The Snowflake Elastic Data Warehouse

Author

Benoit Dageville, Thierry Cruanes, Marcin Zukowski, Vadim Antonov, Artin Avanes, Jon Bock, Jonathan Claybaugh, Daniel Engovatov, Martin Hentschel, Jiansheng Huang, Allison W. Lee, Ashish Motivala, Abdul Q. Munir, Steven Pelley, Peter Povinec, Greg Rahn, Spyridon Triantafyllis, Philipp Unterbrunner

Published

14 June 2016 at 2016 ACM SIGMOD International Conference on Management of Data

Nirmala Niraula, Yiwei Gao, Yicheng Liao, Zihao Zhao, Sheikh Munim Tazwar Riddhi

Limitation of Traditional Data Warehouse

- **Fixed Resources:** Limited scalability, designed for static infrastructure.
- **Rigid Data Processing:** Depend heavily on ETL pipelines for structured data.
- **Inflexible Architecture:** Can’t handle unpredictable workloads or semi-structured data efficiently.
- **High Maintenance:** Require significant tuning and physical design to perform well.

What is Snowflake?

Snowflake is a data warehouse built on top of the Amazon Web Services or Microsoft Azure cloud infrastructure and allows storage and computing to scale independently:

- **Cloud-Based Data Warehouse**
- **Unified Data Platform**
- **Modern Solution for Modern Data Needs**

Key Features

Pure SaaS

- No hardware to manage; Snowflake handles updates and maintenance.

Relational Database Support

- Full ANSI SQL and ACID compliance for easy workload migration

Seamless Data Support

- Handles both structured and semi-structured data effortlessly

Elastic Scaling

- Independently scale compute and storage based on demand

Continuous Availability

- Handles node, cluster, and data center failures without downtime

Durability

- Features cloning, undrop, and cross-region backups to safeguard against accidental data loss

Cost-Efficiency

- Pay-as-you-go pricing with data compression for savings

High Security

- End-to-end encryption, role-based access, and robust data protection

Pure Shared-Nothing Architecture

Good for scalability and efficiency

Works well for static and on-premise environments

- **Tight Coupling of Compute and Storage**:
  - Scaling compute requires adding storage, even if unnecessary.
  - Leads to underutilized resources and increased costs.
- **Membership Changes**:
  - Node failures or resizing causes significant data shuffling.
  - Impacts system performance and limits elasticity.

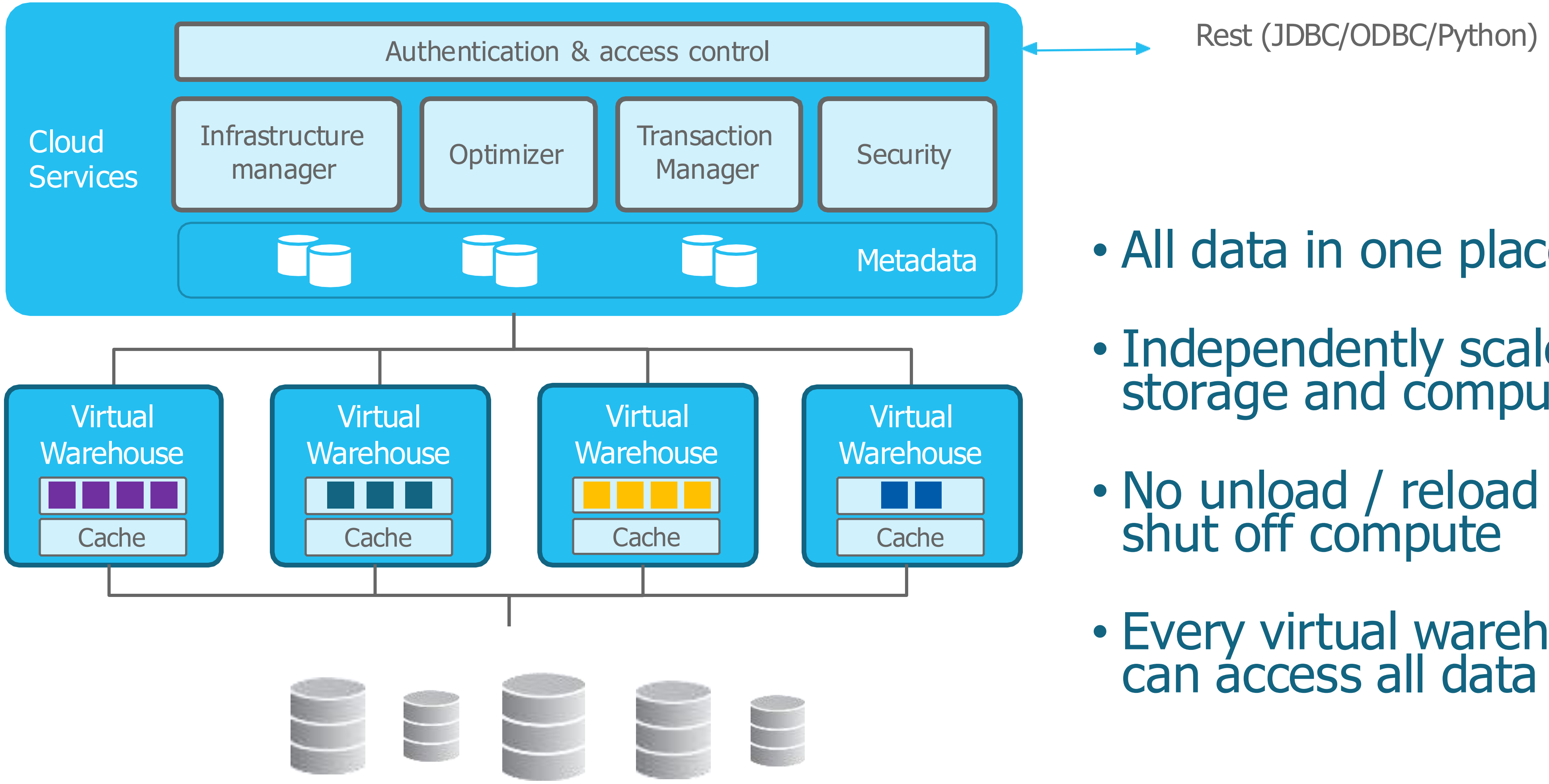
Snowflake's Solution: Separation of Storage and Compute

- **Independent Scaling:**
  - Centralized storage using Amazon S3, unaffected by compute changes.
  - Virtual warehouses that can scale up or down as needed.
- **Elasticity:**
  - Compute resources can be added or paused without moving or changing stored data.
- **Local Caching:**
  - Frequently accessed data is cached on compute nodes for quick retrieval, but permanent data stays in centralized storage.

Benefits of Separating Storage and Compute

- **Flexible Resource Management**:
  - Scale compute up or down based on workload without touching storage.
- **Cost Efficiency**:
  - Pay only for active compute resources; storage costs remain steady.
- **High Availability and Resilience**:
  - Data remains accessible even if compute nodes experience failures.
- **Improved Performance**:
  - No data shuffling required during scaling, ensuring faster response times.
- **Simplicity**:
  - Reduces administrative overhead, making it easier to manage workloads.

Multi-cluster Shared-data Architecture



- All data in one place
- Independently scale storage and compute
- No unload / reload to shut off compute
- Every virtual warehouse can access all data

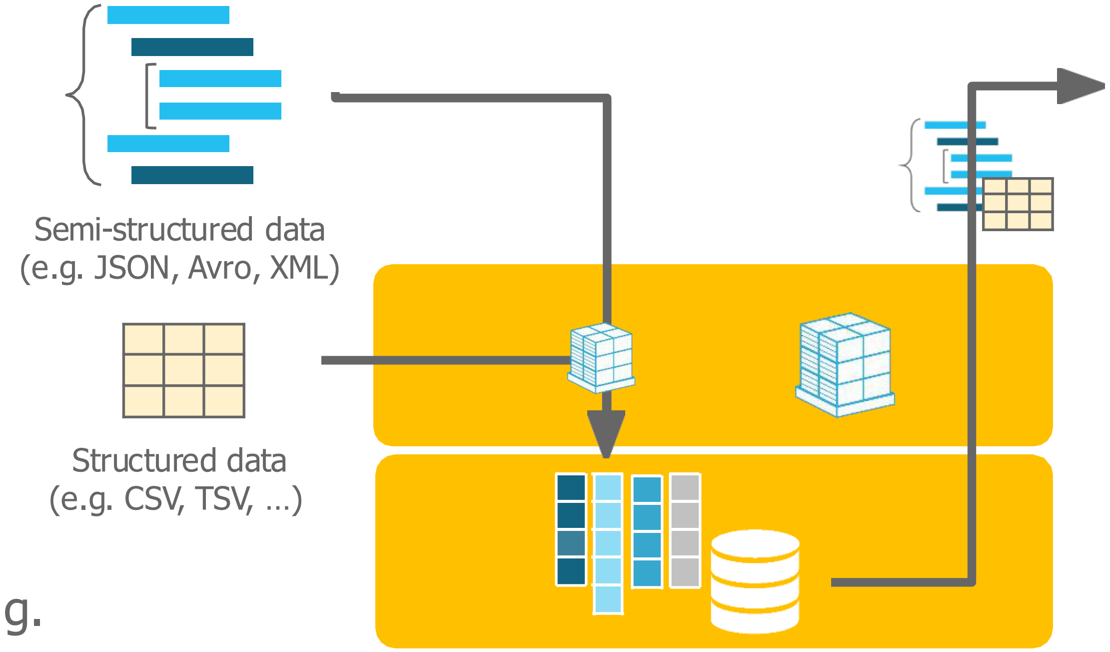
Data Storage Layer

- Stores table data and query results
  - Table is a set of immutable micro-partitions
- Uses tiered storage with Amazon S3 at the bottom
  - Object store (key-value) with HTTP(S) PUT/GET/DELETE interface
  - High availability, extreme durability (11-9)
- Some important differences w.r.t. local disks
  - Performance (sure…)
  - No update-in-place, objects must be written in full
  - But: can read parts (byte ranges) of objects
- Strong influence on table micro-partition format and concurrency control

Virtual Warehouse

- warehouse = Cluster of EC2 instances called worker nodes
- Pure compute resources
  - Created, destroyed, resized on demand
  - Users may run multiple warehouses at same time
  - Each warehouse has access to all data but isolated performance
  - Users may shut down *all* warehouses when they have nothing to run
- T-Shirt sizes: XS to 4XL
  - Users do not know which type or how many EC2 instances
  - Service and pricing can evolve independent of cloud platform

Query Management and Optimization



> SELECT … FROM …

Native support
Loaded in raw form (e.g. JSON, Avro, XML)

Optimized storage
Optimized data type, no fixed schema or transformation required

Optimized SQL querying
Full benefit of database optimizations (pruning, filtering, …)

Concurrency Control

- Designed for analytic workloads
  - Large reads, bulk or trickle inserts, bulk updates
- Snapshot Isolation (SI) [Berenson95]
- SI based on multi-version concurrency control (MVCC)
  - DML statements (insert, update, delete, merge) produce new table versions of tables by adding or removing whole files
  - Natural choice because table files on S3 are immutable
  - Additions and removals tracked in metadata (key-value store)
- Versioned snapshots used also for time travel and cloning

Pruning

- Database adage: The fastest way to process data? Don't.
  - Limiting access only to relevant data is key aspect of query processing
- Traditional solution: B^+-trees and other indices
  - Poor fit for us: random accesses, high load time, manual tuning
- Snowflake approach: pruning
  - AKA small materialized aggregates [Moerkotte98], zone maps [Netezza], data skipping [IBM]
  - Per file min/max values, #distinct values, #nulls, bloom filters etc.
  - Use metadata to decide which files are relevant for a given query
  - Smaller than indices, more load-friendly, no user input required

Feature Highlights

- Expected Features
  - Comprehensive SQL support
  - ACID transactions
  - Stability
- Technical differentiators

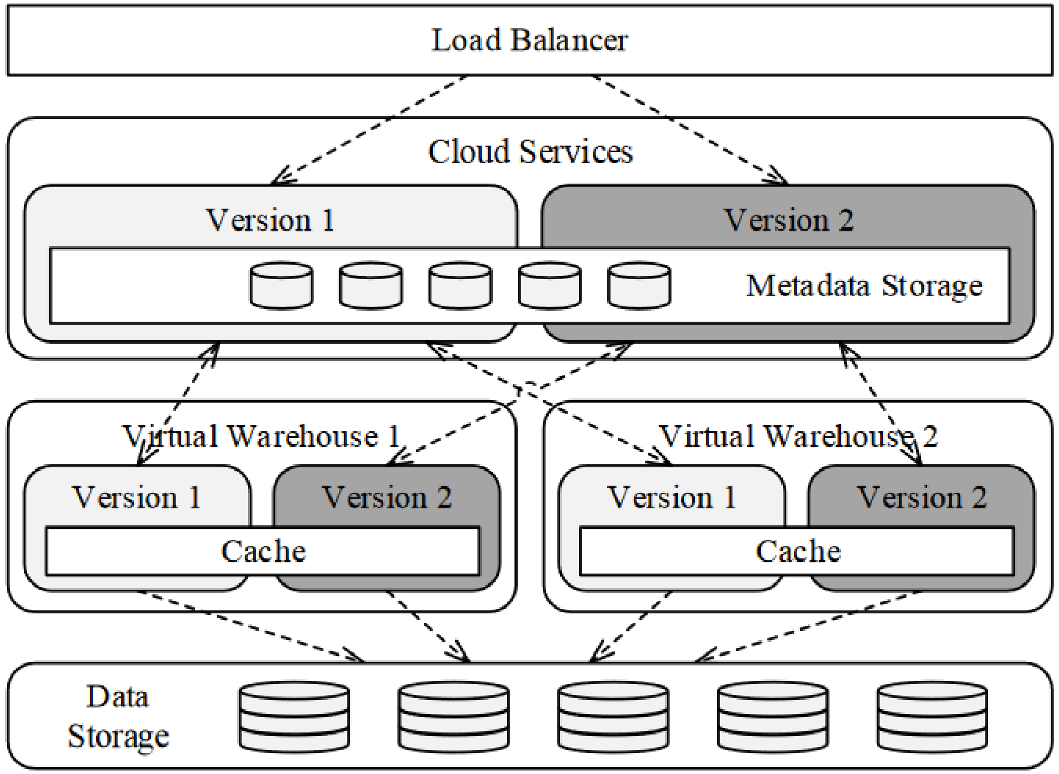
Pure Software-as-a-Service

- Ways to interact with the system
  - Standard database interfaces (JDBC, ODBC, Python PEP-0249)
  - Web browser

Continuous Availability – Fault Resistance

- Data Storage Layer / Metadata Store
  - Uses Amazon S3 storage with multi-availability zone (AZ) replication
  - 99.99% data availability
  - Resilient to full AZ failures
- Virtual Warehouses
  - Located within a single AZ
  - Faults in VWs trigger transparent re-execution or quick replacement of failed nodes
  - Full AZ outages are rare, but they cause VW-based queries to fail

Continuous Availability – Online Upgrade



- New software versions deployed alongside previous versions
- Load balancer directs users progressively to the new version
- Once all queries on the old version finish, older services are decommissioned

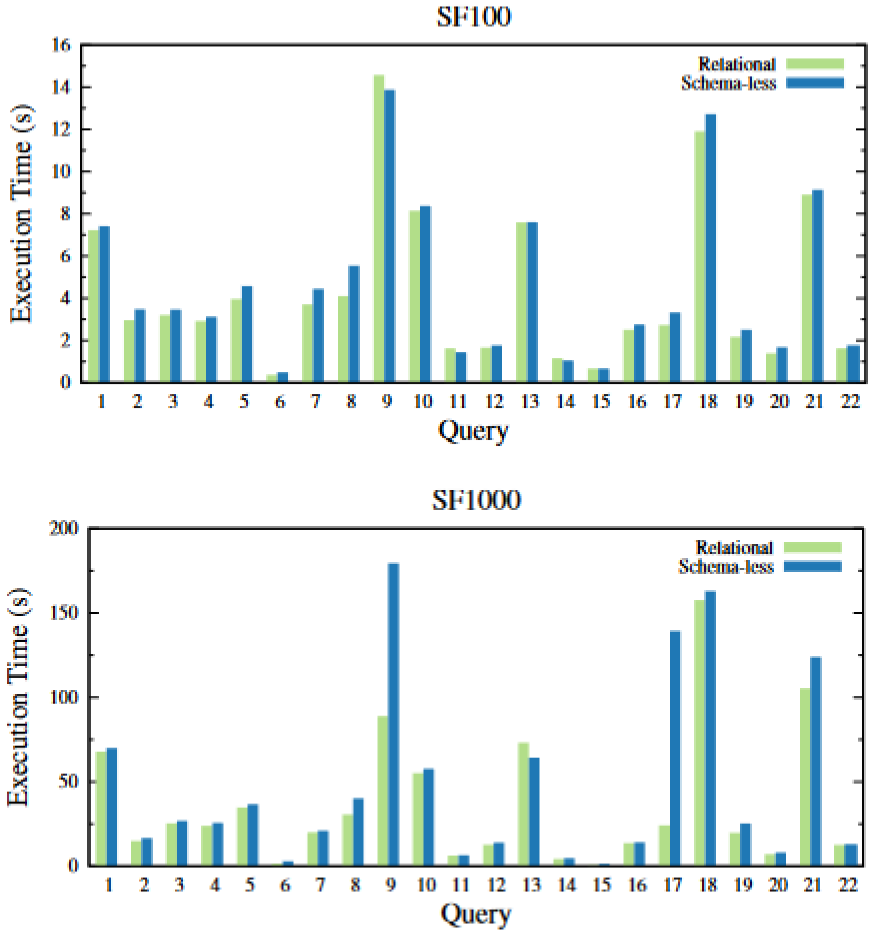
Semi-Structured and Schema-Less Data

- VARIANT
  - any value of native SQL type
  - variable length ARRAYs
  - OBJECTS
- ARRAY
  - arrays of values
- OBJECT
  - JavaScript-like key-value maps

Columnar Storage and Processing

- automatically performs statistical analysis to perform automatic type inference
- corresponding columns are then stored efficiently in columnar format, allowing fast access and materialized aggregates
- Common formats (e.g., JSON/XML) often represent SQL types (e.g., dates) as strings, requiring conversion at write time or read time
- perform optimistic data conversion, and preserve both the result of the conversion and the original string

Performance



- 10% Overhead
- Outliers (Q9, Q17): A bug in join optimization caused additional slowdown for two queries on SF1000

Time Travel

- Write Operations: Insert, update, delete, and merge operations produce new versions by managing entire files
- Removed files retained for a configurable duration
- Perform time travel using the convenient AT or BEFORE syntax
- UNDROP keyword to quick restoration

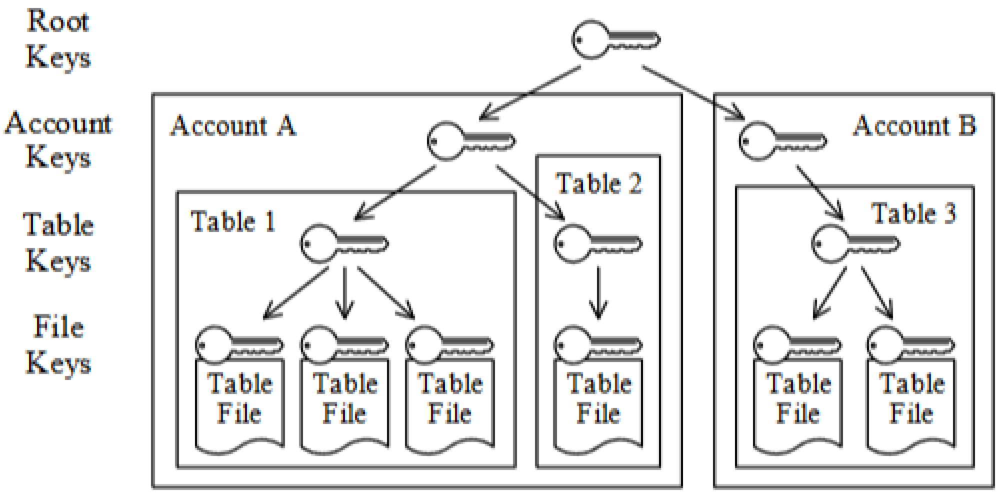
```
SELECT * FROM my_table AT(TIMESTAMP =>
  'Mon, 01 May 2015 16:20:00 -0700'::timestamp);
SELECT * FROM my_table AT(OFFSET => -60*5); -- 5 min ago
SELECT * FROM my_table BEFORE(STATEMENT =>
  '8e5d0ca9-005e-44e6-b858-a8f5b37c5726');
```

```
DROP DATABASE important_db; -- whoops!
UNDROP DATABASE important_db;
```

Cloning

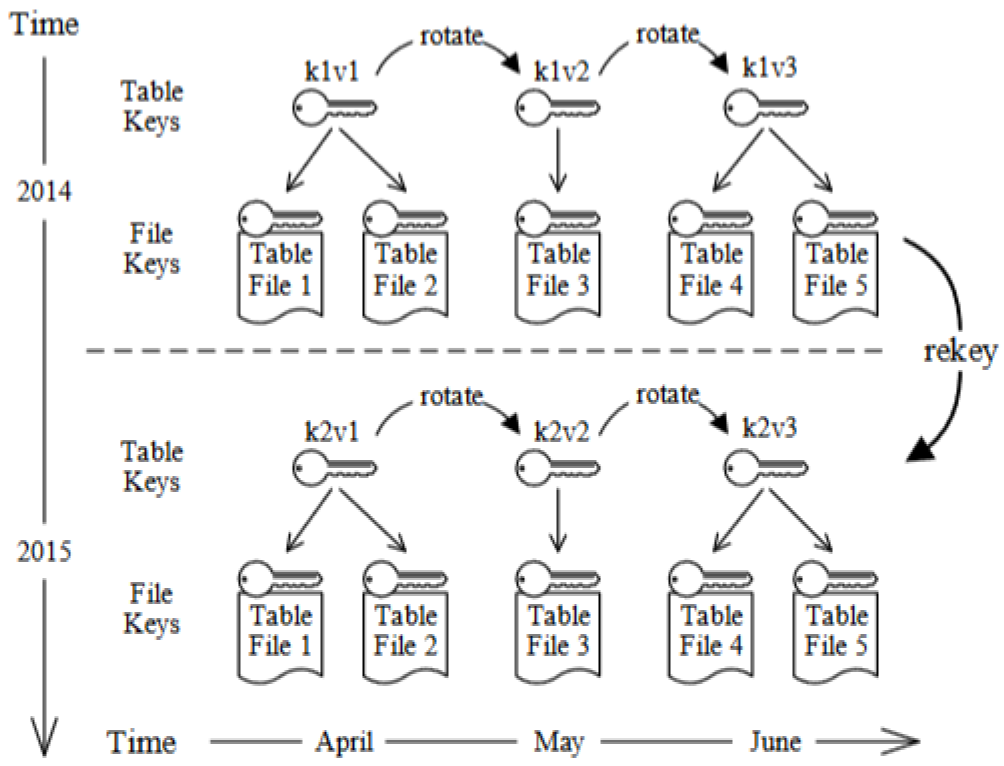
- New keyword CLONE
- Creates new tables, schemas, or databases without physical copies
- Copies metadata, not data files
- Clone to a specific past version using AT or BEFORE

Security - Key Hierarchy



- **AES 256-bit encryption** with a hierarchical key model rooted in AWS CloudHSM
- **Four levels**: root keys, account keys, table keys, and file keys.

Security - Key Life Cycle



- limits the key-usage period using key rotation and rekeying
- **Key rotation** - creates new versions of keys at regular intervals
- **Rekeying** - the process of re-encrypting old data with new keys.

Related Works

- Cloud-based Parallel Database Systems

Feature	Snowflake	Amazon Redshift	Google BigQuery	Azure Warehouse
Architecture (Scaling)	Multi-cluster, shared data (Instantly scale, pause, or resume compute w/o data removement)	Shared-nothing (Scalable but requires data reshuffle)	Parallel query service (Not support full SQL)	Separate storage and compute (Scalable but limits concurrent queries)
Physical Tuning	No manual tuning required	Requires user tuning	Not specified	Requires user tuning
Semi-Structured Data (e.g., JSON)	Native support, optimized	JSON as text only	Supports JSON, limited SQL compatibility	Limited support via external integration
Data Operations	Full Data Manipulation Language (DML), ACID transactions w/o schema definitions	Supports DML	Append-only and require schemas	Limited DML with concurrency cap (<32)

Challenges & Lesson Learnt

- **Initial Architecture Challenges:** Building against the trend (SQL on Hadoop)
  - **Lesson:** Sometimes going against market trends pays off when based on fundamental user needs.
- **Implementation Missteps:** Oversimplified relational operators, Delayed datatype implementation, Postponed resource management
  - **Lesson:** Early architectural decisions have long-lasting impacts
- **Multi-tenancy Challenges:** Complex metadata layer for concurrent users, Node and network failures, Security across multiple dimensions
  - **Lesson:** SaaS architecture creates unique operational complexities

New Challenges & Future Work

- **Users continuously push larger workloads**
  - Need for better skew handling and load balancing
- **More complex queries than anticipated**
  - Requires ongoing optimization without adding complexity
- **Unexpected rapid adoption of semi-structured data**
  - Need to maintain efficient processing while supporting various data formats
  - Balancing SQL functionality with semi-structured data capabilities
- **Users want zero-touch operations**
  - Need for automated support systems
  - Automated performance optimization
  - Enhanced security automation

Conclusion

Main Contribution

- Cloud-Native Architecture
- Flexible Data Support
- Fully Managed SaaS

Limitations

- High Complexity in Multi-Tenancy
- Resource Management Constraints

Future Work

- Improving Skew Handling and Load Balancing
- Enhanced Support for Semi-Structured Data
- Advancing Zero-Touch Operations

Study Questions

1. Explain the rationale behind Snowflake's separation of storage and compute layers. How does this architectural decision address specific limitations in traditional data warehousing solutions, and what unique benefits does it provide for cloud environments?

2. Discuss the role of Snowflake's semi-structured data support, including the VARIANT data type and columnar storage optimization. How does this feature impact data loading, query performance, and flexibility compared to conventional ETL processes in data warehouses?

Thank You!

Any Questions?