NEON

December 7, 2022 6 min read

Time Travel with Serverless Postgres

Data recovery with database branching

One unfortunate scenario you might run into is running a SQL query that accidentally results in data loss. To deal with this issue, you would typically need to have backups and then roll back your database to a previous state.

Neon's database branching feature enables you to create copies of your database at any point in time to restore your data to a previous state within seconds.

This article will cover how Neon's data branching works and how you can use it for disaster recovery. If you prefer to watch a video instead, you can check out this **Developer Days** talk, presented by one of Neon's Co-Founders, **Stas Kelvich**.

All-Things-Branching

What is a branch?

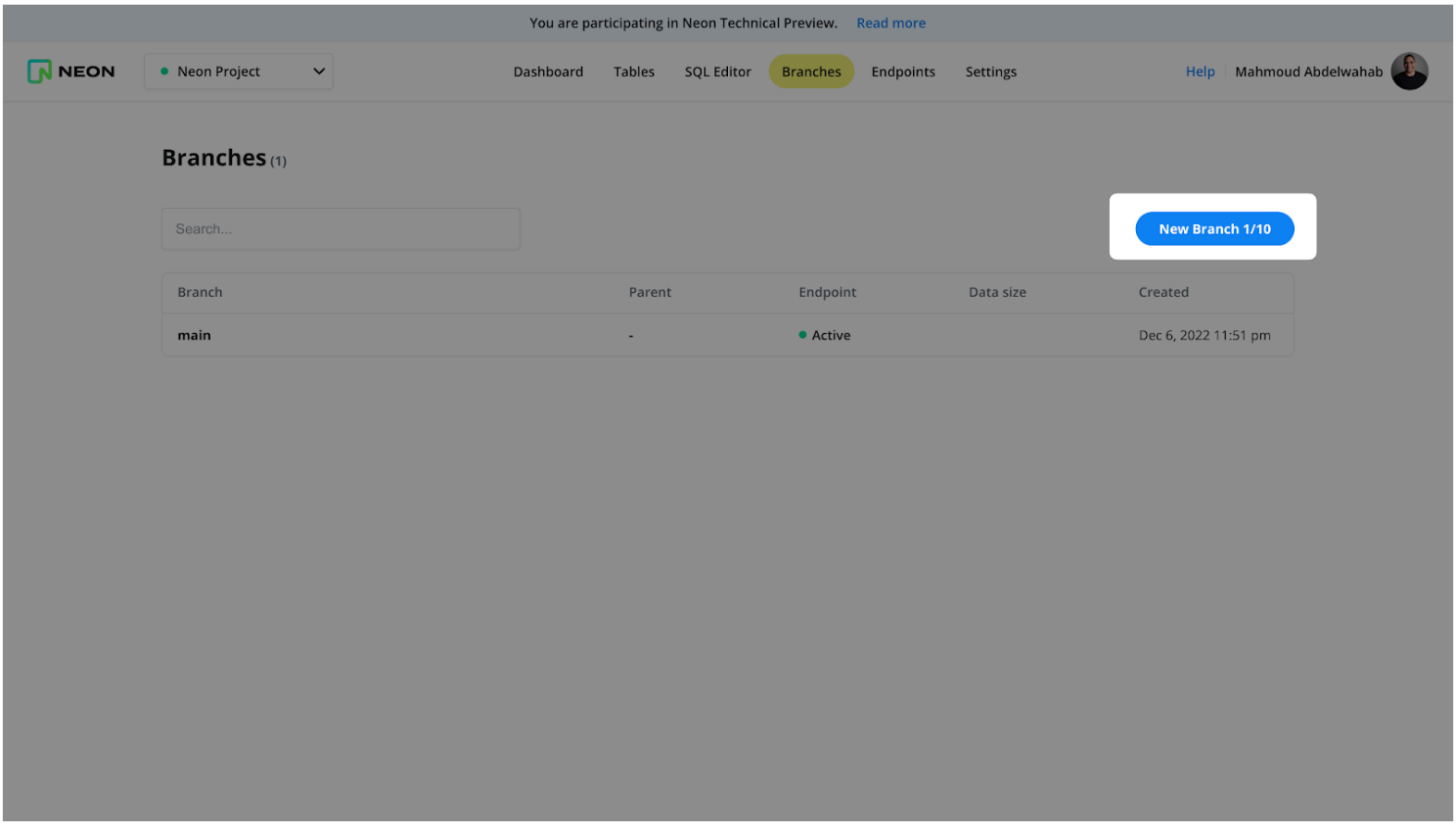
A branch acts as an isolated environment for working with your database. It is a **copy-on-write** clone of your

branch called the main, and you can create more branches depending on your needs.

Creating a branch

To get started, you first need to create a project in the **console**.

To create a branch via the console, navigate to the Branches tab and click on “New Branch”

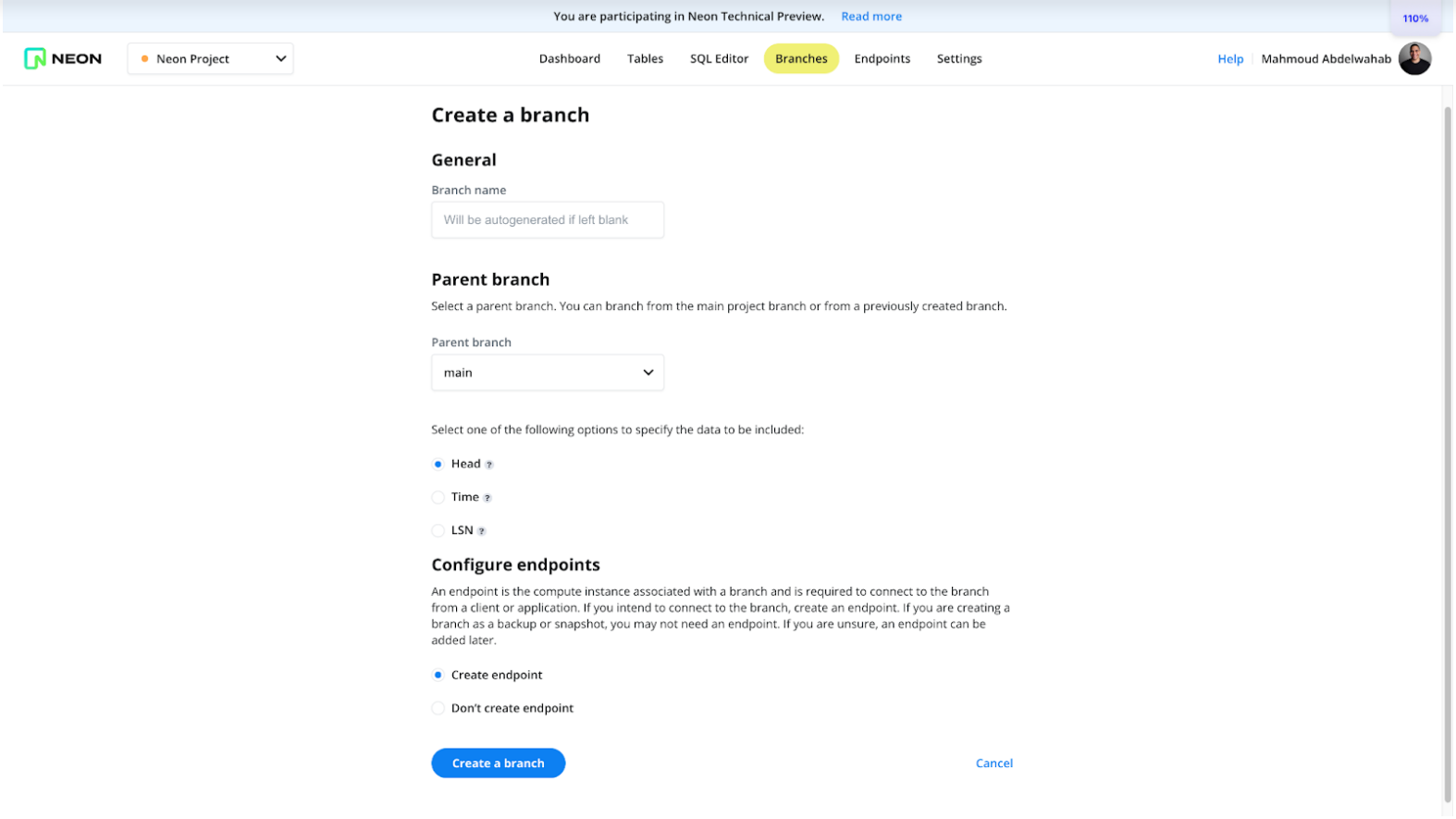


You will then need to select a parent branch. If this is your first branch, then the parent branch will be main. When it comes to which data you want this branch to copy, you have several options:

Head: creates a branch with all data from the parent branch up to the current point in time.

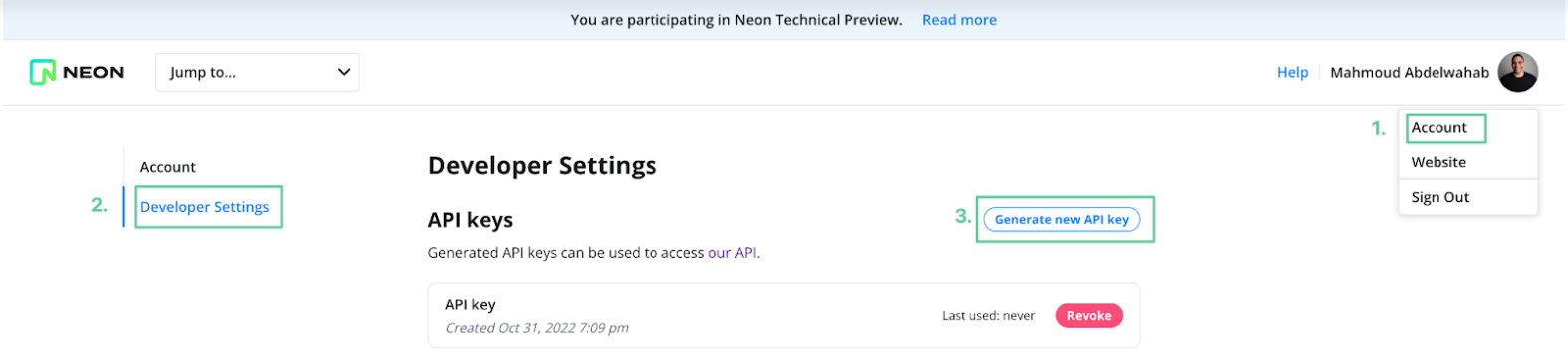
Time: creates a branch and pulls all data from the parent branch up to a certain point in time. This feature enables you to restore your database to a previous state within a specific time window. During the technical preview, we offer a window of 7 days.

LSN: creates a branch and pulls all data from the parent branch up to a certain LSN (**Log Sequence Number**). This is a pointer to a location in the WAL (**Write-Ahead Log**), which is the log of changes made to the database cluster.



If you have a rough idea of when the database was in the correct state, pick the "Time" option.

Alternatively, you can also create branches via the **Neon API**. To do that, you will first need to generate an API key. Go to the profile icon in the upper right corner and choose the "Account" option from the dropdown menu. Next, go to the Developer Settings tab and click on "Generate new API key"



Now that you have an API key, you can send a request to the API to create branches. Here's an example cURL command:

```
curl "https://console.neon.tech/api/v2/projects/${PROJECT_ID}/branches" \
  -H "Accept: application/json" \
  -H "Content-Type: application/json" \
  -H "Authorization: Bearer ${NEON_API_KEY}" \
  -d '{
    "branch": {
      "name": "My new branch!"
    }
  }' | jq
```

You are sending a `POST` request to the create branch endpoint. You are passing the API key in the request's header and you're passing your project ID as a variable. Finally, you're formatting the JSON response using `jq`, an optional third-party tool (see **jq** for more information)

How to get the most accurate rollback

Rolling back your database to a certain point in time can be good enough. However, this solution might not work for you if you are not 100% confident when the data loss occurred precisely.

Fortunately, you can leverage Neon's ability to create point-in-time branches with data up to a particular Log Sequence Number (LSN) to determine precisely when the data loss occurred. Let us take a look at an example:

Imagine you have the following users table:

```
| Name  |
| ----- |
| Bob   |
| Alice |
| Mike  |
| John  |
| Sarah |
| Lilly |
```

Then for some reason, you wanted to remove Alice from the list manually, so you ran the following query:

```
DELETE FROM users WHERE name != "Alice"
```

The next day, you noticed your mistake and discovered that you deleted all users from the table instead of Alice. Unfortunately, you are not sure when exactly the query finished executing and rolling back to an arbitrary date is not an option.

Since Neon offers the ability to create point-in-time branches, you can generate a sequence of branches that include past data in chronological order between two LSNs. You can then go through this list of branches and accurately determine when the disaster occurred.

To generate the list of branches, you will first need to define a start LSN and an end LSN.

To get the start LSN, you can create a branch where you are sure that that data was in a proper state (e.g., one day ago). You can then run the following query to return the LSN

```
SELECT pg_current_wal_flush_lsn()
-- returns a hexadecimal value (e.g. 1E9F3918) this will be the start //value
```

You then need to run the same query on the branch containing the incorrect data. The returned value will be the end LSN.

After generating the list of branches, you will notice that you have a search problem where the goal is to find the LSN that resulted in the incorrect data.

Since we have a sorted sequence of LSN values, we can leverage the "**binary search**" algorithm.

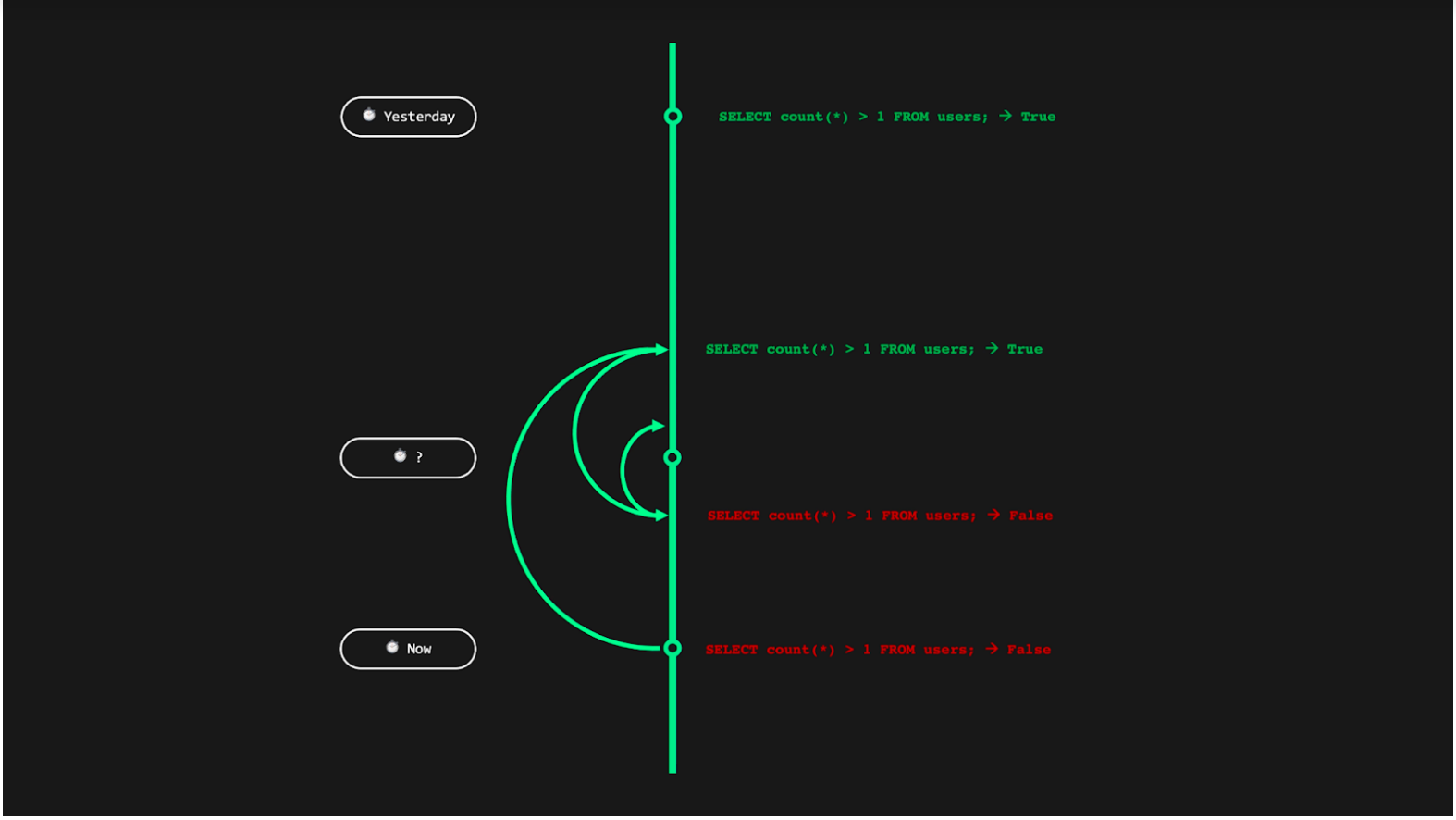
Here is a quick summary of how it works:

1. Divide the list into two halves, and compare the target item with the middle item in the list. If the target is equal to the middle item, you have found it, and the search is over.
2. If the target is less than the middle item, the target must be in the left half of the list. Repeat the same process on the left half of the list until you find the target or narrow it down to a single item.
3. If the target is greater than the middle item, the target must be in the right half of the list. Repeat the same process on the right half of the list until you find the target or narrow it down to a single item.
4. Continue dividing the list and comparing the target with the middle item until you find the target or narrow it down to a single item.

Instead of checking for equality, we will need to write a validation function that ensures the correctness of the database. In our example, we can use the following query:

```
SELECT count (*) FROM users > 1
-- If the row count of the `users` table is greater than 1, then our database
```

Depending on your database schema and the type of data loss that occurred, this validation function will be different.



Here is a complete example script using Python and Neon's API:

```
#!/usr/bin/env python3
import psycopg2
import requests
import os
import time

# We know that, e.g. three days ago everything was fine. It is possible to
# mannualy create a branch at specific time with UI and get it's LSN.
start = 0x2EB3898
# Last lsn, `select pg_current_wal_flush_lsn()`
end   = 0x3779AD8

# Database info
project = "silent-morning-200885"
db_creds = f"admin:{os.environ['PGPASSWORD']}"
headers = {
    "Authorization": f"Bearer {os.environ['NEON_API_KEY']}",
    "Content-Type": "application/json"
}
```

```
def query_branch(query, branch):
    endpoint_name = branch['endpoints'][0]['id']
    connstr = f"postgres://{db_creds}@{endpoint_name}.eu-central-1.aws.neon.t
    conn = psycopg2.connect(connstr)
    cursor = conn.cursor()
    cursor.execute(query)
    result = cursor.fetchall()[0][0]
    print(f"Checking \"{query}\" at lsn \"{branch['branch']['parent_lsn']}\":
    return result

def create_branch(parent_id, lsn):
    branch = requests.post(f'https://console.neon.tech/api/v2/projects/{proje
        headers=headers,
        data=f'{{"endpoints":[{{"type":"read_write"}}],"branch":{{"name":"bra
    ).json()
    print(f"Creating branch at lsn = {lsn}")
    return branch

def delete_branch(branch, lsn):
    branch_id = branch['branch']['id']
    branch = requests.delete(f'https://console.neon.tech/api/v2/projects/{pro
        headers=headers
    ).json()
    print(f"Deleted branch at lsn = {lsn}")
    time.sleep(2)
    return branch

def query_at_lsn(parent_id, query, lsni):
    lsn = f"0/{lsni:X}"
    branch = create_branch(parent_id, lsn)
    ret = query_branch(query, branch)
    delete_branch(branch, lsn)
    return ret

def bsearch_rightmost(parent_id, l, r, query):
    while l < r:
        m = (l + r)//2
        if query_at_lsn(parent_id, query, m):
            l = m + 1
        else:
            r = m
```

```
# Find out name of the main branch
resp = requests.get(f'https://console.neon.tech/api/v2/projects/{project}/bra
main_branch_id = next(b for b in (resp.json()['branches']) if b['name'] == "m
print(f"Main branch id is: \"{main_branch_id}\"")

# Do the bsearch
bsearch_rightmost(main_branch_id, start, end, "SELECT count(*) > 1 FROM users
print("Finishing")
```

After running the following script, you will get the following output which returns the LSN:

```
> python3 neon_bisect.py
Main branch id is: "br-rough-queen-879713
Creating branch at lsn = 0/1628D2B4
Checking "select exists(select name from users where name='neon')" at lsn "0/
Creating branch at lsn = 0/1A6405E6
Checking "select exists(select name from users where name='neon')" at lsn "0/
Creating branch at lsn = 0/18466C4D
Checking "select exists(select name from users where name='neon')" at lsn "0/
Creating branch at lsn = 0/17379F81
Checking "select exists(select name from users where name='neon')" at lsn "0/
Creating branch at lsn = 0/16B0391B
Checking "select exists(select name from users where name='neon')" at lsn "0/
Creating branch at lsn = 0/166C85E8
Checking "select exists(select name from users where name='neon')" at lsn "0/
Creating branch at lsn = 0/168E5F82
Checking "select exists(select name from users where name='neon')" at lsn "0/
Creating branch at lsn = 0/169F4C4F
Checking "select exists(select name from users where name='neon')" at lsn "0/
Creating branch at lsn = 0/1696D5E9
Checking "select exists(select name from users where name='neon')" at lsn "0/
Creating branch at lsn = 0/16929AB6
Checking "select exists(select name from users where name='neon')" at lsn "0/
Creating branch at lsn = 0/16907D1C
Checking "select exists(select name from users where name='neon')" at lsn "0/
Creating branch at lsn = 0/16918BE9
Checking "select exists(select name from users where name='neon')" at lsn "0/
Creating branch at lsn = 0/16921350
Checking "select exists(select name from users where name='neon')" at lsn "0/
```

```
Checking "select exists(select name from users where name='neon')" at lsn "0/
Creating branch at lsn = 0/1692352A
Checking "select exists(select name from users where name='neon')" at lsn "0/
Creating branch at lsn = 0/16924617
Checking "select exists(select name from users where name='neon')" at lsn "0/
Creating branch at lsn = 0/16924E8D
Checking "select exists(select name from users where name='neon')" at lsn "0/
Creating branch at lsn = 0/16924A52
Checking "select exists(select name from users where name='neon')" at lsn "0/
Creating branch at lsn = 0/16924C70
Checking "select exists(select name from users where name='neon')" at lsn "0/
Creating branch at lsn = 0/16924B61
Checking "select exists(select name from users where name='neon')" at lsn "0/
Creating branch at lsn = 0/16924ADA
Checking "select exists(select name from users where name='neon')" at lsn "0/
Creating branch at lsn = 0/16924A96
Checking "select exists(select name from users where name='neon')" at lsn "0/
Creating branch at lsn = 0/16924A74
Checking "select exists(select name from users where name='neon')" at lsn "0/
Creating branch at lsn = 0/16924A85
Checking "select exists(select name from users where name='neon')" at lsn "0/
Creating branch at lsn = 0/16924A8E
Checking "select exists(select name from users where name='neon')" at lsn "0/
Creating branch at lsn = 0/16924A8A
Checking "select exists(select name from users where name='neon')" at lsn "0/
Creating branch at lsn = 0/16924A88
Checking "select exists(select name from users where name='neon')" at lsn "0/
Creating branch at lsn = 0/16924A89
Checking "select exists(select name from users where name='neon')" at lsn "0/
Converged at 0/16924A89
Finishing
```

Now you can create a branch using this LSN value and restore your database to the correct state.

Final thoughts

In this article, you learned how to create branches for your Neon project and how you can leverage Neon's API along with point-in-time branches to restore your database to a correct state.

Neon is currently in **Technical Preview**, meaning you can **sign up** and try out the platform. If you have any feedback, feel free to email us at **feedback@neon.tech**, we would love to hear from you.

Finally, if you would like to keep up with our latest updates, make sure to subscribe to our newsletter down below.

Mahmoud Abdelwahab
Developer Advocate

Subscribe to Newsletter

Your email...

NEON

Made in SF and the World