Supporting a New PostgreSQL Version in Your Extension – A Citus Case Study

Naisila Puka

Software Engineer @ Microsoft

Outline of this talk

- Title keywords intro:
  - Supporting a **New PostgreSQL Version** in Your **Extension** – A **Citus** Case Study

- PG release timeline adventures

- Steps on supporting a new PG version
  - Successful compilation
  - Extension logic sanity
  - PGXX New features integration

PostgreSQL major releases

- A major version every year

- New features, improvements, and bug fixes

- Following a well-defined release schedule

Postgres: Designed to be easily extensible

PG extensions:

- Add custom functionality

- Enhance database capabilities

- Optimize performance

- Overall idea: make the database more adaptable to **specific requirements or use cases.**
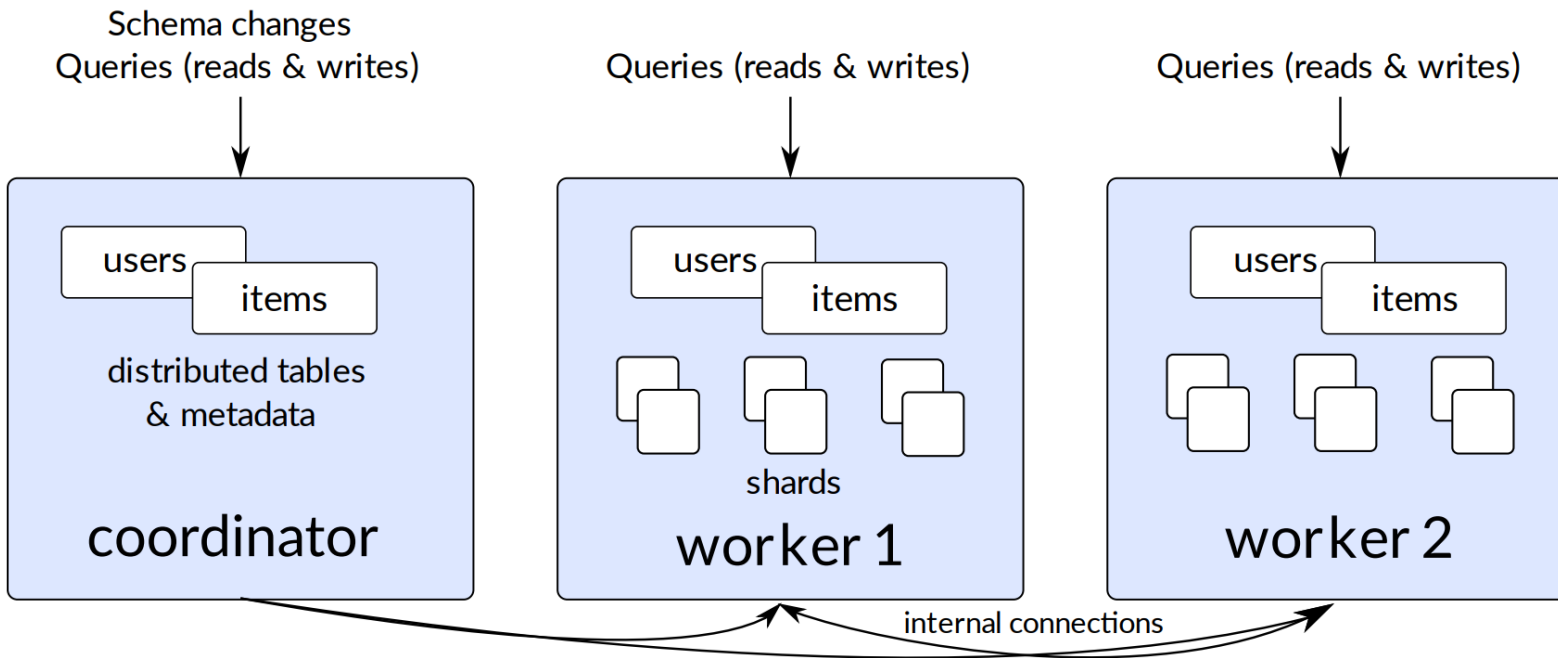
Extensions using PostgreSQL hooks

- Customize PG at various execution points

- Predefined entry points in the server's code where additional functionality can be inserted **without modifying the core source code**

- E.g. hooks at query planning, execution, transaction management

Citus: Extension leveraging PG hooks

Distributed PostgreSQL as an Extension

- Adds the ability to distribute and replicate PostgreSQL tables across a shared-nothing PostgreSQL cluster

- Open-source repo on GitHub: https://github.com/citusdata/citus

- Citus on Azure - Cosmos DB for PostgreSQL

- Citus Utility hook and Columnar Utility hook before Postgres's standard utility process

- Intercepts PostgreSQL's planner, parser and executor

Supporting a New PostgreSQL Version in Your Extension - A Citus Case Study

Questions for your extension:
Is it compatible with the new PG release?

- Does the extension compile successfully?

- How's the test suite doing?

- Do new PG features just work when your extension is installed?

- Do you still want to support earlier PG versions?

- Is your extension still relevant?

- Do you need a strategy here or will a Pull Request fix everything?

Answers for Citus Pre-PG16 support

- Does the extension compile successfully? No

- How's the test suite doing? Some tests fail/crash

- Do new PG features just work when your extension is installed? Most do, some don't

- Do you still want to support earlier PG versions? Yes, 3

- Is your extension still relevant? Yeah, but need to distribute new SQL commands as well to stay coherent

- Do you need a strategy here or will a Pull Request fix everything? A STRATEGY FOR SURE

Supporting a New PostgreSQL Version in Your Extension - A Citus Case Study

Strategy

General: Follow the PG release schedule

1. **Successful compilation**

2. **Extension sanity**
   - **Make sure everything works as before – use your test suite!**

3. PG new features integration
   - **Enhance your extension with PG's newly added features/SQL changes**

Follow the PG release schedule

- Postgres is open-source

- You can follow the commits going into the new release REL_XX_STABLE branch

- Build and run tests regularly to identify potential issues with new PG commits early-on

- Make use of Beta releases and release candidates

Example – PG15 Timeline (2022)

15 Beta 1 – May 19th : SQL/JSON feature
15 Beta 2 – June 30th : improvements to SQL/JSON feature
15 Beta 3 – August 11th
15 Beta 4 – September 8th : SQL/JSON feature is reverted, Citus builds are broken
15 RC 1 – September 29th : message wording change, Citus tests are broken
15 RC 2 – October 6th
15.0 – October 13th : Introduces new function, builds are ok, tests are broken

- SQL/JSON features proposed in beta1 removed as of beta4 => We also reverted our commits for these features.

- Function name conflict ReplicationSlotName after RC2 => renamed to ReplicationSlotNameForNodeAndOwner.

Example – PG16 Timeline (2023)

16 Beta 1 – May 25th
16 Beta 2 – June 29th
Merge compilation changes

16 Beta 3 – August 10th
Merge regression tests sanity changes

16 RC 1 – August 21st
16.0 – September 14th

PG16 compatibility: Resolve compilation issues (#7005)

naisila and **onderkalaci** authored on Jul 21, 2023 (Verified)

This PR provides successful compilation against PG16Beta2. It does some
necessary refactoring to prepare for full support of version 16, in
#6952 .

Adds PG16Beta3 support (#6952)

naisila authored on Aug 17, 2023 (Verified)

DESCRIPTION: Adds PG16Beta3 support

This is the final commit that adds
PG16 compatibility with Citus's current features.

1 - Successful compilation

Update CONFIGURE script to include PGXX

Some variables no longer exist / have been replaced, PG15 e.g:

Value node struct has been removed, replaced by separate Integer, Float, String, and BitString node types

Functions/Objects/Variables/Properties added/changed, PG15 e.g:

```
#if PG_VERSION_NUM >= PG_VERSION_15
#define RelationCreateStorage_compat(RelFileNode rnode, char relpersistence, bool register_delete)
                RelationCreateStorage(rnode, relpersistence, register_delete)
#else
#define RelationCreateStorage_compat(RelFileNode rnode, char relpersistence, bool register_delete)
                RelationCreateStorage(rnode, relpersistence)
```

2 - Extension sanity

- Successful compilation is NOT enough.

- Should update INTERNAL LOGIC accordingly to make sure current features function properly.

Citus Planner Hook Example with PG16

```c
/******************************************************************
 *
 *      Query optimizer entry point
 *
 * To support loadable plugins that monitor or modify planner behavior,
 * we provide a hook variable that lets a plugin get control before and
 * after the standard planning process.  The plugin would normally call
 * standard_planner().
 *
 * Note to plugin authors: standard_planner() scribbles on its Query input,
 * so you'd better copy that data structure if you want to plan more than once.
 *
 ******************************************************************/
PlannedStmt *
planner(Query *parse, const char *query_string, int cursorOptions,
        ParamListInfo boundParams)
{
    PlannedStmt *result;

    if (planner_hook)
        result = (*planner_hook) (parse, query_string, cursorOptions, boundParams);
    else
        result = standard_planner(parse, query_string, cursorOptions, boundParams);
    return result;
}
```

Supporting a New PostgreSQL Version in Your Extension - A Citus Case Study

Citus Technical Documentation

Supporting a New PostgreSQL Version in Your Extension
- A Citus Case Study

planner_hook = distributed_planner

Distributed Query Planner

The distributed query planner is entered through the `distributed_planner` function in `distributed_planner.c`. This is the hook that Postgres calls instead of `standard_planner`.

If the input query is trivial (e.g., no joins, no subqueries/ctes, single table and single shard), we create a very simple `PlannedStmt`. If the query is not trivial, call `standard_planner` to build a `PlannedStmt`. For queries containing a distributed table or reference table, we then proceed with distributed planning, which overwrites the `planTree` in the `PlannedStmt`.

Distributed planning (`CreateDistributedPlan`) tries several different methods to plan the query:

1. Fast-path router planner, proceed if the query prunes down to a single shard of a single table
2. Router planner, proceed if the query prunes down to a single set of co-located shards
3. Modification planning, proceed if the query is a DML command and all joins are co-located
4. Recursive planning, find CTEs and subqueries that cannot be pushed down and go back to 1
5. Logical planner, constructs a multi-relational algebra tree to find a distributed execution plan

Supporting a New PostgreSQL Version in Your Extension - A Citus Case Study

planner_hook = distributed_planner

Distributed Query Planner

The distributed query planner is entered through the `distributed_planner` function in `distributed_planner.c`. This is the hook that Postgres calls instead of `standard_planner`.

If the input query is trivial (e.g., no joins, no subqueries/ctes, single table and single shard), we create a very simple `PlannedStmt`. If the query is not trivial, call `standard_planner` to build a `PlannedStmt`. For queries containing a distributed table or reference table, we then proceed with distributed planning, which overwrites the `planTree` in the `PlannedStmt`.

Distributed planning (`CreateDistributedPlan`) tries several different methods to plan the query:

1. Fast-path router planner, proceed if the query prunes down to a single shard of a single table
2. Router planner, proceed if the query prunes down to a single set of co-located shards
3. Modification planning, proceed if the query is a DML command and all joins are co-located
4. Recursive planning, find CTEs and subqueries that cannot be pushed down and go back to 1
5. Logical planner, constructs a multi-relational algebra tree to find a distributed execution plan

Supporting a New PostgreSQL Version in Your Extension - A Citus Case Study

Fast Path Planner skips cost estimation prior to query distribution

Fast Path Router Planner

The Fast Path Router Planner is specialized in optimizing queries that are both simple in structure and certain to touch a single shard. Importantly, it targets queries on a single shard distributed, citus local or reference tables. This does not mean the planner is restricted to trivial queries; it can handle complex SQL constructs like `GROUP BY`, `HAVING`, `DISTINCT`, etc., as long as these operate on a single table and involve an equality condition on the distribution key (`distribution_key = X`). The main SQL limitation for fast path distributed query planning is the subquery/CTE support. Those are left to the next planner: Router planner.

The aim of this planner is to avoid relying on PostgreSQL's standard_planner() for planning, which performs unnecessary computations like cost estimation, irrelevant for distributed planning. Skipping the standard_planner has significant performance gains for OLTP workloads. By focusing on "shard-reachable" queries, the Fast Path Router Planner is able to bypass the need for more computationally expensive planning processes, thereby accelerating query execution.

Supporting a New PostgreSQL Version in Your Extension - A Citus Case Study

PG16 commit that broke the planner hook

Rework query relation permission checking

alvherre committed on Dec 6, 2022 (Verified)

Currently, information about the permissions to be checked on relations
mentioned in a query is stored in their range table entries. So the
executor must scan the entire range table looking for relations that
need to have permissions checked. This can make the permission checking
part of the executor initialization needlessly expensive when many
inheritance children are present in the range range. While the
permissions need not be checked on the individual child relations, the
executor still must visit every range table entry to filter them out.

This commit moves the permission checking information out of the range
table entries into a new plan node called RTEPermissionInfo. Every
top-level (inheritance "root") RTE_RELATION entry in the range table
gets one and a list of those is maintained alongside the range table.
This new list is initialized by the parser when initializing the range
table. The rewriter can add more entries to it as rules/views are
expanded. Finally, the planner combines the lists of the individual
subqueries into one flat list that is passed to the executor for
checking.

To make it quick to find the RTEPermissionInfo entry belonging to a
given relation, RangeTblEntry gets a new Index field 'perminfoindex'
that stores the corresponding RTEPermissionInfo's index in the query's
list of the latter.

ExecutorCheckPerms_hook has gained another List * argument; the
signature is now:
typedef bool (*ExecutorCheckPerms_hook_type) (List *rangeTable,
                                              List *rtePermInfos,
                                              bool ereport_on_violation);
The first argument is no longer used by any in-core uses of the hook,
but we leave it in place because there may be other implementations that
do. Implementations should likely scan the rtePermInfos list to
determine which operations to allow or deny.

Author: Amit Langote <amitlangote09@gmail.com>
Discussion: https://postgr.es/m/CA+HiwqGjJDmUhDSfv-U2qhKJjt9ST7Xh9JXC_irsA

Currently, information about the permissions to be checked on relations
mentioned in a query is stored in their range table entries. So the
executor must scan the entire range table looking for relations that
need to have permissions checked. This can make the permission checking
part of the executor initialization needlessly expensive when many
inheritance children are present in the range range. While the
permissions need not be checked on the individual child relations, the
executor still must visit every range table entry to filter them out.

table. The rewriter can add more entries to it as rules/views are
expanded. Finally, the planner combines the lists of the individual
subqueries into one flat list that is passed to the executor for
checking.

To make it quick to find the RTEPermissionInfo entry belonging to a
given relation, RangeTblEntry gets a new Index field 'perminfoindex'
that stores the corresponding RTEPermissionInfo's index in the query's
list of the latter.

New entry in PlannedStmt struct

New entry in RangeTblEntry struct

Supporting a New PostgreSQL Version in Your Extension
- A Citus Case Study

INSERT failure in a distributed table

```
CREATE TABLE test_table (id int primary key, name text);

INSERT INTO test_table VALUES (1, 'beana');

SELECT create_distributed_table('test_table', 'y'); ## Citus signature

INSERT INTO test_table VALUES (2, 'erida');

ERROR:  invalid perminfoindex 1 in RTE with relid 25395


What happened?
test_table has the perminfoindex entry as 1
A PlannedStmt struct somewhere missing permInfos list
```

Supporting a New PostgreSQL Version in Your Extension
- A Citus Case Study

One-line fix in the fast path planner

Supporting a New PostgreSQL Version in Your Extension
- A Citus Case Study

Process of committing into the Citus repo

1. Find the relevant PG commit breaking the current logic

   [Rework query relation permission checking · postgres/postgres@a61b1f7 (github.com)](github.com)

2. Fix the logic in Citus and put a reference to the PG commit in the commit description

   [PG16 compatibility - Rework PlannedStmt and Query's Permission Info (... · citusdata/citus@b36c431 (github.com)](github.com)

   ```
   descr: This commit is in the series of PG16 compatibility commits.
   It handles the Permission Info changes in PG16. See below:
   …
   We had crashes because perminfoindexes were not updated in the finalized
   planned statement after distributed planner hook.
   So, basically, everywhere we set a query's or planned statement's rtable
   entry, we need to set the rteperminfos/permInfos accordingly.
   Relevant PG commit:
   a61b1f74823c9c4f79c95226a461f1e7a367764b
   ```

2 - Extension sanity

- Q: How to find all the broken pieces?
  A: Hopefully you have a nice and thorough test suite

- Q: How to keep track of everything?
  A: GitHub issues are a nice way

This issue encapsulates what remains to be fixed in #6952
Check the tests in CI in the linked PR in order to see the exact error which I point to with "seen in" labels.

test-16_check-multi 12/175 failing
test-16_check-multi-1 9/210 failing
test-16_check-columnar 3/42 failing
test-16_check-isolation 2/93 failing
test-16_check-mx 2/68 failing
test-16_check-operations 1/16 failing
test-16_check-vanilla 4/215 failing

☑ Max mem usage changed and it's no longer less than 8MB, seen in `columnar_memory`
  It's still less than 9MB. TopMemoryContext simply has more children.
☑ Fix `ERROR:  duplicate key value violates unique constraint` in `columnar_write_concurrency_index`
☑ Extra `DEBUG:  pathlist hook for columnar table am`, seen in `drop_column_partitioned_table`
☑ Not getting expected `ERROR:  correlated subqueries are not supported when the FROM clause contains a CTE or subquery`, seen in `multi_subquery_in_where_reference_clause`
☑ Fix `ERROR:  relation "rule_table_1" cannot have ON SELECT rules`, seen in `undistribute_table`
  this fails on plain PG16 whereas it succeeds in plain PG15, so not related to Citus
☑ Fix `ERROR:  "rule_table_1" is not a view`, seen in `undistribute_table`
  this fails on plain PG16 whereas it succeeds in plain PG15, so not related to Citus
☑ Not getting expected `ERROR:  cannot alter table because an extension depends on it`, seen in `undistribute_table`
☑ Different `pa_compare_tables` output, seen in `merge`
☑ Fix `ERROR:  EXPLAIN ANALYZE is currently not supported for MERGE INTO ... commands with repartitioning`, seen in `merge`

Supporting a New PostgreSQL Version in Your Extension
- A Citus Case Study

2 - Extension sanity

If possible, in terms of engineering resources,
you can be *even more proactive*

Build and run tests of your extension regularly with
REL_XX_STABLE branch of PostgreSQL

1. Fix build issues instantly,

2. Fix test issues instantly, or document them for later

3 - PG new features integration

Resources to track PostgreSQL XX's improvements/additions.

- Official release notes

- Feature matrix (which features added in which version)

- "Waiting for PGXX" blog www.depesz.com

- pgPedia notes

3 - PG new features integration

Two types of improvements/features

1. Simply work with your extension

2. Need development in your extension

3 - PG new features integration

Stuff that just work with Citus (the majority do!):

- Shards are regular Postgres tables, and queries are sent to shards as regular SQL commands. Any improvement on these are reflected on distributed tables such as performance, index/constraint improvements etc.

- Citus does not interfere with replication, checkpointing, vacuum, logging, monitoring, psql, fdw, contrib modules and many other things. Any improvement on these areas is also reflected.

Supporting a New PostgreSQL Version in Your Extension - A Citus Case Study

3 - PG new features integration

**Stuff that need development/testing to work with Citus:**

When the SQL interface changes, Citus needs to learn how to send it properly to the worker nodes.

- Syntax on a command is expanded

- New command is introduced

- New functions /data types added

Decide what to do based on your resources!

1. Extend the codebase to support new stuff

2. Extend testing for new features that work with your extension, but might need maintenance for future changes

3. Print meaningful error messages for unsupported stuff

In Citus, we tracked these with yet another GitHub issue
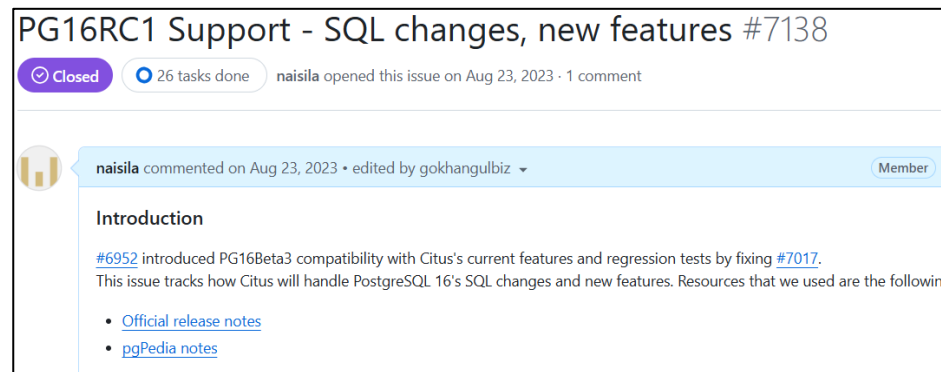
Examples with PG16

1. **Extend the codebase to support new stuff**

Features to support in Citus

We plan to support the following new options:

☑ CREATE COLLATION (rules = …) postgres/postgres@ `30a53b7`
  ⌥ **PG16 - Add rules option to CREATE COLLATION** #7185

☑ TRUNCATE triggers on foreign tables postgres/postgres@ `3b00a94`
  ⌥ **PG16 - Add citus_truncate_trigger for Citus foreign tables** #7170

☑ GENERIC_PLAN option for EXPLAIN postgres/postgres@ `3c05284`
  ⌥ **PG16 - Add GENERIC_PLAN option to EXPLAIN** #7141

☑ VACUUM SKIP_DATABASE_STATS, ONLY_DATABASE_STATS postgres/postgres@ `a46a701`
  ⌥ **PG16 compatibility - new options to vacuum and analyze** #7114

☑ VACUUM PROCESS_MAIN postgres/postgres@ `4211fbd`
  ⌥ **PG16 compatibility - new options to vacuum and analyze** #7114

☑ VACUUM/ANALYZE BUFFER_USAGE_LIMIT postgres/postgres@ `1cbbee0`
  ⌥ **PG16 compatibility - new options to vacuum and analyze** #7114

Supporting a New PostgreSQL Version in Your Extension - A Citus Case Study

Examples with PG16

2. **Extend testing for new features that work with your extension, but might need maintenance for future changes**

Supporting a New PostgreSQL Version in Your Extension - A Citus Case Study

Examples with PG16

3. **Print meaningful error messages for unsupported stuff**

Meaningful error messages for currently unsupported features

For now, we will not provide support for the following, but we will print error messages with possible hints/workarounds for the user:

☑ GRANT … WITH INHERIT postgres/postgres@ e3ce2de
  ⌥ PG16 - Don't propagate GRANT ROLE with INHERIT/SET option #7190

☑ GRANT … WITH SET postgres/postgres@ 3d14e17
  ⌥ PG16 - Don't propagate GRANT ROLE with INHERIT/SET option #7190

☑ Batch insertion during COPY into a foreign table postgres/postgres@ 97da482
  COPY FROM is already not supported for Citus foreign tables
  ⌥ Adds test for COPY FROM failure in Citus foreign tables #7160

☑ ALTER TABLE … SET STORAGE DEFAULT postgres/postgres@ b9424d0
  Already changing storage is not supported and errors out, adding some tests
  ⌥ Add tests for CREATE/ALTER TABLE .. STORAGE in PG16 #7140

☑ CREATE STATISTICS without a user-specified name postgres/postgres@ 624aa2a
  ⌥ PG16 - Throw meaningful error for stats without a name on Citus tables #7136

PG17 progress on Citus

- Successful compilation changes are merged.

- Extension sanity is in progress
  - [PG17Beta2 Support - Regression tests sanity · Issue #7653 · citusdata/citus (github.com)](github.com)

- Implementing new features – not started yet
  - Set to track in [PG17.0 Support – SQL changes, new features · Issue #7708 · citusdata/citus (github.com)](github.com)

- Expected to add PG17 support on Citus by the end of 2024

Revisiting Strategy

General: Follow the PG release schedule

1. **Successful compilation**

2. **Extension sanity**
   - **Make sure everything works as before – use your test suite!**

3. **PG new features integration**
   - **Enhance your extension with PG's newly added features/SQL changes**
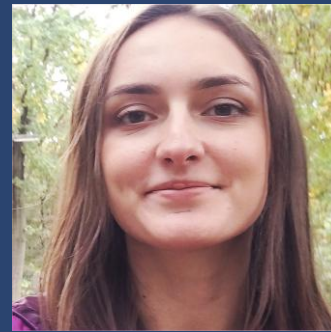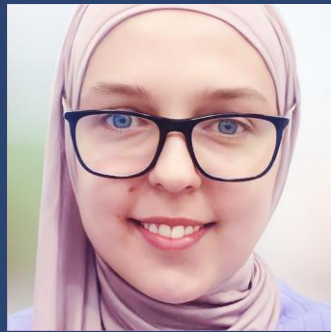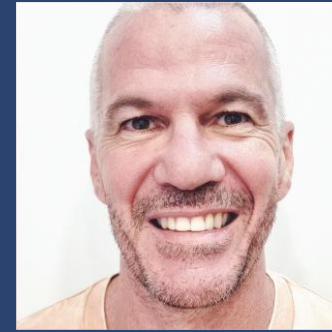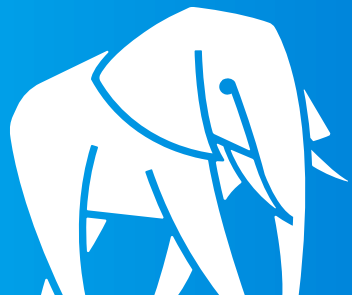
Thank you for your attention!

Q&A

Feedback QR:

Supporting a New PostgreSQL Version in Your Extension
- A Citus Case Study

Microsoft

Meet our
Postgres
team at
PGConf EU
2024

Have you listened to TalkingPostgres.com?

TALKING POSTGRES

WITH CLAIRE GIORDANO

Save the date
June 10-12, 2025

POSETTE:
An Event for Postgres

2025 Now in it's 4th year!

A free & virtual developer event

Subscribe to news → aka.ms/posette-subscribe