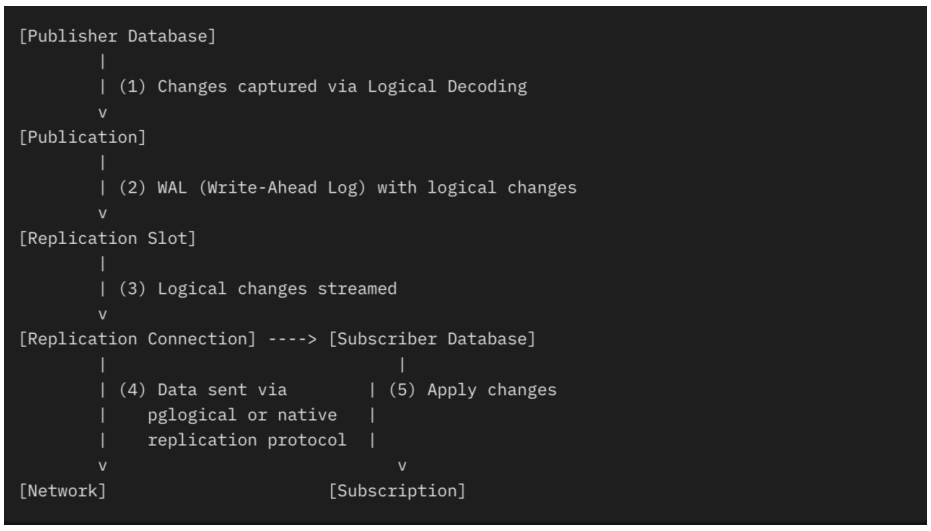

PostgreSQL Logical Replication

Constantin Alexander
Cloud Data Architect | DeltaLake + Apache Iceberg | Apache Flink + Apache Fluss | Data Governance | Ai Ops | ...

September 21, 2025

PostgreSQL Logical Replication: Complete Guide with Examples

Logical replication, is emitting SQL Statements that are in turn applied to the replica, in the same order as on the master database instance. Unlike physical replication, which replicates exact disk block changes, logical replication allows for more flexibility in what gets replicated and how.

Not only SQL Read Capability

Unlike physical replicas (streaming replication) which are read-only by default, logical replicas are fully functional PostgreSQL databases that can:

- Accept read queries
- Accept write queries (though this requires careful consideration)
- Have their own indexes, constraints, and optimizations
- Run different PostgreSQL versions than the publisher
- Geo replication
- Multi master replication

Key Advantages for Read Workloads

Logical replication uses a publish and subscribe model where one or more subscribers subscribe to one or more publications on a publisher node. The subscriber pulls data from the publications they subscribe to, and may subsequently re-publish data to allow cascading replication or more complex configurations.

Key Concepts

Publisher

The database instance that makes data available for replication. It defines what data should be replicated through publications.

Subscriber

The database instance that receives replicated data from publishers by subscribing to publications.

Publication

A set of changes generated from a table or a group of tables, optionally filtered by specific operations (INSERT, UPDATE, DELETE).

Subscription

The downstream side of logical replication. It defines the connection to another database and set of publications to subscribe to.

Prerequisites and Setup

System Requirements

- PostgreSQL 10 or later
- Sufficient WAL retention on the publisher
- Network connectivity between publisher and subscriber

Configuration Parameters

On the **publisher**, ensure these settings in postgresql.conf:

```
# Enable logical replication
wal_level = logical

# Set maximum replication slots (adjust based on number of
subscribers)
max_replication_slots = 10

# Set maximum WAL senders (should be >=
max_replication_slots)
max_wal_senders = 10

# Optional: Set WAL retention (prevents WAL files from
being removed too early)
wal_keep_size = 1GB
```

In pg_hba.conf, allow replication connections:

```
# Allow replication connections
host replication replicator 192.168.1.0/24 md5
host all replicator 192.168.1.0/24 md5
```

After making these changes, restart PostgreSQL.

Practical Example: Setting Up Logical Replication

Let's walk through a complete example of setting up logical replication between two PostgreSQL instances.

Step 1: Prepare the Publisher Database

Connect to the publisher database and create sample data:

```
-- Create a sample database and table
CREATE DATABASE ecommerce;
\c ecommerce

-- Create tables for replication
CREATE TABLE products (
    id SERIAL PRIMARY KEY,
    name VARCHAR(255) NOT NULL,
    price DECIMAL(10,2),
    category VARCHAR(100),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

CREATE TABLE orders (
    id SERIAL PRIMARY KEY,
    product_id INTEGER REFERENCES products(id),
    customer_name VARCHAR(255),
    quantity INTEGER,
    order_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

```
-- Insert sample data
INSERT INTO products (name, price, category) VALUES
    ('Laptop', 999.99, 'Electronics'),
    ('Desk Chair', 199.50, 'Furniture'),
    ('Coffee Mug', 12.99, 'Kitchenware');

INSERT INTO orders (product_id, customer_name, quantity)
VALUES
    (1, 'John Doe', 1),
    (2, 'Jane Smith', 2),
    (3, 'Bob Johnson', 3);
```

Step 2: Create a Replication User

```
-- Create a user for replication
CREATE USER replicator WITH REPLICATION LOGIN PASSWORD
'secure_password';

-- Grant necessary permissions
GRANT SELECT ON ALL TABLES IN SCHEMA public TO replicator;
GRANT USAGE ON SCHEMA public TO replicator;

-- Grant permissions on sequences (for SERIAL columns)
GRANT SELECT ON ALL SEQUENCES IN SCHEMA public TO
replicator;

-- Set default privileges for future tables
ALTER DEFAULT PRIVILEGES IN SCHEMA public GRANT SELECT ON
TABLES TO replicator;
ALTER DEFAULT PRIVILEGES IN SCHEMA public GRANT SELECT ON
SEQUENCES TO replicator;
```

Step 3: Create Publications

Publications define what data to replicate:

```
-- Create a publication for all tables
CREATE PUBLICATION all_tables FOR ALL TABLES;

-- Or create selective publications
CREATE PUBLICATION products_only FOR TABLE products;

CREATE PUBLICATION orders_only FOR TABLE orders;

-- Create a publication with specific operations
CREATE PUBLICATION products_inserts_only FOR TABLE
products WITH (publish = 'insert');

-- View existing publications
SELECT * FROM pg_publication;

-- View tables in publications
SELECT * FROM pg_publication_tables;
```

Step 4: Prepare the Subscriber Database

On the subscriber instance, create the same database structure:

```
-- Create the same database and schema
CREATE DATABASE ecommerce;
\c ecommerce

-- Create identical table structures (without data)
CREATE TABLE products (
    id SERIAL PRIMARY KEY,
    name VARCHAR(255) NOT NULL,
    price DECIMAL(10,2),
    category VARCHAR(100),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
```

```
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

CREATE TABLE orders (
    id SERIAL PRIMARY KEY,
    product_id INTEGER REFERENCES products(id),
    customer_name VARCHAR(255),
    quantity INTEGER,
    order_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

Step 5: Create Subscriptions

On the subscriber, create subscriptions to the publisher's publications:

```
-- Create a subscription to replicate all tables
CREATE SUBSCRIPTION all_tables_sub
    CONNECTION 'host=publisher_ip port=5432
user=replicator password=secure_password dbname=ecommerce'
    PUBLICATION all_tables;

-- View subscription status
SELECT * FROM pg_subscription;

-- Check subscription worker status
SELECT * FROM pg_stat_subscription;

-- View replication slots on the publisher
-- (Run this on the publisher)
SELECT * FROM pg_replication_slots;
```

Step 6: Test the Replication

On the publisher, insert some new data:

```
-- Insert new products
INSERT INTO products (name, price, category) VALUES
    ('Wireless Mouse', 29.99, 'Electronics'),
    ('Standing Desk', 299.00, 'Furniture');

-- Update existing product
UPDATE products SET price = 899.99 WHERE name = 'Laptop';

-- Insert new order
INSERT INTO orders (product_id, customer_name, quantity)
VALUES
    (4, 'Alice Brown', 1);
```

On the subscriber, verify the data:

```
-- Check if data was replicated
SELECT * FROM products ORDER BY id;
SELECT * FROM orders ORDER BY id;
```

Advanced Configuration Examples

Filtered Publications

Create publications that replicate only specific rows:

```
-- Replicate only electronics products (PostgreSQL 15+)
CREATE PUBLICATION electronics_only FOR TABLE products
WHERE (category = 'Electronics');

-- Replicate only recent orders
```

```
CREATE PUBLICATION recent_orders FOR TABLE orders WHERE
(order_date > '2024-01-01');
```

Column-Level Filtering

Replicate only specific columns (PostgreSQL 15+):

```
-- Replicate only specific columns
CREATE PUBLICATION products_basic FOR TABLE products (id,
name, category);
```

Multiple Publications in One Subscription

```
-- Subscribe to multiple publications
CREATE SUBSCRIPTION multi_pub_sub
    CONNECTION 'host=publisher_ip port=5432
user=replicator password=secure_password dbname=ecommerce'
    PUBLICATION products_only, orders_only;
```

Cascading Replication

Set up a subscriber that also acts as a publisher:

```
-- On the middle node, create a publication after
subscribing
CREATE PUBLICATION cascade_pub FOR ALL TABLES;

-- On the final subscriber, subscribe to the middle node
CREATE SUBSCRIPTION cascade_sub
    CONNECTION 'host=middle_node_ip port=5432
user=replicator password=secure_password dbname=ecommerce'
    PUBLICATION cascade_pub;
```

Monitoring and Maintenance

Monitoring Replication Lag

```
-- On the publisher: Check replication slot lag
SELECT
    slot_name,
    plugin,
    slot_type,
    database,
    active,
    restart_lsn,
    confirmed_flush_lsn,
    pg_size_pretty(pg_wal_lsn_diff(pg_current_wal_lsn(),
confirmed_flush_lsn)) AS lag_size
FROM pg_replication_slots;

-- On the subscriber: Check subscription statistics
SELECT
    subname,
    pid,
    received_lsn,
    last_msg_send_time,
    last_msg_receipt_time,
    latest_end_lsn,
    latest_end_time
FROM pg_stat_subscription;
```

Managing Publications and Subscriptions

```
-- Add table to existing publication
ALTER PUBLICATION all_tables ADD TABLE new_table;

-- Remove table from publication
ALTER PUBLICATION all_tables DROP TABLE old_table;

-- Refresh subscription after publication changes
ALTER SUBSCRIPTION all_tables_sub REFRESH PUBLICATION;

-- Disable subscription temporarily
ALTER SUBSCRIPTION all_tables_sub DISABLE;

-- Enable subscription
ALTER SUBSCRIPTION all_tables_sub ENABLE;

-- Drop subscription
DROP SUBSCRIPTION all_tables_sub;

-- Drop publication
DROP PUBLICATION all_tables;
```

Handling Initial Data Synchronization

```
-- Create subscription without initial data copy
CREATE SUBSCRIPTION no_copy_sub
    CONNECTION 'host=publisher_ip port=5432
user=replicator password=secure_password dbname=ecommerce'
    PUBLICATION all_tables
    WITH (copy_data = false);

-- Manually synchronize specific tables
-- First, ensure tables are empty on subscriber, then:
ALTER SUBSCRIPTION no_copy_sub REFRESH PUBLICATION WITH
(copy_data = true);
```

Conflict Resolution

Logical replication can encounter conflicts. Here are common scenarios and solutions:

Primary Key Conflicts

```
-- If you encounter primary key conflicts, you might need
to:
-- 1. Temporarily disable the subscription
ALTER SUBSCRIPTION all_tables_sub DISABLE;

-- 2. Resolve the conflict manually
-- 3. Re-enable the subscription
ALTER SUBSCRIPTION all_tables_sub ENABLE;
```

Handling Schema Changes

```
-- Schema changes must be applied manually to both
publisher and subscriber
-- On both publisher and subscriber:
ALTER TABLE products ADD COLUMN description TEXT;

-- Then refresh the publication on subscriber
ALTER SUBSCRIPTION all_tables_sub REFRESH PUBLICATION;
```

Best Practices

Security

1. Use strong passwords for replication users
2. Restrict network access using pg_hba.conf
3. Use SSL connections for replication
4. Grant minimal necessary permissions to replication users

Performance

1. Monitor replication lag regularly
2. Ensure adequate WAL retention on publisher
3. Consider partitioning large tables
4. Use appropriate work_mem settings for large initial syncs

Maintenance

1. Regularly monitor disk space usage
2. Clean up unused replication slots
3. Test failover procedures
4. Document your replication topology

Troubleshooting Common Issues

Subscription Not Starting

Copy | Share quote

```
-- Check subscription status
SELECT * FROM pg_stat_subscription;

-- Check PostgreSQL logs for errors
-- Look for permission issues or connection problems
```

High Replication Lag

```
-- Check for long-running transactions on publisher
SELECT pid, state, query_start, query
FROM pg_stat_activity
WHERE state = 'active' AND query_start < NOW() - INTERVAL
'1 hour';

-- Check for conflicts on subscriber
-- Look in PostgreSQL logs for conflict messages
```

WAL Files Accumulating

```
-- Check replication slot status
SELECT * FROM pg_replication_slots WHERE active = false;

-- Drop inactive replication slots
SELECT pg_drop_replication_slot('slot_name');
```

Choose Physical Replication When:

- You need simple read-write distribution
- Strong consistency is important
- You want proven, stable technology
- Schema changes need to be automatic
- You have a single geographic region
- Your team has limited database expertise

Choose Logical Replication When:

- You need multi-master capabilities
- You want selective data replication
- You need cross-version replication

- You have complex geographic requirements
- You need schema flexibility between nodes
- You have advanced database expertise

Comments

38 · 7 reposts

Like Comment Share

Add a comment...

No comments, yet.

Be the first to comment.

Start the conversation

Constantin Alexander

Cloud Data Architect | DeltaLake + Apache Iceberg | Apache Flink + Apache Fluss | Data Governance | Ai Ops | 🐘 PostgreSQL support 🐘 | AWS & GCP Specialist | Data Modeling | AI | DevOps

Azadkumar followed

✓ Following