaufman, U. Dirkscneider & W. Hoffman	230619	3990994	0.79
4	Restless and Wild	3	2	1	F. Baltes, R.A. Smith-Diesel, S. Kaufman, U. Dirkscneider & W. Hoffman	252051	4331779	0.79
5	Princess of the Dawn	3	2	1	Deaffy & R.A. Smith-Diesel	375418	6290521	0.79

We can also use ranges, such as applying a 25% discount to those tracks with album_id values between 10-20:

```
UPDATE track
SET unit_price = unit_price * .75
WHERE album_id BETWEEN 10 AND 20
RETURNING *;
```



Query Results

Row count: 120

track_id	name	album_id	media_type_id	genre_id	composer	milliseconds	bytes	unit_price
85	Cochise	10	1	1	Audioslave/Chris Cornell	222380	5339931	0.74
86	Show Me How to Live	10	1	1	Audioslave/Chris Cornell	277890	6672176	0.74
87	Gasoline	10	1	1	Audioslave/Chris Cornell	279457	6709793	0.74
88	What You Are	10	1	1	Audioslave/Chris Cornell	249391	5988186	0.74
89	Like a Stone	10	1	1	Audioslave/Chris Cornell	294034	7059624	0.74
90	Set It Off	10	1	1	Audioslave/Chris Cornell	263262	6321091	0.74
91	Shadow on the Sun	10	1	1	Audioslave/Chris Cornell	343457	8245793	0.74
92	I am the Highway	10	1	1	Audioslave/Chris Cornell	334942	8041411	0.74
93	Exploder	10	1	1	Audioslave/Chris Cornell	206053	4948095	0.74

What you will notice is that anything you can query and filter using the WHERE clause in the SELECT statement can be filtered in the same manner as the UPDATE statement.

Video Transcription

[MUSIC PLAYING] The UPDATE statement can also be utilized to update multiple rows at the same time. However, you do have to be careful that when you're writing the UPDATE statement that it's not going to update every single row in a table if that's not what you intended to do.

Let's take a look at an example here with track table where we have the track_id, the media_type_id and the unit_price. There's multiple different other items we want to key into these three columns.

So with the track_id being the primary key, if we made an update there, it would only affect one option. However, perhaps we have the media_type_id of being 1 and we want to update every instance where the media_type_id, we're going to have a sale and the unit price is going to be changed to $0.75. So how do we do that?

Go ahead and do UPDATE and then table name, track. We're going to set a unit_price equal to $0.75 where the media_type_ID is equal to 1. And that's basically it. It's very similar to how you have the track_id but the only difference in this case here is that it's going to be based on a different column rather than the primary key.

So if we go ahead and do that, we can go back, run this. And we'll see in this case here we'll actually be closer to here, where the unit price is at $0.75 whenever we see an instance of the media_type_id being set to 1.

TRY IT

Your turn! Open the SQL tool by clicking on the LAUNCH DATABASE button below. Then enter in one of the examples above and see how it works. Next, try your own choices for which columns you want the query to provide.

SUMMARY

The UPDATE statement can be used to update multiple rows at the same time.

Source: Authored by Vincent Tran