Internals of logical replication

Vigneshwaran C

FUJITSU

Agenda

- Introduction
- Use cases
- Architecture
- Publication
- Subscription
- Logical replication processes
- Replication of different transaction types
- Replication slot
- PG15 new features

Introduction

FUJITSU

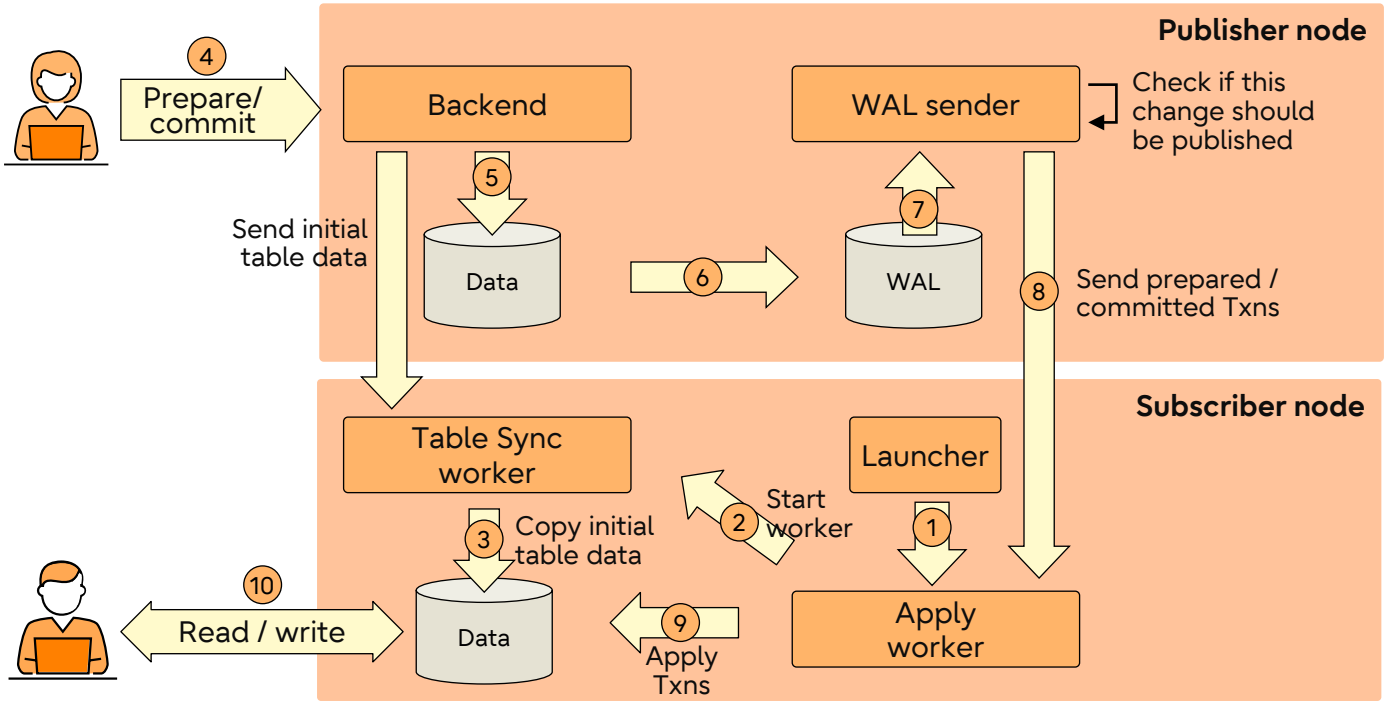
- **Logical replication** is a method of replicating data changes from publisher to subscriber.
- The node where a publication is defined is referred to as *publisher*.
- The node where a subscription is defined is referred to as the *subscriber*.
- Logical replication allows fine-grained control over both data replication and security.
- Logical replication uses a publish and subscribe model with one or more subscribers subscribing to one or more publications on a publisher node.
- Subscribers pull data from the publications they subscribe to and may subsequently re-publish data to allow cascading replication or more complex configurations.

Use cases

FUJITSU

- Sending incremental changes in a single database or a subset of a database to subscribers as they occur.
- Firing triggers for individual changes as they arrive on the subscriber.
- Consolidating multiple databases into a single one (e.g., for analytical purposes).
- Replicating between different major versions of PostgreSQL.
- Replicating between PostgreSQL instances on different platforms (e.g., Linux to Windows).
- Giving access to replicated data to different groups of users.
- Sharing a subset of the database between multiple databases.

Architecture



Publication

FUJITSU

- A publication can be defined on the primary node whose changes should be replicated.
  - A publication is a set of changes generated from a table or a group of tables and might also be described as a change set or replication set.
  - Each publication exists in only one database.
- Publications are different from schemas and do not affect how the table is accessed.
  - Each table can be added to multiple publications if needed.
  - Publications may currently only contain tables and all tables in schema.
- Publications can choose to limit the changes they produce to any combination of INSERT, UPDATE, DELETE, and TRUNCATE, similar to how triggers are fired by particular event types.
  - By default, all operation types are replicated.

Publication

FUJITSU

- When a publication is created the publication information will be added to pg_publication catalog table:

```
postgres=# CREATE PUBLICATION pub_all FOR ALL TABLES;
CREATE PUBLICATION
postgres=# SELECT * FROM pg_publication;
  oid  |    pubname    | pubowner | puballtables | pubinsert | pubupdate | pubdelete | pubtruncate | pubviaroot
-------+---------------+----------+--------------+-----------+-----------+-----------+-------------+------------
 16392 | pub_alltables |       10 | t            | t         | t         | t         | t           | f
(1 row)
```

- Information about table publication is added to pg_publication_rel catalog table:

```
postgres=# CREATE PUBLICATION pub_employee FOR TABLE employee;
CREATE PUBLICATION
postgres=# SELECT oid, prpubid, prrelid::regclass FROM
pg_publication_rel;
  oid  | prpubid | prrelid
-------+---------+----------
 16407 |   16406 | employee
(1 row)
```

Publication

FUJITSU

- Information about schema publications is added to pg_publication_namespace catalog table:

```
postgres=# CREATE PUBLICATION pub_sales_info FOR TABLES IN SCHEMA marketing, sales;
CREATE PUBLICATION
postgres=# SELECT oid, pnpubid, pnnspid::regnamespace FROM pg_publication_namespace;
  oid  | pnpubid |  pnnspid
-------+---------+-----------
 16410 |   16408 | marketing
 16411 |   16408 | sales
(2 rows)
```

- The view pg_publication_tables provides information about the mapping between publications and information of tables they contain.

```
postgres=# CREATE PUBLICATION pub_data_all FOR TABLE data;
CREATE PUBLICATION
postgres=# CREATE PUBLICATION pub_data_blue FOR TABLE data WHERE (rgb = 'B');
CREATE PUBLICATION
postgres=# SELECT * FROM pg_publication_tables;
    pubname    | schemaname | tablename | attnames |     rowfilter
---------------+------------+-----------+----------+-------------------
 pub_data_all  | public     | data      | {id,rgb} |
 pub_data_blue | public     | data      | {id,rgb} | (rgb = 'B'::text)

(2 rows)
```

Subscription

FUJITSU

- A subscription is the downstream side of logical replication.
  - A subscription defines the connection to another database and set of publications (one or more) to which it wants to subscribe.
- The subscriber database behaves in the same way as any other PostgreSQL instance and can be used as a publisher for other databases by defining its own publications.
- A subscriber node may have multiple subscriptions if desired.
  - It is possible to define multiple subscriptions between a single publisher-subscriber pair, in which case care must be taken to ensure that the subscribed publication objects don't overlap.
- Each subscription will receive changes via one replication slot.
  - Additional replication slots may be required for the initial data synchronization of pre-existing table data and those will be dropped at the end of data synchronization.

Subscription

FUJITSU

- When a subscription is created, the subscription information will be added to pg_subscription catalog table:

```
postgres=# CREATE SUBSCRIPTION sub_alltables
CONNECTION 'dbname=postgres host=localhost port=5432'
PUBLICATION pub_alltables;
NOTICE:  created replication slot "sub_alltables" on publisher
CREATE SUBSCRIPTION
postgres=# SELECT oid, subdbid, subname, subconninfo, subpublications FROM pg_subscription;
  oid  | subdbid |    subname       |              subconninfo                | subpublications
-------+---------+------------------+-----------------------------------------+-----------------
 16393 |       5 | sub_alltables    | dbname=postgres host=localhost port=5432 | {pub_alltables}
(1 row)
```

- The subscriber will connect to the publisher and get the list of tables that the publisher is publishing.

Subscription

FUJITSU

- In our earlier example, we created pub_alltables to publish data of all tables, the publication relations will be added to pg_subscription_rel catalog tables:

```
postgres=# SELECT srsubid, srerelid::regclass FROM pg_subscription_rel;
 srsubid | srrelid
---------+---------
   16399 |   accounts
   16399 |   account_roles
   16399 |   roles
   16399 |   department
   16399 |   employee
(5 rows)
```

- Subscriber connects to the publisher and creates a replication slot, whose information is available in pg_replication_slots:

```
postgres=# SELECT slot_name, plugin, type, datoid, database, temporary, active,
active_pid, restart_lsn, confrm_flush_lsn FROM pg_replication_slots;
 slot_name     | plugin   | slot_type | datoid | database | temporary | active | active_pid | restart_lsn | confirmed_flush_lsn
---------------+----------+-----------+--------+----------+-----------+--------+------------+-------------+---------------------
 sub_alltables | pgoutput | logical   |      5 | postgres | f         | t      |      24473 | 0/1550900   | 0/1550938
(1 row)
```

Subscription

FUJITSU

- Subscribers add the subscription stats information to pg_stat_subscription:

```
postgres=# SELECT subid, subname, received_lsn FROM pg_stat_subscription;
subid  | subname         |   received_lsn
-------+-----------------+---------------
 16399 | sub_alltables    | 0/1550938
(1 row)
```

- The initial part of the CREATE SUBSCRIPTION command will be completed and returned to the user.
- The remaining work will be done in the background by replication launcher, walsender, apply worker and table sync worker after the CREATE SUBSCRIPTION command is completed.

Processes - Replication launcher

FUJITSU

- This process is started by the postmaster during the start of the instance.
- The logical replication worker launcher uses the background worker infrastructure to start the logical replication workers for every enabled subscription.

```
vignesh 24438 /home/vignesh/postgres/inst/bin/postgres -D subscriber
vignesh 24439 postgres: checkpointer
vignesh 24440 postgres: background writer
vignesh 24442 postgres: walwriter
vignesh 24443 postgres: autovacuum launcher
vignesh 24444 postgres: logical replication launcher
```

- The launcher process will periodically check the pg_subscription catalog table to see if any subscription has been added or enabled.

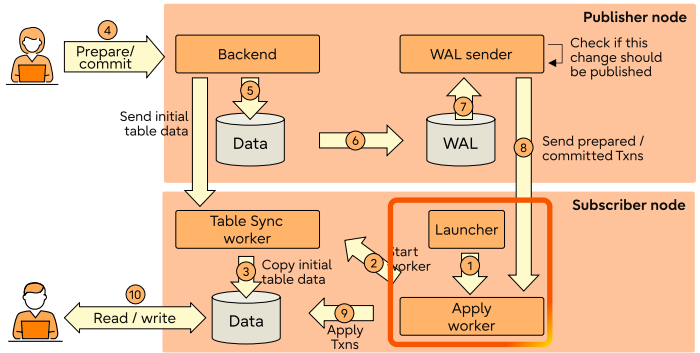
Processes - Replication launcher

FUJITSU

- Once the launcher process identifies that a new subscription has been created or enabled, it will start an apply worker process.
- The apply worker running can be seen from the process list:

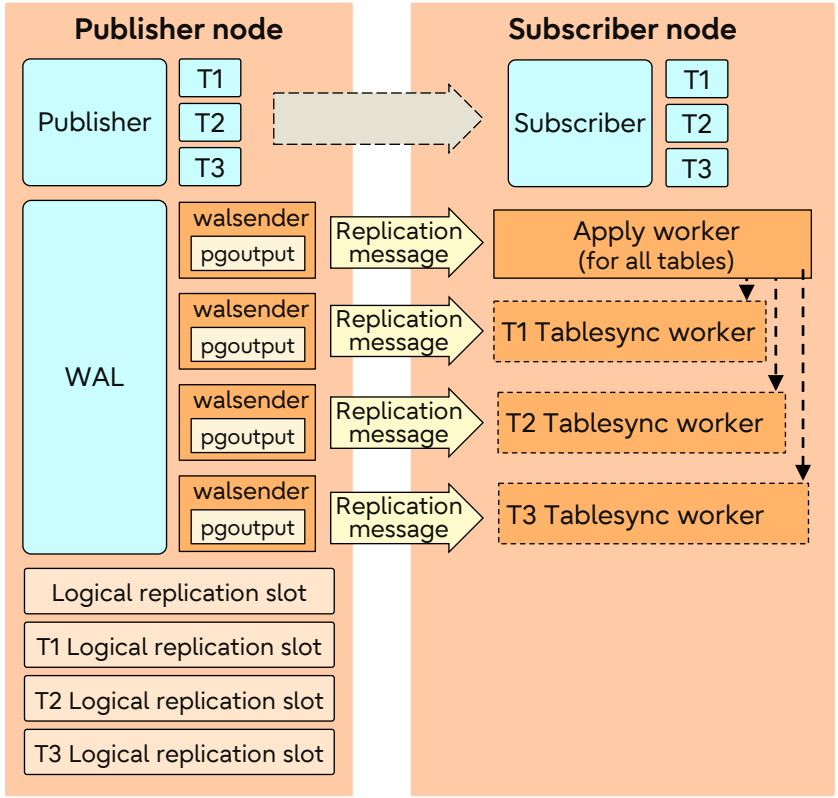
```
vignesh 24438 /home/vignesh/postgres/inst/bin/postgres -D subscriber
vignesh 24439 postgres: checkpointer
vignesh 24440 postgres: background writer
vignesh 24442 postgres: walwriter
vignesh 24443 postgres: autovacuum launcher
vignesh 24444 postgres: logical replication launcher
vignesh 24472 postgres: logical replication apply worker for subscription 16399
vignesh 24473 postgres: walsender vignesh postgres 127.0.0.1(55020) START_REPLICATION
```

The above information illustrates step 1 mentioned in the architecture



Processes - Apply worker

FUJITSU



- The Apply worker will iterate through the table list and launch table sync workers to synchronize the tables.
- Each table will be synchronized by one table sync worker.
- Multiple table sync workers (one for each table) will run in parallel based on **max_sync_workers_per_subscription** configuration.
- Table synchronization workers are taken from the pool defined by **max_logical_replication_workers** configuration.

Processes - Apply worker

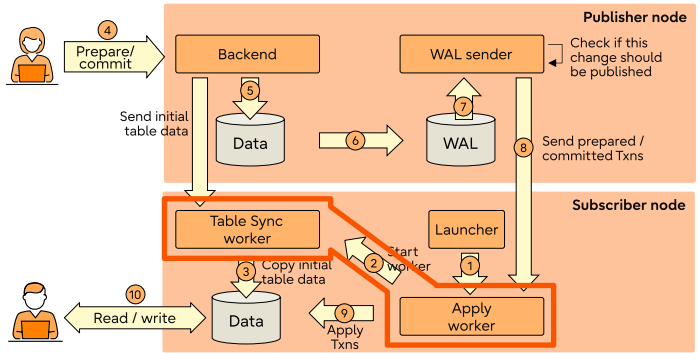
FUJITSU

- The apply worker will wait until the table sync worker copies the initial table data and sets the table state to *ready* state in pg_subscription_rel.

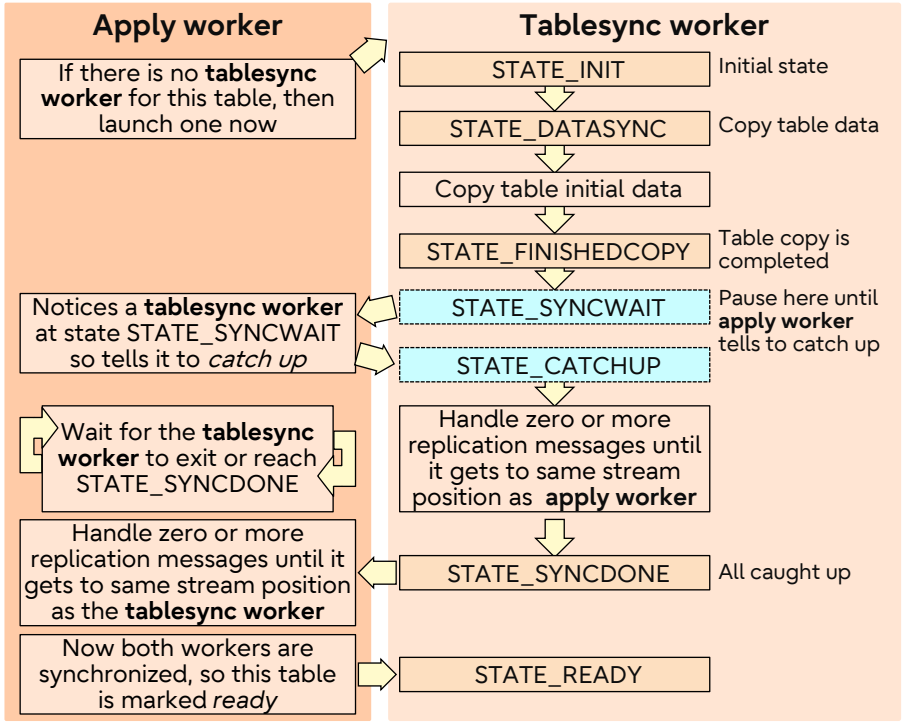
```
postgres=# SELECT srsubid, srrelid::regclass, srsubstate, srsublsn FROM pg_subscription_rel;
 srsubid |    srrelid     | srsubstate | srsublsn
---------+----------------+------------+-----------
   16399 | accounts       | r          | 0/156B8D0
   16399 | account_roles  | r          | 0/156B8D0
   16399 | department     | r          | 0/156B940
   16399 | employee       | r          | 0/156B940
   16399 | roles          | r          | 0/156B978
(5 rows)
```

- Note: Currently, DDL operations are not supported by logical replication. Only DML changes will be replicated.

The above information illustrates step 2 mentioned in the architecture

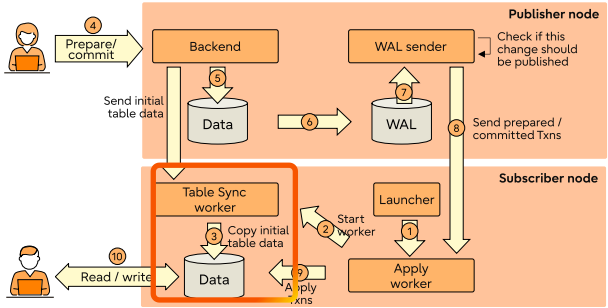


Processes – Tablesync worker



- The initial data synchronization is done separately for each table, by a separate table sync worker.
- Create a replication slot with **USE_SNAPSHOT** option and copy table data with COPY command.
- Table sync worker will request the publisher to start replicating data from the publisher.
- Table sync worker will synchronize data from walsender until it reaches the syncworker's LSN set by the apply worker.

The information to the left illustrates step 3 mentioned in the architecture



Processes - Walsender

FUJITSU

- Takes care of sending WAL from the primary server to a single recipient.
- Started by the postmaster when the subscriber connects to the publisher and requests WAL.
- Reads the WAL record by record and decodes the WAL to get the tuple data and size.
- Queues this change into the **reorderbufferqueue**.
  - The **reorderbufferqueue** collects individual pieces of transactions in the order they are written to the WAL. When a transaction is completed, it will reassemble the transaction and call the output plugin with the changes.
  - If the **reorderbufferqueue** exceeds *logical_decoding_work_mem*, then find the largest transaction and evict it to disk.

Processes - Walsender

FUJITSU

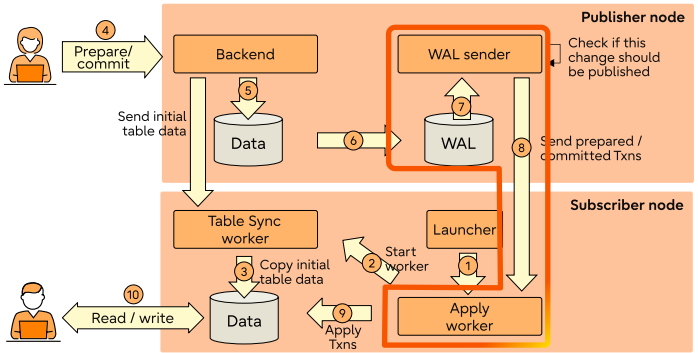
- If streaming is enabled, then this transaction data will be sent to subscriber, but will be applied in the subscriber only after the transaction is committed in the publisher.
- Once the transaction is committed:
  - Check if this relation should be published (based on ***ALL TABLES*** or ***TABLE* list** or ***TABLES IN SCHEMA* list** specified in the publication).
  - Check if this operation should be published (based on what user has specified for ***publish*** option – insert/update/delete/truncate).
  - Change the publish relation ID if ***publish_via_partition_root*** *is set*. In this case the relation ID of the ancestor will be sent.
  - Check and filter this data if it satisfies the condition specified by **row filter/column list**.

Processes - Walsender

FUJITSU

- This transaction data will be sent to the subscriber.
- Update the statistics like txn count, txn bytes, spill count, spill bytes, spill txns, stream count, stream bytes, stream txns.

The above information illustrates steps 7 and 8 mentioned in the architecture



Replicating incremental changes

FUJITSU

	Walsender	Apply worker
Txn start	• Begin message includes final_lsn of the DML txn, commit time, and txn ID.	• Receives begin message which will include the final_lsn of the DML, txn, commit time, and txn ID.
	• Sends relation info: relation ID, schema name, relation name, attribute info, and relation kind.	• Gets relation information which includes relation ID, schema name, relation name, attribute information, and relation kind. Stores this info in a hash map.
Insert Update Delete	• Sends relation ID and tuple info (for updates, send old and new tuple info) Tuple info includes number of columns. Each column will have the column type, length and column data.	• Creates a tuple. — Ins Upd Del • Fills the tuple with the received values. — Ins Upd Del • Fills the required default columns. — Ins Upd Del • Finds indices associated with result relation and populates values. — Ins Upd Del • Finds the tuple in the table. — Upd Del • Inserts / updates / deletes tuple — Ins Upd Del
Truncate	• Sends truncate flags, number of relations, and the relation ID.	• Gets truncate flags, number of relations, and relation ID information. • For each table specified: truncates the relation and all associated objects such as indices and toast table.
Commit	• Sends commit information, which includes LSN details and commit time.	• Gets commit information, which includes LSN details and commit time. • Commits transaction in subscriber. • Updates origin LSN. • Stores the flush position of local LSN and remote LSN in a map. • Periodically sends recv, write, and flush positions to walsender.

The above information illustrates step 9 mentioned in the architecture

Apply Worker Failure handling

FUJITSU

- **If the apply worker fails due to an error, the apply worker process will exit.**
- The apply worker will have maintained the origin LSN during the last transaction commit.
- The replication launcher will periodically check if the subscription worker is running. If the launcher identifies that it is not, then it will restart the worker for the subscription.
- The apply worker will request **start_replication** streaming from the last origin LSN that was committed.
- Walsender will start streaming transactions from the origin LSN (last committed transaction) requested by the apply worker.

Apply Worker Failure handling

FUJITSU

- Whenever the apply worker encounters a constraint error such as duplicate constraint error, check constraint error, etc, it will exit and repeat the steps mentioned in the previous slide.
- There is an option to skip the LSN in case of errors - user can **set skip lsn** of the failing transaction in this case.
  - If user sets to skip LSN, the apply worker will check if the transaction matches the LSN specified, skip this transaction, and proceed to the next one.

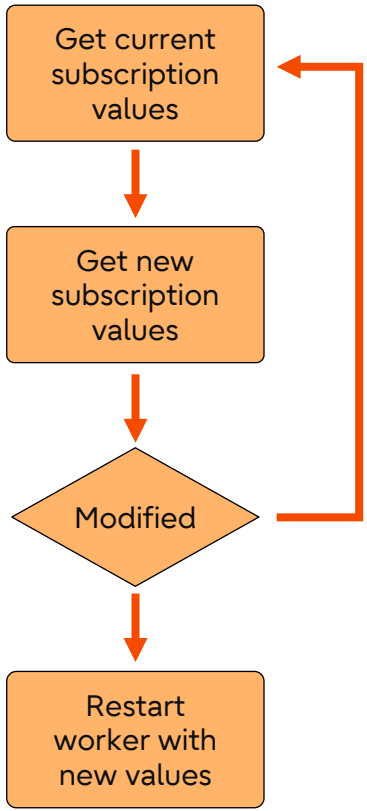
Apply Worker Failure handling

FUJITSU

- Whenever the apply worker encounters a constraint error such as duplicate constraint error, check constraint error, etc, it will exit and repeat the steps mentioned in the previous slide.
- The user can **disable_on_error** instead of repeatedly trying the steps.
  - In this case, any error in the apply worker will be caught using try() /catch(), and the subscription will be disabled before the apply worker exists.
  - As the subscription is disabled, the launcher will not restart the apply worker for the subscription.

Altering a subscription

- The apply worker will periodically check the current subscription values against the new ones - if they have been changed, the apply worker will exit.
- The launcher will restart the apply worker after the latter exits.
- The apply worker will load the new subscription values from pg_subscription system table.
- The apply worker will apply the changes using the newly modified values.



How synchronous_commit is achieved

FUJITSU

- Create subscription with synchronous_commit option as ‘on’ in the subscriber.
- In the publisher:
  - Set synschronous_standby_names to the subscription name using "ALTER SYSTEM SET synchronous_standby_names..." command in the publisher
  - Reload the configuration using pg_reload_conf
  - Verify that is_sync option is enabled in pg_stat_replication.

Subscriber

```
postgres=# CREATE SUBSCRIPTION sync
CONNECTION 'dbname=postgres host=localhost port=5432'
PUBLICATION sync
WITH (synchronous_commit = 'on');
NOTICE:  created replication slot "sync" on publisher
CREATE SUBSCRIPTION
```

Publisher

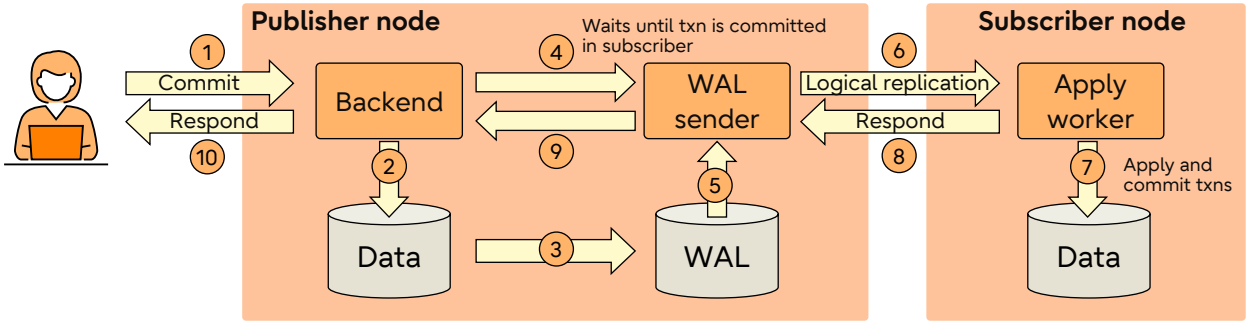
```
postgres=# ALTER SYSTEM SET synchronous_standby_names TO 'sync';
ALTER SYSTEM

postgres=# SELECT pg_reload_conf();
 pg_reload_conf
----------------
 t
(1 row)

postgres=# SELECT application_name, sync_state = 'sync' AS is_sync
FROM pg_stat_replication
WHERE application_name = 'sync';
 application_name | is_sync
------------------+---------
 sync             | t
(1 row)
```

How synchronous_commit is achieved

FUJITSU



1 2 Backend in the publisher performs transactions

3 Backend generates the WAL records for transactions

4 Backend adds the LSN and backend information to the syncrep queue to be awakened by the walsender

5 6 Walsender decodes WAL records of the corresponding txns, sends txns to the subscriber, and waits until txn is complete

7 Apply worker applies the transactions

8 Subscriber confirms the transaction is committed

9 Walsender awakens the backend

10 Backend commits the transaction successfully

Replication slot

FUJITSU

- As mentioned earlier, each (active) subscription receives changes from a replication slot on the remote (publishing) side.
- Additional table synchronization slots are normally transient, created internally to perform initial table synchronization and dropped automatically when they are no longer needed.
  - These table synchronization slots have generated names:
    - pg_%u_sync_%u_%llu

subscription oid table relid system identifier sysid

Replication slot

FUJITSU

- Normally, the remote replication slot is created automatically when the subscription is created during *CREATE SUBSCRIPTION* and it is dropped automatically when the subscription is dropped during *DROP SUBSCRIPTION*.
- In some situations, however, it can be useful or necessary to manipulate the subscription and the underlying replication slot separately.
- Replication slots provide an automated way to ensure that the primary does not remove WAL segments until they have been received by all standbys.

Row filters

FUJITSU

- An optional WHERE clause can be specified.
- This information is stored in pg_publication_rel catalog table:

```
postgres=# CREATE PUBLICATION active_departments FOR TABLE departments WHERE (active IS TRUE);
CREATE PUBLICATION
postgres=# SELECT oid, prpubid, prrelid, pg_get_expr (prqual, prrelid) FROM pg_publication_rel;
  oid  | prpubid | prrelid |   pg_get_expr
-------+---------+---------+-----------------
 16457 |   16456 |   16426 | (active IS TRUE)
(1 row)
```

- Rows that don't satisfy this WHERE clause will be filtered by the publisher.
- This allows a set of tables to be partially replicated.
- During table sync only the table data that satisfies the row filter will be copied to the subscriber.

Row filters

FUJITSU

```
CREATE PUBLICATION pub_data_red
FOR TABLE data
WHERE (rgb = 'R');
```

```
CREATE PUBLICATION pub_data_blue
FOR TABLE data
WHERE (rgb = 'B');
```

Data

id	rgb
1	R
2	R
3	G
4	B
5	G
6	R
7	B
8	B
9	R
10	G

Subscriber 1

```
CREATE SUBSCRIPTION sub_r
CONNECTION …
PUBLICATION pub_data_red;
```

id	rgb
1	R
2	R
6	R
9	R

Subscriber 2

```
CREATE SUBSCRIPTION sub_r_b
CONNECTION …
PUBLICATION pub_data_red,
            pub_data_blue;
```

Multiple row filter expressions for the same table will be **OR-ed** together

id	rgb
1	R
2	R
4	B
6	R
7	B
8	B
9	R

Row filters

FUJITSU

- If the subscription has several publications in which the same table has been published with different filters (for the same publish operation):
  - The expressions get OR 'ed, and rows satisfying any of the expressions are replicated.
- If the subscription has several publications in which some publication is defined for ALL TABLES or TABLES IN SCHEMA publication where the table belongs to the referred schema:
  - ALL TABLES publication and TABLES IN SCHEMA publication take precedence and the publish treat as if there are no row filters.

Row filters transformation

FUJITSU

- For insert operations, the publisher checks if the new row satisfies the row filter condition to determine if the new record should be sent to the subscriber or skipped.
- For delete operations, the publisher checks if the row satisfies the row filter condition to determine if the operation should be sent to the subscriber or skipped.
- The update operation is handled in a slightly different manner:
  - If neither the old row nor the new one match the row filter condition:
    - Update is skipped.
  - If the old row does not satisfy the row filter condition, but the new one does:
    - Transform the update to insertion of new row on the subscriber.
  - If the old row satisfies the row filter condition but the new one does not:
    - Transform the update into deletion of old row from the subscriber.
  - If both the old row and the new one satisfy the row filter condition:
    - Send the data as an update to the subscriber, without any transformation.

Column List

FUJITSU

- An optional column list clause can be specified.
- This information is stored in pg_publication_rel catalog table:

```
postgres=# CREATE PUBLICATION users_filtered FOR TABLE users (user_id, firstname);
CREATE PUBLICATION
postgres=# SELECT * FROM pg_publication_rel;
  oid  | prpubid | prrelid | prqual | prattrs
-------+---------+---------+--------+---------
 16453 |   16452 |   16436 |        | 1 2
(1 row)

postgres=# SELECT * FROM pg_publication_tables;
   pubname     | schemaname | tablename |       attnames       | rowfilter
---------------+------------+-----------+----------------------+-----------
users_filtered | public     | users     | {user_id, firstname} |
(1 row)
```

- Columns not included in this list are not sent to the subscriber.
- This allows the schema on the subscriber to be a subset of the publisher schema.

Column List

FUJITSU

student Table

stud_id	name	dob	phone	course_id	email	photo
1001	steve	01-01-2004	999999999	251	steve@test.com	steve.jpeg
1002	leo	02-02-2004	888888888	252	leo@test.com	leo.jpeg
1003	thom	03-03-2004	777777777	253	thom@test.com	thom.jpeg
1004	jobs	04-04-2004	666666666	254	jobs@test.com	jobs.jpeg
1005	gates	05-05-2004	555555555	254	gates@test.com	gates.jpeg

```
CREATE PUBLICATION pub_student
FOR TABLE student
(stud_id,name,phone,email);
```

Subscriber

```
CREATE SUBSCRIPTION sub_student
CONNECTION ...
PUBLICATION pub_student;
```

student Table

stud_id	name	phone	email
1001	steve	999999999	steve@test.com
1002	leo	888888888	leo@test.com
1003	thom	777777777	thom@test.com
1004	jobs	666666666	jobs@test.com
1005	gates	555555555	gates@test.com

Column List

FUJITSU

- During the initial table synchronization, only columns included in the column list are copied to the subscriber.
- When sending incremental transaction changes, publisher will check for the relation information and send to the subscriber the values for the columns that match the specified column list. The other columns are skipped.
- For partitioned tables, publish_via_partition_root determines whether the column list for the root or the leaf relation will be used.
  - If the parameter is 'false' (the default), the list defined for the leaf relation is used.
  - Otherwise, the column list for the root partition will be used.
- Specifying a column list when the publication also publishes FOR TABLES IN SCHEMA is not supported.
- There's currently no support for subscriptions comprising several publications where the same table has been published with different column lists.

Advantages of row filters and column list

FUJITSU

- The row filter and column list features provide the following advantages:
  - Reduces network traffic (increase performance) by replicating only a small subset of a large data table.
  - Provides only the data that is relevant to a subscriber node.
  - Acts as a form of security by hiding sensitive information (not replicating credit card number).

Replicating TABLES IN SCHEMA

FUJITSU

- One or more schemas can be specified in FOR TABLES IN SCHEMA.
- This information is maintained in the pg_publication_namespace catalog table:

```
postgres=# CREATE PUBLICATION sales_publication FOR TABLES IN SCHEMA marketing, sales;
CREATE PUBLICATION
postgres=# SELECT oid, pnpubid, pnnspid::regnamespace FROM pg_publication_namespace;
  oid  | pnpubid | pnnspid
-------+---------+------------
 16450 |   16449 | marketing
 16451 |   16449 | sales
(2 rows)
```

- During the initial table synchronization, only tables that belong to the specified schema are copied to the subscriber.
- When sending the incremental transaction changes, publisher will check if this transaction's relation belongs to one of the schemas and publish only those changes.

Replicating TABLES IN SCHEMA

FUJITSU

- If the subscription has several publications in which some publication is defined for all table, then all tables publication will be given higher precedence and all the table data will be sent to subscription.
- Any new table created in the schema after creation of publication will be automatically added to the publication.
  - Similarly, tables removed from the schema will be automatically removed from the publication.
- But data of newly created tables (after creation of subscription) will not be replicated automatically - the user will have to run ALTER SUBSCRIPTION ... REFRESH PUBLICATION, which will fetch the missing tables and take care of synchronizing the data from the publisher.
- ALL TABLES replication is similar to TABLES IN SCHEMA publication, except that it will replicate all tables data instead of replicating only the tables present in the schema.

Appendix

FUJITSU

Recommended reading:

- Logical Replication Tablesync Workers
- Logical replication of tables in schema in PostgreSQL 15
- How to gain insight into the pg_stat_replication_slots view by examining logical replication
- Column lists in logical replication publications
- Introducing publication row filters
- Addressing logical replication conflicts using ALTER SUBSCRIPTION SKIP

Thank you

Internals of logical replication

Vigneshwaran C