Lessons Learned

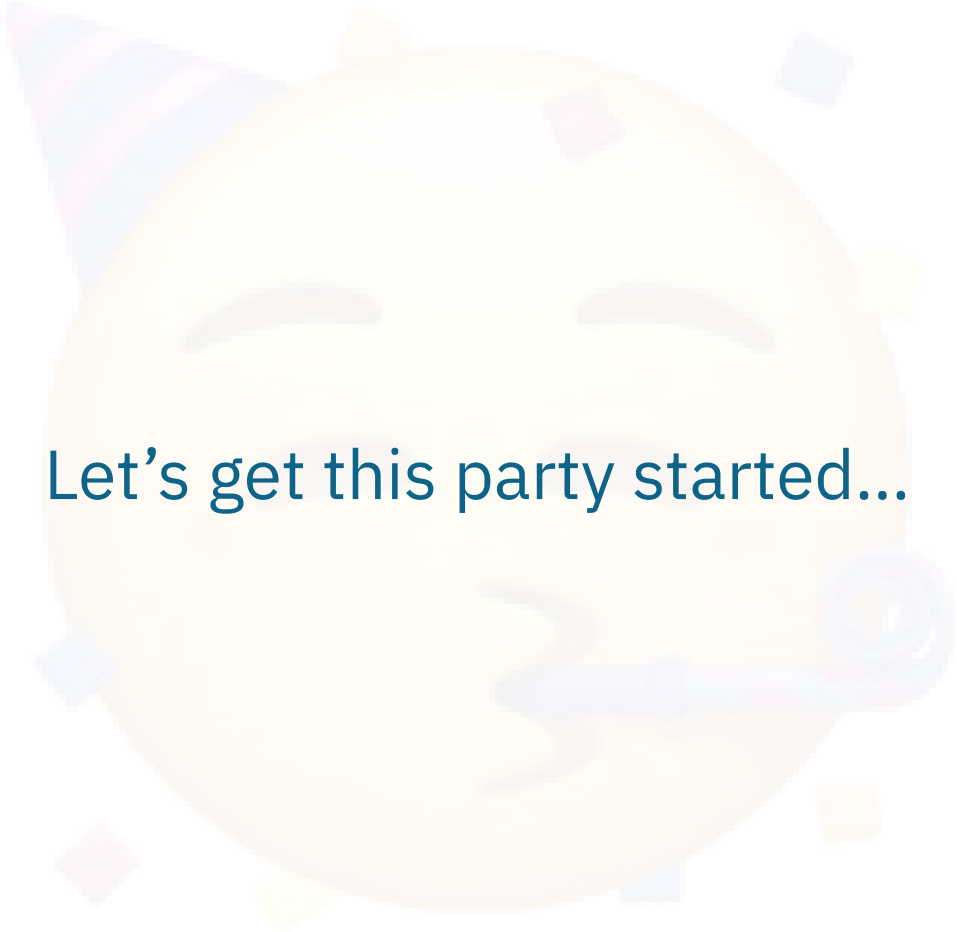
Extending Postgres To Its Limits

Jonah H. Harris, Chief Technology Officer
Mason Sharp, Vice President of Server Technologies
Curt Kolovson, Sr. Principal Software Engineer

PGConf NYC 2023

Introduction

- Why am I here?
- What's this all about?
- What am I getting out of this?



Let’s get this party started...

MariaDB

Wait a minute! MariaDB is a fork of MySQL...

What the #!@$ are you doing at a Postgres conference?!? 🤔

CASEY CONFERENCE

Random PGConf NYC 2023 Attendee

MariaDB

About Us

- Jonah Harris
  - CTO @ MariaDB plc
  - Longtime PostgreSQL user (1999), community contributor (early 2000s), and professional internals developer (2005+)
- Curt Kolovson
  - Member of original POSTGRES project @ UC Berkeley
  - HP, VMWare, and MariaDB
- Mason Sharp (in Finland, but here in spirit)
  - ExtenDB, EnterpriseDB, Stado, StormDB, PG-XC, Postgres-XL, Immuta, ...

A Brief History of Extensibility

- Stonebraker
- Extensible for research
  - Ability to swap out different algorithms and components of the system.
  - Object-relational data model
    - User-defined types, operator, access methods, casts, etc.
- Result of this groundbreaking work…
  - Illustra
  - Postgres95 -> PostgreSQL
  - Today (Example pgvector)
    - User-defined Type
    - User-defined Operators
    - User-defined Indices (IVFFLAT, HNSW)

Extensibility Goals

- Features & Functionality
  - Custom Access & Indexing Methods
  - Custom Data Types & Operators
  - Federation
  - Procedural Languages
- Industry Solutions
  - Compatibility
  - Distributed/Clustered
  - Geospatial
  - Graph
  - HTAP
  - OLAP
  - Performance
  - Time Series
- Why Extensibility Was Necessary

APIs, Hooks, and Extensions

- APIs
  - Standardized interfaces which allow developers to craft extensions that enhance and broaden Postgres' core capabilities.
    - Foreign Data Wrappers (FDW), Table/Index Access Methods, Background Workers, Types, Operators, …
- Hooks
  - Internal interfaces which enables developers to modify, replace, or inject custom behaviors into Postgres' standard operations.
    - ProcessUtility_hook, post_parse_analyze_hook, planner_hook, …
- Extensions
  - A modular framework which enables developers to add bespoke features and integrations, enriching the core database functionality.
    - PostGIS, pg_stat_statements, pgcrypto, hstore, earthdistance, …

Challenges Encountered

- Types of Challenges Faced
  - Missing APIs
  - Missing Hooks
    - Protocol
    - Parsing
      - Lack of access to raw parse tree
      - Hooks into specific areas (pre-parse-analyze, ...)
- Specific PostgreSQL Limitations
  - Multi-process/multi-threaded
- Impact on Goals
  - Still have to fork pg to add specific APIs/hooks
    - Babelfish
    - OrioleDB
    - ...

Innovative Implementations

- Implemented Solutions
  - Why were they created?
  - How were they created?
  - What challenges did they encounter?
  - What was their resulting effectiveness/results?

Innovative Implementations

Columnar Database

- Hydra (github.com/hydradatabase/hydra)
  - Problem
    - How do we create a solid columnar implementation for Postgres?
  - Solution
    - Extension forked from the last version of Citus' FDW-based columnar implementation
    - Rewritten to use the table access manager interface
    - Uses custom scan nodes
    - Uses hooks for object access, utility statement handling, and planner
    - UDFs to perform specific tasks
  - Challenges
    - More challenges with original cstore_fdw
  - Effectiveness & Results
    - Have you seen this thing!?! It's amazing!

Innovative Implementations

Columnar Database

- Alternatives
  - Greenplum (Apache)
  - Citus (AGPL)
  - Original cstore_fdw (Apache)

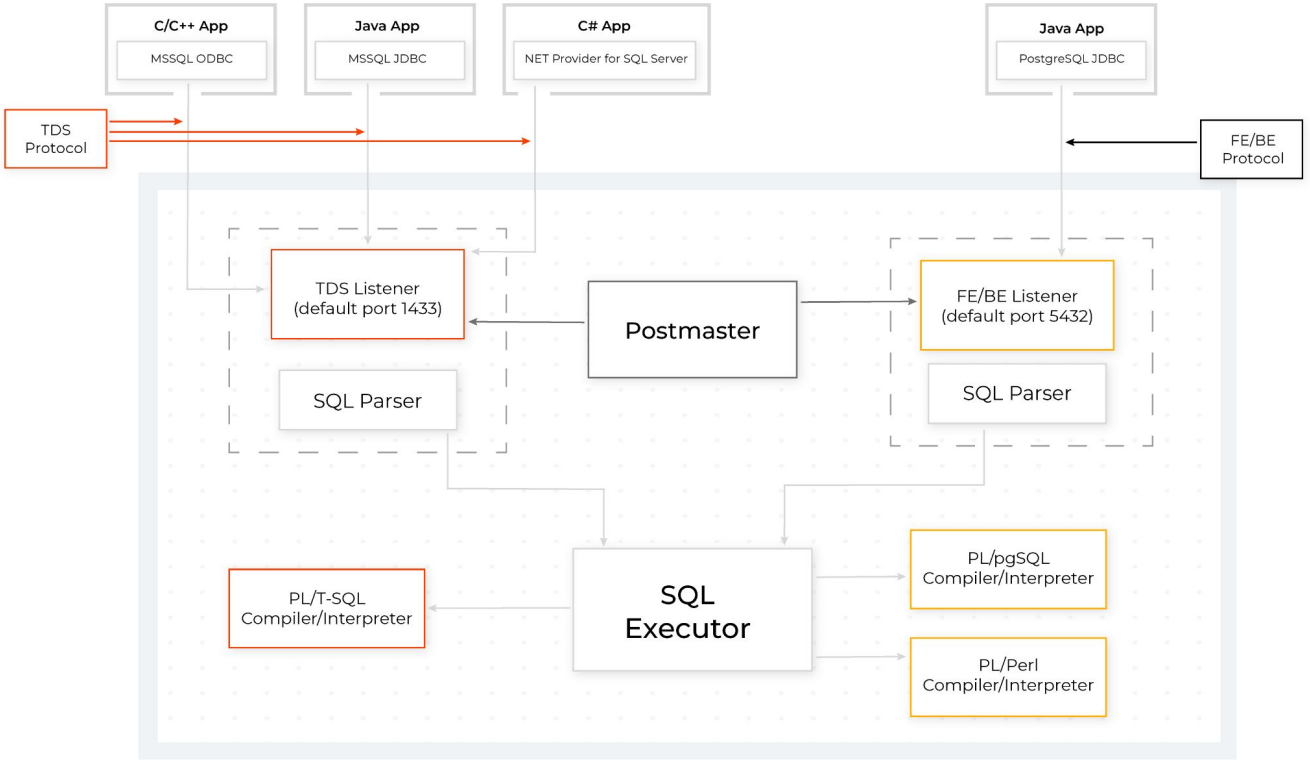
Innovative Implementations

Database Compatibility

- AWS Babelfish (github.com/babelfish-for-postgresql)
  - Problem
    - How do we provide customers a simple migration path from SQL Server to Postgres?
  - Solution
    - Extension (with a forked Postgres for added hooks)
    - TDS Protocol Plugin
    - TransactSQL Implementation
    - Additional Data Type
  - Challenges
    - Missing hooks…
    - Using the PL as the base execution engine
    - Single vs. Multi-database access

Innovative Implementations

Database Compatibility (Continued)



Innovative Implementations

Database Compatibility (Continued)

- Effectiveness & Results
  - I can’t speak for AWS, but IMNSHO…
- Alternatives
  - EnterpriseDB
  - IvorySQL
  - NEXTGRES

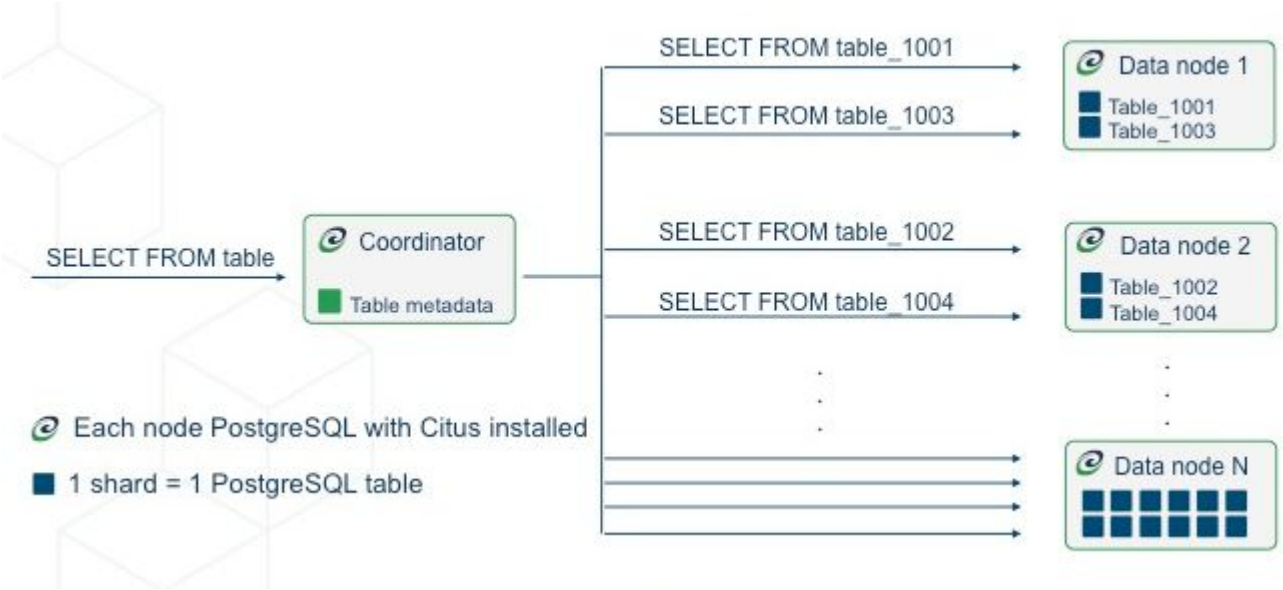
Innovative Implementations

Distributed Database

- Citus (github.com/citusdata/citus/tree/main)
  - Problem
    - How do we distribute load across multiple nodes?
  - Solution
    - Extension
    - Background Workers
    - Uses hooks for object access, utility statement handling, and planner
  - Challenges
    - Synchronization across nodes
    - Distributed Transactions
    - Data Locality Optimizations

Innovative Implementations

Distributed Database (Continued)



MariaDB

Innovative Implementations

Distributed Database (Continued)

- Effectiveness & Results
  - Successful
  - Well-designed

- Alternatives
  - EnterpriseDB’s BDR
  - Huawei’s GaussDB/openGauss (forked from Postgres-XC)
  - pgEdge (not exactly distributed, but similar)
  - Tencent’s TBase (forked from Postgres-XL)
  - yugabyteDB

There’s a lot we haven’t mentioned...

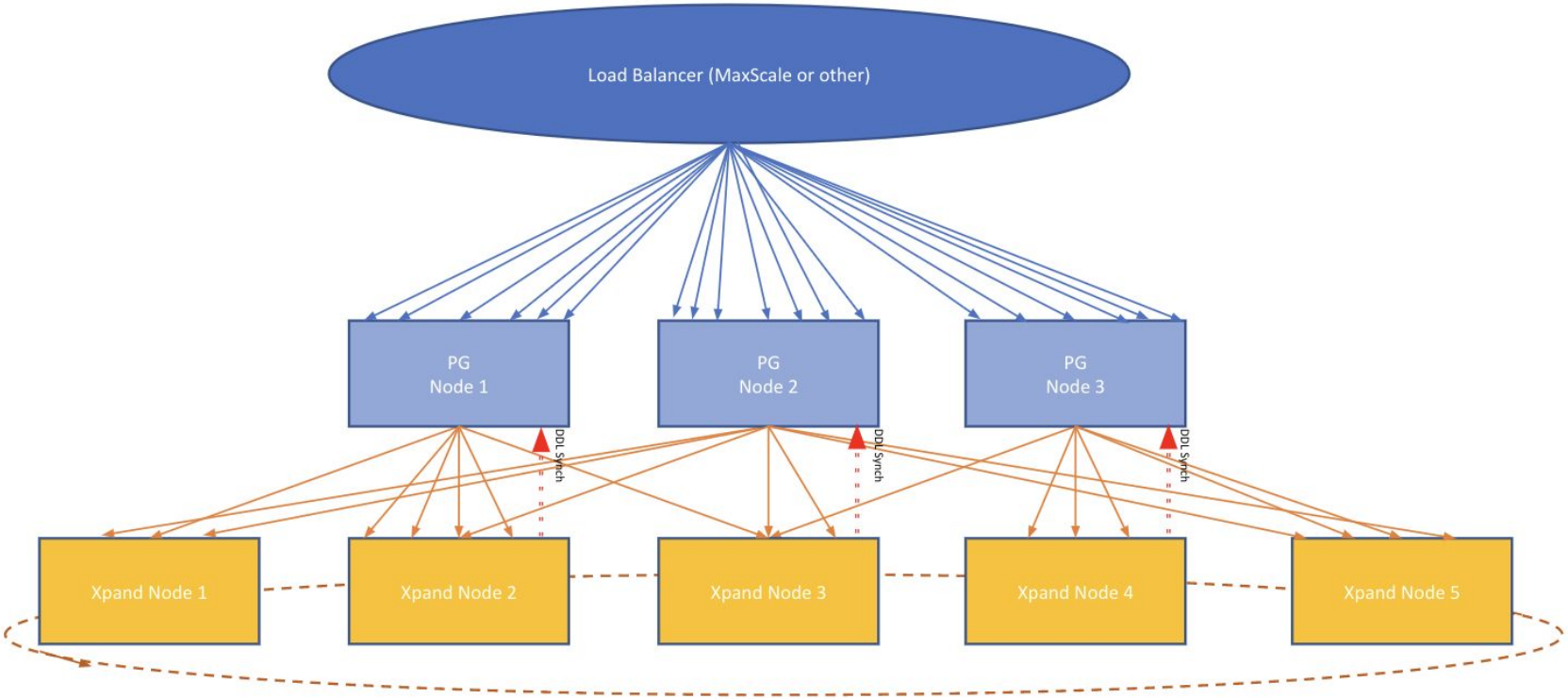
Case Study #1

MariaDB PostgreSQL with Xpand (aka Xgres)

- Problem
  - We have a great distributed SQL database (Xpand), but a problem with adoption.
    - Prior CEO recognized the obvious benefits to being in the Postgres market.
    - Wanted to offer a version of Postgres with better HA and scalability.
- Solutions Explored
  - “The Plugin” (which shall not be named)
  - Google Spanner/MaxScale Protocol Proxy (passthrough SQL)
- Approach (Really needed Postgres as base)
  - Extension
    - Hybrid DML based on EnterpriseDB’s mysql_fdw
    - DDL and Simple DML based on database links

Case Study #1 (Continued)

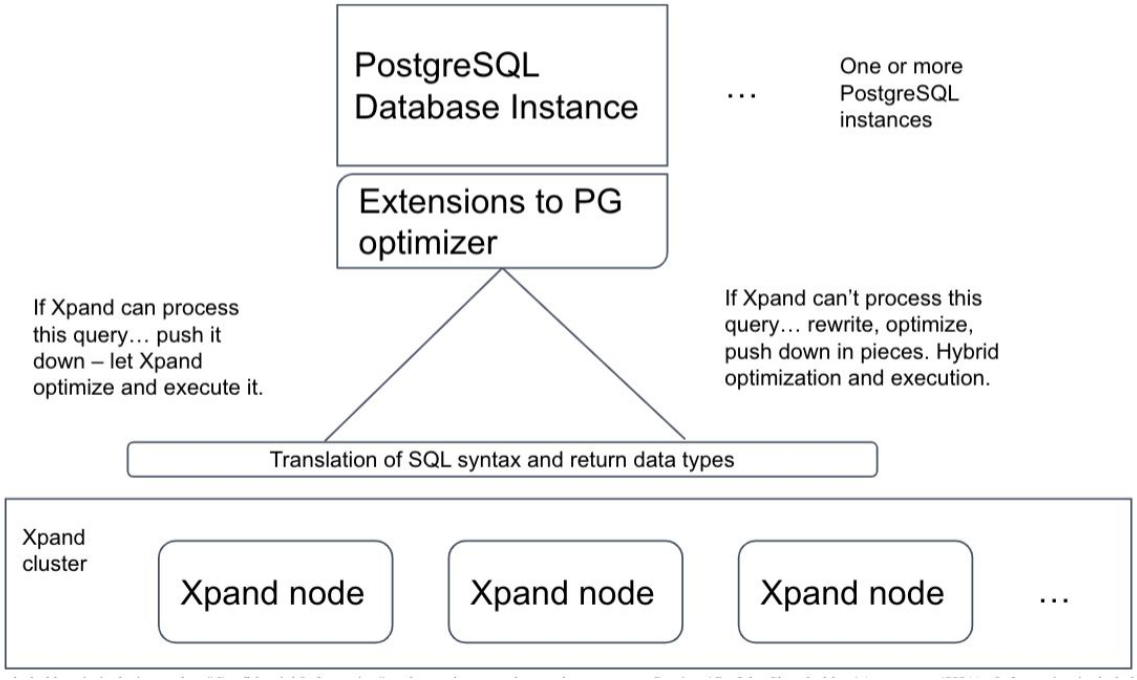
MariaDB PostgreSQL with Xpand (aka Xgres)



MariaDB

Case Study #1 (Continued)

MariaDB PostgreSQL with Xpand (aka Xgres)



MariaDB

Case Study #1 (Continued)

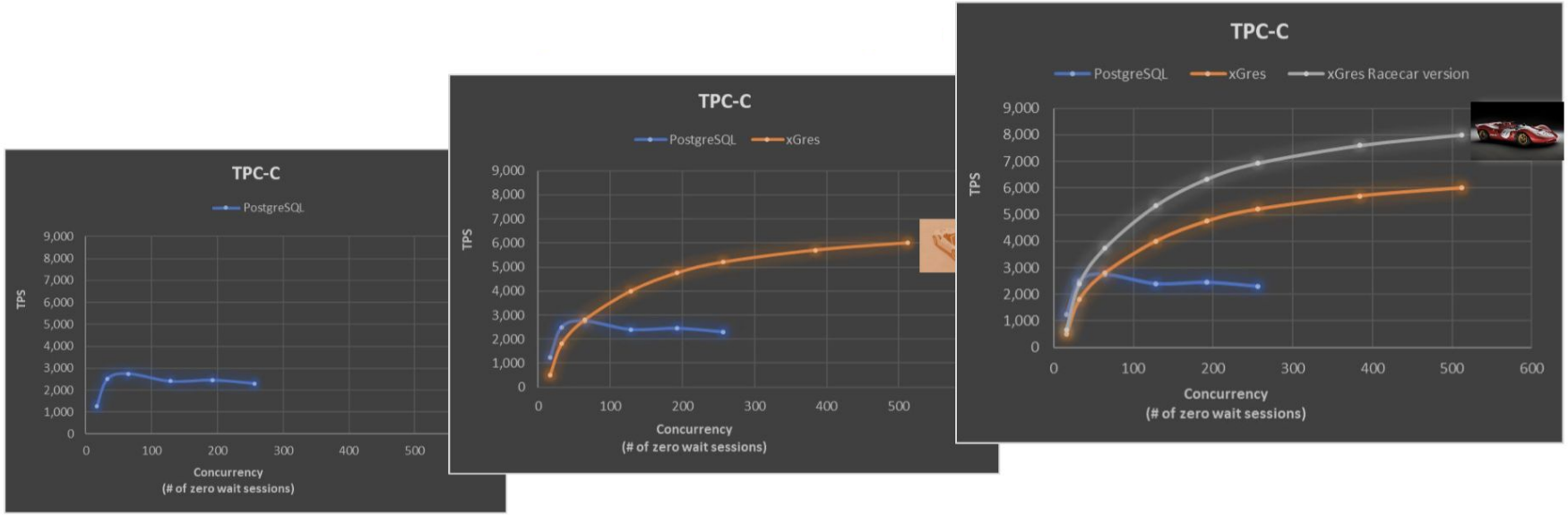
MariaDB PostgreSQL with Xpand (aka Xgres)

- Challenges
  - mysql_fdw issues (fixed bugs, added support for COPY, extended to support more use-cases, etc.)
  - Reparsing (parse tree discarding)
  - Syntactic/Semantic Differences (Xpand-specific SQL generation/Timezones, Data Types, etc.)
  - Statement caching (to reduce round-trips)
  - Implicit COMMIT
  - Keeping multiple stateless Postgres instances in sync.
  - Allocator thrashing (per-statement execution overhead)
  - Deferred constraints

- Results (~85% of pure Xpand)

Case Study #1 (Continued)

MariaDB PostgreSQL with Xpand (aka Xgres)



MariaDB

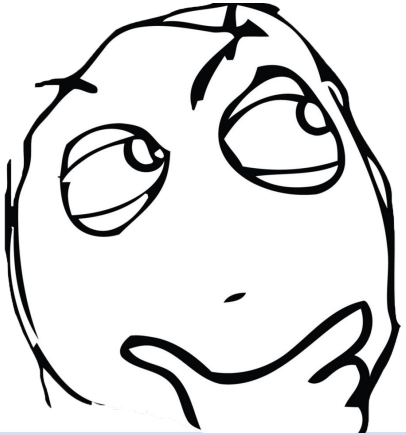
I believe if Michael Stonebraker were to rewrite PostgreSQL today, he would probably follow the same principles as CockroachDB.

He would build a natively distributed, cloud-native, multi-region, multi-cloud solution that is still compatible with PostgreSQL.

DOMENIC RAVITA

Principal Product Evangelist, Cockroach Labs

Sweet... where is it?



Case Study #1 (Continued)

MariaDB PostgreSQL with Xpand (aka Xgres)

- Status
  - Shelved (for now)... *But, don't worry, we're working on something much better...*

- What are we doing with our improvements to mysql_fdw?
  - When we get a chance to separate those back out, we will contribute them back.

Case Study #2

Immuta Query Engine (Proxy)

- Problem
  - How do we provide customers with the following data governance and data access controls across all their enterprise databases?
    - Attribute Based Access Control
    - Access workflows
    - PII auto-detection
    - Privacy controls
      - Row and column masking according to policy
      - Differential Privacy
      - K-anonymity
      - Randomized Response
- Approach
  - Extension
    - Based on FDW API
    - Hooks, hooks, and more hooks...

Case Study #2 (Continued)

Immuta Query Engine (Proxy)

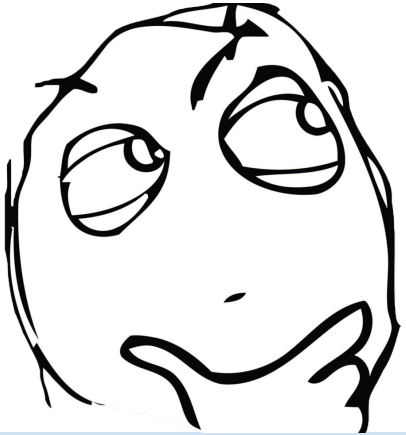
- Challenges
  - Supporting 20+ Database Engines
    - MariaDB, MySQL, Oracle, SQL Server, DB2, Hive, Presto…
  -
  - Intelligent Push-Down
    - IN subqueries, FROM subqueries, UNION, Window functions, complex expressions…
  - Query Rewriting
    - Determine if masking policies may be applied
    - Walk expressions, see if all elements can be pushed down
    - Examine FROM and IN subqueries, see if they can be pushed down, fold into parent query
  - Distributed Federated Joins
    - Standard FDW queries all of the rows and ships them to PostgreSQL for local joining
      - SQL Server table A with 20 M rows being joined with MariaDB table B with 500 rows

Case Study #2 (Continued)

Immuta Query Engine (Proxy)

- Results
  - Single, multi-database-optimized FDW
  - Awesome Distributed Federated Joins
    - Create new execution node type
    - Recognize different DB-type|host|port|database via planner hook
    - do not allow local join pushdown
    - Use new plan node for distributed join
    - Cost-based optimizer, may or may not be chosen, custom costing logic
    - During execution, rows from small side read and incorporated into one or more run time generated SELECTs by having the values appear as a fake table in the FROM clause
  - Optimized Plan Deparsing
    - Examine final UNION plans, fold into a single SELECT if possible

Sweet... where is it?

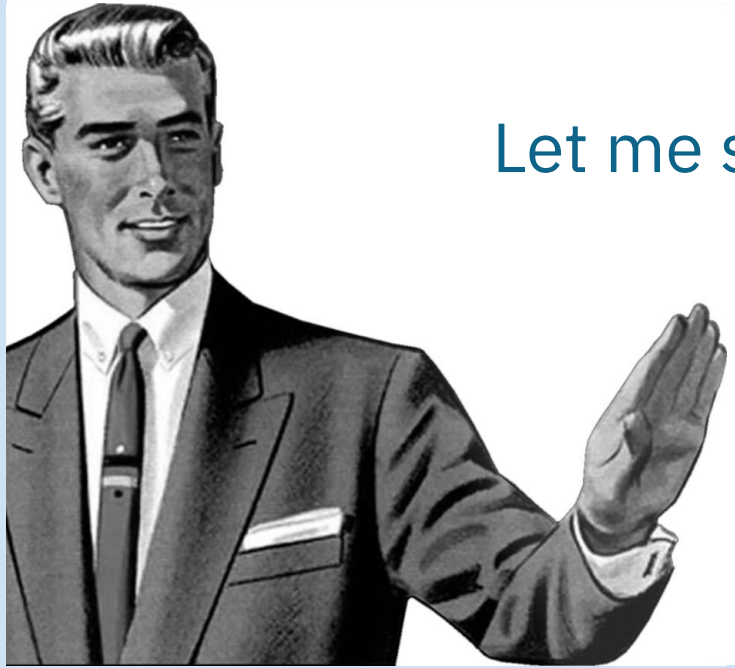


An FDW optimized for both OLTP & OLAP... WIN!

Wow! Postgres is ultra-extensible...

Let’s extend Postgres to do ALL THE THINGS!

MariaDB



Let me stop you right there...

MariaDB

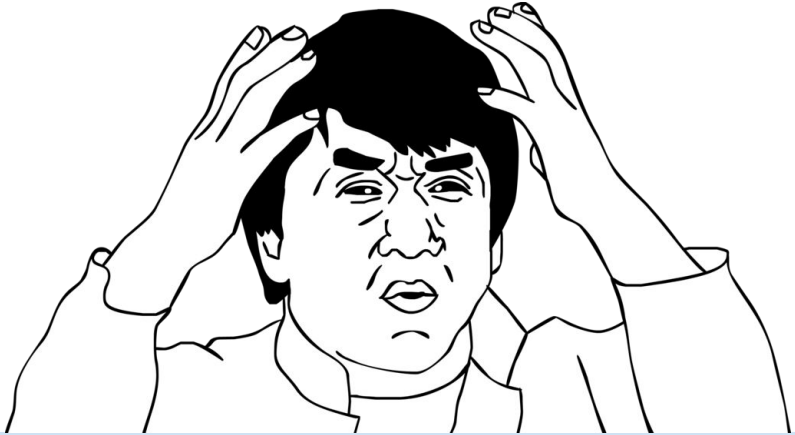
My personal opinion is that Postgres almost reaches its “limit of evolution” or is close to it.

Making some major changes such as multithreading, undo log, columnar store with vector executor requires so much changes and cause so many conflicts with existed code that it will be easier to develop new system from scratch rather than trying to plugin new approach in old architecture.

KONSTANTIN KNIZHNIK

Well-Known Database Internals Engineer & PostgreSQL Contributor, now @ Neon

First, you say it’s ultra-extensible. Then that. What?



Pull to push model, agreed. Otherwise, meh...

The Future of PostgreSQL Extensibility

Our Predictions and Desires

- There will continue to be feature/functionality specific additions like pgvector, but also more exhaustive (e.g. postgresml).
- There will be more holistic extensions created.
- There will continue to be more hooks added.
- There will be more optimized storage engines added.
- Desired Features
  - Additional hooks around the protocol, parsing, and parse analysis.
  - Perhaps custom parse/plan nodes and an API?
- Importance for the Community

How do we make this happen?

My feeling about this is if you want to use Oracle, go use Oracle.

Don't ask PG to take on a ton of maintenance issues so you can have a frankenOracle.

TOM LANE

PostgreSQL Core Team Member

FrankenSQL: The All-The-Things Database

- Responsibility
  - What is the community responsible for?
    - Adding reasonable hooks and APIs.
    - Not worrying about maintaining compatibility with those across versions.
  - What are extension authors responsible for?
    - Everything else.
  - How do we meet in the middle?
    - The usual *lively* discussion on the mailing lists.

Key Takeaways

- Historical Roots
- The Evolution of Extensibility
- Key Features of Postgres Extensibility
- Amazing Extensions
- Lessons Learned
- Future Extensibility

PostgreSQL builds the foundation; we sculpt the skyline.

The PostgreSQL community's depth-over-breadth approach and focus on core strengths are more than mere strategy; they’re the scaffolding of a thriving ecosystem booming with commercial opportunity.

JONAH H. HARRIS

PostgreSQL Contributor & Chief Technology Officer @ MariaDB plc

MariaDB

Thank You

- You!
- The PostgreSQL Community
- Steve Touw @ Immuta for allowing us to share their architecture.


MariaDB