



## AWS Open Source Blog

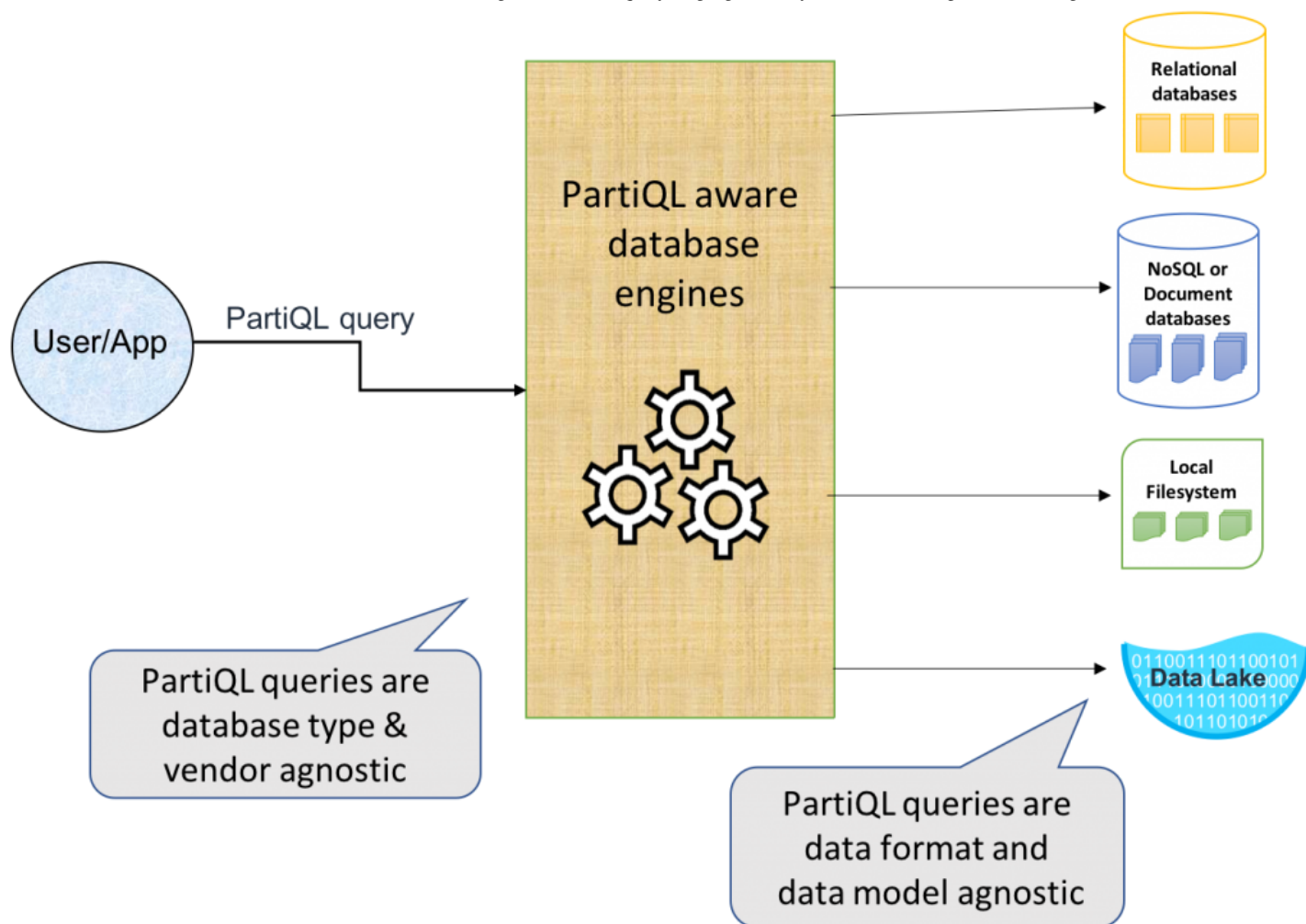
# Announcing PartiQL: One query language for all your data

by Yannis Papakonstantinou, Almann Goo, Brad Ruppert, Jon Wilsdon, and Prasad Varakur | on 01 AUG 2019 | in [Amazon Redshift](#), [Glacier Select](#), [Open Source](#), [S3 Select](#) | [Permalink](#) | [💬 Comments](#) | [↪ Share](#)

Data is being gathered and created at rates unprecedented in history. Much of this data is intended to drive business outcomes but, according to the [Harvard Business Review](#), “...on average, less than half of an organization’s structured data is actively used in making decisions...”

The root of the problem is that data is typically spread across a combination of relational databases, non-relational data stores, and data lakes. Some data may be highly structured and stored in SQL databases or data warehouses. Other data may be stored in NoSQL engines, including key-value stores, graph databases, ledger databases, or time-series databases. Data may also reside in the data lake, stored in formats that may lack schema, or may involve nesting or multiple values (e.g., [Parquet](#), JSON). Every different type and flavor of data store may suit a particular use case, but each also comes with its own query language. The result is tight coupling between the query language and the format in which data is stored. Hence, if you want to change your data to another format, or change the database engine you use to access/process that data (which is not uncommon in a data lake world), or change the location of your data, you may also need to change your application and queries. This is a very large obstacle to the agility and flexibility needed to effectively use data lakes.

Today we are happy to announce [PartiQL](#), a SQL-compatible query language that makes it easy to efficiently query data, regardless of where or in what format it is stored. As long as your query engine supports PartiQL, you can process structured data from relational databases (both transactional and analytical), semi-structured and nested data in open data formats (such as an Amazon S3 data lake), and even schema-less data in NoSQL or document databases that allow different attributes for different rows. We are open sourcing the PartiQL [tutorial](#), [specification](#), and a [reference implementation](#) of the language under the Apache2.0 license, so that everyone can participate, contribute, and use it to drive widespread adoption for this unifying query language.



The PartiQL open source will make it easy for developers to parse and embed PartiQL in their own applications. The implementation supports users parsing PartiQL queries into abstract syntax trees that their applications can analyze or process and supports interpreting PartiQL queries directly.

PartiQL solves problems we faced within Amazon. It is already being used by Amazon S3 Select, Amazon Glacier Select, [Amazon Redshift](#) Spectrum, Amazon Quantum Ledger Database ([Amazon QLDB](#)), and Amazon internal systems. Also, [Amazon EMR](#) pushes down PartiQL queries to S3 Select. More AWS services will add support in the coming months. Outside of Amazon, Couchbase also looks forward to supporting PartiQL in the [Couchbase Server](#).

We look forward to the creators of data processing engines diving deep into PartiQL, and joining us in solving a problem that affects all users of data, across all industries.

## Why we built it

We developed PartiQL in response to Amazon's own needs to query and transform vast amounts and varieties of data – not just SQL tabular data, but also nested and semi-structured data – found in a variety of formats and storage engines. Amazon's retail business already had vast sets of semi-structured data, most often in the [Ion format](#). Amazon's retail business, led by Chris Suver, was in pursuit of an SQL-like query language. Multiple AWS services, such as [QLDB](#), saw the benefits of schema-optional, document-oriented data models, but also wanted

to leverage existing SQL knowledge and tools. Finally, the AWS relational database services like [Redshift](#), and the many existing clients of SQL, needed to expand into accessing the non-relational data of the data lake, while maintaining strict backwards compatibility with SQL. At the same time, the database research community, with query language works like UCSD's [SQL++](#), was showing that it is possible to devise clean, well-founded query languages that stay very close to SQL, while having the power needed to process nested and semi-structured data.

**Don Chamberlin**, creator of the SQL language specification, says: "As JSON and other nested and semi-structured data formats have increased in importance, the need for a query language for these data formats has become clear. The approach of adapting SQL for this purpose has the advantage of building on our industry's investment in SQL skills, tools, and infrastructure. The [SQL++](#) proposal of Dr. Yannis Papakonstantinou, and languages based on SQL++ such as PartiQL, have shown that the extensions to SQL needed for querying semistructured data are fairly minimal. I hope that these small language extensions will help to facilitate a new generation of applications that process data in JSON and other flexible formats, with and without predefined schemas."

We therefore set out to create a language that offers strict SQL compatibility, achieves nested and semi-structured processing with minimal extensions, treats nested data as a first-class citizen, allows optional schema, and is independent of physical formats and data stores.

The result was PartiQL, which provides a simple and consistent way to query data across a variety of formats and services. This gives you the freedom to move your data across data sources, without having to change your queries. It is backwards-compatible with SQL, and provides extensions for multi-valued, nested, and schema-less data, which blend seamlessly with the join, filtering, and aggregation capabilities of standard SQL.

## PartiQL design tenets

The following design tenets captured our design goals and were fundamental to PartiQL:

- **SQL compatibility:** PartiQL facilitates adoption by maintaining compatibility with SQL. Existing SQL queries will continue to work (that is, they will maintain their syntax and semantics) in SQL query processors that are extended to provide PartiQL. This avoids any need to rewrite existing SQL, and makes it easy for developers and business intelligence tools to leverage PartiQL.
- **First-class nested data:** The data model treats nested data as a fundamental part of the data abstraction. Consequently, the PartiQL query language provides syntax and semantics that comprehensively and accurately access and query nested data, while naturally composing with the standard features of SQL.
- **Optional schema and query stability:** PartiQL does not require a predefined schema over a dataset. It is designed to be usable by database engines that assume the presence of a schema (be it schema-on-write or schema-on-read) or schemaless engines. Technically, the result of a working query does not change as a schema is imposed on existing data, so long as the data itself remains the same. It is thus easier to provide consistent access to multiple stores, despite the different schema assumptions of the participating engines.

- **Minimal extensions:** PartiQL has a minimum number of extensions over SQL. The extensions are easy to understand, lend themselves to efficient implementation, and compose well with each other and with SQL itself. This enables intuitive filtering, joining, aggregation, and windowing on the combination of structured, semi-structured, and nested datasets.
- **Format independence:** PartiQL syntax and semantics are not tied to any particular data format. A query is written identically across underlying data in JSON, Parquet, ORC, CSV, Ion, or other formats. Queries operate on a comprehensive logical type system that maps to diverse underlying formats.
- **Data store independence:** PartiQL syntax and semantics are not tied to a particular underlying data store. Thanks to its expressiveness, the language is applicable to diverse underlying data stores.

Past languages have addressed subsets of these tenets. For example, Postgres JSON is SQL-compatible, but does not treat the JSON nested data as a first-class citizen. Semi-structured query languages treat nested data as first-class citizens, but either allow occasional incompatibilities with SQL, or do not even look like SQL. PartiQL is the first language to address this full set of tenets.

As you would expect from its design tenets, PartiQL will be both easy and familiar for SQL users. It has been in use by several customers of Amazon Redshift Spectrum since 2018:

**Annalect** is Omnicom's global data and analytics arm, providing purpose-built, scalable solutions that make data actionable, and is the driving force behind Omnicom's revolutionary precision marketing and insights platform, *Omni*. "PartiQL enables us to query nested data with Amazon Redshift Spectrum directly in Amazon S3 without un-nesting, and will also enable us to easily bring nested data from Amazon S3 into local tables in Amazon Redshift using standardized language," said Eric Kamm, senior engineer and architect at Annalect. John Briscoe, Director of Data and Operations at Annalect added: "We're also excited that it will give us consistent query syntax from one data platform to another, allowing for easier development of multi-data platform applications and easier developer onboarding."

**Steven Moy, Software Engineer at Yelp:** "PartiQL addresses the critical missing piece in a poly-store environment — a high-level declarative language that works across multiple domain-specific data stores. At Yelp, we leverage multiple AWS data stores (Redshift, S3, DynamoDB) technology to deliver the best local businesses to users and best way to reach local audiences for local business owners. With Amazon Redshift Spectrum, Yelp enables eight times the amount of data to help our developer communities make data informed decisions, and we look forward to taking that partnership a step further with PartiQL which will allow Yelp's developers to focus their time on creating delightful user experiences instead of mastering a new query language or solving classic consistency problems."

Unlike traditional SQL, the PartiQL query language also meets the needs of NoSQL and non-relational databases. PartiQL has already been adopted by the [Amazon Quantum Ledger Database \(QLDB\)](#) as their query language.

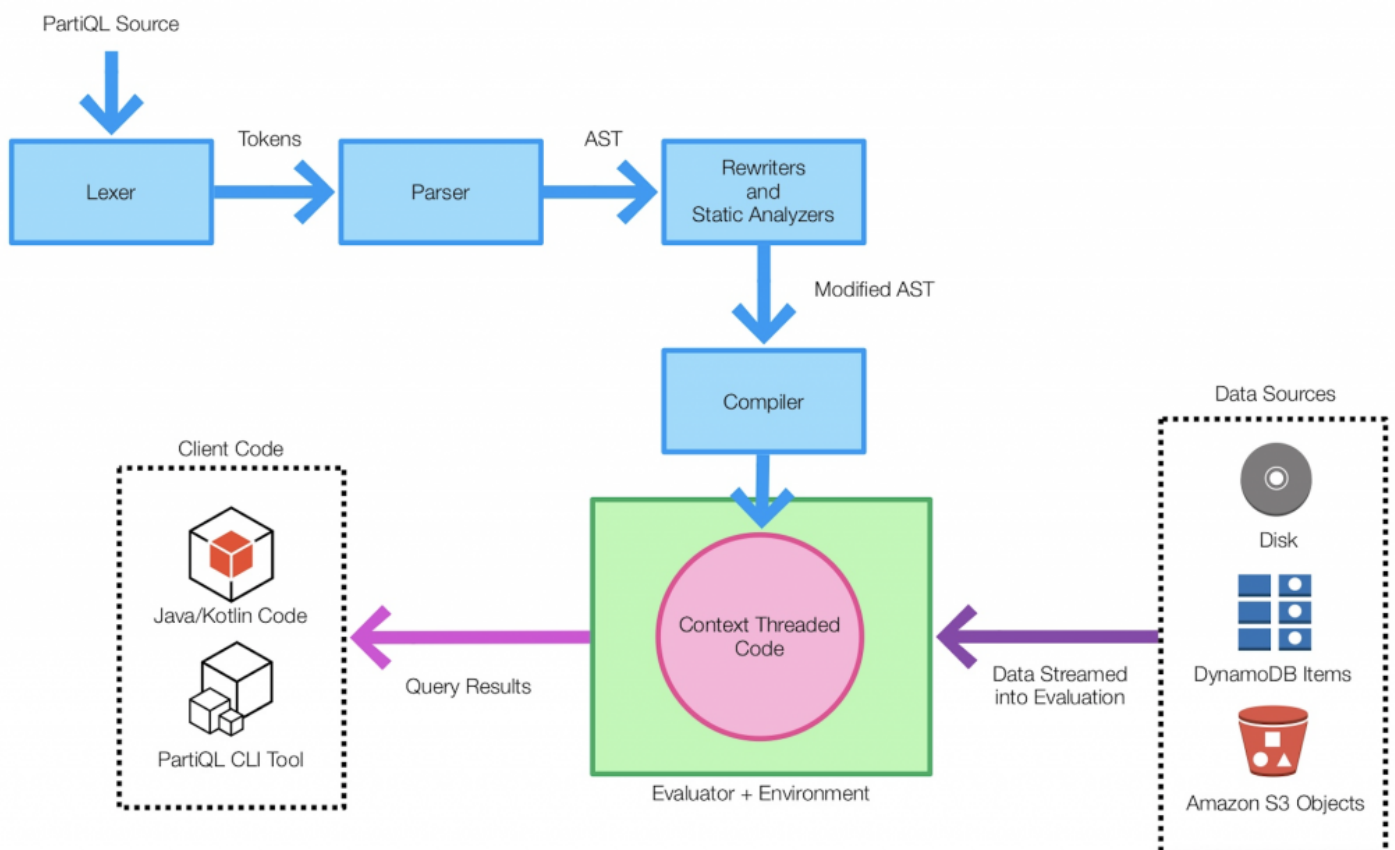
**Andrew Certain, AWS Senior Principal Engineer and Amazon Quantum Ledger Database (QLDB) architect**, says about the choice of PartiQL: “QLDB needed a flexible, document-oriented data model so that users can easily store and process both structured and semi-structured data, without the burden of defining and evolving a schema. At the same time, QLDB wanted to benefit from the wide knowledge of SQL. PartiQL has greatly served both purposes. Its extensions for accessing nested and semistructured data are few, powerful and quite intuitive.” QLDB, currently in preview mode, is one of the AWS services that adopted PartiQL.

The **Couchbase Server**, which utilizes a JSON-based document-oriented data model, is also looking forward to adopting PartiQL:

**Ravi Mayuram, Senior Vice President of Engineering and CTO of Couchbase**, says: “As a pioneer in bringing SQL to JSON when we introduced N1QL more than three years ago, Couchbase believes that the foundations on which SQL was built for relational databases are just as sound for a JSON data model and database. PartiQL is a welcome next step in this convergence, and we look forward to supporting it.”

## PartiQL reference engine

### PartiQL Reference Implementation Architecture



This diagram shows, at a very high level, the PartiQL reference implementation. We are open sourcing the lexer, parser, and compiler for PartiQL query expressions. We provide a library that can be embedded or used as a standalone tool for running queries. A user could use this library to simply validate PartiQL queries, or to embed a PartiQL evaluator to process data within their system. The library provides a data interface to bind to whatever data back end an application may have, and provides out-of-the-box support for Ion and JSON.

## Getting started

The PartiQL open source implementation provides an interactive shell (or Read Evaluate Print Loop (REPL)) which allows users to write and evaluate PartiQL queries.

### Prerequisites

PartiQL requires the Java Runtime (JVM) to be installed on your machine. You can obtain the latest version of the Java Runtime from [OpenJDK](#), [OpenJDK for Windows](#), or [Oracle](#).

Follow the instructions for [Installing the JDK Software and Setting](#) `JAVA_HOME` to the path where your Java Runtime is installed.

### Download the PartiQL REPL

Each [release of PartiQL](#) comes with an archive that contains the PartiQL REPL as a zip file.

You may have to click on [Assets](#) to see the zip and tgz archives. Download the latest `partiql-cli` zip archive to your machine. The file will append PartiQL's release version to the archive, i.e., `partiql-cli-0.1.0.zip`.

Expand (unzip) the archive on your machine, which yields the following folder structure (where `...` represents elided files/directories):

```
├─ partiql-cli
│   ├─ bin
│   │   ├─ partiql
│   │   └─ partiql.bat
│   └─ lib
│       └─ ...
├─ README.md
└─ Tutorial
    ├─ code
    │   └─ ...
    ├─ tutorial.html
    └─ tutorial.pdf
```

The root folder `partiql-cli` contains a `README.md` file and three subfolders:

1. `bin` contains startup scripts: `partiql` for macOS and Unix systems and `partiql.bat` for Windows systems. Execute these files to start the REPL.
2. `lib` contains all the necessary Java libraries needed to run PartiQL.
3. `Tutorial` contains the tutorial in `pdf` and `html` form. The subfolder `code` contains three types of files:
  1. Data files with the extension `.env`. These files contain PartiQL data that we can query.
  2. PartiQL query files with the extension `.sql`. These files contain the PartiQL queries used in the tutorial.
  3. Sample query output files with the extension `.output`. These files contain sample output from running the tutorial queries on the appropriate data.

## Running the PartiQL REPL

### Windows

Run (double-click on) `partiql.bat`. This should open a command-line prompt and start the PartiQL REPL, which displays:

```
Welcome to the PartiQL REPL!  
PartiQL>
```

### macOS (Mac) and Unix

1. Open a terminal and navigate to the `partiql-cli` folder. The folder name will have the PartiQL version as a suffix, i.e., `partiql-cli-0.1.0`.
2. Start the REPL by typing `./bin/partiql` and pressing ENTER, which displays:

```
Welcome to the PartiQL REPL!  
PartiQL>
```

## Testing the PartiQL REPL

Write a simple query to verify that your PartiQL REPL is working. At the `PartiQL>` prompt type:

```
PartiQL> SELECT * FROM [1,2,3]
```

and press `ENTER` *twice*. The output should look similar to:

```
PartiQL> SELECT * FROM [1,2,3]
```

```
|
```

```
=== '
```

```
<<
```

```
{
```

```
  '_1': 1
```

```
},
```

```
{
```

```
  '_1': 2
```

```
},
```

```
{
```

```
  '_1': 3
```

```
}
```

```
>>
```

```
---
```

```
OK! (86 ms)
```

```
PartiQL>?
```

Congratulations! You have successfully installed and run the PartiQL REPL. The PartiQL REPL is now waiting for more input.

To exit the PartiQL REPL, press:

- **Control+D** in macOS or Unix
- **Control+C** on Windows

or close the terminal/command prompt window.

## Loading data from a file

An easy way to load the necessary data into the REPL is use the **-e** switch when starting the REPL and provide the name of a file that contains your data:

```
./bin/partiql -e Tutorial/code/q1.env
```

You can then see what is loaded in the REPL's global environment using the **special** REPL command

**!global\_env**, i.e.,

```
{
  'id': 3,
  'name': 'Bob Smith',
  'title': NULL
},
{
  'id': 4,
```



```
'name': 'Susan Smith',
'title': 'Dev Mgr'
},
{
  'id': 6,
  'name': 'Jane Smith',
  'title': 'Software Eng 2'
}
>>
}
}
---
```

OK! (6 ms)

## How to participate in PartiQL

PartiQL is fully open sourced under the Apache2.0 license. We welcome your contributions in further expanding the specification, building the technology and increasing its adoption & mindshare in the user community. [Learn more about PartiQL](#).

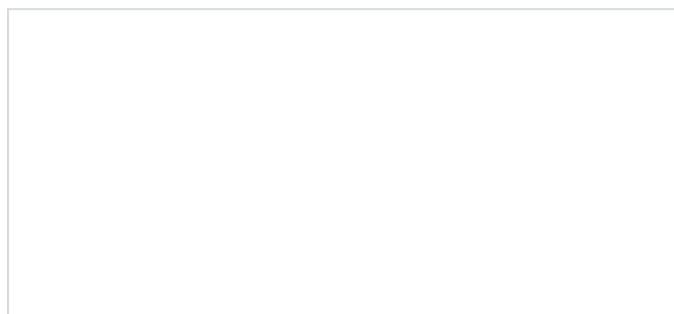
You can contribute to the project by sending a pull request for a [good first issue](#). File an [issue](#) if there are bugs or missing features. Read through the [tutorial](#) to understand PartiQL syntax, how it extends SQL, and take a step-by-step walkthrough. Want to learn about every little detail of PartiQL? Read through the [specification](#). You can also talk to us in the [PartiQL Discourse community](#).

## re:Invent 2019!

Learn more about PartiQL in these re:Invent sessions:

- [OPN207 – PartiQL: One query language for all of your data](#)
- [OPN308 PartiQL: Solution integration and joining the community](#)

TAGS: [storage](#)



**AWS Podcast**

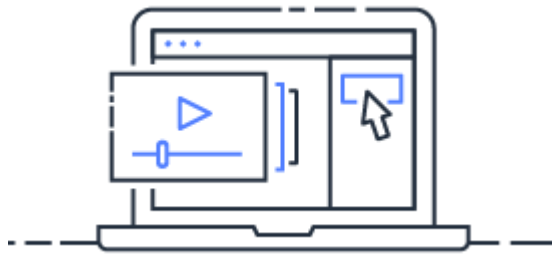
Subscribe for weekly AWS news and interviews

[Learn more »](#)

**AWS Partner Network**

Find an APN member to support your cloud business needs

[Learn more »](#)

**AWS Training & Certifications**








Free digital courses to help you develop your skills

[Learn more »](#)

## Resources

[Open Source at AWS](#)  
[Projects on GitHub](#)

## Follow

-  [AWS Open Source](#)
-  [AWS Cloud](#)
-  [Facebook](#)
-  [LinkedIn](#)
-  [Twitch](#)
-  [Open Source RSS](#)
-  [Email Updates](#)

### Amazon Managed Service for Grafana

Create unified, interactive data visualizations

[Learn more »](#)

## Related Posts

---

[Sharing Amazon Redshift data securely across Amazon Redshift clusters for workload isolation](#)

[AWS partners accelerate customer innovation with SAP Beyond Infrastructure offerings](#)

[Announcing Amazon Redshift federated querying to Amazon Aurora MySQL and Amazon RDS for MySQL](#)

[Building high-quality benchmark tests for Amazon Redshift using SQLWorkbench and psql](#)

[Getting the most out of your analytics stack with Amazon Redshift](#)

[How Etleap Integrates with Amazon Redshift Data Sharing to Provide Isolation of ETL and BI Workloads](#)

[Introducing Amazon Redshift RA3.xlplus nodes with managed storage](#)

[Optimizing tables in Amazon Redshift using Automatic Table Optimization](#)

