New Publication Syntax and Sequence Synchronisation in PostgreSQL 19 and Beyond

Ajin Cherian & Peter Smith



FUJITSU

FUJITSU-PUBLIC

© Fujitsu 2026

FUJITSU

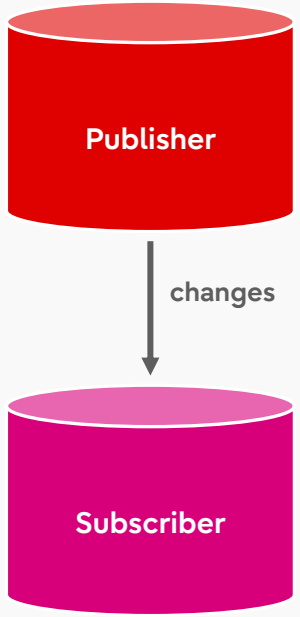
Sequence Synchronisation: Closing gaps in logical replication

Ajin Cherian
- PostgreSQL Significant Contributor

© Fujitsu 2026

Agenda

1. **Logical replication 101**
   What it is, publishers & subscribers
2. **A short history**
   From logical decoding in 9.4 to PostgreSQL 19
3. **The real killer use case**
   Near-zero-downtime major version upgrades
4. **pg_createsubscriber**
   Automating the upgrade setup (PG17+)
5. **The missing piece: sequences**
   The problem, and REFRESH SEQUENCES in PG19



© Fujitsu 2026

Two ways to replicate PostgreSQL

FUJITSU

Physical (streaming) replication

Ships the WAL stream itself

- Exact binary copy of the whole cluster
- Same major version & platform only
- Standby is read-only (hot standby)
- Great for HA and failover



WAL records, streamed as a continuous byte stream

Logical replication

Ships decoded row changes

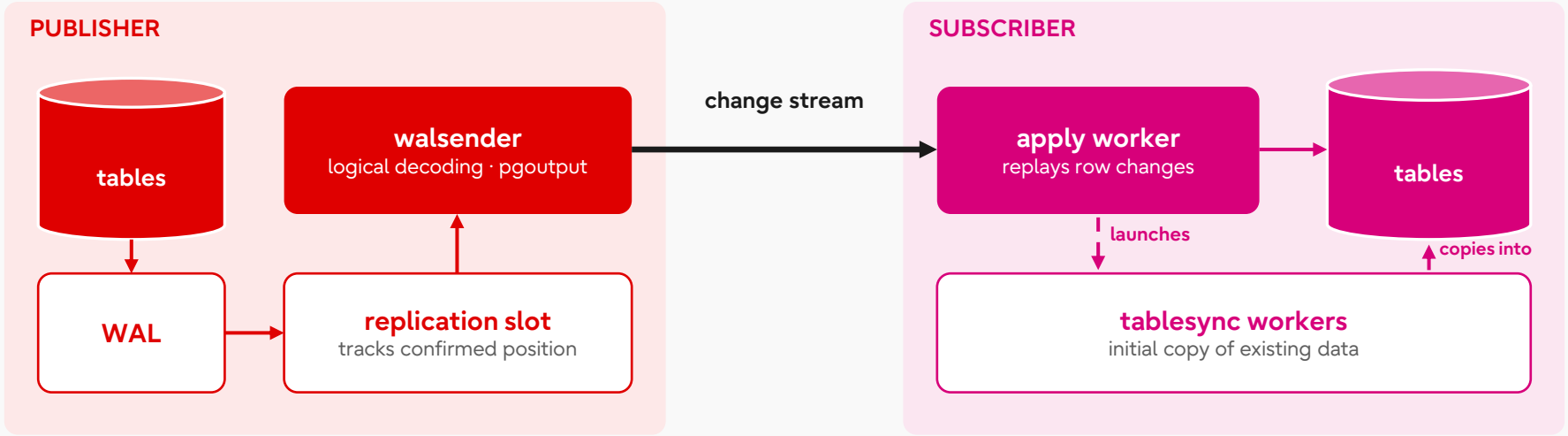
- Choose what to replicate: tables, rows, columns
- Works across major versions
- Subscriber stays writable
- Great for upgrades, migrations, integration

```
INSERT  orders   (42, 'widget')
UPDATE  users    id=3: plan='pro'
DELETE  cart     id=7
```

decoded change messages (pgoutput), not SQL

© Fujitsu 2026

Publishers and subscribers



PUBLISHER

```
CREATE PUBLICATION app_pub
    FOR TABLE orders, customers;
```

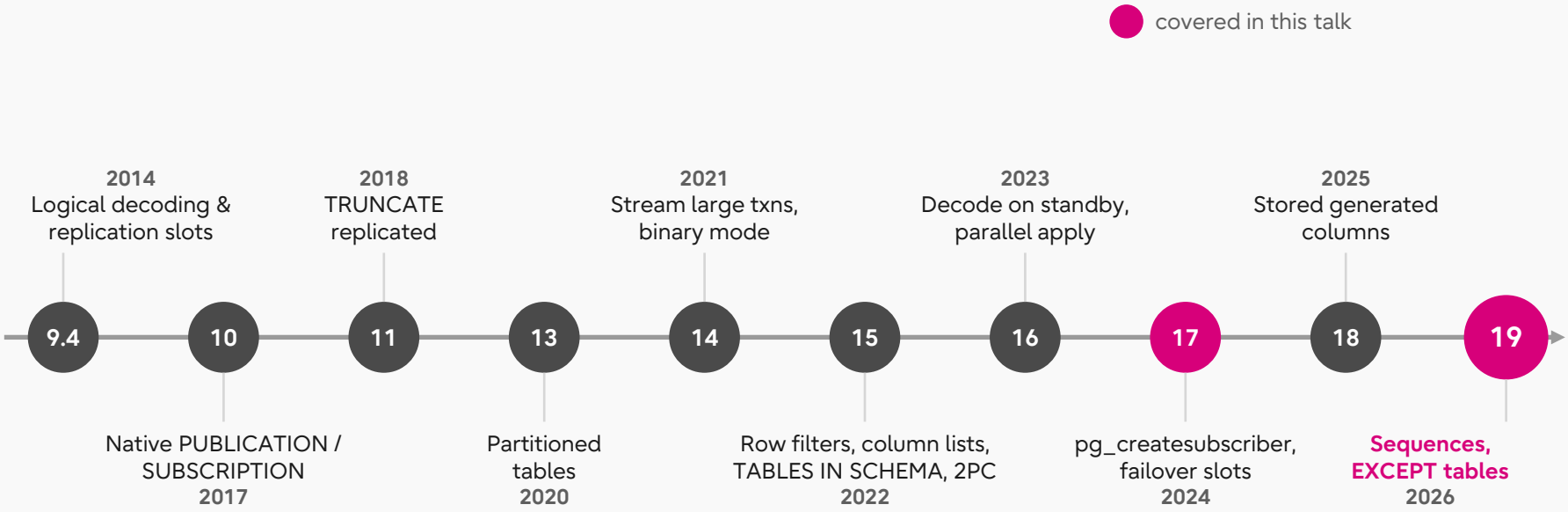
SUBSCRIBER

```
CREATE SUBSCRIPTION app_sub
  CONNECTION 'host=pub dbname=app'
  PUBLICATION app_pub;
```


© Fujitsu 2026

A short history of logical replication

FUJITSU



A decade of steady gap-filling - **PG19 closes one of the last big ones for upgrades.**

© Fujitsu 2026

Intended use vs. actual use

FUJITSU

THE VISION

Logical standbys

- An alternative to physical standbys
- Writable replicas, extra indexes and tables
- Replicate just a subset of data
- Data distribution & consolidation

IN PRACTICE

A key use: major upgrades

- Physical replication cannot cross major versions
- Logical replication can
- So the new version can be kept in sync while the old one keeps serving traffic

Downtime shrinks to a **switchover** instead of the whole upgrade.

© Fujitsu 2026

Why major version upgrades hurt

FUJITSU

Illustrative, not benchmarks



pg_dump / pg_restore

Everything must happen offline; grows with data size



pg_upgrade (--link / --clone)

Fast, but the cluster is offline while it runs



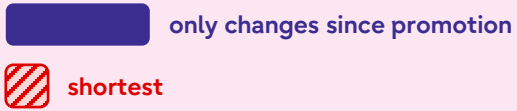
Logical replication alone

Fresh subscriber must copy every table first



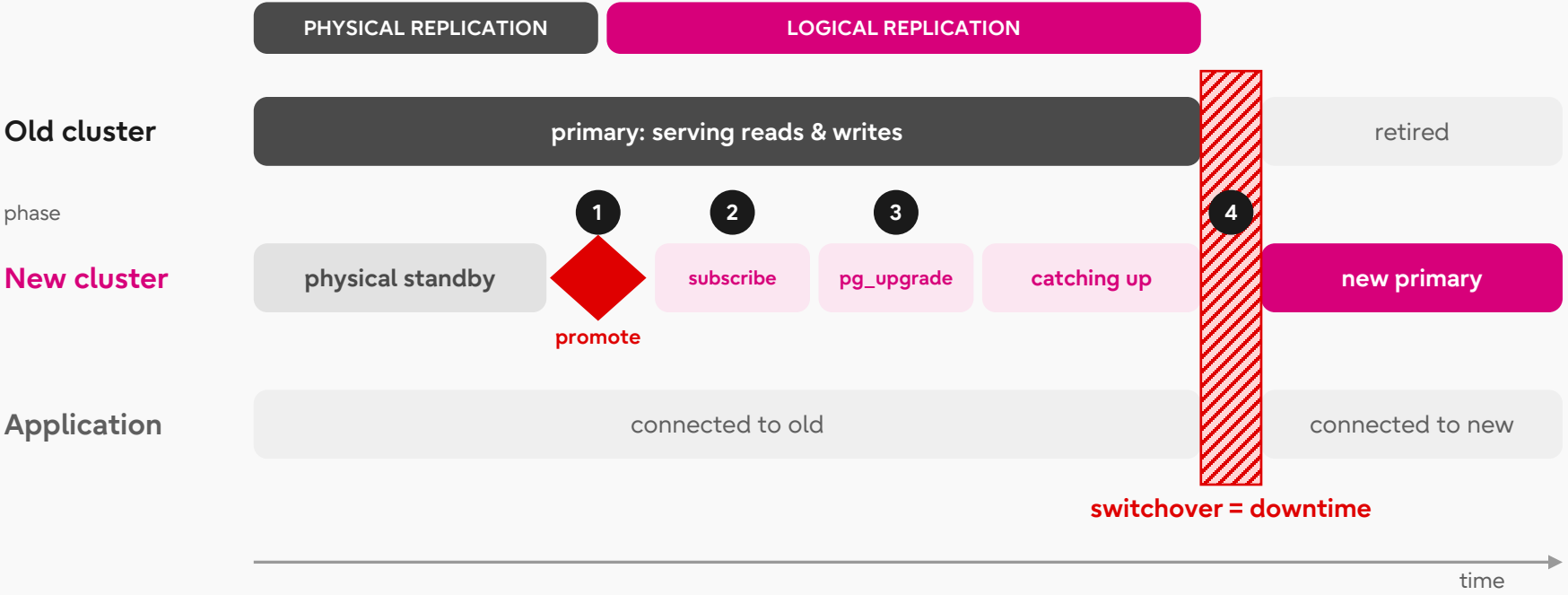
Physical + logical replication

Start from a physical standby (pg_createsubscriber)



© Fujitsu 2026

How it shrinks the downtime



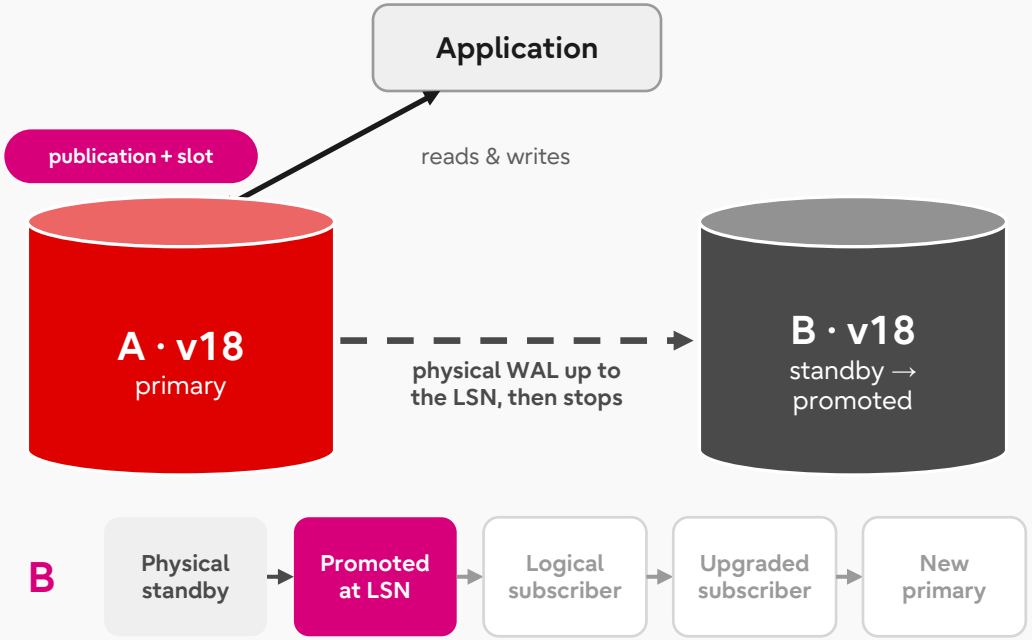
Everything slow happens while production is running. **Only the red sliver is downtime.**

© Fujitsu 2026

Phase 1 - Slot, then promote B

FUJITSU

0 Physical standby | 1 Promote at LSN | 2 Subscribe | 3 pg_upgrade | 4 Switchover | 5 Rebuild HA



Commands (on A and on B)

```
# 1. on B: stop it first
pg_ctl -D /data/v18 stop

-- 2. on A
CREATE PUBLICATION upg_pub
  FOR ALL TABLES;

SELECT lsn FROM
  pg_create_logical_replication_slot(
    'upg_slot', 'pgoutput');

# 3. on B: recovery target, start
recovery_target_lsn = '0/3000148'
recovery_target_action = 'promote'
recovery_target_inclusive = true
pg_ctl -D /data/v18 start
```

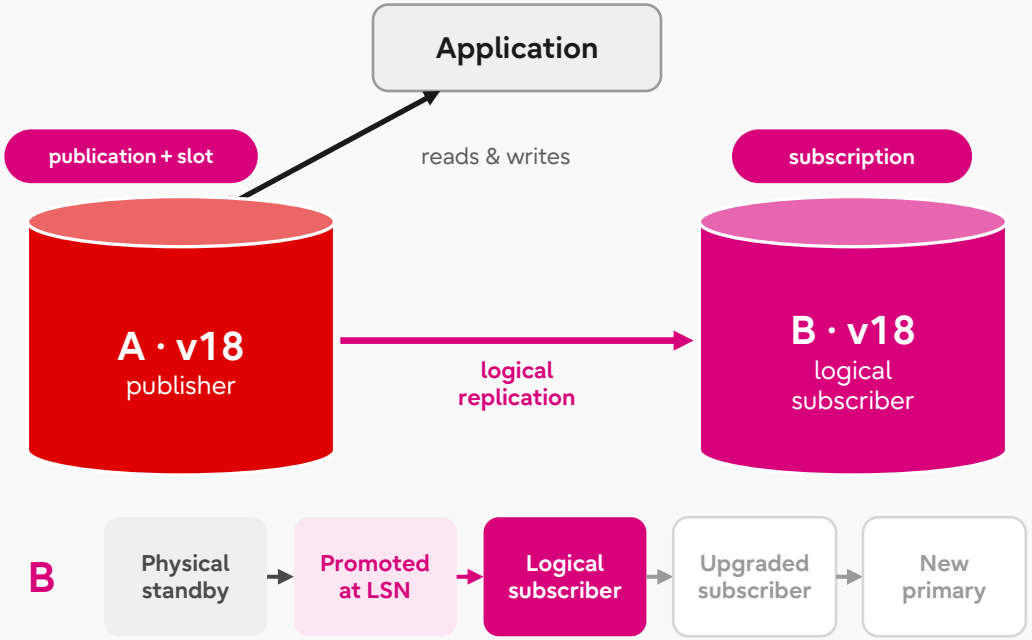
- Stop B first, so it can't replay past the LSN
- Slot's LSN = hand-off: physical before, logical after

© Fujitsu 2026

Phase 2 - B becomes a subscriber

FUJITSU

0 Physical standby | 1 Promote at LSN | **2 Subscribe** | 3 pg_upgrade | 4 Switchover | 5 Rebuild HA



Commands (on B)

```
CREATE SUBSCRIPTION upg_sub
  CONNECTION 'host=A dbname=app'
  PUBLICATION upg_pub
  WITH (create_slot = false,
        slot_name = 'upg_slot',
        copy_data = false,
        enabled = false);

SELECT pg_replication_origin_advance(
  'pg_' || <sub oid>, '0/3000148');

ALTER SUBSCRIPTION upg_sub ENABLE;
```

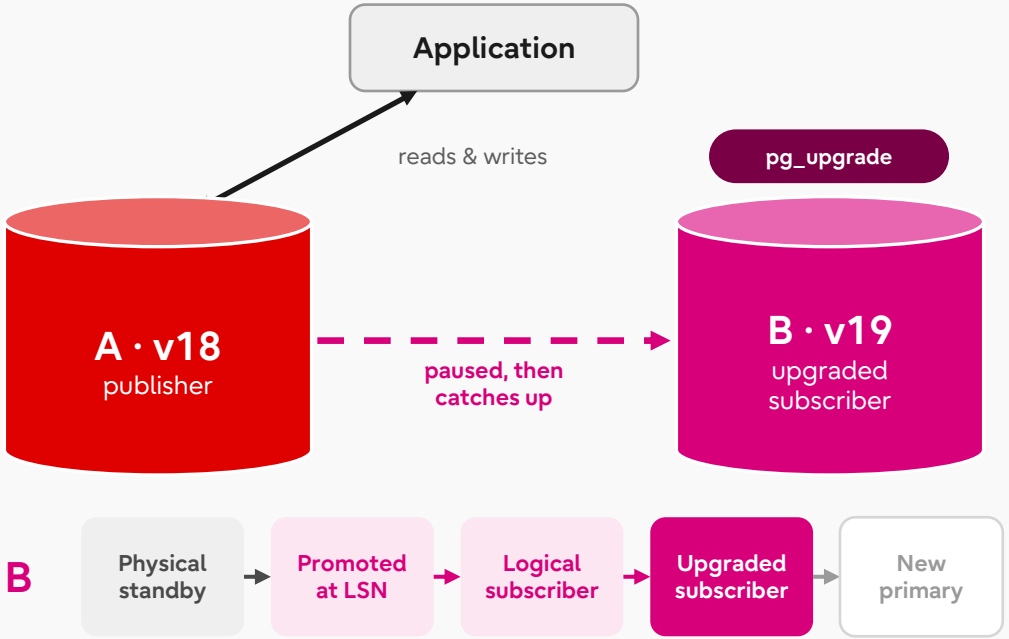
No initial copy: B already has the data. **Phases 1 + 2 are fiddly and error-prone.** Hold that thought.

© Fujitsu 2026

Phase 3 - Upgrade B with pg_upgrade

FUJITSU

0 Physical standby | 1 Promote at LSN | 2 Subscribe | **3 pg_upgrade** | 4 Switchover | 5 Rebuild HA



Commands (on B)

```
ALTER SUBSCRIPTION upg_sub DISABLE;

pg_ctl -D /data/v18 stop

initdb -D /data/v19

pg_upgrade --link \
  -b /usr/pg18/bin -B /usr/pg19/bin \
  -d /data/v18 -D /data/v19

pg_ctl -D /data/v19 start

ALTER SUBSCRIPTION upg_sub ENABLE;
```

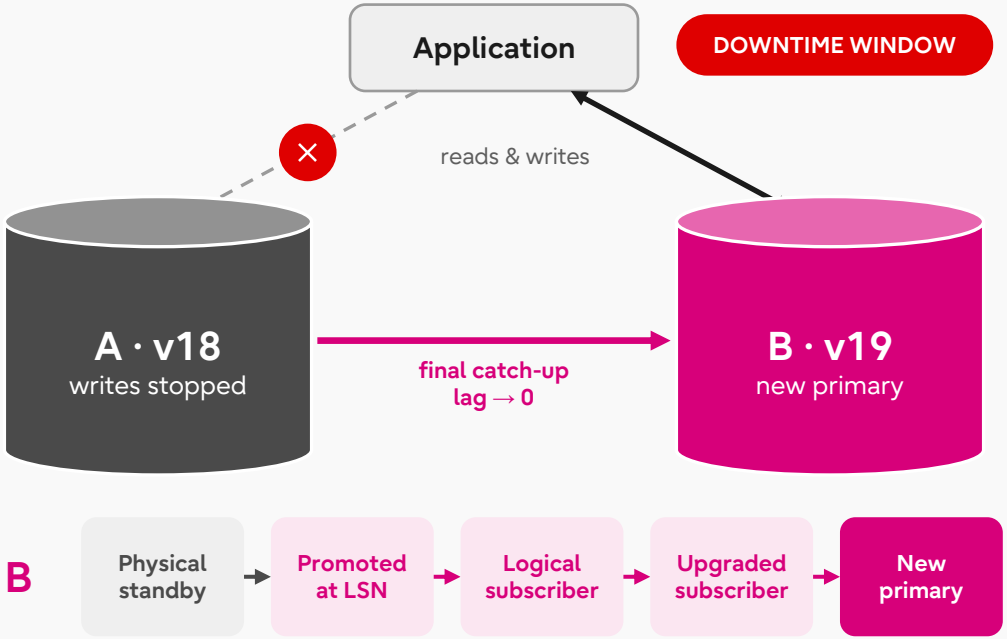
- Old cluster (B) must be PG17+ for pg_upgrade to keep the subscription and its progress
- A keeps taking writes; its slot holds WAL until B catches up

© Fujitsu 2026

Phase 4 - Switchover

FUJITSU

0 Physical standby | 1 Promote at LSN | 2 Subscribe | 3 pg_upgrade | **4 Switchover** | 5 Rebuild HA



Commands (on A and on B)

```
-- on A: block writes, note target LSN
SELECT pg_current_wal_lsn();
  -- e.g. 0/5A3C1F8

-- on A: repeat until true,
-- then repoint the app to B
SELECT confirmed_flush_lsn
         >= '0/5A3C1F8'
  FROM pg_replication_slots
 WHERE slot_name = 'upg_slot';
```

Is B really ready for writes?

Tables are in sync... but is everything? **Hold that thought too.**

© Fujitsu 2026

From a checklist to one command

FUJITSU

0 Physical standby | 1 Promote at LSN | 2 Subscribe | 3 pg_upgrade | 4 Switchover | 5 Rebuild HA

Before: by hand

1. Create publication on A
2. Create logical slot, note the LSN
3. Set recovery_target_lsn on B
4. Restart B and wait for promotion
5. Create disabled subscription on B
6. Look up the subscription's origin
7. Advance the origin to the LSN
8. Enable the subscription
9. Clean up standby slots / settings

After: pg_createsubscriber (PG17+)

```
# B stopped; run on B's host
pg_createsubscriber \
  -D /data/v18 \
  -P 'host=A dbname=app' \
  -d app \
  --publication=upg_pub \
  --subscription=upg_sub
```

- **`--dry-run`** to validate first
- Target must be a physical standby of the source, same major version
- Changes B's system identifier - no way back

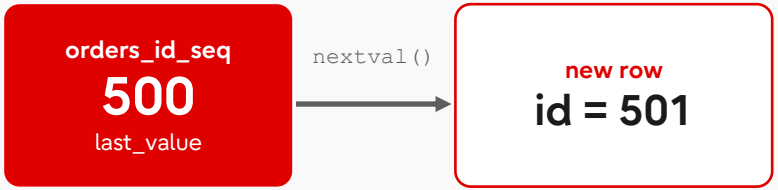
© Fujitsu 2026

Sequences 101

FUJITSU

```
CREATE TABLE orders (
  id   serial PRIMARY KEY,
  item text
);
-- serial implicitly creates
-- sequence orders_id_seq with
-- DEFAULT nextval('orders_id_seq')

INSERT INTO orders (item)
  VALUES ('widget');  -- id = nextval()
```



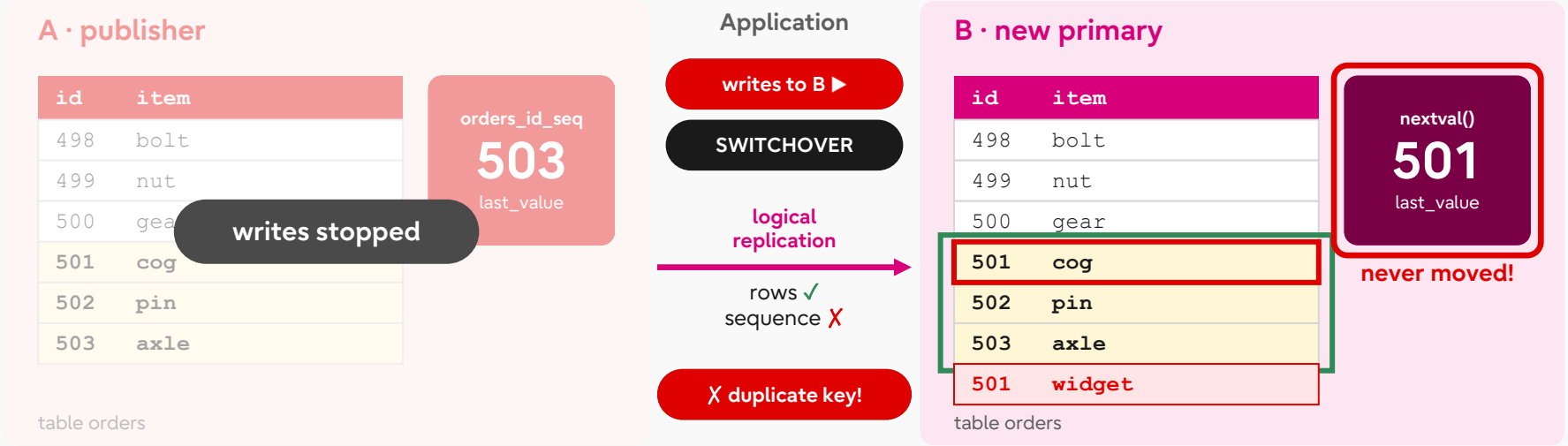
Replicating the rows **does not** move the counter.

1. **A separate object** — The counter lives outside the table
2. **Non-transactional** — nextval() is never rolled back - gaps are normal
3. **Logged ahead** — WAL written ahead in batches (typically 32 values), not per call

© Fujitsu 2026

What goes wrong: a worked example

FUJITSU




```
B=# INSERT INTO orders (item) VALUES ('widget');
ERROR:  duplicate key value violates unique constraint "orders_pkey"
DETAIL:  Key (id)=(501) already exists.
```

Rows replicated perfectly, sequence stale → **every insert fails right after go-live.**

© Fujitsu 2026

PG19: sequences join the publication

FUJITSU

```
-- publisher: new publication
CREATE PUBLICATION pub
  FOR ALL TABLES, ALL SEQUENCES;

-- ...or extend an existing one
ALTER PUBLICATION pub
  SET ALL TABLES, ALL SEQUENCES;

-- subscriber
CREATE SUBSCRIPTION sub
  CONNECTION '...' PUBLICATION pub;
```

When are sequence values synced?

`CREATE SUBSCRIPTION`
Initial values (copy_data = true)

`REFRESH PUBLICATION`
Newly added sequences

`REFRESH SEQUENCES`
Re-sync sequences it already knows about

A snapshot, not a stream. Sequence values are copied when you ask for it - they do not follow the publisher continuously.

© Fujitsu 2026

Inside REFRESH SEQUENCES

FUJITSU

A · PUBLISHER

B · SUBSCRIBER

1. `ALTER SUBSCRIPTION sub REFRESH SEQUENCES;` returns at once
2. Every subscribed sequence set to **'i' (INIT)** in pg_subscription_rel
3. A **sequence sync worker** starts (uses a max_sync_workers_per_subscription slot)
4. Fetch publisher sequence data, **100 per batch**
5. **Definitions must match;** on mismatch: log the error, retry later
6. **Set the local value directly** a state copy, no WAL replay
7. **'r' (READY);** store publisher page_lsn in `srsublsn`

`pg_get_sequence_data()`
last_value, is_called, page_lsn
+ sequence definition



It's asynchronous

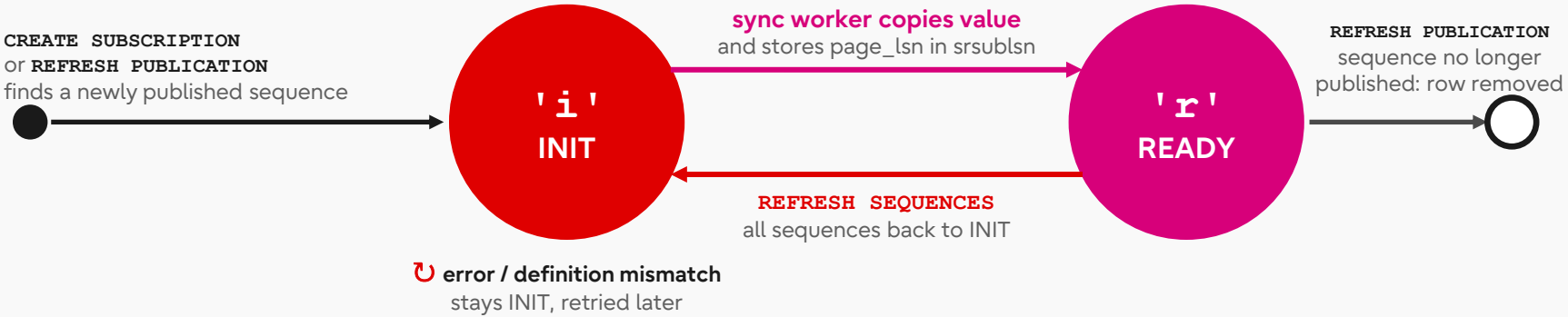
The command returns before any value is copied. Check that the sync has finished before you switch over.

```
-- any sequence not READY?
SELECT srrelid::regclass,
       srsubstate
  FROM pg_subscription_rel r
  JOIN pg_class c
    ON c.oid = r.srrelid
 WHERE c.relkind = 'S'
   AND srsubstate <> 'r';
```

0 rows = all sequences in sync

© Fujitsu 2026

Sequence sync states



Only two states

Tables go i → d → f → s → r (copy, then catch up). Sequences are a single value copy: i → r.

READY ≠ up to date

READY means synced as of srsublsn. The publisher keeps moving; run REFRESH SEQUENCES again before switchover.

© Fujitsu 2026

When to refresh sequences

FUJITSU

1. **Before switchover**

   After writes on the publisher stop, before the app moves. The one that matters for upgrades.

2. **Before any planned failover**

   To a logical replica you intend to write to.

3. **Periodically (optional)**

   Keeps a replica roughly current. Never replaces the final refresh.

4. **After fixing a mismatch**

   Once ALTER SEQUENCE aligns definitions on the subscriber.

Added sequences to the publication? REFRESH PUBLICATION first - REFRESH SEQUENCES only covers sequences the subscription already knows about.

© Fujitsu 2026

The upgrade workflow, completed

FUJITSU

pg_create-subscriber | pg_upgrade | Stop writes | Catch up | **Sync sequences** | Switch writes | Rebuild HA

NEW in PG19

Needs PG19+ on A and B: applies to upgrades from PG19, e.g. 19 → 20

A Publication already has sequences

```
-- on B
ALTER SUBSCRIPTION sub
  REFRESH PUBLICATION;
ALTER SUBSCRIPTION sub
  REFRESH SEQUENCES;
```

B Publication has tables only

```
-- on A
ALTER PUBLICATION pub SET
  ALL TABLES, ALL SEQUENCES;
-- on B
ALTER SUBSCRIPTION sub
  REFRESH PUBLICATION;
ALTER SUBSCRIPTION sub
  REFRESH SEQUENCES;
```

C Separate sequence publication

```
-- on A
CREATE PUBLICATION pub_seq
  FOR ALL SEQUENCES;
-- on B
CREATE SUBSCRIPTION sub_seq
  ... PUBLICATION pub_seq;
ALTER SUBSCRIPTION sub_seq
  REFRESH PUBLICATION;
ALTER SUBSCRIPTION sub_seq
  REFRESH SEQUENCES;
```

Then: wait until every sequence in pg_subscription_rel is READY ('r'), **then redirect writes.**

© Fujitsu 2026

Things to keep in mind

FUJITSU

Publisher must run PostgreSQL 19+

An 18 → 19 upgrade still needs setval(). The payoff starts with upgrades from 19 → 20 onward.

Snapshot, not stream

Values are copied only when a sync is triggered

Definitions must match

Type, increment, min/max, cycle... fix with ALTER SEQUENCE

Sequences must exist

DDL isn't replicated - create them on the subscriber first

Shares sync workers

Counts toward max_sync_workers_per_subscription

Monitor it

sync_seq_error_count in pg_stat_subscription_stats

LSN checks aren't exact

Drift within a cached block of ~32 values is invisible

© Fujitsu 2026

Key takeaways

FUJITSU

1

Upgrades are the killer use case

Logical replication crosses major versions, so downtime shrinks to a switchover.

2

pg_createsubscriber (PG17+)

Turns an existing physical standby into a logical subscriber - no initial copy.

3

Sequences were the gap

Tables replicate. Sequences don't – duplicate keys right after go-live.

4

PG19 closes it

Publish ALL SEQUENCES; run REFRESH SEQUENCES after writes stop, before switchover.

© Fujitsu 2026

References

1. **Vigneshwaran C - Closing a critical gap in PostgreSQL upgrade workflows**
   postgresql.fastware.com/blog/closing-a-critical-gap-in-postgresql-upgrade-workflows-with-sequence-synchronization
2. **PostgreSQL 19 docs - 29.7 Replicating Sequences**
   postgresql.org/docs/19/logical-replication-sequences.html
3. **PostgreSQL 19 docs - ALTER SUBSCRIPTION**
   postgresql.org/docs/19/sql-altersubscription.html
4. **PostgreSQL 19 docs - ALTER PUBLICATION**
   postgresql.org/docs/19/sql-alterpublication.html
5. **PostgreSQL 19 docs - pg_createsubscriber**
   postgresql.org/docs/19/app-pgcreatesubscriber.html
6. **PostgreSQL 19 release notes**
   postgresql.org/docs/19/release-19.html

© Fujitsu 2026

Part 2

Logical Replication
New Syntax in PostgreSQL 19+

Sequences, EXCEPT clauses,
and the catalogs behind them



Peter Smith
- PostgreSQL Significant Contributor

© Fujitsu 2026

Agenda

- **Syntax Evolution**
- **System Catalogs**
- **Sequences**
  - Publishing
  - Synchronising
  - Example
- **Publication Exclusions**
  - FOR ALL TABLES EXCEPT
  - Example
  - PG 20+



© Fujitsu 2026

Syntax Evolution

© Fujitsu 2026

Syntax Evolution – 1 of 2

FUJITSU

```
CREATE PUBLICATION pub_name
```

	PG version
FOR ALL TABLES	10
FOR TABLE t1	10
FOR TABLE t1 (c1,c2)	15
FOR TABLE t1 WHERE (condition)	15
FOR TABLES IN SCHEMA s1	15
FOR ALL TABLES EXCEPT (TABLE t1,t2)	19
FOR ALL SEQUENCES	19
FOR ALL SEQUENCES EXCEPT (SEQUENCE seq1,seq2)	20+ ?
FOR TABLES IN SCHEMA s1 EXCEPT (TABLE t1,t2)	20+ ?

© Fujitsu 2026

Syntax Evolution – 2 of 2

FUJITSU

```
ALTER PUBLICATION pub_name
```

	PG version
`ADD/SET TABLE …`	10
`ADD/SET TABLES IN SCHEMA s1`	15
`SET ALL TABLES` `EXCEPT (TABLE t1,t2)`	19
`SET ALL SEQUENCES`	19
`SET ALL SEQUENCES` `EXCEPT (SEQUENCE seq1,seq2)`	20+ ?
`ADD/SET TABLES IN SCHEMA s1` `EXCEPT (TABLE t1,t2)`	20+ ?

```
ALTER SUBSCRIPTION sub_name
```

`REFRESH PUBLICATION`	10
`REFRESH SEQUENCES`	19

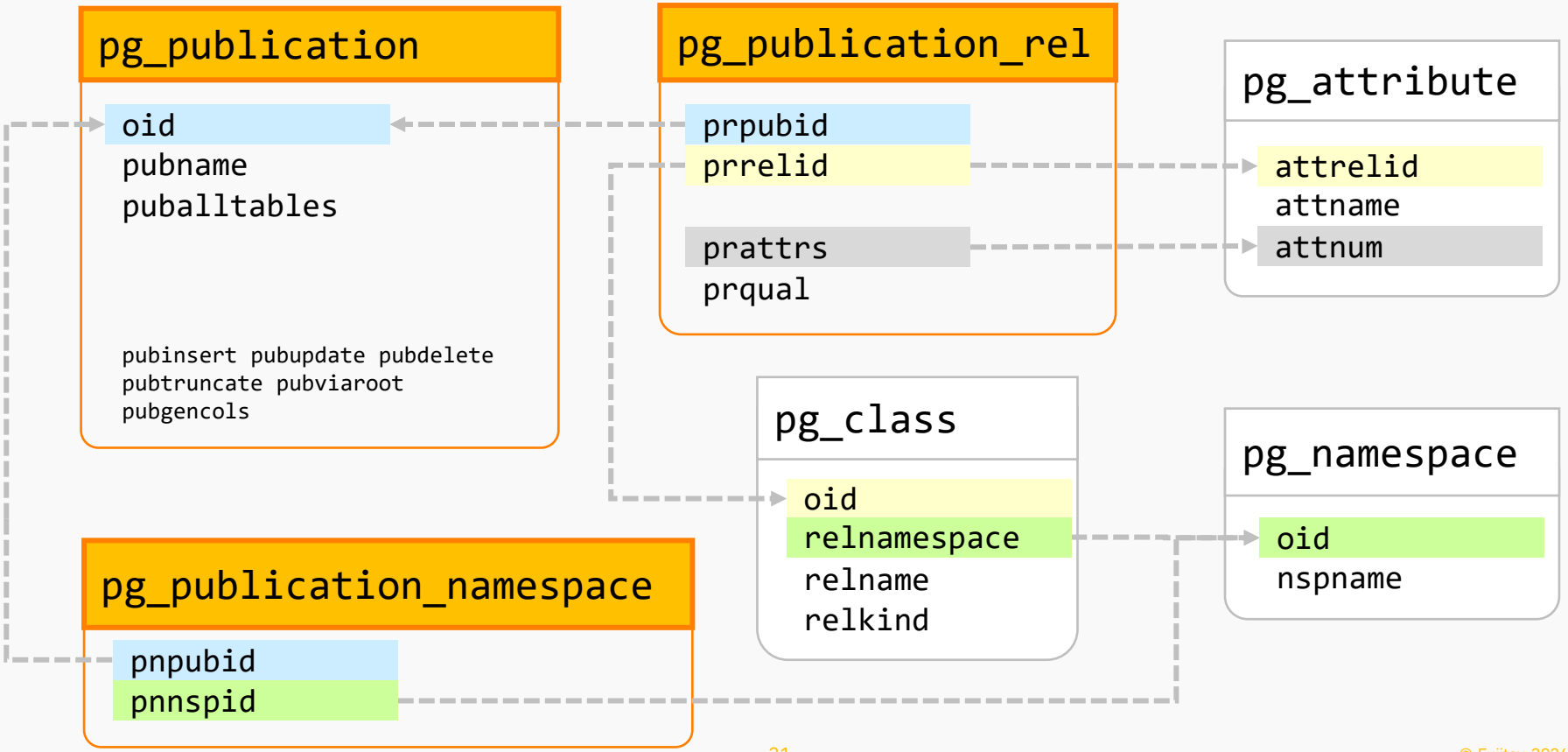
© Fujitsu 2026

System Catalogs PG18

© Fujitsu 2026

Publication Catalogs – PG18

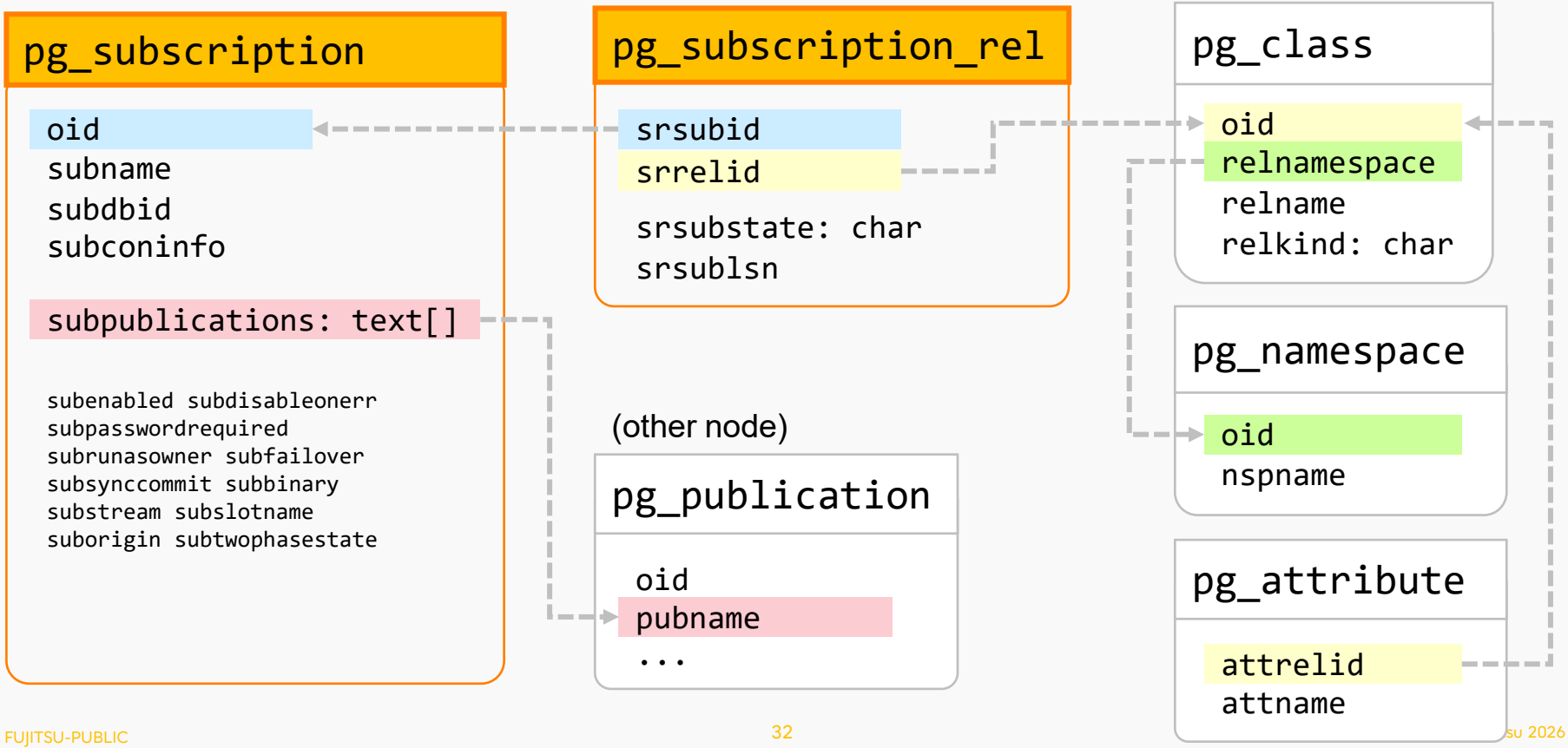
FUJITSU



© Fujitsu 2026

Subscription Catalogs – PG18

FUJITSU



SEQUENCE synchronisation

© Fujitsu 2026

SEQUENCE synchronisation

FUJITSU

- PG18 only replicates table data, e.g.
  - `CREATE PUBLICATION pub_name FOR ALL TABLES;`
- Recap from Part 1
  - We need some way to synchronise subscriber sequence values with the publisher, to avoid problems after promotion
- New syntax
  - `CREATE PUBLICATION pub_name FOR ALL SEQUENCES;`
  - `ALTER PUBLICATION pub_name SET ALL SEQUENCES;`
  - `ALTER SUBSCRIPTION sub_name REFRESH SEQUENCES;`

© Fujitsu 2026

SEQUENCE publication

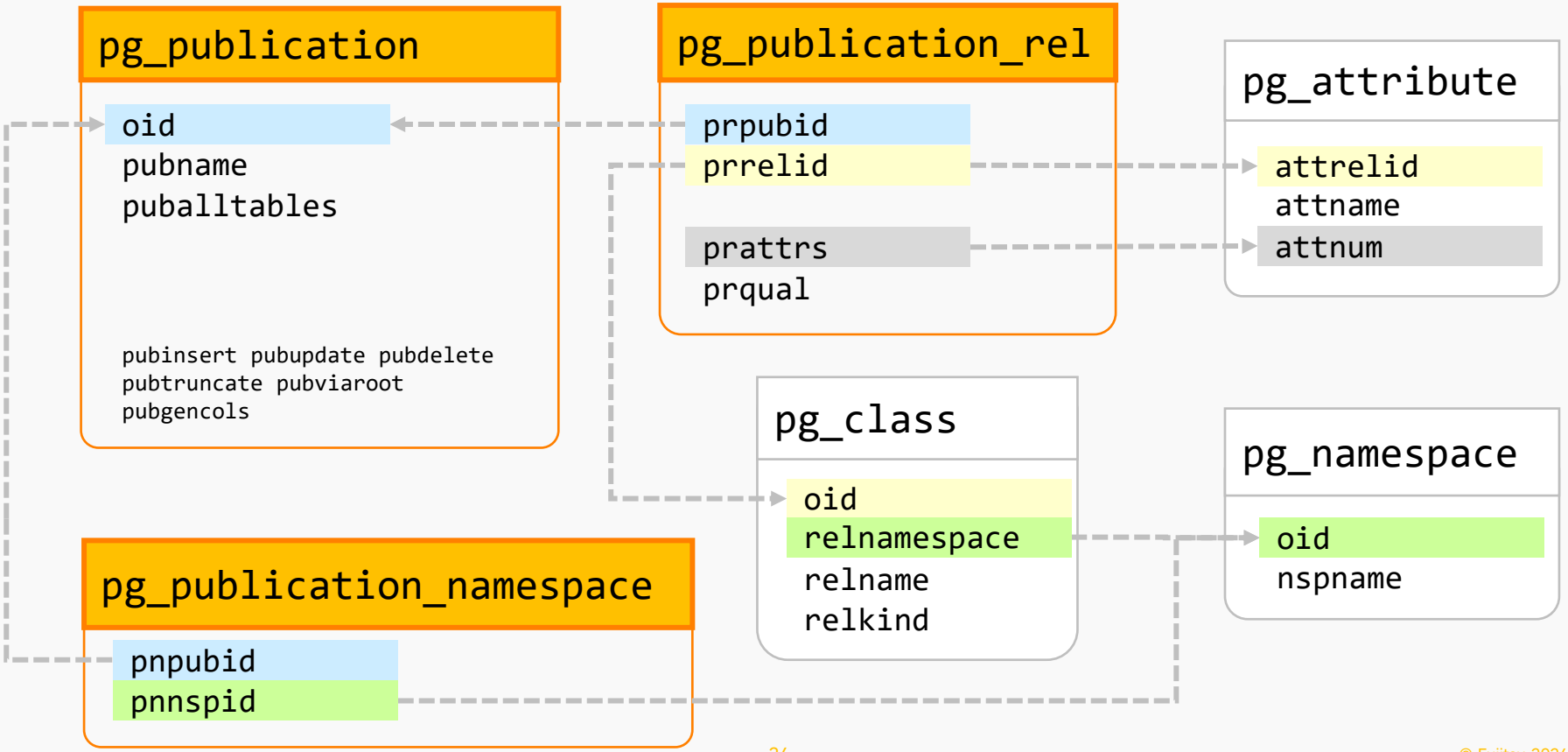
FUJITSU

- Syntax
  - `CREATE PUBLICATION pub_name FOR ALL SEQUENCES;`
  - `ALTER PUBLICATION pub_name SET ALL SEQUENCES;`
- Catalog
  - `pg_publication.puballsequences : bool (true)`
- Effect
  - Only the flag is set.

© Fujitsu 2026

Publication Catalogs – PG18

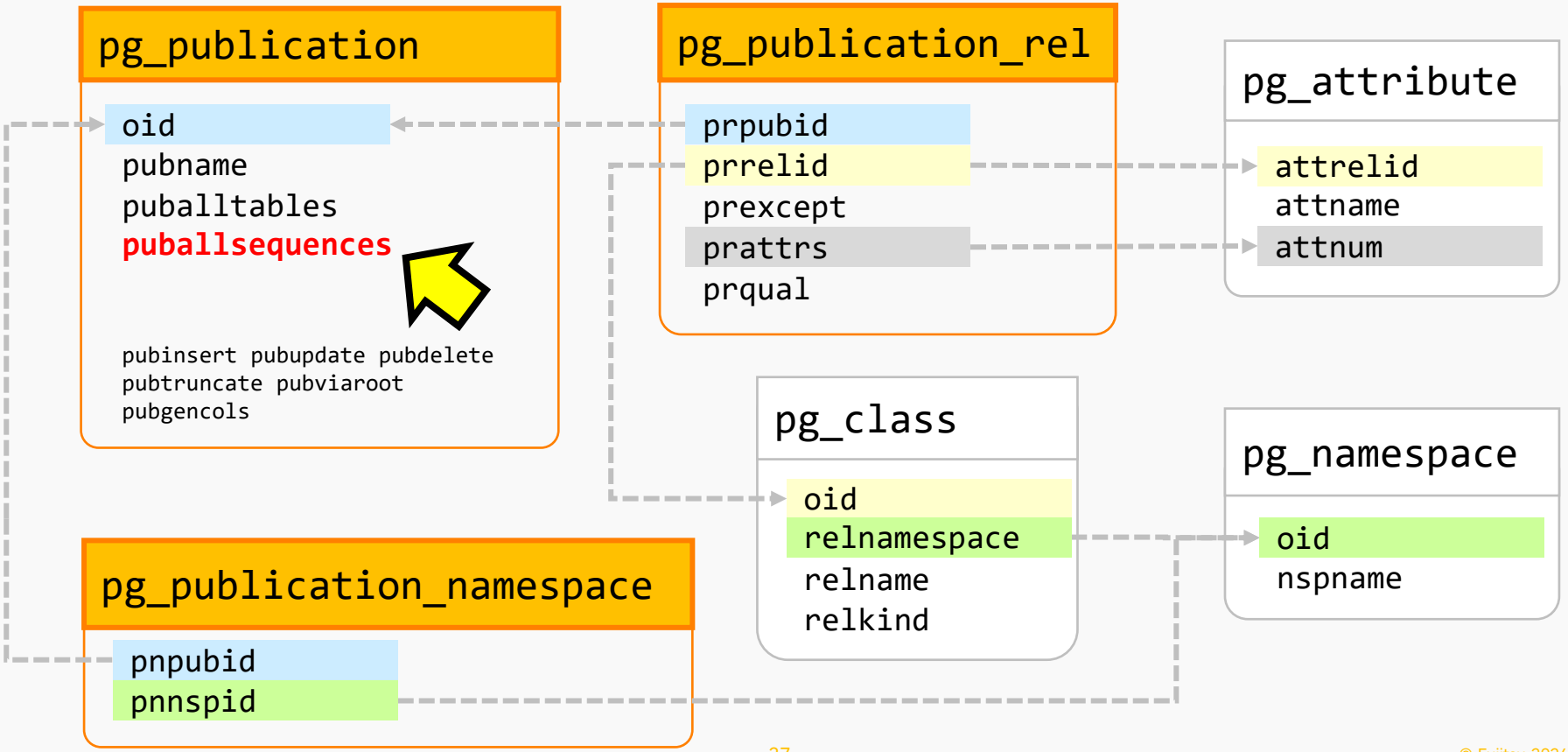
FUJITSU



© Fujitsu 2026

Publication Catalogs – PG19

FUJITSU



© Fujitsu 2026

SEQUENCE publication – flexible syntax

FUJITSU

- Combinations
  - `CREATE PUBLICATION pub_name FOR ALL SEQUENCES;`
  - `CREATE PUBLICATION pub_name FOR ALL SEQUENCES, ALL TABLES;`
  - `CREATE PUBLICATION pub_name FOR ALL TABLES, ALL SEQUENCES;`

© Fujitsu 2026

SEQUENCE refresh

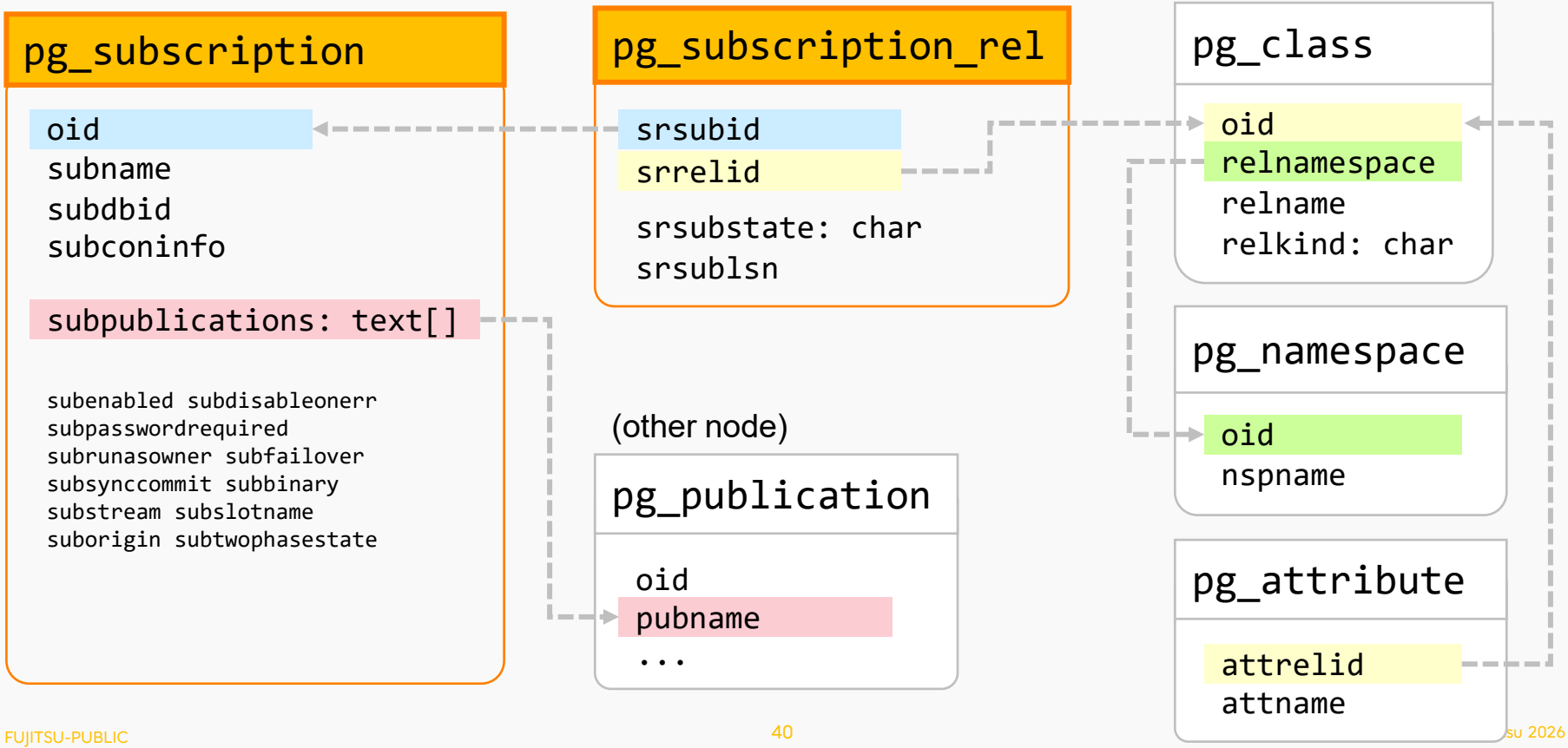
Values are not replicated as they change on the publisher

- Value synchronisation is “triggered” by these commands:
  - `CREATE SUBSCRIPTION sub_name … WITH (copy_data=true);`
  - `ALTER SUBSCRIPTION sub_name REFRESH PUBLICATION;`
  - `ALTER SUBSCRIPTION sub_name REFRESH SEQUENCES;`
- Catalog (no changes)
  - Sequences are just *relations* in `pg_subscription_rel`
- Effect
  - `pg_subscription_rel` rows are added/removed, or state updated
  - A **sequencesync** worker batch-processes `INIT`-state sequences

© Fujitsu 2026

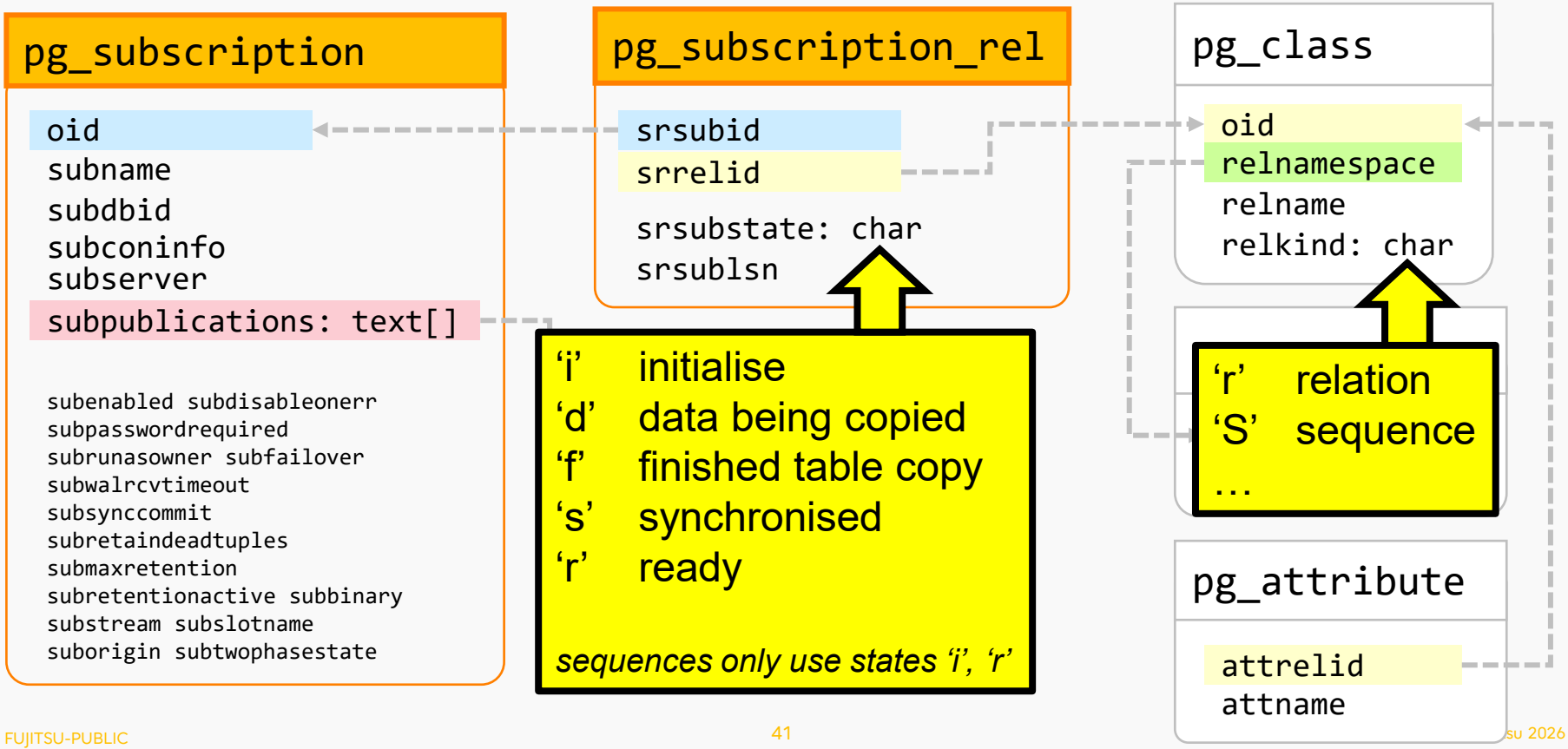
Subscription Catalogs – PG18

FUJITSU



Subscription Catalogs – PG19

FUJITSU



INIT state

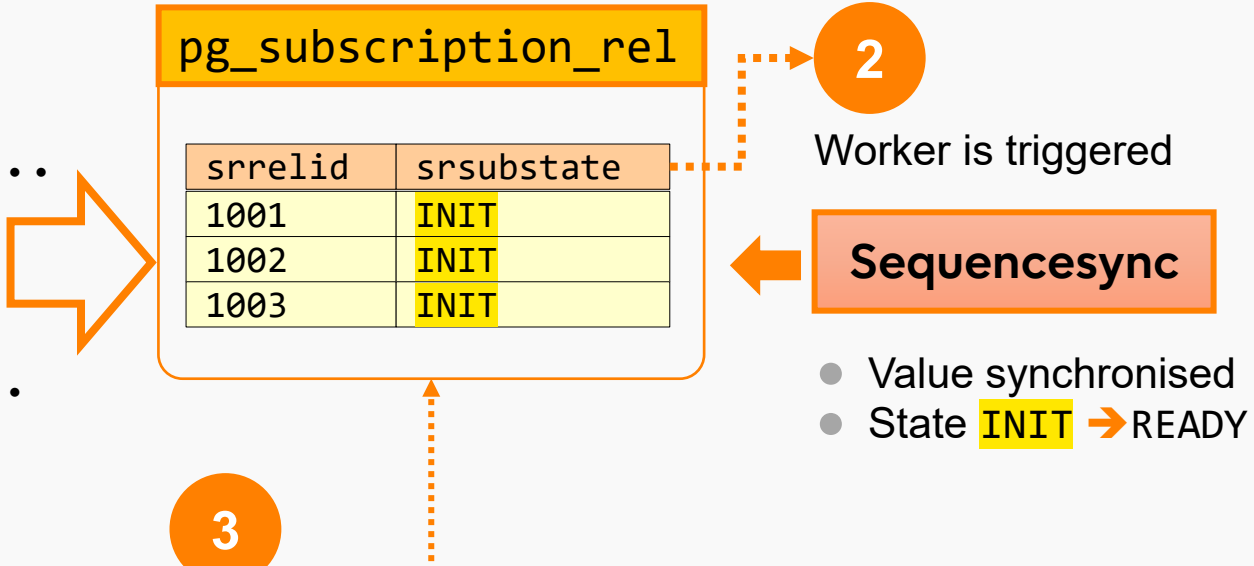
FUJITSU

1

CREATE SUBSCRIPTION ...

ALTER SUBSCRIPTION ...
REFRESH PUBLICATION

- Fetches from publisher
- Add/remove sequences
- State set `INIT` for new ones (unless `copy_data` is false)



© Fujitsu 2026

INIT → READY state

FUJITSU

- Apply worker checks `pg_subscription_rel`
- For tables with `INIT` state
  - Apply worker starts a **tablesync** worker (per table)
  - `INIT` → `DATASYNC` → `FINISHED_COPY` → `SYNCDONE` → `READY`
- For sequences with `INIT` state
  - Apply worker starts a **sequencesync** worker (only one, batches)
  - `INIT` → `READY`

© Fujitsu 2026

SEQUENCE refresh – Example

FUJITSU

```
CREATE PUBLICATION pub FOR ALL TABLES, ALL SEQUENCES;
```

PUBLISHER

Tables

relid	relname
101	my_tab1
102	my_tab2

Sequences

relid	relname
1001	my_seq1
1002	my_seq2

SUBSCRIBER

pg_subscription_rel

srrelid	srsubstate

© Fujitsu 2026

SEQUENCE refresh – Example

FUJITSU

```
CREATE SUBSCRIPTION sub ... PUBLICATION pub;
```

PUBLISHER

Tables

relid	relname
101	my_tab1
102	my_tab2

Sequences

relid	relname
1001	my_seq1
1002	my_seq2



SUBSCRIBER

pg_subscription_rel

srrelid	srsubstate
101	INIT
102	INIT
1001	INIT
1002	INIT

Relations are populated with INIT state

© Fujitsu 2026

SEQUENCE refresh – Example

FUJITSU

Workers are triggered …

PUBLISHER

Tables

relid	relname
101	my_tab1
102	my_tab2

Sequences

relid	relname
1001	my_seq1
1002	my_seq2

SUBSCRIBER

pg_subscription_rel

srrelid	srsubstate
101	INIT
102	INIT
1001	INIT
1002	INIT

Workers

Tablesync: 101

Tablesync: 102

Sequencesync

Workers are started for relations with INIT state

© Fujitsu 2026

SEQUENCE refresh – Example

FUJITSU

Values are copied, states become READY, workers finish.

PUBLISHER

Tables

relid	relname
101	my_tab1
102	my_tab2

Sequences

relid	relname
1001	my_seq1
1002	my_seq2

SUBSCRIBER

pg_subscription_rel

srrelid	srsubstate
101	READY
102	READY
1001	READY
1002	READY

READY state

© Fujitsu 2026

SEQUENCE refresh – Example

FUJITSU

Publisher adds more tables and sequences.

PUBLISHER

Tables

relid	relname
101	my_tab1
102	my_tab2
103	my_tab3

Sequences

relid	relname
1001	my_seq1
1002	my_seq2
1003	my_seq3

SUBSCRIBER

pg_subscription_rel

srrelid	srsubstate
101	READY
102	READY
1001	READY
1002	READY

Subscription is unaware

© Fujitsu 2026

SEQUENCE refresh – Example

FUJITSU

ALTER SUBSCRIPTION sub REFRESH SEQUENCES;

PUBLISHER

Tables

relid	relname
101	my_tab1
102	my_tab2
103	my_tab3

Sequences

relid	relname
1001	my_seq1
1002	my_seq2
1003	my_seq3

SUBSCRIBER

pg_subscription_rel

srrelid	srsubstate
101	READY
102	READY
1001	INIT
1002	INIT

The “known” sequences are reset to INIT state

© Fujitsu 2026

SEQUENCE refresh – Example

FUJITSU

Workers are triggered …

PUBLISHER

Tables

relid	relname
101	my_tab1
102	my_tab2
103	my_tab3

Sequences

relid	relname
1001	my_seq1
1002	my_seq2
1003	my_seq3

SUBSCRIBER

pg_subscription_rel

srrelid	srsubstate
101	READY
102	READY
1001	INIT
1002	INIT

Workers

Sequencesync

Workers are started for relations with INIT state

© Fujitsu 2026

SEQUENCE refresh – Example

FUJITSU

Values are copied, states become READY, workers finish.

PUBLISHER

Tables

relid	relname
101	my_tab1
102	my_tab2
103	my_tab3

Sequences

relid	relname
1001	my_seq1
1002	my_seq2
1003	my_seq3

SUBSCRIBER

pg_subscription_rel

srrelid	srsubstate
101	READY
102	READY
1001	READY
1002	READY

READY state

© Fujitsu 2026

SEQUENCE refresh – Example

FUJITSU

ALTER SUBSCRIPTION sub REFRESH PUBLICATION;

PUBLISHER

Tables

relid	relname
101	my_tab1
102	my_tab2
103	my_tab3

Sequences

relid	relname
1001	my_seq1
1002	my_seq2
1003	my_seq3



SUBSCRIBER

pg_subscription_rel

srrelid	srsubstate
101	READY
102	READY
1001	READY
1002	READY
103	INIT
1003	INIT

Fetch to update the subscription's "known" relations.

Added relations get INIT state

© Fujitsu 2026

SEQUENCE refresh – Example

FUJITSU

Workers are triggered …

PUBLISHER

Tables

relid	relname
101	my_tab1
102	my_tab2
103	my_tab3

Sequences

relid	relname
1001	my_seq1
1002	my_seq2
1003	my_seq3

SUBSCRIBER

pg_subscription_rel

srrelid	srsubstate
101	READY
102	READY
1001	READY
1002	READY
103	INIT
1003	INIT

Workers

Tablesync: 103

Sequencesync

Workers are started for relations with INIT state

© Fujitsu 2026

SEQUENCE refresh – Example

FUJITSU

Values are copied, states become READY, workers finish.

PUBLISHER

Tables

relid	relname
101	my_tab1
102	my_tab2
103	my_tab3

Sequences

relid	relname
1001	my_seq1
1002	my_seq2
1003	my_seq3

SUBSCRIBER

pg_subscription_rel

srrelid	srsubstate
101	READY
102	READY
1001	READY
1002	READY
103	READY
1003	READY

Stale values?

READY state

© Fujitsu 2026

SEQUENCE refresh – Example

FUJITSU

```
ALTER SUBSCRIPTION sub REFRESH SEQUENCES;
```

PUBLISHER

Tables

relid	relname
101	my_tab1
102	my_tab2
103	my_tab3

Sequences

relid	relname
1001	my_seq1
1002	my_seq2
1003	my_seq3

SUBSCRIBER

pg_subscription_rel

srrelid	srsubstate
101	READY
102	READY
1001	INIT
1002	INIT
103	READY
1003	INIT

The “known” sequences are reset to INIT state

© Fujitsu 2026

SEQUENCE refresh – Example

FUJITSU

Workers are triggered …

PUBLISHER

Tables

relid	relname
101	my_tab1
102	my_tab2
103	my_tab3

Sequences

relid	relname
1001	my_seq1
1002	my_seq2
1003	my_seq3

SUBSCRIBER

pg_subscription_rel

srrelid	srsubstate
101	READY
102	READY
1001	INIT
1002	INIT
103	READY
1003	INIT

Workers

Sequencesync

Workers are started for relations with INIT state

© Fujitsu 2026

SEQUENCE refresh – Example

FUJITSU

Values are copied, states become READY, workers finish.

PUBLISHER

Tables

relid	relname
101	my_tab1
102	my_tab2
103	my_tab3

Sequences

relid	relname
1001	my_seq1
1002	my_seq2
1003	my_seq3

SUBSCRIBER

pg_subscription_rel

srrelid	srsubstate
101	READY
102	READY
1001	READY
1002	READY
103	READY
1003	READY



© Fujitsu 2026

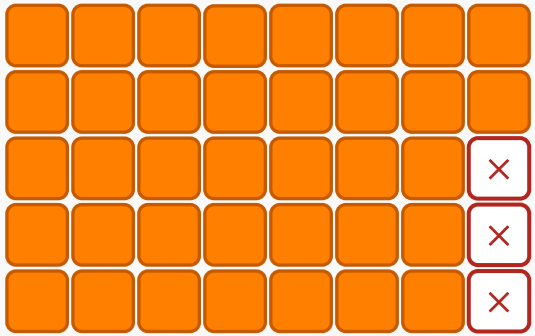
Publication exclusions

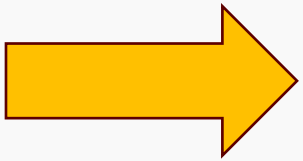
© Fujitsu 2026

Publication Excluding Tables – PG18

FUJITSU

- Let's say we have many tables
- And we want to publish most of them, but not all…
- `CREATE PUBLICATION pub_name FOR ALL TABLES;`





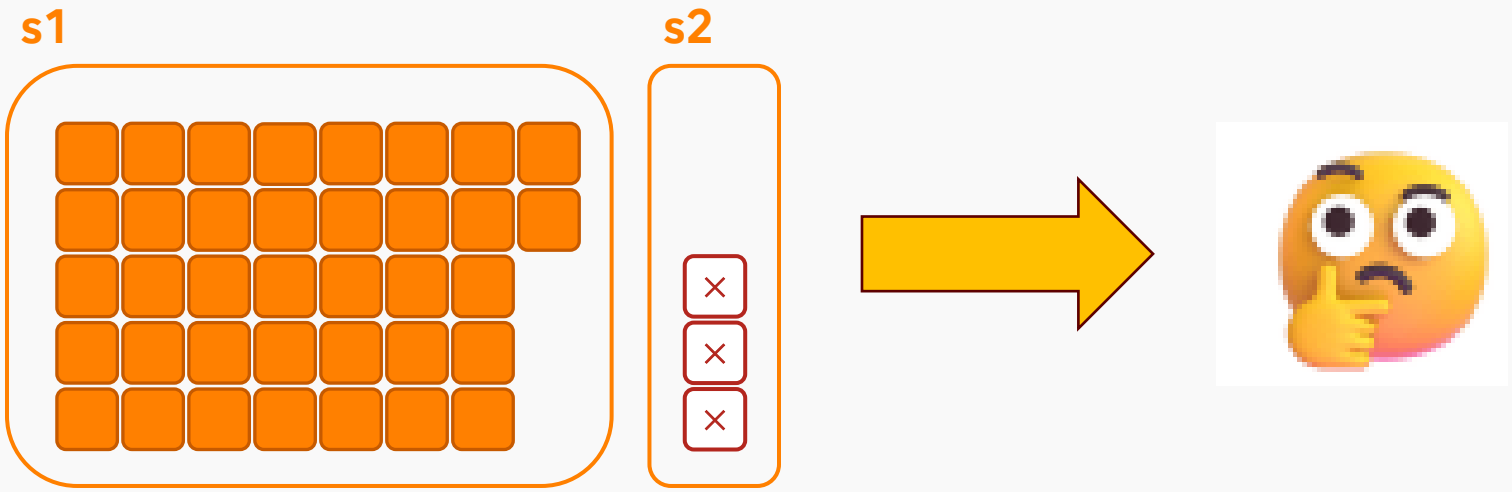


© Fujitsu 2026

Publication Excluding Tables – PG18

FUJITSU

- Rearrange tables into schemas, name what ones to publish
- `CREATE PUBLICATION pub_name FOR TABLES IN SCHEMA s1;`



© Fujitsu 2026

Publication Excluding Tables – PG18

FUJITSU

- Name every table to be published

- ```
  CREATE PUBLICATION pub_name FOR TABLE
  t01,t02,t03,t04,t05,t06,t07,t08,
  t09,t10,t11,t12,t13,t14,t15,t16,
  t17,t18,t19,t20,t21,t22,t23,
  t25,t26,t27,t28,t29,t30,t31,
  t33,t34,t35,t36,t37,t38,t39    ;
  ```



- But “future” tables are not in the publication

© Fujitsu 2026
  ```

FOR ALL TABLES EXCEPT

FUJITSU

- Publish every table, name only tables to be excluded

- ```
  CREATE PUBLICATION pub_name
  FOR ALL TABLES EXCEPT (TABLE t24,t32,t40);
  ```

- ```
  ALTER PUBLICATION pub_name
  SET ALL TABLES EXCEPT (TABLE t24,t32,t40);
  ```

SET overwrites previous exclusions

© Fujitsu 2026

FOR ALL TABLES EXCEPT – flexible syntax

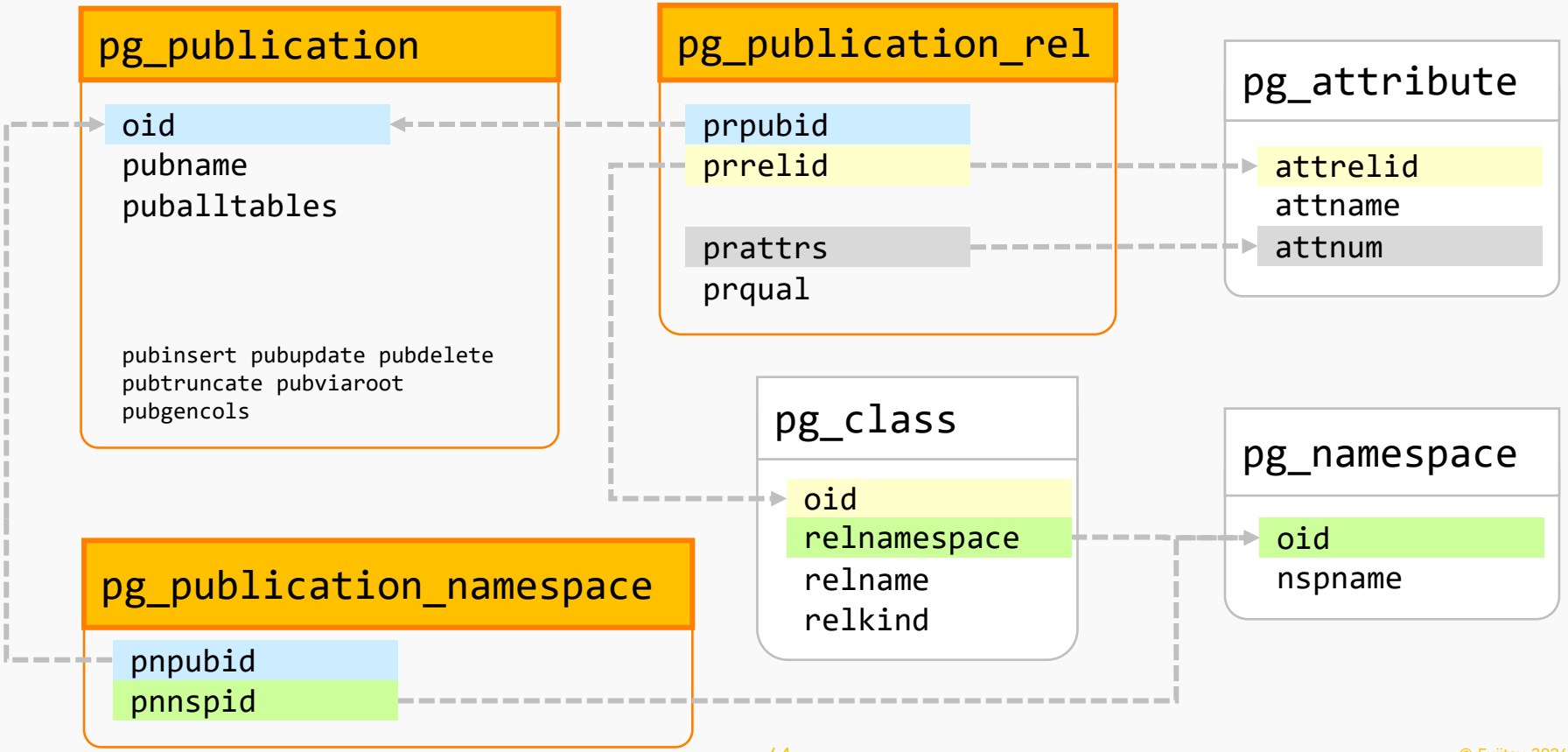
FUJITSU

- These are the same
  - `CREATE PUBLICATION pub_name FOR ALL TABLES EXCEPT (TABLE t24,t32,t40);`
  - `CREATE PUBLICATION pub_name FOR ALL TABLES EXCEPT (TABLE t24, TABLE t32,t40);`
  - `CREATE PUBLICATION pub_name FOR ALL TABLES EXCEPT (TABLE t24, TABLE t32, TABLE t40);`

© Fujitsu 2026

Publication Catalogs – PG18

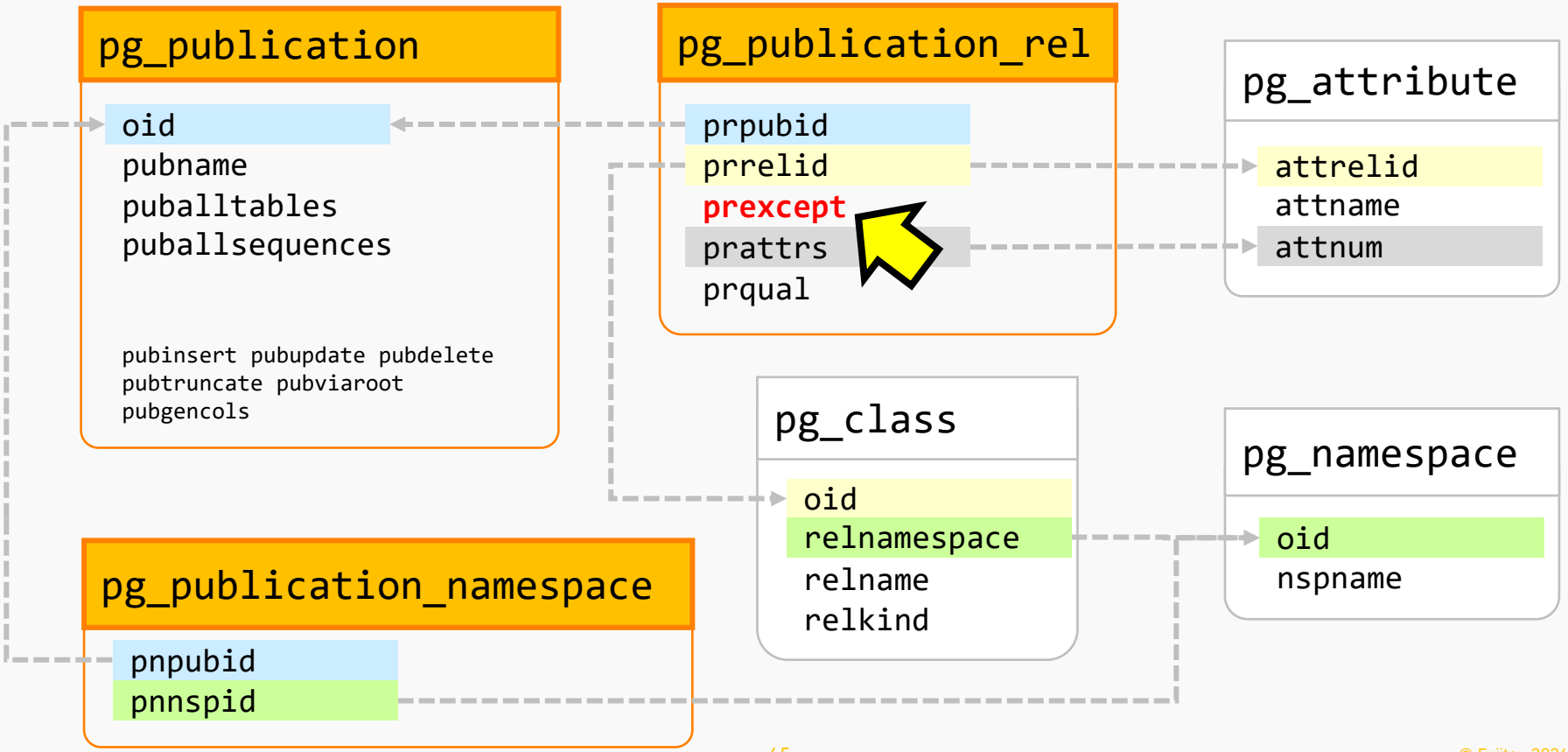
FUJITSU



© Fujitsu 2026

Publication Catalogs – PG19

FUJITSU

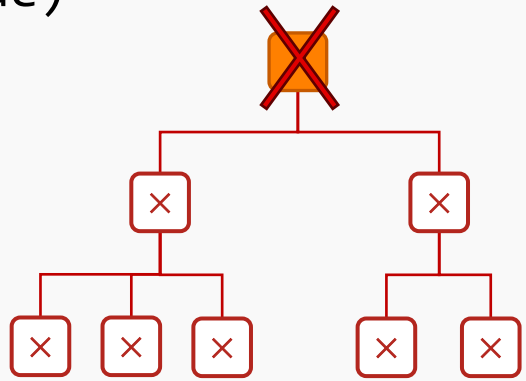


© Fujitsu 2026

FOR ALL TABLES EXCEPT

FUJITSU

- Catalogs
  - `pg_publication.puballtables : bool (true)`
  - `pg_publication_rel.prexcept : bool (true)`
- Partitions
  - Cannot specify partitions
  - Excluding a root excludes partitions beneath it



- psql
  - “describe” commands **\d** and **\dRp+** will show the exclusions

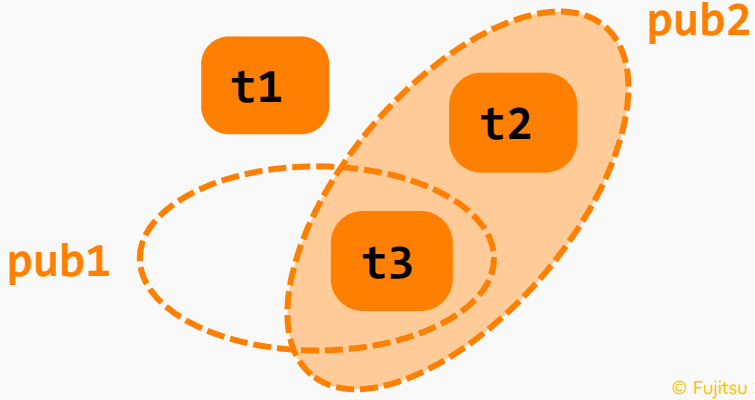
© Fujitsu 2026

Example - 1 of 2

```
-- create some test tables
CREATE TABLE t1(a int);
CREATE TABLE t2(a int);
CREATE TABLE t3(a int);

-- pub1 publishes all tables except t1 and t2
CREATE PUBLICATION pub1 FOR ALL TABLES EXCEPT (TABLE t1,t2);

-- pub2 publishes tables t2 and t3
CREATE PUBLICATION pub2 FOR TABLE t2,t3;
```



© Fujitsu 2026

Example – 2 of 2

FUJITSU

-- describe the publication, now shows which tables are excluded

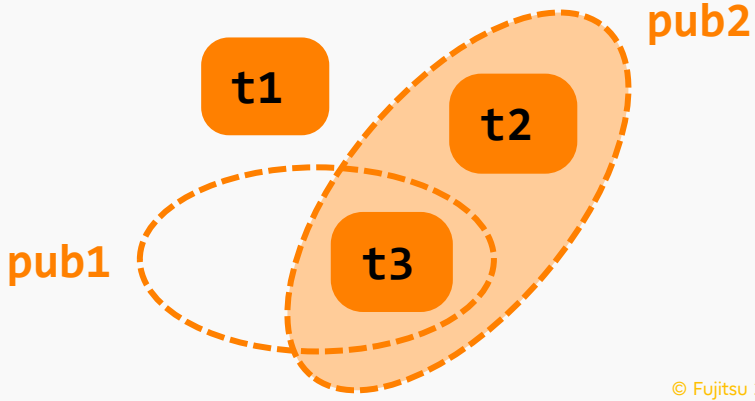
\dRp+ pub1

```
                                                Publication pub1
  Owner   | All tables | All sequences | Inserts | Updates | Deletes | Truncates | Generated columns | Via root | Description
----------+------------+---------------+---------+---------+---------+-----------+-------------------+----------+-------------
 postgres | t          | f             | t       | t       | t       | t         | none              | f        |
Except tables:
    "public.t1"
    "public.t2"
```

-- describe the table, now shows which publications are excluding it

\d t2

```
                Table "public.t2"
 Column |  Type   | Collation | Nullable | Default
--------+---------+-----------+----------+---------
 a      | integer |           |          |
Included in publications:
    "pub2"
Excluded from publications:
    "pub1"
```



© Fujitsu 2026

Proposals - 1 of 2

FUJITSU

- `CREATE PUBLICATION pub_name`
  `FOR ALL SEQUENCES EXCEPT (SEQUENCE seq1,seq2);`

- Catalog (no changes)
  - `pg_publication.puballsequences : bool (true)`
  - `pg_publication_rel.prexcept : bool (true)`

© Fujitsu 2026

Proposals - 2 of 2

FUJITSU

- `CREATE PUBLICATION pub_name`
  `FOR TABLES IN SCHEMA s1 EXCEPT (TABLE t1,t2);`

- Catalog (no changes)
  - `pg_publication_namespace`
  - `pg_publication_rel.prexcept : bool (true)`

- Can only exclude tables from the associated schema

- Partitions
  - Same rules as `FOR ALL TABLES EXCEPT (...)`

© Fujitsu 2026

Future exclusions?

FUJITSU

- EXCEPT could be extended to more places
  - `... FOR ALL TABLES EXCEPT (SCHEMA s1);`
  - `... FOR ALL SEQUENCES EXCEPT (SCHEMA s1);`
  - `... FOR SEQUENCES IN SCHEMA;`
  - `... FOR SEQUENCES IN SCHEMA EXCEPT (SEQUENCE seq1);`
  - `... FOR TABLE t1 EXCEPT (COLUMN c1,c2,c3);`

© Fujitsu 2026

Almost the end

© Fujitsu 2026

Summary

FUJITSU

Topic	System Catalogs	What Happens?
FOR ALL SEQUENCES	pg_publication.puballsequences	Sets flag
REFRESH PUBLICATION	pg_subscription_rel	Fetches. Now adds/removes sequences as well as tables
REFRESH SEQUENCES	pg_subscription_rel.srsubstate	Resets “known” sequences to INIT
sequencesync worker	pg_subscription_rel.srsubstate	Batch processing of INIT sequences: • Synchronises values. • States → READY
FOR ALL TABLES EXCEPT	pg_publication_rel.prexcept	Excludes from publication
psql \d psql \dRp+		Updated to show exclusions
TABLES IN SCHEMA EXCEPT SEQUENCES EXCEPT	(no changes)	Active proposals on pgsql-hackers

References

FUJITSU

- Online Documentation
  - PG19 CREATE PUBLICATION
  - PG19 Replicating Sequences
- EXCEPT Proposal Threads
  - Support EXCEPT for TABLES IN SCHEMA publications
  - Support EXCEPT for ALL SEQUENCES publications
- Blogs
  - Excluding tables from publications in PostgreSQL 19
  - Upgrade workflows with sequence synchronization

© Fujitsu 2026

The end



© Fujitsu 2026

FUJITSU

Thank you!

Ajin Cherian · Peter Smith

© Fujitsu 2026