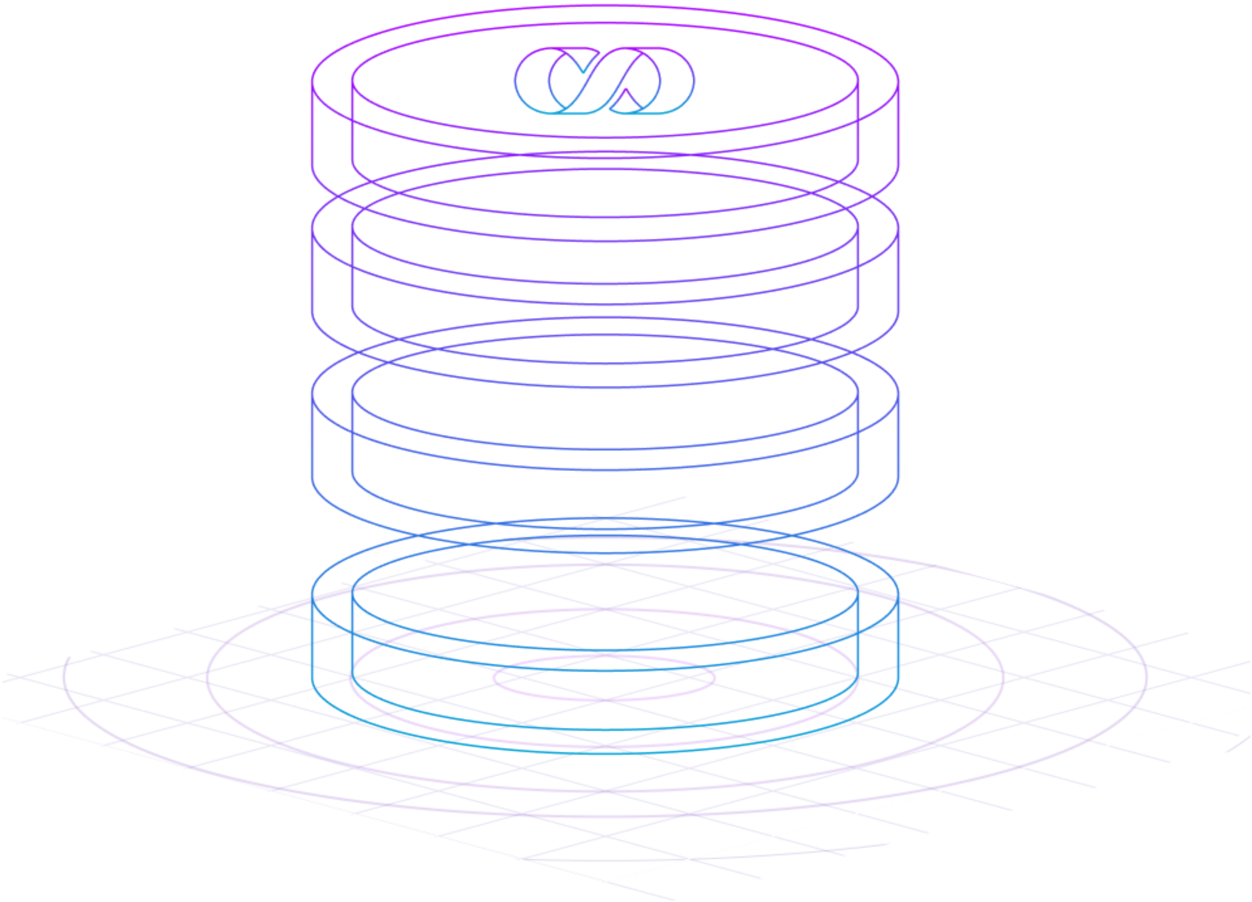
TECHNICAL WHITEPAPER

—

Introduction to MemSQL



2019

Table of Contents

Introduction

This whitepaper introduces MemSQL as a modern data platform with the speed, scale and cost efficiencies to generate business value and insights from operational data.

You'll learn how MemSQL is designed and built to ingest data at high speeds, scale-out efficiently and deliver record-breaking query performance with familiar relational SQL.

Data Platform Landscape

Historically, databases fit into one of two categories: those optimized for online transactional processing (OLTP) and those optimized for online analytical processing (OLAP). Transactional systems are traditionally separate from Analytical processing systems, even though they could potentially be combined into a single system. Transaction systems are often revenue generating, have strict availability requirements, and are viewed as mission critical.

OLAP running on Data warehouse systems are used to analyze large amounts of data and require a lot of computing resources. They are not generally as mission critical.

Combining data warehouse and transaction systems in a single database generally results in the transaction workload suffering and thus affecting the business adversely. Hence, separation of the two has become standard.

A third type of data platform, called an operational data store (ODS), is used to support operational analytics. This data platform allows the business to have near real-time visibility into rapidly changing events, such as orders and/or customer interaction. Success of the operational data store is driven by the ability to handle streaming ingest of data with a high number of concurrent analytical queries.

An ODS receives transactions from an OLTP system in a minimally intrusive manner using techniques such as extract transform and load(ETL) or change data capture. It also serves as a source for the data warehouse. Traditionally, an ODS would serve as an up-to-date replica of operational data for a variety of analytic requirements. However, over the years, more data sources have driven a massive increase in data volume and velocity creating a series of challenges for legacy ODS technologies to keep up.

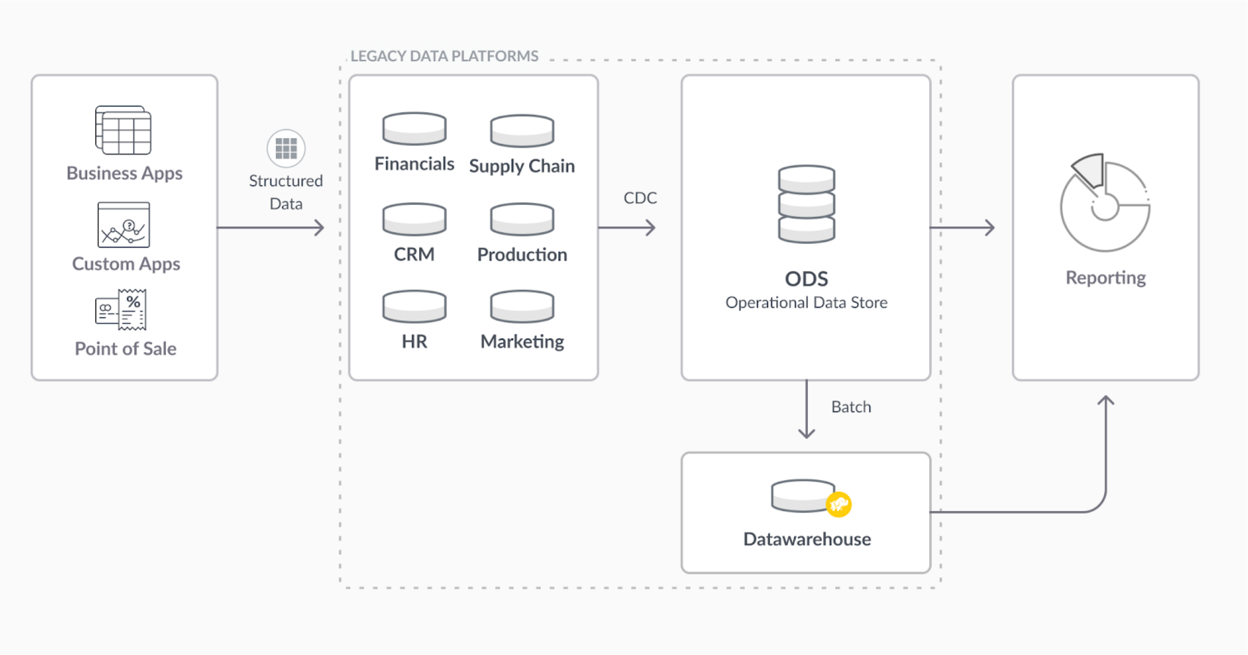


Figure 1. Legacy multi-tiered ODS architecture

How MemSQL Modernizes Data Platforms

First released in 2011, MemSQL is a third generation RDBMS written in C/C++. MemSQL is designed to run efficiently on modern systems; both multi-core systems with a big memory footprint and lower-powered edge computing devices.

MemSQL is ANSI SQL-Compatible and natively supports structured, semi-structured, and unstructured (full-text search) data. With built-in connectors to Kafka, Spark, S3, and Hadoop, as well as legacy transactional systems, MemSQL easily integrates with a broad ecosystem to cover both real-time streaming and batch workloads.

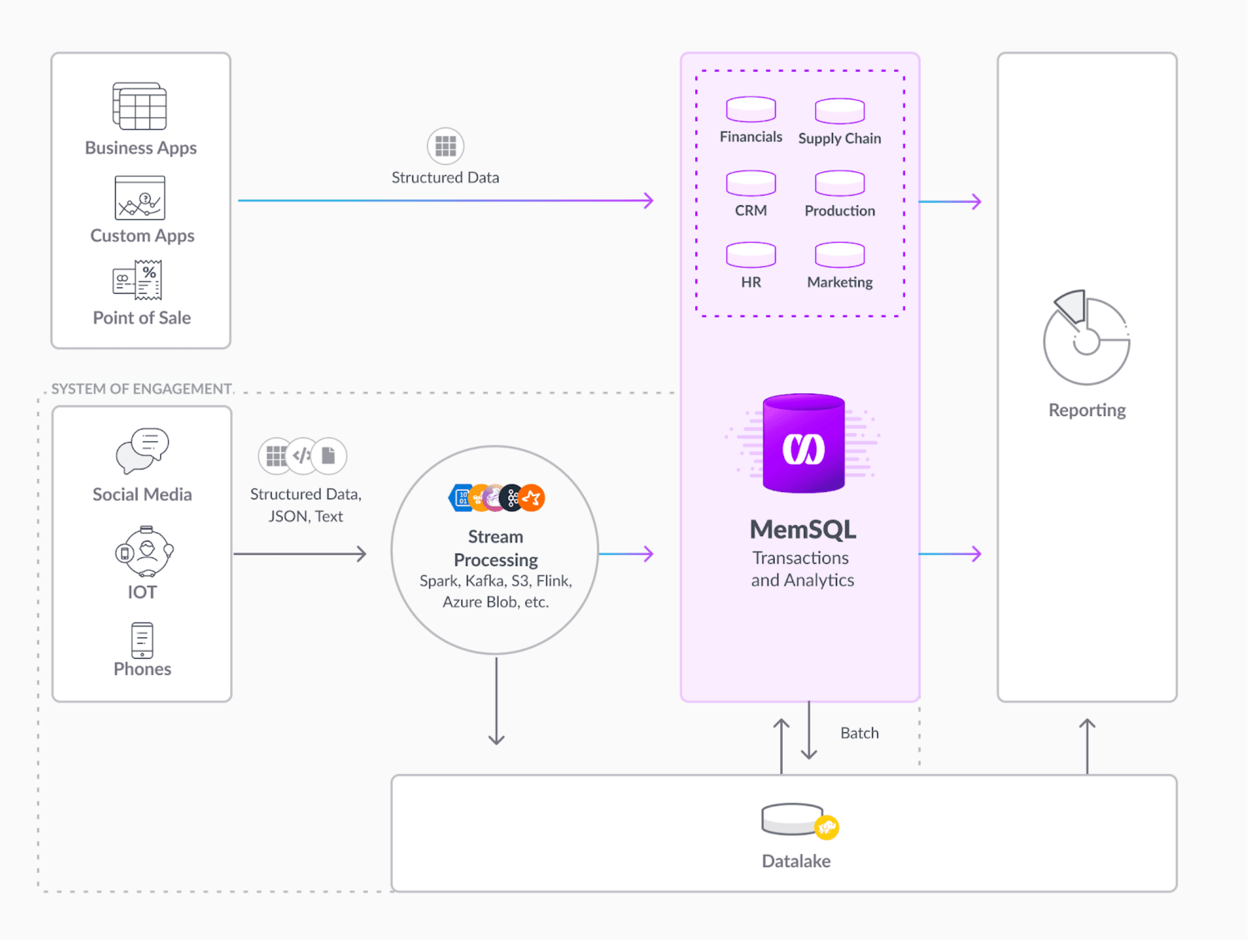


Figure 2. Modernizing legacy data platforms with MemSQL

With support for JSON and Documents, MemSQL can ingest data from modern sources such as mobile phones, social media, and smart devices and provide both transactional and analytical capabilities on a single platform with the ease and familiarity of SQWith MemSQL, all enterprise features, such as partitioning, security, high availability (HA), and disaster recovery (DR), are included in the product and not licensed separately. As a modern and efficient platform, workloads often run on less hardware when compared to legacy systems. Thus cost savings are realized in reduced software spend, the reduction of data silos through decoupling transactions from analytic systems, the cost of deployment, and maintenance costs in comparison to legacy vendors.

MemSQL can run legacy transactional workloads and serve as the ODS or data hub that performs as an operational analytic backbone to power real-time decisions across reports, interactive dashboards, data science, and more.

Core MemSQL Technical Concepts

In this section we will go over some key technical concepts that underpin MemSQL's architecture. It's these key technical differences that differentiate it from other solutions and make it a solution of choice for customers.

Code Generation and Compiled Plans

When a query is submitted to a database, the query is interpreted, an execution plan generated and the query is executed. Optimizing this process is critical for performance. MemSQL embeds an industrial compiler (LLVM) for low-level optimizations along hot code paths - optimizations that are not possible when executing via interpretation. This approach also takes advantage of newer instruction sets that are available with modern cpus.

When a MemSQL server encounters a query shape for the first time, it generates a just-in-time execution plan, written in C++, which is incrementally compiled to machine code as it processes the query. This has two-fold benefits allowing optimal query performance with first run queries while offering extreme fast response time to repeat queries. Each compiled plan is cached to prepare for future invocations of the given query. When future queries match an existing parameterized query plan template, MemSQL bypasses code generation and executes the query immediately using the cached plan.

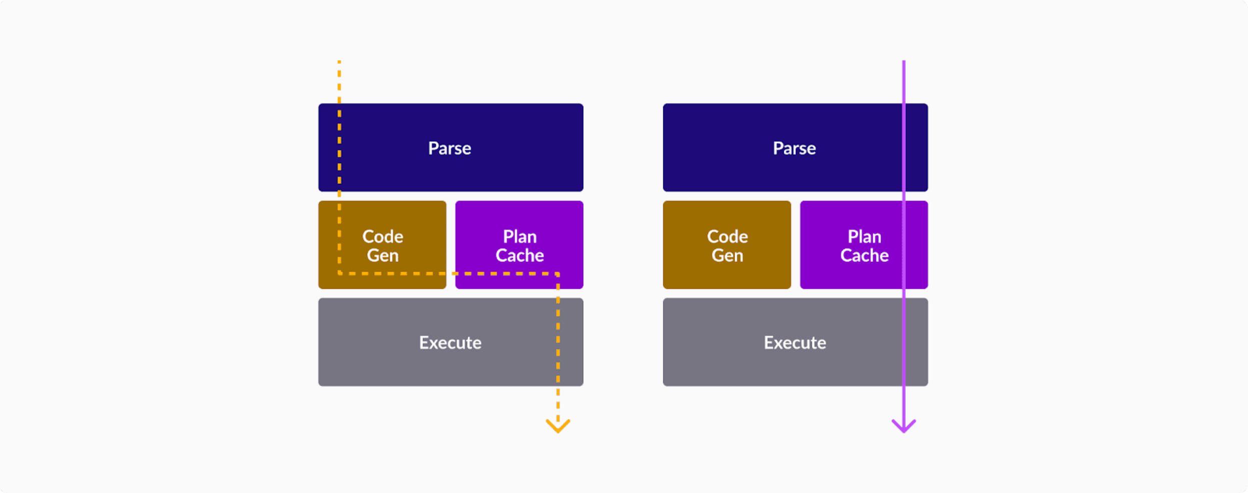


Figure 3. Compiled plans in MemSQL utilizing and bypassing the code generator

After code generation, the compiled plans are saved for later use in a plan cache. Each MemSQL node has its own plans and plan cache. A plan cache consists of two layers: the in-memory plan cache and the on-disk plan cache. Plans stored in the in-memory plan cache remain until they expire, or until the memSQL node restarts. When a plan expires, it stays put in the on-disk plan cache, and is loaded back into memory the next time the query is executed. By interpreting SQL statements and implementing compiled query plans, MemSQL removes interpretation overhead and minimizes code execution paths.

Another key feature of MemSQL's compiled plans is that they do not pre-specify values for the parameters. Query parameters are dynamically extracted from a query template, producing a normalized query that is then transformed into a specialized native representation (MemSQL Plan Language, or MPL). The generated execution plan is written in C++ and compiled to machine code. When queries that match the query template are executed, MemSQL substitutes the parameter values, allowing the request to reuse already-compiled plans and run quickly. Additionally, compiled plans are also reused across server restarts, so they need to be only compiled once in an application's lifetime.

Lock-Free data structures

Traditional databases use locks (latches and enqueues) to manage serialization and they run into issues such as deadlocks or priority inversion, when processes block each other as they complete execution. This results in performance and scalability issues with increasing volumes of concurrent read and write operations.

Lock-free data structures that enable better scalability and performance are at the core of MemSQL's engine. Every component of the engine is built using lock-free data structures including queues, stacks, hash tables, skip lists, and linked lists. For superior memory management and efficiently managing transaction state, lock-free queues and stacks are used throughout the system. In the area of code generation, lock-free hash tables are used to map query shapes to the compiled plans in the plan cache. MemSQL also implements lock-free skip lists and hash tables that are kept in memory for fast, random access to data. Lock-free skip lists are the primary index-backing data structure in MemSQL. Compared to B-trees for disk-based databases, skip lists perform extremely well in-memory and under high concurrency, thereby delivering better scalability.

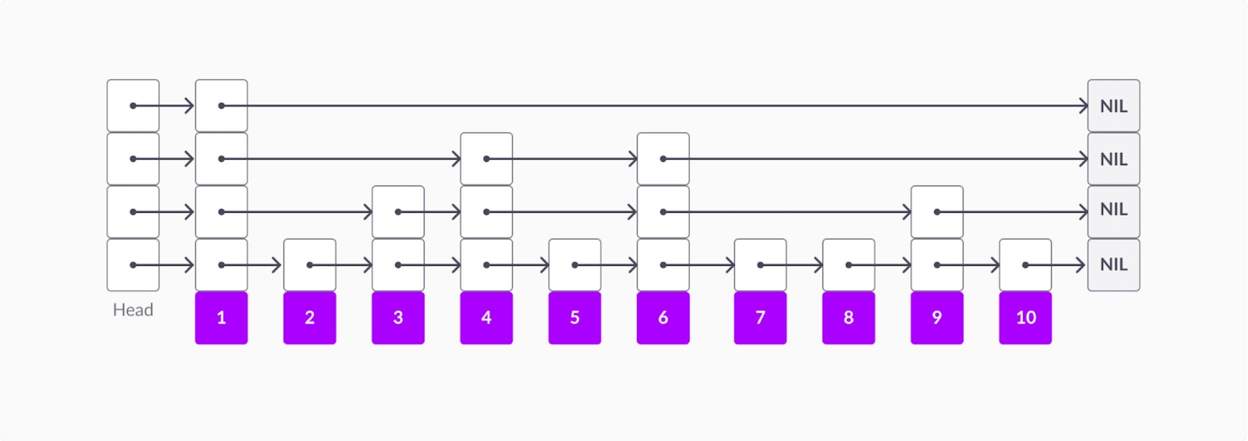


Figure 4. Skiplist index

Multi-Version Concurrency Control

MemSQL favors parallelism and uses multi-version concurrency control (MVCC) to prevent queries from blocking each other in multi-threaded applications. Today's real-time applications - especially those with high volumes of streaming data, or mixed read and write workloads - cannot tolerate the performance loss that comes with database locking.

By having a concept of versioning, and preserving older versions, MVCC allows the database to reduce the number of read-write conflicts among operations. Versions in MemSQL are implemented as a lock-free linked list. Each time a transaction modifies a row, MemSQL creates a new version that sits on top of the existing one. The new version is visible only to the transaction that performed the write - when accessing the same row, read queries see the old version.

Modified rows are queued for garbage collection "behind the scenes," so that old versions are efficiently cleaned up, without the need for a full-table scan. MVCC delivers efficiency and consistency across transactions. A lock in MemSQL only occurs in the case of a write-write conflict on the same row. MemSQL takes a row level lock in this case because it's easier to program around - the alternative would be to fail the second transaction, which requires the programmer to resolve the failure.

Implementing lock-free data structures with MVCC enables MemSQL to avoid locking on both reads and writes when updating tables. As a result, writes can operate at greater throughput, while a large number of concurrent reads happen simultaneously. Since reads and writes never block one another, this minimizes query stalls and allows for greater parallelism - concurrent threads can modify the same object, and even if one thread stalls or stops in the middle of an operation, the remaining threads can carry on processing data.

Multi-version concurrency control summary:

- Every write creates a new version of row
- Commits are atomic
- Old versions are garbage-collected
- Reads are never blocked
- Row-level locking for writes include DELETE
- Allows for online ALTER TABLE

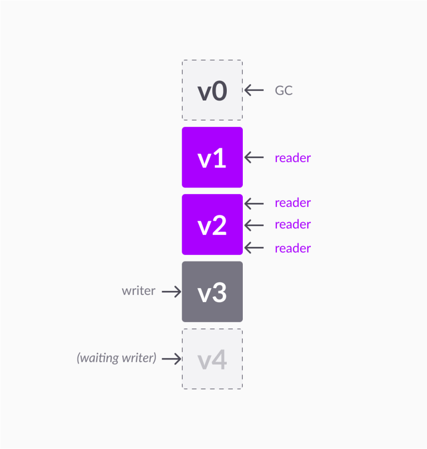


Figure 5. MVCC control summary

Disk-optimized columnstores and memory-optimized rowstore tables

MemSQL supports storing and processing data using two types of data stores: a completely in-memory rowstore and a disk-backed columnstore. Rowstores and columnstores differ both in storage format (row vs. column) and in storage medium (RAM vs. disk). MemSQL allows querying rowstore and columnstore data together in the same query.

The rowstore is typically used for highly concurrent online transaction processing (OLTP) and mixed OLTP/analytical workloads. The whole data set is kept in memory - providing fast writes and supporting thousands of concurrent queries.

Rowstores uses a transaction log to avoid disk I/O bottlenecks on writes. Transactions are committed to disk as logs and periodically compressed as snapshots of the entire database.

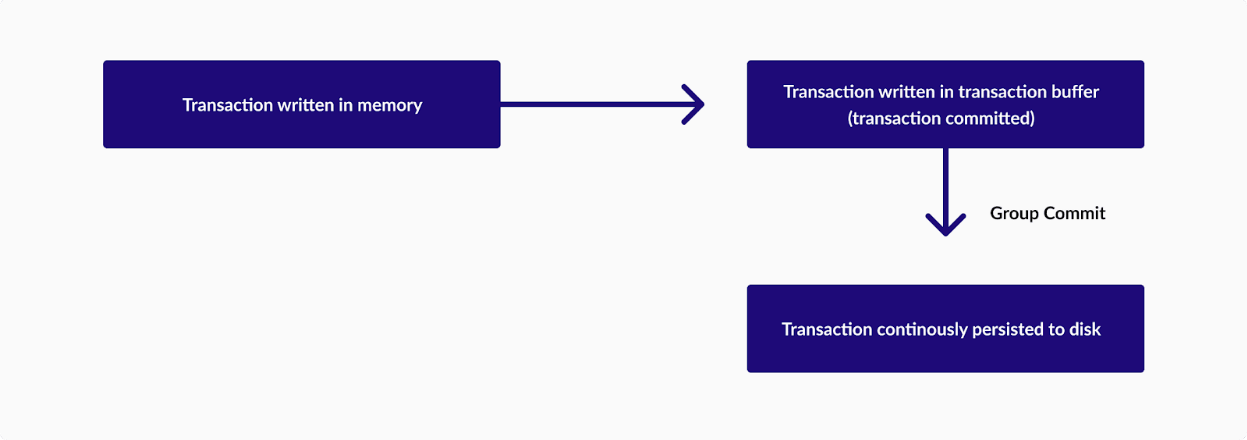

Transaction written in memory
Transaction written in transaction buffer (transaction committed)
Group Commit
Transaction continously persisted to disk

Figure 6. MemSQL designed for durability

To restore a database, MemSQL loads the most recent snapshot and replays remaining transactions from the log. The granularity of logging and frequency of snapshots are both configurable. Because MemSQL only writes logs and snapshots to disk, all disk I/O is sequential. In-memory writes are serialized into a transaction buffer. A background process pulls groups of transactions and persists them to disk.

MemSQL's disk-based columnstore features up to 80% compression and is capable of storing petabytes of data. Columnstores are optimized for complex queries over large data sets that don't fit in memory. The user determines whether tables are stored as rowstores or columnstores at table definition time.

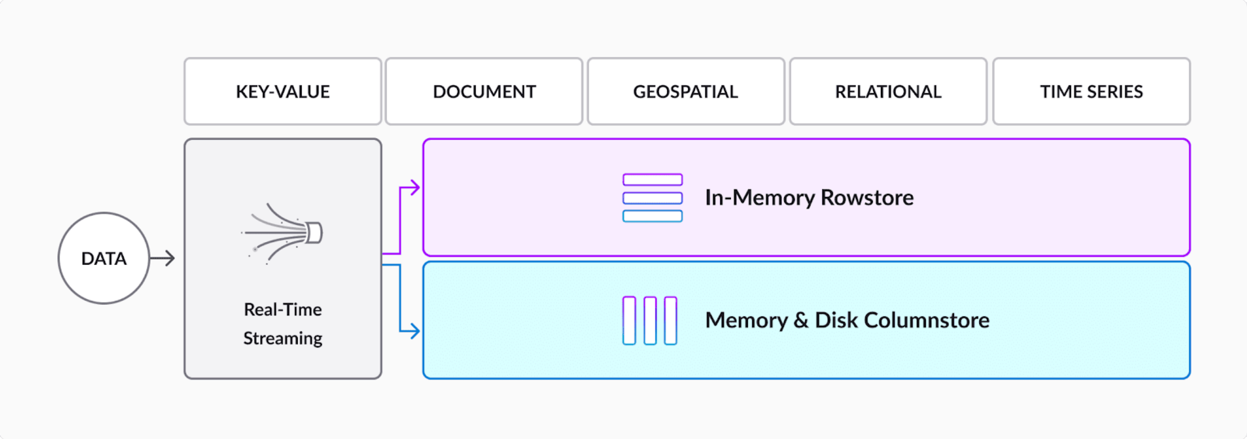


Figure 7. Rowstores and columnstores in MemSQL

MemSQL supports indexing on Rowstore (Skiplist and Hash indexes) and Columnstore (Hash indexes) to efficiently retrieve rows as needed.

Distributed Query Processing

MemSQL supports fast distributed query processing, with a query optimizer that is fully aware of data distribution, and a query execution system that takes advantage of compilation and vectorization , achieving 10X to 100X performance gains. The following diagram illustrates MemSQL query processing at a high level.

Query Optimization

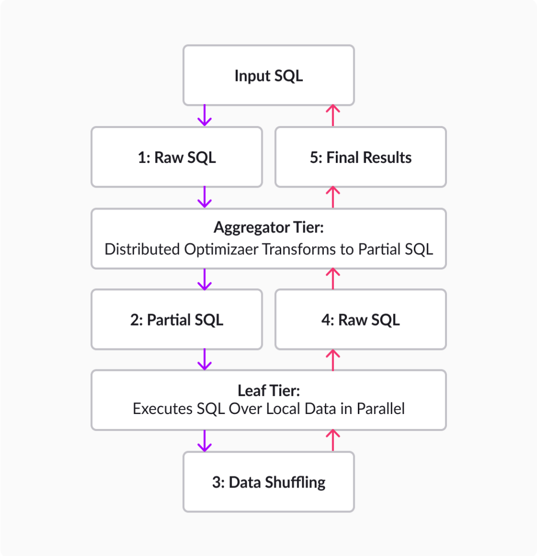


Figure8. Query optimization

The MemSQL Query Optimizer uses search and heuristics, driven by cost models based on statistics, to find high-quality distributed query execution plans. The optimizer is fully aware of data distribution and can use broadcast, shuffle, local-global aggregation, semi-join reduction, and co-located join operations to solve queries with limited and judicious use of data movement across the cluster. Statistics and summary information available to the optimizer include distinct count information, histograms, and high-quality random samples of data from both row store and column store tables.

The MemSQL Query Optimizer is a modular component in the database engine. The optimizer framework is divided into three major modules:

(1) Rewriter: The Rewriter applies SQL-to-SQL rewrites on the query. Depending on the characteristics of the query and the rewrite itself, the Rewriter decides whether to apply the rewrite using heuristics or cost; the cost being the distributed cost of running the query. The Rewriter intelligently applies certain rewrites in a top-down fashion while applying others in a bottom-up manner, and also interleaves rewrites that can mutually benefit from each other.

(2) Enumerator: The Enumerator is a central component of the optimizer, which determines the distributed join order and data movement decisions as well as local join order and access path selection. It considers a wide search space of various execution alternatives and selects the best plan, based on the cost models of the database operations and the network data movement operations. The Enumerator is also invoked by the Rewriter to cost transformed queries when the Rewriter wants to perform a cost-based query rewrite.

(3) Planner: The Planner converts the chosen logical execution plan to a sequence of distributed query and data movement operations. The Planner uses SQL extensions called RemoteTables and ResultTables to represent a series of Data Movement Operations and local SQL Operations using a SQL-like syntax and interface, making it easy to understand, flexible, and extensible.

Query Execution

MemSQL query execution technology tends to be superior overall to query execution technology in legacy database systems, sometimes by up to a factor of 10 or more. MemSQL's query execution technology is thus often a motivating factor to move applications to MemSQL to get lower TCO, a better user experience, or enable applications that were not feasible before. MemSQL parameterizes queries, compiles them, and stores them in a plan cache. On subsequent executions, MemSQL takes a query plan from the cache and runs it so it need not be compiled again.

Unlike established database products, MemSQL compilation translates a query all the way to machine code. This, combined with in-memory row store storage structures designed with code generation in mind, allows query processing rates on the order of 20 million rows per second per core against an in-memory skip list row store table. That is about 10X faster than the per-core processing rate for scans of the B-tree indexes found in most legacy databases, in many cases.

For columnstore tables, MemSQL uses a high-performance vectorized query execution engine that can operate on blocks of 4K rows at a time, very efficiently. This vectorized execution engine also makes use of single-instruction, multiple-data (SIMD) instructions available on Intel and compatible processors that support the AVX-2 instruction set. Processing rates on columnstore tables are often over 100 million rows per second per core, and sometimes as much as 2 billion rows per second per core when using SIMD and operations on encoded (compressed) data.

MemSQL also supports high-performance data movement for broadcast and shuffle operations. This implementation gets is speed by sending data over the wire in its native in-memory format, so it does not have to be serialized on the sending side or deserialized on the receiving side. Rather, it can be operated on directly after it is received, saving CPU instructions, and thus total execution time.

Query Processing Summary

Together, compiled query plans, flexible storage options, lock-free data structures, MVCC and a mature optimizer allows for better performance on multiple cores with high data accessibility, even under high concurrency. This is part of the "secret sauce" that makes MemSQL different than other data platforms out there.

Core Architecture

MemSQL utilizes a distributed, shared-nothing architecture that runs on a cluster of servers, and leverages memory and disk infrastructure for high throughput on concurrent workloads. No two nodes in a MemSQL cluster share CPU, memory, or disk.

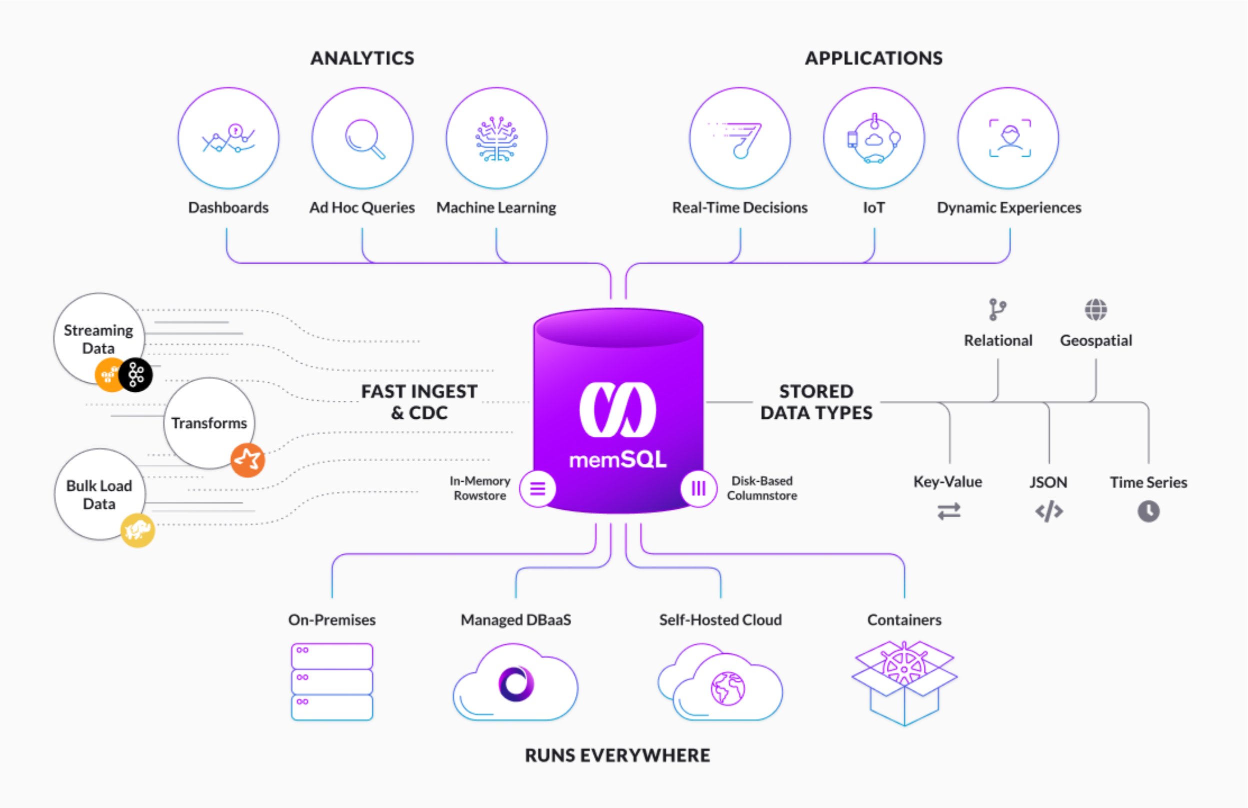


Figure 9. MemsSQL architecture

Our architecture is built for horizontal scalability on commodity hardware, in your data center or in the cloud. MemSQL enables high performance and fault tolerance on large data sets and high-velocity data.

Key Components of a MemSQL Cluster

As shown in Figure 5, a MemSQL cluster consists of aggregator nodes and leaf nodes. The aggregator serves as a query interceptor and router, manages cluster metadata and is responsible for cluster monitoring and failover. A leaf node is a MemSQL server instance that stores data and executes queries issued by the aggregator.

In typical deployments, the aggregator-to-leaf node ratio is generally 1:5. Increasing the number of aggregators can improve operations like data loading and can allow for MemSQL to process more client requests concurrently. Applications serving many clients have a higher aggregator-to-leaf ratio, and those with more demanding storage requirements need more leaves per aggregator.

Client applications connect to an aggregator, which serves as the query router in the cluster. When the client sends a SQL query, the aggregator will parse, compile and distribute the query across the leaf nodes. In the leaf node, MemSQL may further optimize the query as needed and execute on the local store of data. This allows MemSQL to maintain high query performance even with rapidly changing data. The leaf nodes quickly compute the query results and send them back to the aggregator. The aggregator then aggregates the results from each leaf and sends the final result back to the client.

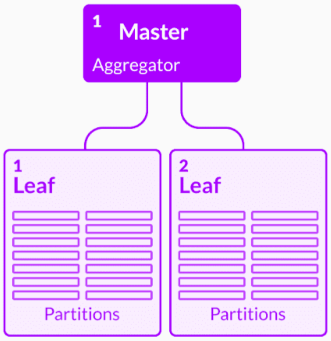


Figure 10. Massively parallel processing for query execution

Database Partitions and Sharding

When a user creates a database in MemSQL, it is always partitioned (a minimum of 2 partitions).

As seen below , a database is a sum of all of its partitions. Partitions reside on the leaf nodes.

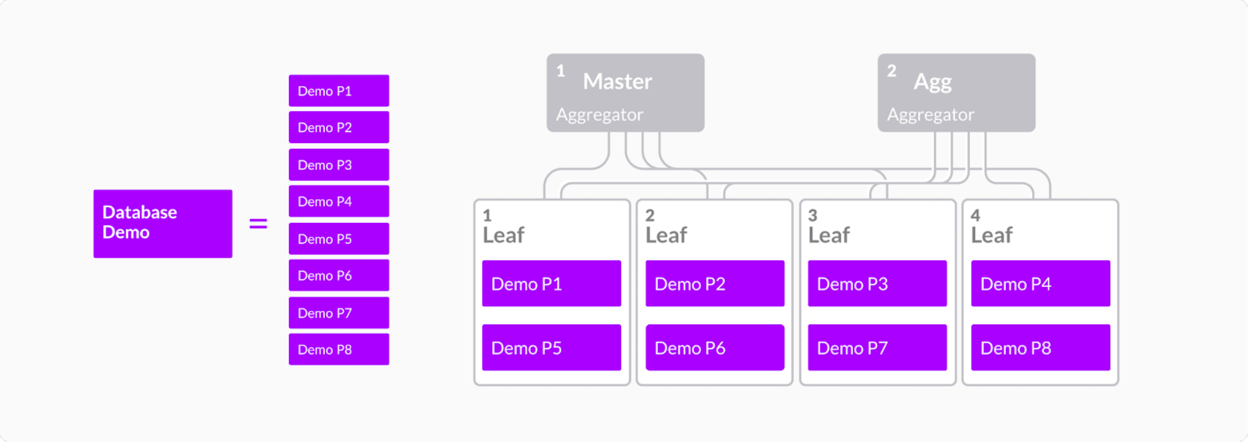


Figure 11. Using leaf nodes as partitions with MemSQL

Each partition in-itself is implemented as a database on a leaf. When a sharded table is created, it is split according to the number of partitions of its encapsulating database. Each partition will hold a slice of the table.

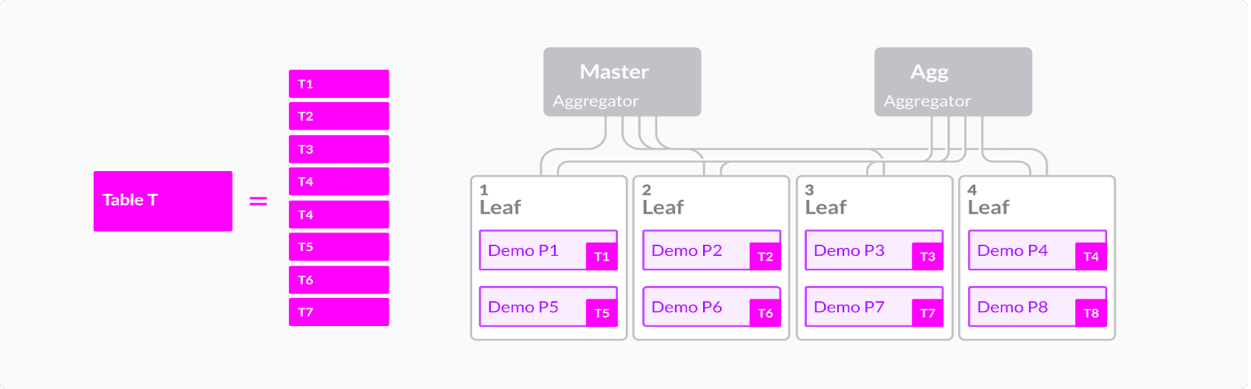


Figure 12. Leveraging shard keys for distributed tables

Sharded and Reference tables

MemSQL supports both distributed (or sharded) and reference (or duplicated) tables. Both table formats can be as rowstore or columnstore tables. For sharded tables, the primary key acts as the hash and each shard is stored on the respective leaf nodes. For reference tables, the table is replicated to all nodes (including aggregators) and is well suited for smaller, slowly changing tables.

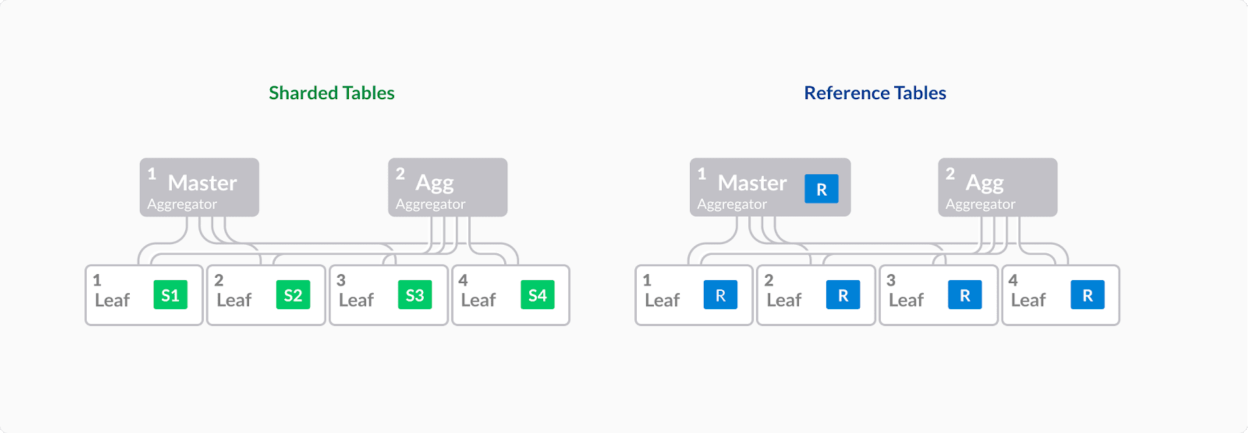


Figure 13. Sharded vs Reference Table designsData Types

MemSQL supports a variety of data types, including integers, timestamp, string types like CHAR and VARCHAR, and compound types such as computed columns, ENUM and SET. Additional complex data types that are supported include geospatial, full text (search capability), and semi-structured JSON data.

Parallel Data Ingest with Pipelines

MemSQL Pipelines is a MemSQL database feature that ingests data from external sources in a continuous manner. As a built-in component of the database, Pipelines can extract, transform, and load external data without the need for third-party tools or middleware.

Pipelines are robust, scalable, highly performant, and supports fully distributed workloads.

Pipelines scales with MemSQL clusters as well as distributed data sources like Kafka, Amazon S3 and HDFS. Pipelines data is loaded in parallel from the data source to MemSQL leaves, which improves throughput by bypassing the aggregator. Additionally, Pipelines has been optimized for low lock contention and concurrency.

The architecture of Pipelines ensures that transactions are processed exactly once, even in the event of failover. Pipelines makes it easier to debug each step in the ETL process by storing exhaustive metadata about transactions, including stack traces and stderr messages.

MemSQL Pipelines Data Flow

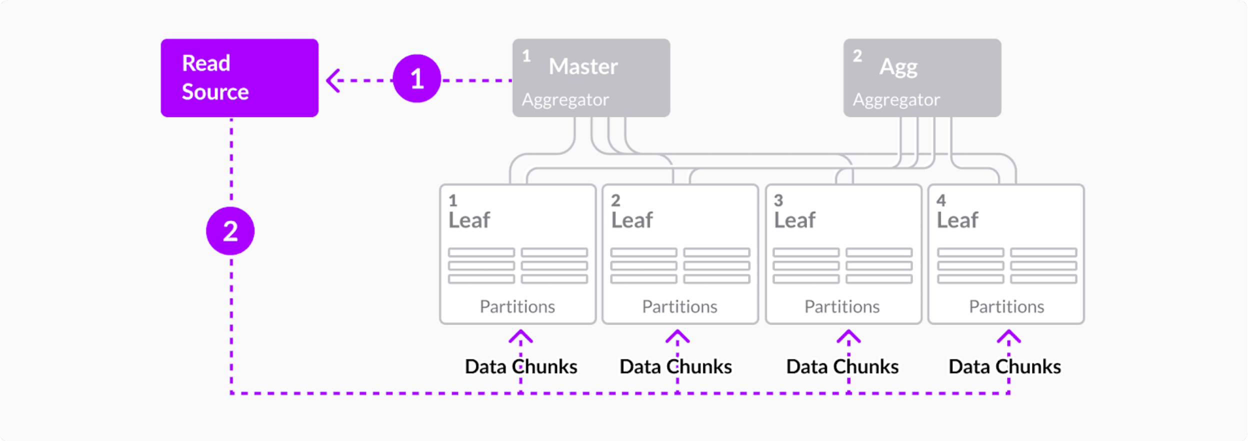


Figure 14. Parallel data ingest in MemSQL using Pipelines

Cluster Management

In this section, we'll go over the key cluster management aspects of MemSQL, including the workload manager, distributed storage, and security.

MemSQL's distributed system allows clusters to be scaled out at any time to provide increased storage capacity and processing power. Sharding occurs automatically and the cluster re-balances data and workload distribution. Data remains highly available and nodes can go down with little effect on performance. As Data is distributed and the cluster is self-healing and elastic, it allows for scale-out/in data processing.

With tiered storage, you can take advantage of MemSQL's memory-optimized rowstore tables for high-speed query processing or ingestion, or disk-optimized columnstore

tables for analytics. MemSQL has a SQL query optimizer that runs on both row-based and column-based tables. This gives you the ability to do transactional processing, analytic processing, or both at once, using the best table structure for each workload.

Dynamic Cluster resizing

MemSQL features powerful but simple cluster management with dynamic cluster operations and no single point of failure. You can add or remove nodes - leaves or aggregators - to the cluster at any time while keeping the cluster online, even while running a workload.

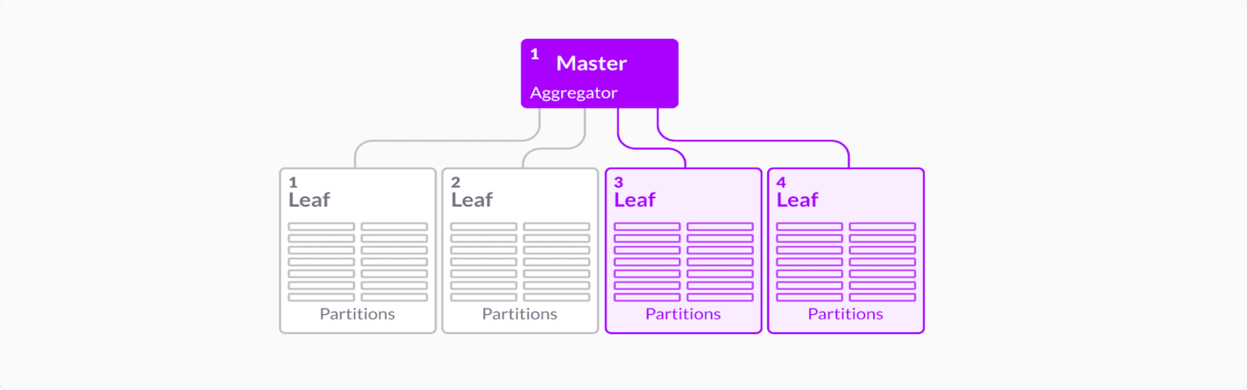


Figure 15. Scaling-out a MemSQL cluster

Data Replication

A MemSQL cluster is resilient to failure with automatic failover and self-healing capabilities. MemSQL allows you to store a redundant copy of data within a cluster. Leaf nodes are organized into availability groups such that each node is paired with a node in the other availability group. Each leaf node has a pair that replicates its data, and can be configured to do so synchronously or asynchronously. In case of node failure, MemSQL restores data and promotes replica partitions to put the cluster back online.

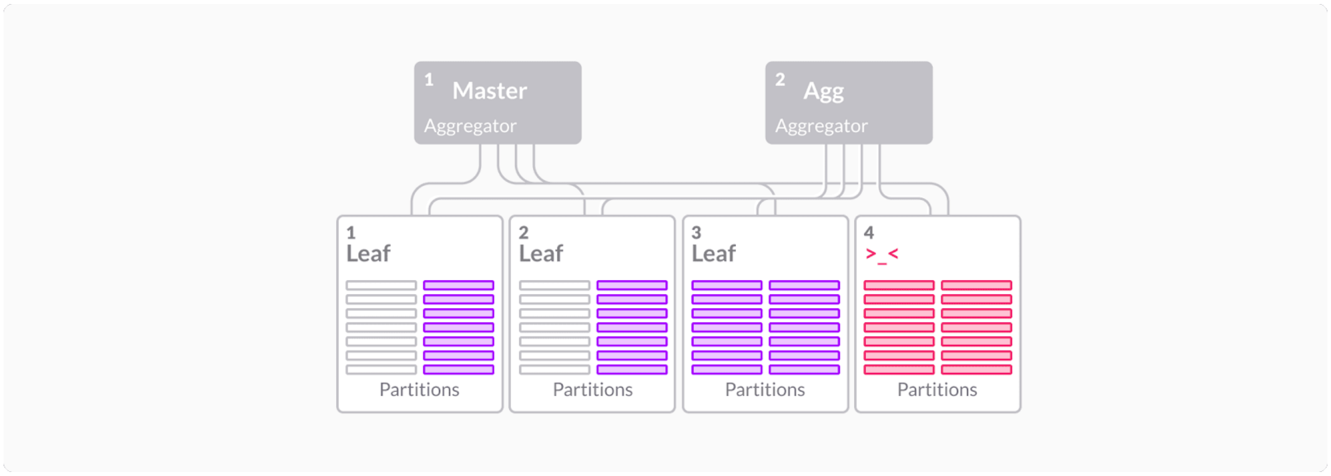


Figure 16. Replicas in MemSQL are promoted to master

MemSQL also supports fully automatic cross-data center replication that can be provisioned with a single command. The replica cluster stores a read-only copy of data asynchronously replicated from the primary cluster. In the event of a major failure in the primary cluster, MemSQL can promote the secondary cluster, immediately making it a "full" MemSQL cluster. In addition to providing disaster recovery assurance, the secondary cluster can also be used for heavy read-only workloads.

Feedback-driven Workload Manager

MemSQL automatically manages cluster workloads functions that limit execution of queries that require fully-distributed execution to ensure that they are matched with available system resources. Using built-in ML functions, workload management intelligently estimates the number of connections and threads needed to execute queries that require reshuffle and broadcast operations, and admits the query only if workload management can assign the necessary resources.

Workload management also estimates the amount of memory required to execute queries and only runs queries if sufficient memory is expected to be available.

Queries that are not immediately executed are queued and are executed when system resources become available. Workload management improves overall query execution efficiency and prevents workload surges from overwhelming the system. It allows queries to run successfully when the system is low on connections, threads, or memory.

Resource pools include the following:

- **Memory Percentage**: This is the percentage of memory resources allocated to the pool
- **Query Timeout**: The number of seconds specifying the time after which a query running in the pool will be automatically terminated
- **Soft and Hard CPU Limit Percentage**: This is the percentage of CPU resources allocated to the pool
- **Maximum Concurrency**: The maximum number of concurrent SQL queries that are allowed to run cluster-wide across all aggregators

MemSQL Security

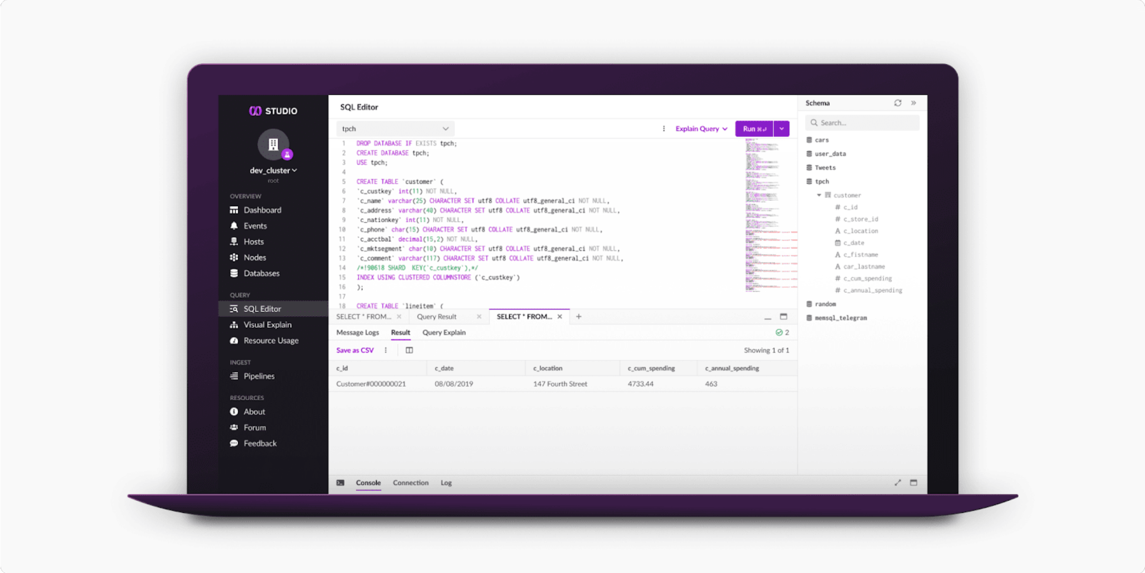
Security is an important aspect of any data platform. To meet regulatory and compliance requirements, MemSQL supports several security features in the areas of authentication, authorization, auditing, and encryption.

Existing account access can be easily managed via PAM (Pluggable Authentication Module), SAML, or GSSAPI (Kerberos) authentication support. MemSQL also implements RBAC to protect sensitive data for tens of thousands of distinct users and their specific access roles.

MemSQL's auditing feature provides configurable database logging to a secure external location to support information security tasks such as tracking user access. Data can be encrypted at ingest time and is distributed across nodes over TLS.

MemSQL Studio

The MemSQL Studio interface lets you monitor, debug and interact with all of your MemSQL Clusters. Designed to be lightweight, easy to deploy, and easy to upgrade, MemSQL Studio provides the tools you need to maintain cluster health without the overhead of complex, heavyweight, and error-prone client software. The built-in query profiler delivers historical usage statistics to shed light on what queries and resources are utilizing the most time. You can visualize and diagnose query bottlenecks and compute resources to ensure optimal performance and availability.



```
DROP DATABASE IF EXISTS tpch;
CREATE DATABASE tpch;
USE tpch;

CREATE TABLE `customer` (
`c_custkey` int(11) NOT NULL,
`c_name` varchar(25) CHARACTER SET utf8 COLLATE utf8_general_ci NOT NULL,
`c_address` varchar(40) CHARACTER SET utf8 COLLATE utf8_general_ci NOT NULL,
`c_nationkey` int(11) NOT NULL,
`c_phone` char(15) CHARACTER SET utf8 COLLATE utf8_general_ci NOT NULL,
`c_acctbal` decimal(15,2) NOT NULL,
`c_mktsegment` char(10) CHARACTER SET utf8 COLLATE utf8_general_ci NOT NULL,
`c_comment` varchar(117) CHARACTER SET utf8 COLLATE utf8_general_ci NOT NULL,
/*!90618 SHARD  KEY(`c_custkey`),*/
INDEX USING CLUSTERED COLUMNSTORE (`c_custkey`)
);

CREATE TABLE `lineitem` (
```

c_id	c_date	c_location	c_cum_spending	c_annual_spending
Customer#000000021	08/08/2019	147 Fourth Street	4733.44	463

Figure 17. MemSQL Studio tool

MemSQL Studio turns user actions into standard SQL queries that are run against your MemSQL Cluster. Results are then displayed back to you in the form of tables and graphics that help you understand your cluster better. Conceptually, MemSQL Studio is a UI on top of the MemSQL database engine itself, pairing the stability and security guarantees of the command line with the ease of use of a visual UI. MemSQL Studio also comes with an in-built visual SQL client, so it can be used safely alongside other tools such as MemSQL Ops.

Cloud-native Support and Managed Service

As a cloud-native database, MemSQL can deploy across hybrid, multi-cloud, and on-premises environments. The MemSQL Kubernetes Operator provides an easy way to deploy and manage data infrastructure on private or public clouds.

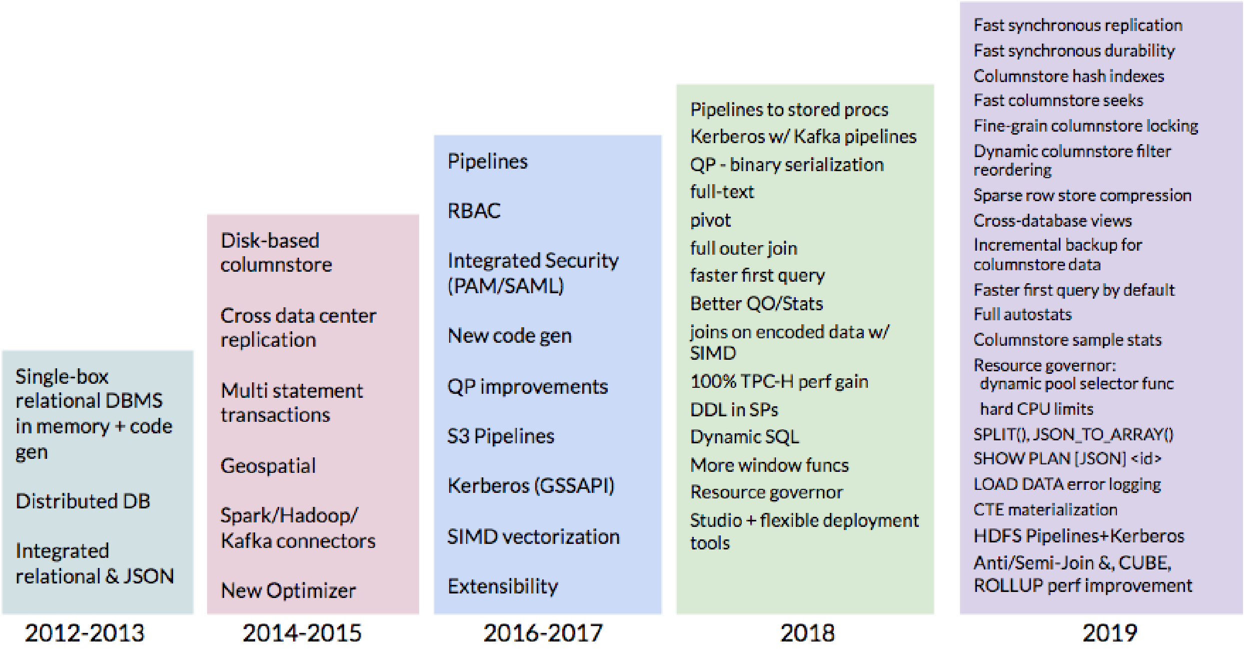
Managing a cluster is simple with the Kubernetes Operator. With the Kubernetes command-line interface (CLI) and the Kubernetes API, the Operator can be used the same way as other standard Kubernetes tools. The commands and operations are the same across all the major clouds, public and private, as well as in on-premises environments. You describe the state of the cluster that you want; Kubernetes creates it, and then maintains that state for you.

MemSQL also provides Managed Services on AWS/Azure/GCP Public Cloud platforms. MemSQL manages all management aspects of the cluster freeing the customer of the necessity to maintain a dedicated staff for administration of the cluster.

MemSQL Innovation History

MemSQL offers a full featured database platform through 6 plus years of innovative engineering. The chart below describes some of the notable advances over time.

MemSQL History and Roadmap



2012-2013

- Single-box relational DBMS in memory + code gen
- Distributed DB
- Integrated relational & JSON

2014-2015

- Disk-based columnstore
- Cross data center replication
- Multi statement transactions
- Geospatial
- Spark/Hadoop/ Kafka connectors
- New Optimizer

2016-2017

- Pipelines
- RBAC
- Integrated Security (PAM/SAML)
- New code gen
- QP improvements
- S3 Pipelines
- Kerberos (GSSAPI)
- SIMD vectorization
- Extensibility

2018

- Pipelines to stored procs
- Kerberos w/ Kafka pipelines
- QP - binary serialization
- full-text
- pivot
- full outer join
- faster first query
- Better QO/Stats
- joins on encoded data w/ SIMD
- 100% TPC-H perf gain
- DDL in SPs
- Dynamic SQL
- More window funcs
- Resource governor
- Studio + flexible deployment tools

2019

- Fast synchronous replication
- Fast synchronous durability
- Columnstore hash indexes
- Fast columnstore seeks
- Fine-grain columnstore locking
- Dynamic columnstore filter reordering
- Sparse row store compression
- Cross-database views
- Incremental backup for columnstore data
- Faster first query by default
- Full autostats
- Columnstore sample stats
- Resource governor:
  - dynamic pool selector func
  - hard CPU limits
- SPLIT(), JSON_TO_ARRAY()
- SHOW PLAN [JSON] <id>
- LOAD DATA error logging
- CTE materialization
- HDFS Pipelines+Kerberos
- Anti/Semi-Join &, CUBE, ROLLUP perf improvement

Figure 18. MemSQL innovation history

Conclusion

Modern enterprises requires a data platform that is versatile, cost efficient, and performant. Not only does the data platform needs to support and improve legacy workloads, but also be able to deliver on new business requirements.

MemSQL is a modern data platform that is well suited to meet today's demanding requirements. It offers an easy migration path from legacy platforms, is cloud-friendly, and supports modern workloads seamlessly.

MemSQL allows for infrastructure convergence, simplicity, and support for predictive capabilities in a cost-effective and highly performant manner.

Appendix: MemSQL Capability Checklist

MemSQL Capabilities	MemSQL
ARCHITECTURE	
Distributed Shared Nothing Scale out Architecture	●
ACID transactions	●
High Availability and Disaster Recovery	●
Lock-free synchronization	●
Multi-version concurrency control	●
Distributed Query execution	●
Deployment flexibility (Multi-Cloud and On-Premises)	●
Compressed columnar table format on disk	●
Row store in memory	●
QUERY	
SQL-92	●
SQL-99 OLAP Extensions	●
SQL-2003 extensions	●
Multi-statement transaction	●

SELECT FOR UPDATE	●
Procedural Language with support for Procedures, UDF, TVF and UDAF	●
Full text search	●
Vectorization and single instruction, multiple data (SIMD)	●
Operations on encoded data	●
Bloom filter pushdown for hash join	●
Local join support (hash, merge, nested loop)	●
Distributed join support (broadcast and reshuffle)	●
Query optimization and Auto Statistics	●
Oracle compatibility extensions	●
Adaptive query compilation and execution	●
Just-in-time (JIT) compilation	●
STORAGE	
Rowstore In-Memory (with compression)	●
Rowstore skiplist indexes In-Memory	●
Columnstore on Disk	●
Columnstore Hash indexes on Disk	●
Sharded and Reference tables	●
Temporary tables	●
DATA TYPES	

JSON	●
Relational	●
Full text	●
Geospatial	●
INGESTION	
Native pipeline for data ingestion	●
LOAD DATA enabling bulk loads	●
Native Parallel Ingest from many data sources (Linux File System, S3, Azure Blob Store, HDFS, Kafka)	●
Supports most popular formats (CSV, Avro, JSON)	●
Load to stored procedures (ELT)	●
Pipelines with transform scripts	●
CLUSTER OPERATIONS	
Zero downtime node management (add/remove)	●
Automatic recovery from node failures	●
Rolling production upgrades	●
Automated full backup	●
Database Replication across Geographies to enable disaster recovery	●
Management and Monitoring tools	●
Online operational capabilities (add/remove nodes/re-balance etc.)	●

Monitoring UI with Visual Explain plan	●
Troubleshooting UI	●
Automated Workload management	●
SECURITY	
Auditing	●
Strict Mode	●
Encryption	●
RBAC	●
Authentication (GSSAPI/Kerberos)	●
ECOSYSTEM	
Looker	●
Zoomdata	●
Tableau	●
Streamsets	●
SAS Access	●
Informatica	●
Data Virtuality pipes	●
Talend	●
Power BI	●
CLIENT DRIVERS	
MariaDB command-line client	●

MariaDB C/C++ connector	●
JDBC	●
ODBC	●
MySQL Connector	●
DBD-MariaDB Perl library	●
MySQL Connector/NET (C# and other .NETlanguages)	●