In this lesson we are looking at the SQL SEQUENCE, which is going to allow is to give our user ids incremental values without us having to do anything.

Let’s again revisit the contents of our users’ table.

SELECT \* FROM users;

|  |  |
| --- | --- |
| id  Integer | name  character varying (100) |
| 1 | Mary |
| 2 | Rolf |

Now we will try to enter a name in this table with out entering any id.

INSERT INTO users(name)

VALUES(Jose);

* ERROR

We get an ERROR from Postgres while trying to enter a user’s name with out entering his id. Postgres tells us that the ID field cannot be NULL as it is a PRIMARY KEY and it needs to have a value entered to it.

That’s why in Postgres either we need to enter a unique number to the id field or else Postgres shows us error. What if we have thousands of names in our table and when we have to enter a new name, we do not know what numbers are not used in the table? Then which number should we enter for the new name?

Here’s where SEQUENCE comes in. A Sequence in Postgres is not available in many other SQL management systems, so we need to use it carefully. In other systems we have things like auto-increment, but we are not going to investigate that as it is not available in Postgres.

In Postgres SQL we need to create a sequence. A Sequence is just a generator of numbers sequentially and the numbers are unique because they one comes after another.

The first thing we need to do is create a sequence and give it a name.

CREATE SEQUENCE users\_id\_seq;

Now we can run it, but there is one problem is that it is going to start at 1 and that is going to give us an error as we already have two users in our table and numbers 1 and 2 would contradict with new users starting with 1 and 2. That is why we can tell Postgres to start our sequence from 3.

CREATE SEQUENCE users\_id\_seq START 3;

After running the above query, Postgres will start the sequence from 3.

Now we need to change the default value of ID column in our users’ table. To do that we will introduce a new method called ALTER TABLE.

ALTER TABLE users

ALTER COLUMN id

SET DEFAULT nextval(‘users\_id\_seq’);

In this query we have first defined the ALTER TABLE command followed by the table name where we are altering and then we have defined the ALTER COLUMN command followed by the column name and then finally we have defined SET DEFAULT command followed by the nextval method, this method lets Postgres know that we want it to give the next unique value that is available and in parentheses we have mentioned the column name where we have asked for the unique value. Then we just run the query and it is executed successfully.

The last thing that we need to do is to tell Postgres that although the ID column is using the sequence, the id column also own the sequence, and what that achieves is that when we delete the table for example, then the SEQUENCE will also get deleted. That makes sense because we are not going to reuse the SEQUENCE in another table.

Let’s tell Postgres that this SEQUENCE is owned by the ID column in our table.

ALTER SEQUENCE users\_id\_seq OWNED BY users.id;

Now we will try entering the name of the user that we were trying to enter before getting introduced to the SEQUENCE command.

INSERT INTO users(name)

VALUES (‘jose’);

SELECT \* FROM users;

|  |  |
| --- | --- |
| id  integer | name  character varying (100) |
| 1 | Mary |
| 2 | Rolf |
| 3 | Jose |

As we can see we get the id number 3 for our newest entry to the users’ table. That is how we create a SEQUENCE and how apply it to a column. This allows us to auto-increment ids like this one.