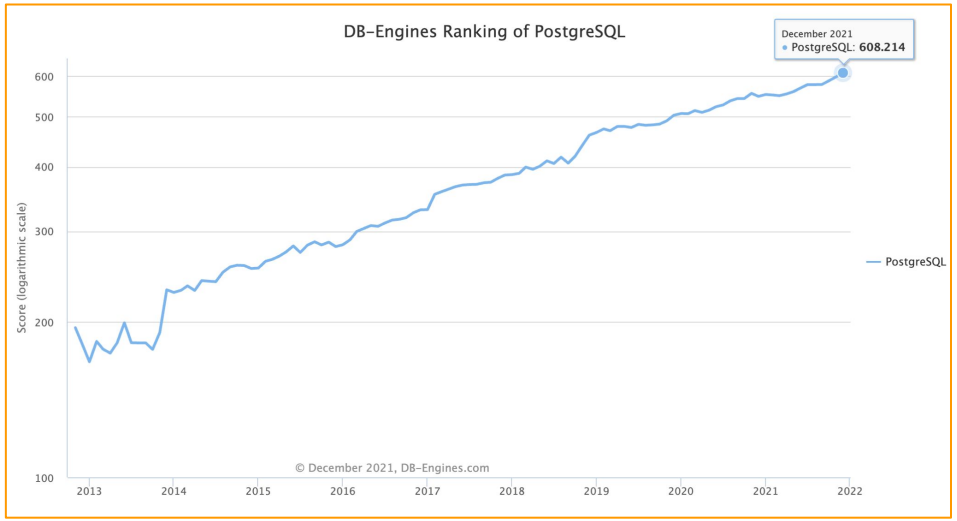
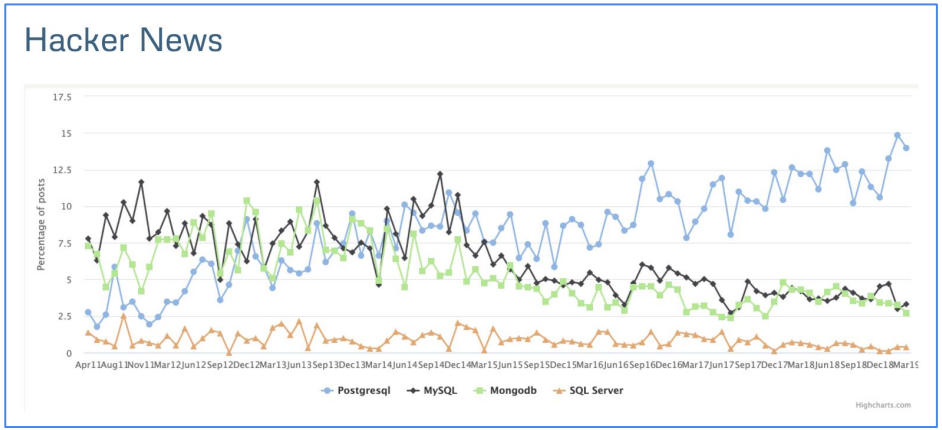
Analyzing PostgreSQL Compatibility: Amazon Aurora, Spanner, YugabyteDB and CockroachDB

Karthik Ranganathan

PostgreSQL is Becoming the Standard API for Operational Databases



PostgreSQL adoption per DB Engines



What Makes PostgreSQL Awesome?

- **Best in class functionality:**
  It offers its users a huge (and growing) set of features.

- **More than a relational database:**
  Supports concepts like in-database functions, user-defined data types, full text search, recursive graph queries, time series support and document queries.

- **Robustness:**
  The speed, security and robustness make it suitable for most applications.

- **Extensible:**
  Extensions, write modules in other programming languages without recompiling.

- **Open source:**
  PostgreSQL is fully open source under a very permissive license, no strings attached.

The PostgreSQL API is Important for New DBs

- **Developer familiarity:**
  Many developers use it, and a compatible API is the easiest way to onboard developers and reduce friction to adoption

- **Comprehensive, robust features:**
  Many critical applications run on PostgreSQL, including ones migrated from Oracle, SQL Server and DB2. This means the feature set is proved to work.

- **Ubiquitous adoption:**
  Many new applications, frameworks and services are built on PostgreSQL, and it's adoption is fast on the rise.

PostgreSQL compatibility is a no-brainer for any new database trying to offer a unique feature set

PostgreSQL Compatible DBs - Amazon Aurora, CockroachDB, YugabyteDB

MySQL and PostgreSQL Compatible

The Amazon Aurora database engine is fully compatible with existing MySQL and PostgreSQL

Amazon **Aurora**

Familiar, consistent SQL

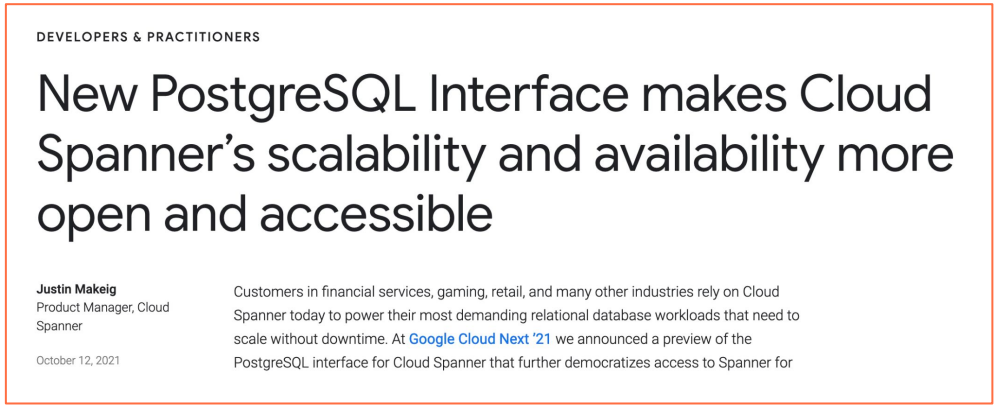
Use a language you already know and tools you already love with Postgres-compatible SQL syntax.

CockroachDB

Get instantly productive with a Postgres compatible RDBMS.

yugabyteDB

Even Google Spanner Announced PostgreSQL Compatibility in October



DEVELOPERS & PRACTITIONERS

New PostgreSQL Interface makes Cloud Spanner's scalability and availability more open and accessible

Justin Makeig
Product Manager, Cloud Spanner

October 12, 2021

Customers in financial services, gaming, retail, and many other industries rely on Cloud Spanner today to power their most demanding relational database workloads that need to scale without downtime. At Google Cloud Next '21 we announced a preview of the PostgreSQL interface for Cloud Spanner that further democratizes access to Spanner for

According to Google:

"PostgreSQL has emerged as the "API" for operational databases. PostgreSQL is a mature, open-source database with a large and growing ecosystem supported by multiple vendors. PostgreSQL's rapid growth and solid technical foundation have made it a safe choice for forward-looking organizations that value flexibility."

Wait… But What Exactly is PostgreSQL Compatibility?

Are All Postgres Compatible Databases the Same?

Levels of PostgreSQL Compatibility

Wire Compatibility

- Can use PostgreSQL client drivers
- Supports many languages (Java, JDBC, Go, etc.)
- Can use *psql* (the PG CLI) to explore the DB
- Developers can start building an app with ease because of familiarity

Syntax Compatibility

- Can parse PG syntax such as data types, DDL and DML
- Execution could be different though
- In some cases, statements are simply ignored
- Developers can use some PG tools & frameworks

Feature Compatibility

- Supports advanced features of PostgreSQL beyond ANSI SQL
- E.g: triggers, stored procedures, RLS, etc.
- Equivalent feature can differ in syntax, in this case developers need to re-learn equivalent feature and modify their apps

Runtime Compatibility

- Matches the PostgreSQL execution semantics at runtime
- Implies all other forms of compatibility
- Existing PG apps can run against the DB
- Developers that know PostgreSQL will instantly be at home

A Few Points on the Compatibility Levels

- **Compatibility levels follow the adoption curve:**

  Exploring a new DB (wire-protocol compatibility with psql)

  → Build and run a basic app and using simple tools (syntax compatibility)

  → Run more apps, some of which are more complex (feature & runtime compatibility)

  → Switching to the new DB as a default for a class of applications

- **Degree of compatibility is not all or nothing:**

  Databases can be higher or lower in their compatibility with PostgreSQL at any of these levels

- **Runtime compatibility implies all other forms of compatibility:**

  A database can support an application only to the degree it can support the other forms of compatibility

How Do the Various DBs Stack Up?

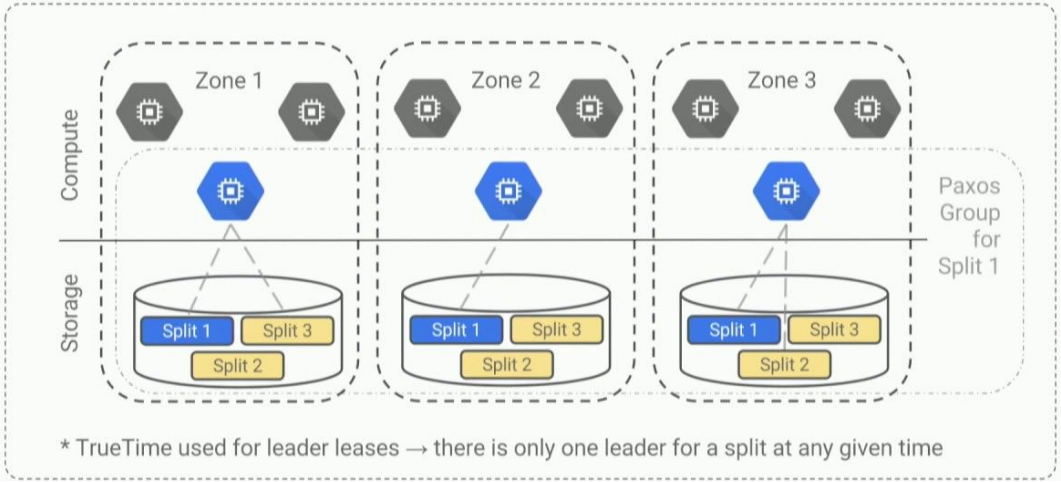
Compatibility	Google Spanner PostgreSQL	CockroachDB	Amazon Aurora PostgreSQL	YugabyteDB
Wire-protocol	Moderate	High	High	High
Syntax	Low	High	High	High
Feature Equivalence	Low	Low	High	High
Runtime	Low	Low	High	High

Let us look at how these databases build PostgreSQL compatibility

1. Google Spanner PostgreSQL

Google Spanner

- Google's "globally distributed database service"
- The first horizontally scalable relational database ever
- Started out with a proprietary API
  - Added SQL next
  - Most recently added a PostgreSQL API

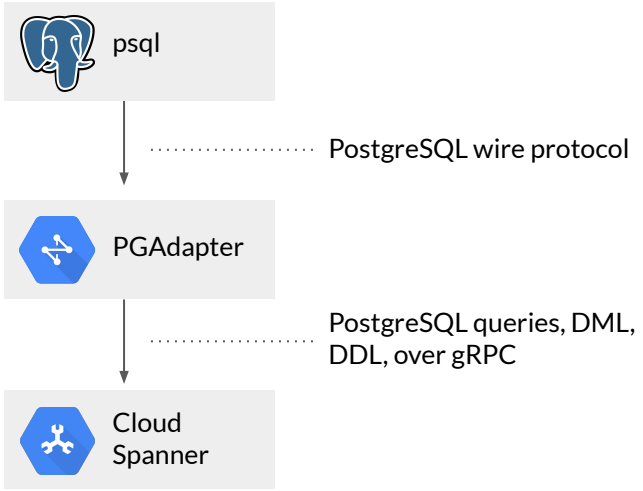


Design Point for PostgreSQL Compatibility in Google Spanner

- ***Highly available, horizontally scalable by design*** (unlike PostgreSQL)
- ***Distributed query processing*** (unlike PostgreSQL which is single node)
- ***Distributed storage layer*** (unlike PostgreSQL which is single node only)
- ***Started out with a custom SQL dialect***
- ***PostgreSQL is being added as an alternate API*** to interact with the database

How Google Spanner Achieves PostgreSQL Compatibility

- Google Spanner PostgreSQL *transpiles* the PostgreSQL statements to equivalent Spanner statements
- The transpiler is called pgadapter
- It runs as a sidecar next to the application
- The application connects to the transpiler
- The transpiler converts and issues Google Spanner statements to the database



Overall, has low PostgreSQL compatibility

What Does This Mean for Developers?

- A PG application may not run as-is
- May need to understand runtime semantics and workarounds for missing features

Familiarity and portability are the goals, not 100% compatibility

100% PostgreSQL compatibility is not the goal. We've focused on familiarity and portability, providing easier access to Spanner's consistency and availability at scale without reducing deployment flexibility.

Cloud SQL for PostgreSQL will continue to be the best choice in Google Cloud for applications that need the highest level of compatibility with PostgreSQL. Cloud SQL is the most direct route to lift and shift PostgreSQL applications to the cloud to take advantage of the operational and performance benefits of a fully managed service.Database Migration Service streamlines migrations with no extra charge.

Source: https://cloud.google.com/blog/topics/developers-practitioners/postgresql-interface-adds-familiarity-and-portability-cloud-spanner

Wire-Protocol Compatibility: *moderate*

The following are not supported, making it moderately compatible

- Functions
- `COPY` protocol
- Prepared statement `DESCRIBE`
- SSL
- PSQL meta-commands other than:
  - `\d <table>`
  - `\dt <table>`
  - `\dn <table>`
  - `\di <table>`
  - `\l`

Some of these could be important, such as `COPY` and the various other meta commands, however they may not always be blockers. Hence wire-protocol compatibility is moderate.

Source: https://cloud.google.com/spanner/docs/reference/postgresql/data-types

Syntax Compatibility: *low*

Google Spanner current does not support several PostgreSQL data types:

- `SERIAL` data type
- `DATE`
- `TIMESTAMP WITHOUT TIME ZONE`
- `CHAR`
- `INTERVAL`
- `ARRAY`

`SERIAL`, `CHAR`, `DATE` and `TIMESTAMP` are quite common in practice, making syntax compatibility very difficult

Source: https://cloud.google.com/spanner/docs/reference/postgresql/data-types

Feature Compatibility: *low*

Google Spanner current does not support several PostgreSQL features, including the following:

- Ecosystem clients
- Stored procedures
- Triggers
- SERIAL
- Privileges
- Fine-grained concurrency control
- Sequences
- SAVEPOINT
- Partial indexes
- Extensions
- Foreign data wrappers
- User-defined data types
- Functions
- Operators

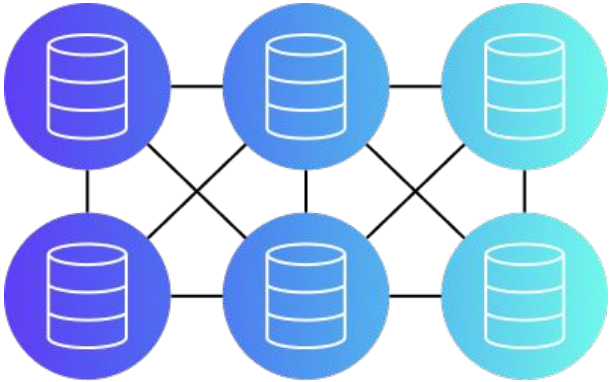
The lack of many of the above features makes feature compatibility very difficult to achieve. It would require building many of these features in the application layer.

Source: https://cloud.google.com/spanner/docs/reference/postgresql/data-types

2. CockroachDB

CockroachDB

- Relational NewSQL database for OLTP workloads - distributed, cloud native
- PostgreSQL wire compatible, achieved by rewriting the query layer in Go
- Available as “source available” software and fully managed cloud service



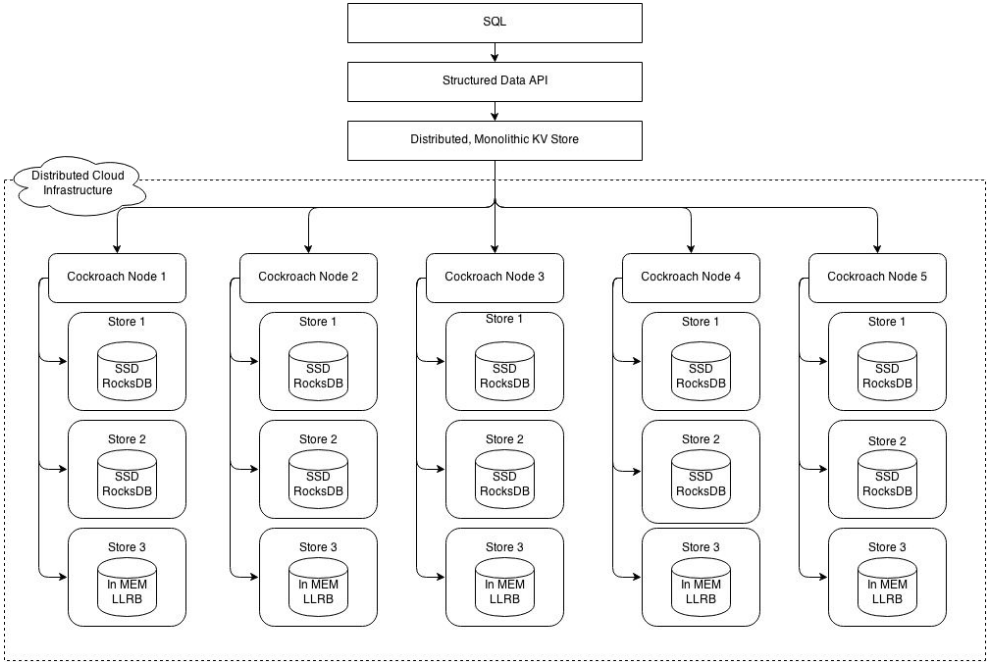
Design Point for PostgreSQL Compatibility in CockroachDB

- ***Highly available, horizontally scalable by design*** (unlike PostgreSQL)
- ***Distributed query processing*** (unlike PostgreSQL which is single node)
- ***Distributed storage layer*** (also unlike PostgreSQL which is single node only)
- ***PostgreSQL wire-protocol compatible, and somewhat syntax compatible***

How CockroachDB Achieves PostgreSQL Compatibility

- Re-implements the PostgreSQL API in Go
- Supports PostgreSQL wire protocol
- Tries hard to support a majority of PostgreSQL syntax
- Not feature or runtime compatible

It is extremely difficult to achieve runtime compatibility by re-implementing an existing DB



What Does This Mean for Developers?

- A PG application may not run as-is
- May need to understand runtime semantics and workarounds for missing features

> However, CockroachDB does not support some of the PostgreSQL features or behaves differently from PostgreSQL because not all features can be easily implemented in a distributed system. This page documents the known list of differences between PostgreSQL and CockroachDB for identical input. That is, a SQL statement of the type listed here will behave differently than in PostgreSQL. Porting an existing application to CockroachDB will require changing these expressions.

Source: https://www.cockroachlabs.com/docs/stable/postgresql-compatibility.html#sql-compatibility

Feature Compatibility: *low*

Unsupported PostgreSQL features

The following PostgreSQL features are not supported in CockroachDB v21.2:

- Stored procedures and functions
- Triggers
- Events
- User-defined functions (UDF)
- FULLTEXT functions and indexes
- Drop primary key
- XML Functions
- Column-level privileges
- XA syntax

- Additional features may be missing
  - Row level security
  - Column privileges
  - Tablespaces
  - DECLARE cursors
  - etc.
- Users of PostgreSQL tend to leverage these features extensively
- Many ecosystem projects use features such as these (example: GraphQL)
- These are critical for apps moving from Oracle, SQL Server, DB2, etc

The lack of many of the above features makes feature compatibility very difficult to achieve.
It would require building many of these features in the application layer.

Source: https://www.cockroachlabs.com/docs/stable/postgresql-compatibility.html#unsupported-features

Runtime Compatibility in Google Spanner and CockroachDB

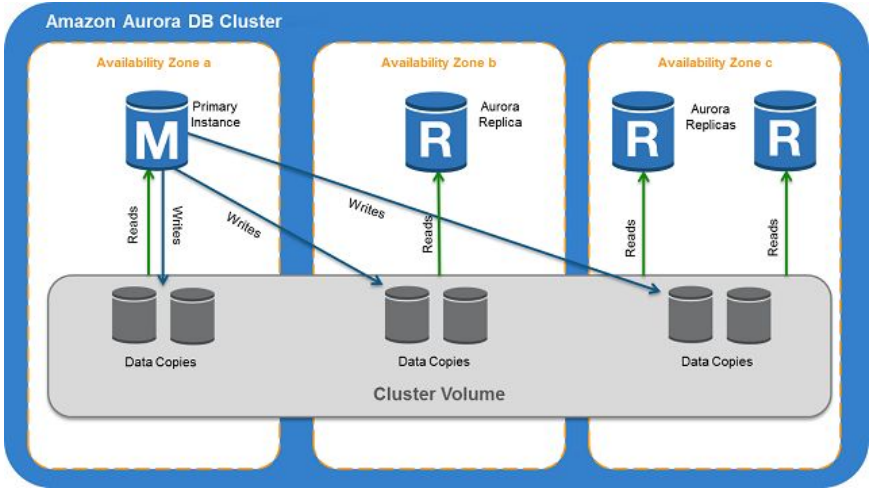
- Both support only the SERIALIZABLE isolation level
- PostgreSQL supports the following:
  - Serializable
  - Repeatable Read
  - Read Committed (default)
- Most relational databases (Oracle, SQL Server, DB2, etc) use Read Committed as default isolation level

Any application moving from these DBs to Google Spanner or CockroachDB would need to reason carefully about transaction semantics, making large scale migrations hard.

3. Amazon Aurora PostgreSQL

Amazon Aurora

- Amazon's fully managed relational distributed database-as-a-service
- MySQL and PostgreSQL compatible database engines
- Reuses query layer (MySQL & Postgres) similar to Yugabyte design
- Priced according to instance type and the number of cores; backup, storage, and networking costs extra
- Shared storage architecture



Design Point for PostgreSQL Compatibility in Amazon Aurora

- ***Crash resilient*** (unlike PostgreSQL)
- ***Not horizontally scalable by design for writes*** (like PostgreSQL)
- ***Predominantly single node query processing*** (like PostgreSQL)
- ***Distributed storage layer*** (unlike PostgreSQL which is single node only)
- ***Full PostgreSQL feature compatibility*** (like PostgreSQL)
- ***Nearly complete PostgreSQL runtime compatibility*** (like PostgreSQL)

PostgreSQL Compatibility in Amazon Aurora

- Reuses the PostgreSQL code
- Applications continue to interact with PG code base
- Aurora re-implements the storage sub-system
  - Data files
  - Write-ahead logging
  - Crash recovery
- Runtime semantics are preserved across all aspects
  - Authentication
  - Authorization
  - Query and syntax
  - Features supported

PostgreSQL Compatibility in Amazon Aurora

- Any app that uses PG automatically runs on Aurora
- Any developer with a working knowledge of PostgreSQL is instantly familiar with Aurora
- Distributed storage sub-system may cause some higher latencies and poorer performance
    - Not an issue in some cases
    - App may need to be modified a bit in other cases

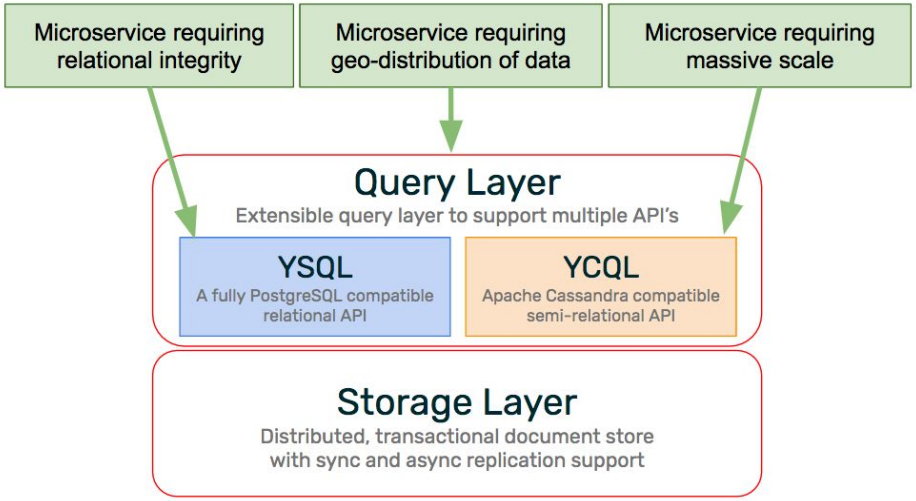
MySQL and PostgreSQL Compatible

The Amazon Aurora database engine is fully compatible with existing MySQL and PostgreSQL open source databases, and adds support for new releases regularly. This means you can easily migrate MySQL or PostgreSQL databases to Aurora using standard MySQL or PostgreSQL import/export tools or snapshots. It also means the code, applications, drivers, and tools you already use with your existing databases can be used with Amazon Aurora with little or no change. Learn more: MySQL | PostgreSQL

4. YugabyteDB YSQL

YugabyteDB

- Relational distributed SQL database for OLTP workloads - distributed, cloud native
- PostgreSQL wire compatible, achieved by reusing the PostgreSQL query layer
- 100% open source, runs in any cloud

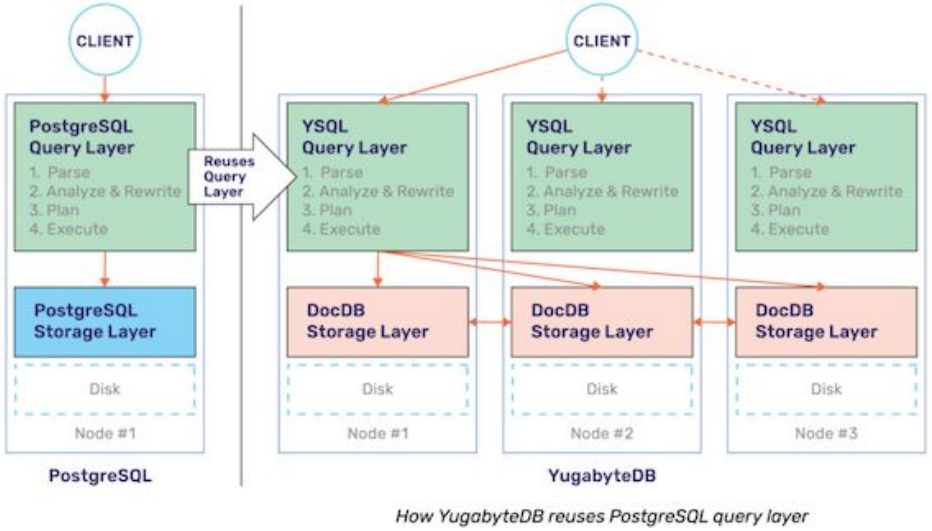


Design Point for PostgreSQL Compatibility in YugabyteDB

- ***Highly available, horizontally scalable by design*** (unlike PostgreSQL)
- ***Distributed query processing*** (unlike PostgreSQL which is single node)
- ***Distributed storage layer*** (also unlike PostgreSQL which is single node only)
- ***Full PostgreSQL feature compatibility is the goal*** (like PostgreSQL)
- ***Nearly complete PostgreSQL runtime compatibility is the goal*** (like PostgreSQL)

PostgreSQL Compatibility with YugabyteDB

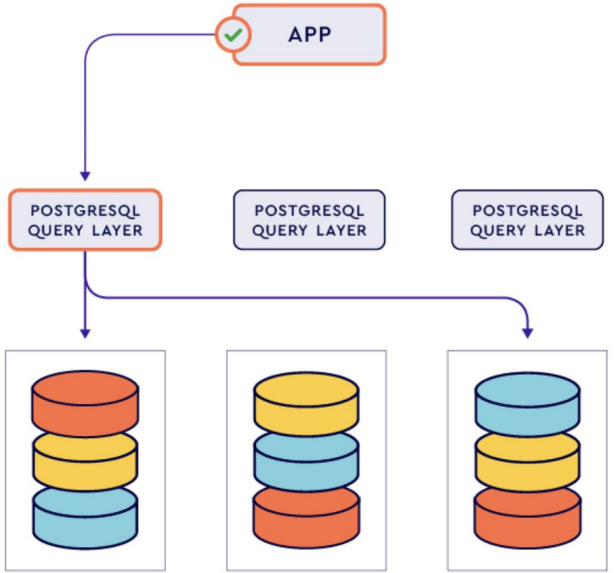
- Reuses the *upper half* of PostgreSQL
  - Query parser, planner, optimizer, executor
  - Similar to Amazon Aurora
- Re-implements the *lower half*
  - Transaction processing subsystem
  - Storage system
  - Automatic data sharding
  - Similar to Google Spanner
- Runtime semantics are preserved
  - Authentication
  - Authorization
  - Query and syntax
  - Features supported



How YugabyteDB reuses PostgreSQL query layer

PostgreSQL Compatibility with YugabyteDB

- Any app that uses PG automatically runs on YugabyteDB YSQL
- Any developer with a working knowledge of PostgreSQL is instantly familiar with YSQL
- Distributed query layer enables scalability and geo-distribution
- Distributed storage sub-system may cause some higher latencies and poorer performance
    - Not an issue in some cases
    - App may need to be modified a bit in other cases



Why all this matters… let's run through some examples.

Common Example Scenario: Auto-update row timestamp

Let's say we have a users table. How to update the update_at column timestamp automatically on modification?

```
CREATE TABLE users (
  id         SERIAL NOT NULL PRIMARY KEY,
  name       TEXT,
  details    TEXT,
  created_at TIMESTAMPTZ NOT NULL DEFAULT NOW(),
  updated_at TIMESTAMPTZ NOT NULL DEFAULT NOW()
);
```

The PostgreSQL Compatible Approach

PostgreSQL developers would:

1. Create a function

```
CREATE OR REPLACE FUNCTION trigger_set_timestamp()
  RETURNS TRIGGER
  LANGUAGE plpgsql
  AS $body$
BEGIN
  NEW.updated_at = transaction_timestamp();
  RETURN NEW;
END;
$body$;
```

2. Attach it to a trigger

```
CREATE TRIGGER set_updated_at
  BEFORE UPDATE ON users
  FOR EACH ROW EXECUTE PROCEDURE trigger_set_timestamp();
```

- Can copy-paste from Google search
- Requires runtime compatibility

- Works on:
  - Amazon Aurora PostgreSQL
  - YugabyteDB YSQL

- Does not work on:
  - Google Spanner
  - CockroachDB

The Google Spanner Way

Writing a commit timestamp using a DML statement

You use the `PENDING_COMMIT_TIMESTAMP` function to write the commit timestamp in a DML statement. Cloud Spanner selects the commit timestamp when the transaction commits.

```
UPDATE Performances SET LastUpdateTime = PENDING_COMMIT_TIMESTAMP()
   WHERE SingerId=1 AND VenueId=2 AND EventDate="2015-10-21"
```

But You Learn the Hard Way

Note: After you call the `PENDING_COMMIT_TIMESTAMP` method, the table and any derived index is unreadable to any future SQL statements in the transaction. You must write commit timestamps as the last statement in a transaction to prevent the possibility of trying to read the table. If you try to read the table, then Cloud Spanner returns an error.

Feature equivalent, but not runtime compatible.

- Is this going to work for all your apps?
- Or is this something to worry about?

The CockroachDB Way

Spent a bunch of time trying to figure out the equivalent feature...

We currently support `DEFAULT current_timestamp()` when creating a table, but do not support the additional `ON UPDATE current_timestamp()` default, as described here in the mysql docs: https://dev.mysql.com/doc/refman/8.0/en/timestamp-initialization.html. The consequence (I believe) is that users would need to manage updates in app in order to adopt CRDB.

wzrdtales commented on Apr 27, 2020

some ping after almost 2 years now :)

Remove recommendations to use `crdb_internal.mvcc_timestamp` in favor of `ON UPDATE` cockroachdb/docs#11028

Source: https://github.com/cockroachdb/cockroach/issues/28281

Found a way, but unclear if the right way

And then, I can verify the data is there (note the addition of the `crdb_internal_mvcc_timestamp` system column):

```
root@:26257/defaultdb> select *, crdb_internal_mvcc_timestamp from person order by 3 desc limit 5;
                  id                  |         name         |  crdb_internal_mvcc_timestamp
--------------------------------------+----------------------+---------------------------------
  08c6ab92-507c-4f43-91f7-e0f8547628a0 | d9ea3d4601d0d21406d7 | 1614815527857987000.0000000000
  18e511e9-20a8-464e-bb76-94415b76ee4f | fa6a8a20c8089b7f6e5b | 1614815527857987000.0000000000
  207843e3-d52e-4502-9b96-998e0bef6ff6 | e9299d79f668c336db8a | 1614815527857987000.0000000000
  2869a943-ca3f-4bd9-ab05-55a924aa7ec2 | 236a9fd4c590e72552c9 | 1614815527857987000.0000000000
  41985298-057e-4364-af19-99be48569fa9 | 813fe18e4fbd431fb0a5 | 1614815527857987000.0000000000
(5 rows)

Time: 2ms total (execution 2ms / network 0ms)
```

Source: https://dzone.com/articles/a-contrived-example-to-illustrate-how-to-track-las

Is this the recommended way?

- Definitely not runtime compatible.
- Is there an equivalent feature?
- If the feature name crdb_internal_mvcc_timestamp has internal, is it production ready?
- Is this going to work for all your apps?

There are many other such scenarios…

… where you need to know that the database will just work.

Remember...

Only PostgreSQL is 100% PostgreSQL Compatible.

But for the rest of the DBs,

higher PostgreSQL compatibility is always better!

Thank You

Join our community
yugabyte.com/slack

Try for yourself
download.yugabyte.com