UNIQUE to Validate Data

by Sophia

WHAT'S COVERED

In this lesson, you will explore the use of the UNIQUE constraint to ensure that the data in a column or columns are unique across all rows, in two parts. Specifically, this lesson will cover:

1. The UNIQUE Constraint
2. ALTER TABLE Statement

1. The UNIQUE Constraint

The **UNIQUE** constraint's purpose is to ensure that all values in a column are different. As you learned earlier in this challenge, the PRIMARY KEY constraint automatically enforces the UNIQUE constraint. However, the UNIQUE constraint can be applied to other columns too, to ensure that they also contain unique values.

When the UNIQUE constraint is applied to a column, and you try to insert a new row, the system will make sure that no duplicate value exists in any existing records. If there is a duplicate value, the system rejects the insertion and displays an error message. The same is true if you try to update a row in a way that would violate the constraint.

For example, recall the contact table we created in an earlier lesson, which used the contact_id column as the primary key. Its other columns were username and password. Although the username column is not the primary key, it should still contain a unique value for each record. To ensure that, we would add the UNIQUE constraint to the clause that creates the username column, like this:

```
CREATE TABLE contact(
contact_id SERIAL PRIMARY KEY,
username VARCHAR(50) UNIQUE,
password VARCHAR(50)
);
```

If we inserted a row or updated a row into the table that had the same username as an existing row, we should see an error similar to the following:

Query Results

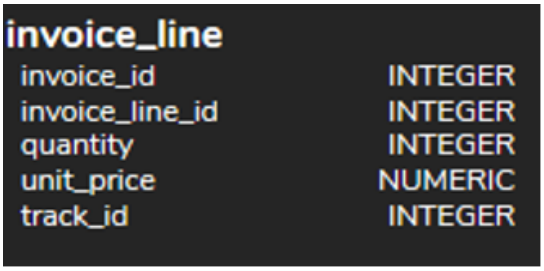
Query failed because of: error: duplicate key value violates unique constraint "contact_username_key"

We can also change this to set the UNIQUE constraint as a table constraint by doing the following:

```
CREATE TABLE contact(
contact_id SERIAL PRIMARY KEY,
username VARCHAR(50),
password VARCHAR(50),
UNIQUE(username)
);
```

Notice that in the above example, UNIQUE is used as a function that applies to the entire table.

Using the UNIQUE() function as a table-level constraint can also be useful in a table where there are multiple foreign-key columns and the combination of them should be unique. A good example of this is in our invoice_line table:



The invoice_line has two foreign keys: the invoice_id that references the invoice_id in the invoice table and the track_id in the track table. For a given invoice, the track_id should exist only once because if a customer purchased more than one track in the same order, the quantity would be incremented in the column. In this case, the invoice_id and the track_id together should be unique. We could do this by adding the following in the CREATE TABLE statement as a table constraint:

```
UNIQUE(invoice_id, track_id)
```

This will ensure that for any given invoice_line row, the combination of the invoice_id and track_id must be unique in the entire table.

TERM TO KNOW

UNIQUE

UNIQUE constraints ensure that all values in a column are different.

2. ALTER TABLE Statement

The best time to add constraints to a table is during its creation. However, that's not your only opportunity. We can also add the UNIQUE constraint on an existing table through the ALTER TABLE statement. However, if the table already contains data, and the existing data in the table violates the constraint you are trying to add, the table alteration will fail.

To add a UNIQUE constraint on an existing table, we can use the ALTER TABLE statement. Here's the syntax for that:

```
ALTER TABLE <tablename> ADD CONSTRAINT <constraintname> UNIQUE(<column>);
```

This statement uses the ADD CONSTRAINT clause to indicate that you want to add a constraint. It then uses the UNIQUE function to specify which column(s) to make unique.

For example, suppose that in the earlier example of creating the contact table, you had neglected to include the UNIQUE(username) clause. You could add it with the following statement:

ALTER TABLE contact ADD CONSTRAINT username_unique UNIQUE(username);

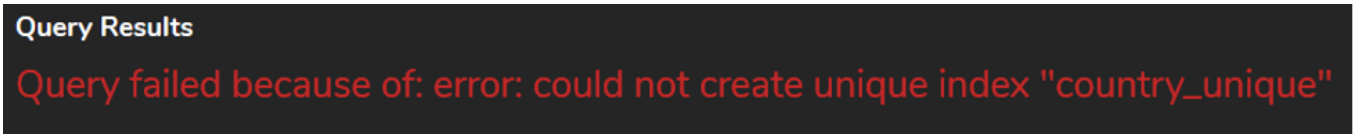
The ADD CONSTRAINT clause creates a name for the constraint; in the above example, that name is username_unique, but it could be anything. For example, it could just as easily be unique_username.

The UNIQUE function is what actually assigns the constraint to the field(s).

As previously mentioned, the ALTER TABLE statement will not run if the existing data in the table violates the constraint being applied. For example, in looking at the customer table, suppose we tried to create a unique constraint on the country column:

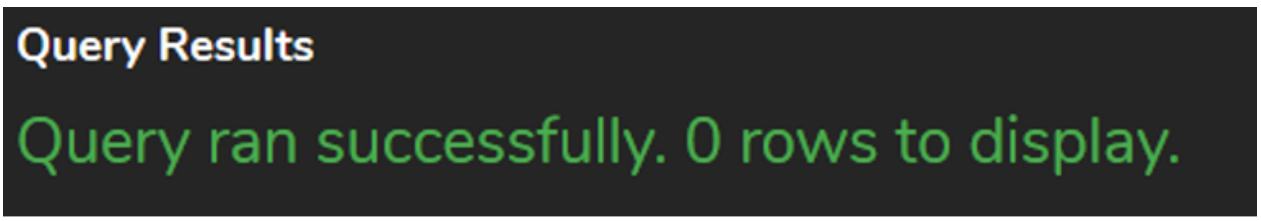
```
ALTER TABLE customer ADD CONSTRAINT country_unique UNIQUE(country);
```

An error message would appear, like this one:



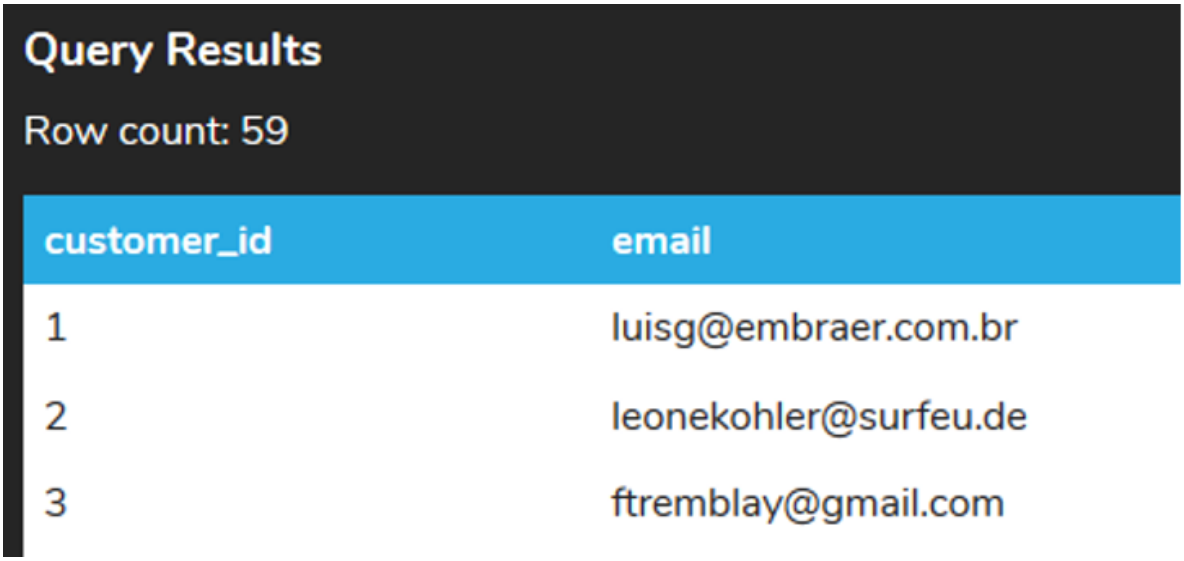
The constraint could not be added because there are at least two records already in the table that have the same value for the country column. However, we could add a constraint on the customer's email, which is unique:

```
ALTER TABLE customer ADD CONSTRAINT email_unique UNIQUE(email);
```



After running the above statement to force the email column to be unique, we would no longer be able to change the email column's content for any record where that uniqueness would be violated. To test this, first display a list of all the customer IDs and their emails:

```
SELECT customer_ID, email
FROM customer;
```



Query Results

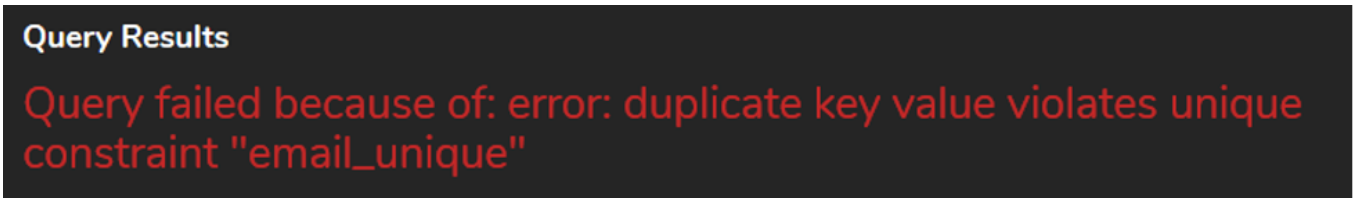
Row count: 59

customer_id	email
1	luisg@embraer.com.br
2	leonekohler@surfeu.de
3	ftremblay@gmail.com

Consider if we tried to set the customer with the customer_id equal to 1 to have the same email address as what customer_id equal to 3 has.

```
UPDATE customer
SET email = 'ftremblay@gmail.com'
WHERE customer_id = 1;
```

We could see the following result:



Query Results

Query failed because of: error: duplicate key value violates unique constraint "email_unique"

WATCH

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

During this lesson, you learned that the **UNIQUE constraint** in PostgreSQL is a feature that ensures the uniqueness of values within one or more columns. Applying the UNIQUE constraint to a column or group of columns ensures that no duplicate values will exist. This constraint maintains data integrity by preventing the insertion or updating of records with conflicting values. It can be specified at the time of table creation or added later.

When adding a table-level UNIQUE constraint, you can use the UNIQUE() function with the desired column(s) in its parentheses. The UNIQUE constraint is best applied when creating the table, but it can be applied later using the **ALTER TABLE statement**, provided that none of the existing data in the table would violate the new constraint.

Source: THIS TUTORIAL WAS AUTHORED BY DR. VINCENT TRAN, PHD (2020) AND Faithe Wempen (2024) FOR SOPHIA LEARNING. PLEASE SEE OUR TERMS OF USE.

TERMS TO KNOW

UNIQUE

UNIQUE constraints ensure that all values in a column are different.