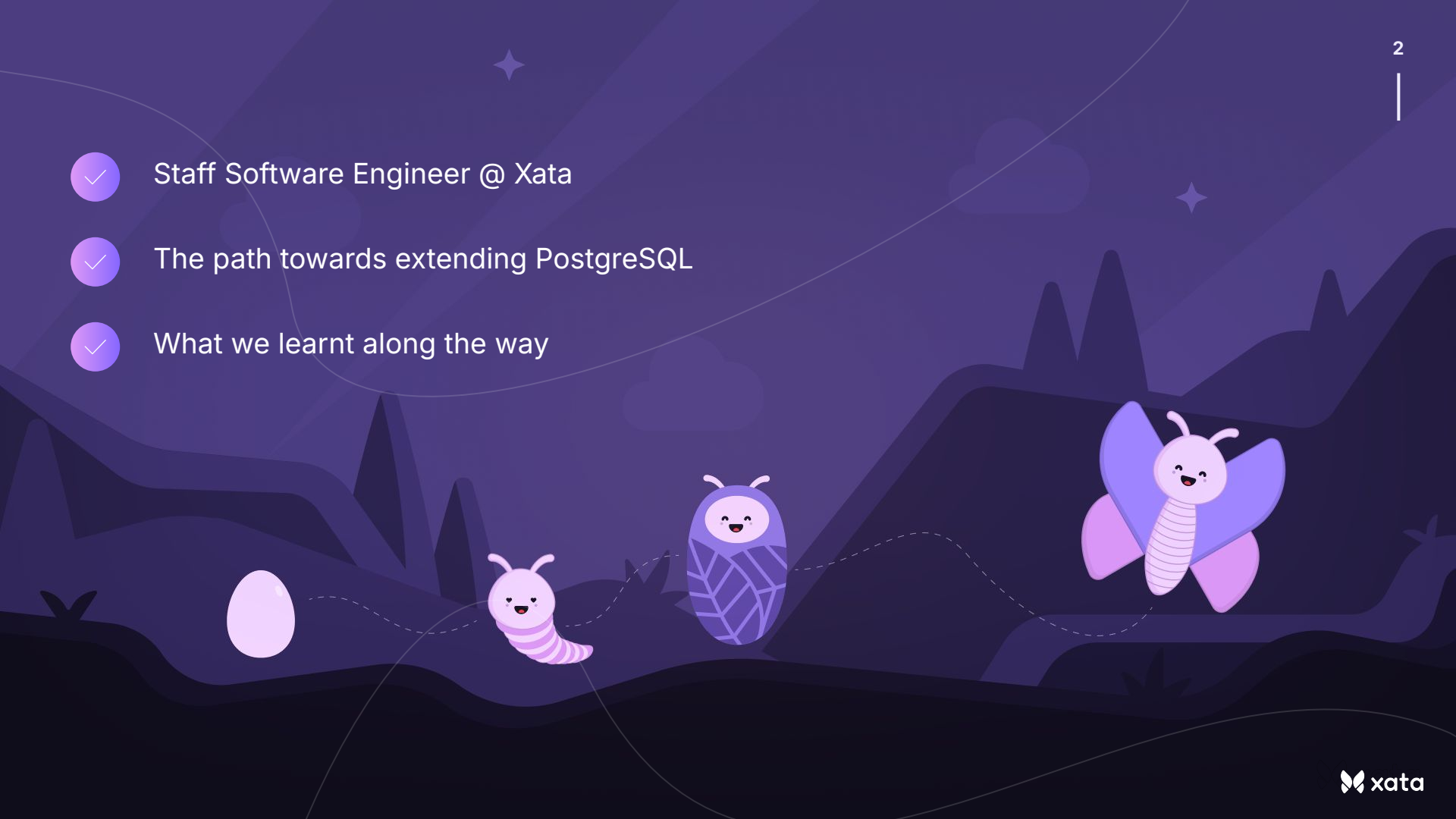
Replicating schema changes with PostgreSQL

Esther Minano Sanz

Jan 2025

- Staff Software Engineer @ Xata
- The path towards extending PostgreSQL
- What we learnt along the way



Content review

01 Replication in PostgreSQL

02 Logical replication of DDL

03 PostgreSQL → Elasticsearch

04 Webhooks using logical replication

05 pgstream

Replication in PostgreSQL

DDL

Data Definition Language

CREATE, ALTER, DROP...

DML

Data Manipulation Language

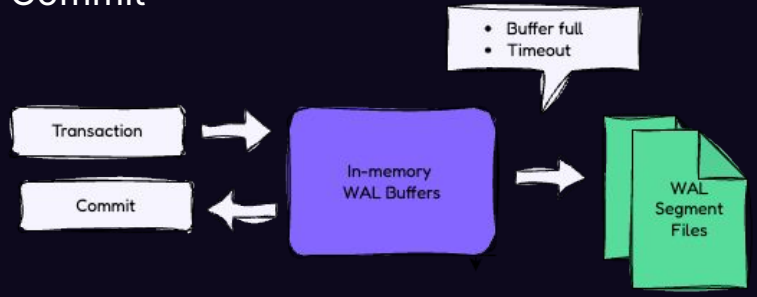
INSERT, UPDATE, DELETE,...

When to use replication?

- Sync data between PostgreSQL database servers
  - High availability
  - Load balancing
  - Backup and disaster recovery
- Sync data from PostgreSQL to any other data store
  - Analytics offloading
- React to PostgreSQL events in near real time

Write Ahead Log

- Sequential transaction log of database changes
- Append only file
- Transaction → WAL buffer → WAL segment file → Commit
- Critical for crash recovery
- Reduced disk operations
- Log Sequence Number - XLOG record unique ID



Types of replication

PHYSICAL REPLICATION

Continuous streaming of WAL records over the network. Best for replication in master-replica architectures.

FEATURES:

- 1:1 Data consistency
- File and record based log support
- Resilience to data loss
- No primary disruption

LIMITATIONS:

- Limited flexibility
- Version compatibility
- High network usage
- Target database must be read-only

LOGICAL REPLICATION

Continuous streaming of logically decoded WAL changesets over the network. Best for replication in publisher/subscriber architectures.

FEATURES:

- Selective replication
- Cross version compatibility
- Transaction level integrity
- Target database can be writable

LIMITATIONS:

- Increased server load
- Data consistency challenges
- No DDL/schema changes
- Replica identity required

Logical Replication of DDL

Logical replication of DDL

- Stateful replication
  - Table created in source database to keep state
  - Uses triggers and functions
- Stateless replication
  - No table or state required on source database

Stateful DDL Replication

Stateful DDL replication 1/9

- Create a dedicated table to track schema changes
- Capture schema on DDL events via triggers
- Insert captured schema details into dedicated schema log table
- Replicate events from schema log table

Stateful DDL replication 2/9

Extracting the schema

- Table information (schema, table name, oid)
  - pg_namespace, pg_class
- Column information (type, default, nullable, unique)
  - pg_attribute, pg_attrdef, pg_type, pg_catalog, pg_enum
- Primary key columns
  - pg_index, pg_attribute
- Constraints (check, unique, foreign keys)
  - pg_constraint
- Indices
  - pg_index

Stateful DDL replication 3/9

- Get all table names/ids for a given database schema

```
SELECT DISTINCT
    pg_namespace.nspname AS schema_name,
    pg_class.relname AS table_name,
    pg_class.oid AS table_oid
FROM pg_namespace
        RIGHT JOIN pg_class ON pg_namespace.oid = pg_class.relnamespace AND pg_class.relkind IN ('r', 'p')
WHERE pg_namespace.nspname = schema_name;
```

Stateful DDL replication 4/9

- Extract column information

```
SELECT
    pg_attribute.attname AS column_name,
    format_type(pg_attribute.atttypid, pg_attribute.atttypmod) AS column_type,
    pg_get_expr(pg_attrdef.adbin, pg_attrdef.adrelid) AS column_default,
    NOT ( pg_attribute.attnotnull OR pg_type.typtype = 'd' AND pg_type.typnotnull) AS column_nullable,
    (EXISTS
        (SELECT 1 FROM pg_constraint WHERE conrelid = pg_attribute.attrelid AND ARRAY[pg_attribute.attnum::int] @ conkey::int[] AND contype = 'u')
    OR EXISTS
        (SELECT 1 FROM pg_index JOIN pg_class ON pg_class.oid = pg_index.indexrelid WHERE indrelid = pg_attribute.attrelid AND indisunique
      AND ARRAY[pg_attribute.attnum::int] @ pg_index.indkey::int[])
    ) AS column_unique,
    pg_catalog.col_description(table_oids.table_oid,pg_attribute.attnum) AS column_description
FROM pg_attribute
        JOIN table_oids ON pg_attribute.attrelid = table_oids.table_oid
        JOIN pg_type ON pg_attribute.atttypid = pg_type.oid
        LEFT JOIN pg_attrdef ON pg_attribute.attrelid = pg_attrdef.adrelid AND pg_attribute.attnum = pg_attrdef.adnum
WHERE pg_attribute.attnum >= 1 -- less than 1 is reserved for system resources
  AND NOT pg_attribute.attisdropped; -- will be `true` if column is being dropped
```

Stateful DDL replication 5/9

- Aggregate per table full information (columns, primary keys)

```
SELECT
    columns.table_name AS table_name,
    columns.table_oid AS table_oid,
    jsonb_agg(jsonb_build_object(
            'name', columns.column_name,
            'type', columns.column_type,
            'default', columns.column_default,
            'nullable', columns.column_nullable,
            'unique', columns.column_unique,
            description, columns.column_description
        )) AS table_columns,
    (SELECT COALESCE(json_agg(pg_attribute.attname), '[]'::json)
       FROM pg_index, pg_attribute
       WHERE
           indrelid = columns.table_oid AND
           pg_attribute.attrelid = columns.table_oid AND
           pg_attribute.attnum = any(pg_index.indkey)
           AND indisprimary
     ) AS primary_key_columns
FROM columns
GROUP BY table_name, table_oid, table_pgs_id;
```

Stateful DDL replication 6/9

- Create a function to get the schema in JSON format

```
CREATE OR REPLACE  FUNCTION get_schema(schema_name  TEXT) RETURNS jsonb
   LANGUAGE  SQL
   SET search_path  = pg_catalog,pg_temp
   AS $$
       <...>
       SELECT
           jsonb_build_object(
                   'tables',
                   jsonb_agg(jsonb_build_object(
                           'oid', tables.table_oid,
                           'name', tables.table_name,
                           'columns', tables.table_columns,
                           'primary_key_columns' , tables.primary_key_columns
                       ))
               ) AS schema_view_json
       FROM tables;
$$;
```

Stateful DDL replication 7/9

- Create a schema log table to keep track of schema details

```
CREATE TABLE IF NOT EXISTS schema_log (
  id uuid PRIMARY KEY DEFAULT gen_random_uuid(),
  version BIGINT NOT NULL,
  schema_name TEXT NOT NULL,
  schema JSONB NOT NULL,
  created_at TIMESTAMP NOT NULL DEFAULT NOW(),
);

CREATE UNIQUE INDEX IF NOT EXISTS schema_log_version_uniq ON schema_log(schema_name, version);
```

Stateful DDL replication 8/9

- Create a function to populate schema_log table with schema view

```
CREATE OR REPLACE FUNCTION log_schema() RETURNS event_trigger
   LANGUAGE plpgsql
   SECURITY DEFINER
   SET search_path = pg_catalog,pg_temp
   AS $$
DECLARE
   rec_objid oid; -- used for deletes
   rec_schema_name text;
   schema_version bigint;
   is_system_schema boolean;
BEGIN
   IF tg_tag = 'DROP SCHEMA' AND tg_event = 'sql_drop' THEN
       SELECT object_name INTO rec_schema_name FROM pg_event_trigger_dropped_objects() WHERE object_type = 'schema' LIMIT 1;
       IF rec_schema_name IS NOT NULL THEN
           SELECT COALESCE((SELECT version+1 FROM "schema_log" WHERE schema_name = rec_schema_name ORDER BY version DESC LIMIT 1), 1) INTO schema_version;
           -- a dropped schema log entry is constant, has no tables AND has the dropped flag set to true.
           INSERT INTO "schema_log" (version, schema_name, schema) VALUES (schema_version, rec_schema_name, '{"tables": null, "dropped": true}'::jsonb);
           -- remove all log entries of current schema, with the exception of 'dropped' entry
           DELETE FROM "schema_log" WHERE schema_name = rec_schema_name AND ((schema->'dropped')::bool IS NULL OR NOT (schema->'dropped')::bool);
       END IF;
   elsif tg_tag = 'DROP TABLE' AND tg_event = 'sql_drop' THEN
       SELECT objid, schema_name INTO rec_objid, rec_schema_name FROM pg_event_trigger_dropped_objects() WHERE object_type = 'table' LIMIT 1;
       IF rec_schema_name IS NOT NULL THEN
           SELECT COALESCE((SELECT version+1 FROM "schema_log" WHERE schema_name = rec_schema_name ORDER BY version DESC LIMIT 1), 1) INTO schema_version;
           INSERT INTO "schema_log" (version, schema_name, schema) VALUES (schema_version, rec_schema_name, get_schema(rec_schema_name));
       END IF;
   elsif tg_event = 'ddl_command_end' THEN
       IF tg_tag = 'CREATE SCHEMA' THEN
           SELECT object_identity INTO rec_schema_name FROM pg_event_trigger_ddl_commands() WHERE object_type = 'schema' AND command_tag = 'CREATE SCHEMA' LIMIT 1;
       elsif tg_tag = 'CREATE TABLE' THEN
           SELECT schema_name INTO rec_schema_name FROM pg_event_trigger_ddl_commands() WHERE object_type = 'table' AND command_tag = 'CREATE TABLE' LIMIT 1;
       elsif tg_tag = 'ALTER TABLE' THEN
           SELECT schema_name INTO rec_schema_name FROM pg_event_trigger_ddl_commands() WHERE object_type IN ('table', 'table column') AND command_tag = 'ALTER TABLE' LIMIT 1;
       END IF;

       IF rec_schema_name IS NOT NULL THEN
           SELECT COALESCE((SELECT version+1 FROM "schema_log" WHERE schema_name = rec_schema_name ORDER BY version DESC LIMIT 1), 1) INTO schema_version;
           INSERT INTO "schema_log" (version, schema_name, schema) VALUES (schema_version, rec_schema_name, get_schema(rec_schema_name));
       END IF;
   END IF;
END;
$$;
```

Stateful DDL replication 9/9

- Create triggers to log schema changes

```
CREATE EVENT TRIGGER log_schema_create_alter_table ON ddl_command_end EXECUTE FUNCTION log_schema();
CREATE EVENT TRIGGER log_schema_drop_schema_table ON sql_drop WHEN tag IN ('DROP TABLE', 'DROP SCHEMA') EXECUTE FUNCTION log_schema();
```

Stateless DDL Replication

Stateless DDL replication

- Rely on logical replication messages
  - Column added/dropped
    - ‘R’ relation message sent + new schema
    - New row
  - Column type change
    - ‘T’ type relation message sent + new schema
    - New row
- Perform a diff of new and previous schema
  - `SELECT * FROM <table> LIMIT 0;`
  - Inspect row field descriptors

Stateless vs Stateful DDL replication

	STATEFUL	STATELESS
Flexible schema representation	✓	✗
Requires dedicated table	✓	✗
Requires triggers and functions	✓	✗
No additional server impact	✗	✓
Low maintenance cost	✗	✓
Schema diff required	✓	✓

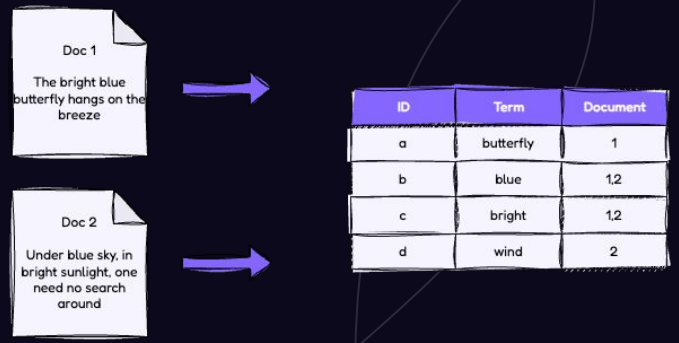
PostgreSQL → Elasticsearch

Why?

- PostgreSQL Full Text Search
  - Requires reindexing on every write
  - Increased load on search read queries
- Offload operational and performance complexity

ElasticSearch

- Search and analytics engine
- NoSQL document store
  - Structured data encoded in JSON
  - Unique identifier
- Index
  - Collection of documents
  - Inverted indices (content → location)
- Mapping
  - How documents and fields are stored and indexed
  - Append supported
  - Most mapping changes require reindexing



Lessons learnt 1/2

- Column type/value mapping
  - No dedicated array types in ElasticSearch
  - Explicit mapping of vectors
  - Timestamp regex match
- Avoid reindexing when possible
  - Renames
    - Use immutable unique identifiers for column mappings
    - Use aliases
  - Column data type changes
    - Create a new field in the mapping for the new type
    - Old data would not be available until it’s reindexed

Lessons learnt 2/2

TOAST columns

The Oversized Attribute Storage Technique

- Values that don't fit in a page are stored separately by PostgreSQL
- If they are not updated, they are not included in the replication event
- Enable `REPLICA IDENTITY FULL`

Out of order events

- Use primary key(s) as event ID
- Use LSN (64 bit integer) as event version

Denormalization

- Relational relationships are lost
- Use Elasticsearch ingest pipelines

Webhooks using logical replication

Limitations of PostgreSQL Triggers

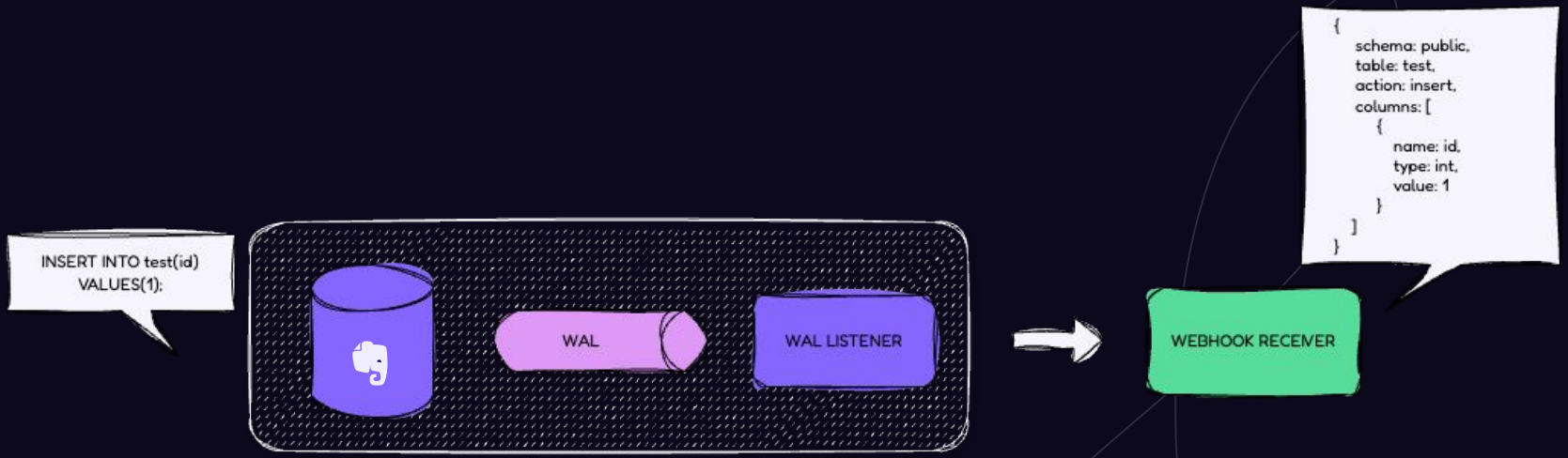
- Increased server load
  - Infinite loops, increased CPU usage, database locking
  - Potential performance impact, or database crashing
- Data inconsistency
  - Conflicting triggers applied to the same tables
- Execution errors
- Maintenance overhead

Should be used sparingly, and be executed quickly and efficiently

Webhooks using PostgreSQL logical replication

- Get notifications for relevant change events on near real time
- No performance impact on workload
  - Decouple slow event processors from database
- Integrate with any service
- Low maintenance overhead
  - Manage notification subscriptions without impacting the database

PostgreSQL replication using webhooks



https://github.com/xataio/pgstream

OSS

Open source CLI tool and library written in Go and made for PostgreSQL.

Modular

Designed to be easy to expand and configure, to multiply the use case coverage.

DDL tracking

Continuous replication of schema changes along with data.

Search outputs

Out of the box support to replicate to Elasticsearch and Opensearch.

Webhook notifications

Lightweight webhook integration supported with subscription server included.

Fast initial snapshots

Supports snapshots of selected database tables on startup.

Coming up soon...

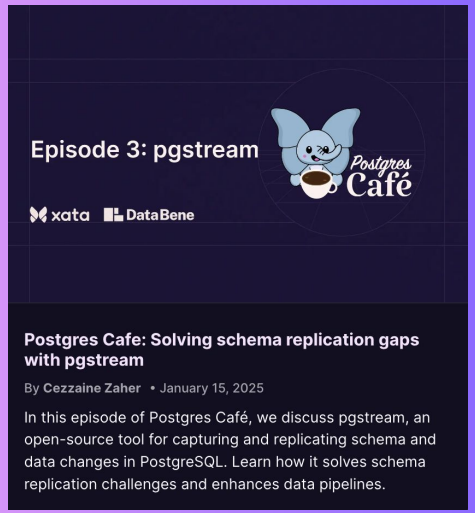
Postgres to Postgres replication

Improved filtering

Anonymisation and subsetting support

For more

https://github.com/xataio/pgstream



https://xata.io/blog

Thank you!

@xata