Implementing Bi-Directional Replication in PostgreSQL

In today’s fast-paced digital world, ensuring that your data is always up-to-date and accessible is crucial. For businesses using PostgreSQL, replication is a key feature that helps achieve this. While many are familiar with streaming replication, bi-directional replication offers unique advantages that can enhance data availability and reliability.

In this blog post, we’ll explore what bi-directional replication is, how it differs from streaming replication, and provide a practical example to setup bi directional replication in PostgreSQL

What is Bi-Directional Replication?

Bi-directional replication, often referred to as multi-master replication, allows data to be written and read from multiple database nodes. This means that changes made on one node are automatically replicated to the other node, and vice versa. This setup ensures that all nodes have the same data, providing high availability and fault tolerance.

Bi-directional replication offers many advantages i.e.

- With bi-directional replication, if one node fails, the other can continue to handle read and write operations, minimizing downtime.
- It allows the distribution of the load across multiple nodes, improving performance and response times.
- Changes can be made on any node, making it easier to manage and update data without a single point of failure.
- Bi-directional replication can help reduce latency by allowing data to be read from the nearest node, especially in geographically distributed environments. This can improve the user experience and application performance.

Setting up Bi-Directional Replication in PostgreSQL

Let’s consider a practical example where we have set up PostgreSQL in Docker with two servers running PostgreSQL 17. One server runs on port 5432 and the other on port 5433. We have a table named employee with some sample data.

Create an employee table on both of the nodes

```
CREATE TABLE employee (employee_id SERIAL PRIMARY KEY,first_name VARCHAR(50) NOT
NULL,last_name VARCHAR(50) NOT NULL,email VARCHAR(100) UNIQUE NOT NULL,phone_number
VARCHAR(15),hire_date DATE NOT NULL,job_title VARCHAR(50),salary DECIMAL(10, 2),department_id
INT,created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,updated_at TIMESTAMP DEFAULT
CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP);Copy to Clipboard
```

Populate the table with dummy data

```
INSERT INTO employee (first_name, last_name, email, phone_number, hire_date, job_title, salary,
department_id)VALUES('John', 'Doe', 'john.doe@example.com', '555-1234', '2023-01-15',
'Software Engineer', 80000.00, 1),('Jane', 'Smith', 'jane.smith@example.com', '555-5678',
'2023-02-20', 'Project Manager', 95000.00, 2),('Emily', 'Johnson', 'emily.johnson@example.com',
'555-8765', '2023-03-10', 'Data Analyst', 70000.00, 1),('Michael', 'Brown',
'michael.brown@example.com', '555-4321', '2023-04-05', 'UX Designer', 75000.00, 3);Copy to
Clipboard
```

Before creating the **PUBLICATION** and **SUBSCRIPTION** on both nodes, I wanted to address a potential issue you may encounter when setting up bi-directional replication.

You might see warnings related to data origins. This can occur because the initial data copy might include data from other sources, which could lead to inconsistencies.

```
postgres=# create subscription sub2 CONNECTION 'host=localhost dbname=postgres port=5433'
publication pub2 WITH (origin = none);WARNING: subscription "sub2" requested copy_data with
origin = NONE but might copy data that had a different originDETAIL: The subscription being
created subscribes to a publication ("pub2") that contains tables that are written to by other
subscriptions.HINT: Verify that initial data copied from the publisher tables did not come
from other origins.NOTICE: created replication slot "sub2" on publisherCREATE SUBSCRIPTIONCopy
to Clipboard
```

What's Happening?
We are creating a subscription to replicate data from a publication. The warning is about potential data issues.

The Warning
It says that when you copy data initially, you might accidentally include data that came from other sources, not just the original database. In simple words data that is already present inside the table might get replicated again For example, if you have a table employee with existing records and you set up a new subscription, those records might be copied again to the subscribing node, even if they were already there

What to Do?
Use another option copy_data = false it can help avoid this issue by not copying existing data during the subscription setup, ensuring only new changes are replicated.

Create a PUBLICATION on Node 1

```
CREATE PUBLICATION pub1 FOR TABLE employee;Copy to Clipboard
```

Create a PUBLICATION on Node 2

```
CREATE PUBLICATION pub2 FOR TABLE employee;Copy to Clipboard
```

Create a SUBSCRIPTION on Node 1 and Node 2 respectively

```
CREATE SUBSCRIPTION sub2 CONNECTION 'host=localhost dbname=postgres port=5432' PUBLICATION
pub2 WITH (copy_data = false, origin = none);

CREATE SUBSCRIPTION sub1 CONNECTION 'host=localhost dbname=postgres port=5433' PUBLICATION
pub1 WITH (copy_data = false, origin = none);Copy to Clipboard
```

Both nodes are now ready. If we insert a record in Node 1, it will replicate to Node 2, and vice versa. Since we used the option copy_data = false when creating the SUBSCRIPTION, PostgreSQL will not replicate the existing data from Node 1 to Node 2.

By understanding its advantages and how to set it up correctly, you can ensure your data is always synchronized across multiple nodes. Whether you're dealing with high traffic or need a fail-safe system, bi-directional replication can be a valuable tool in your PostgreSQL toolkit. If you have any questions or need further assistance, feel free to reach out!