# Data Modeling with Snowflake: A Practical Guide to Accelerating Snowflake Development Using Universal Data Modeling Techniques

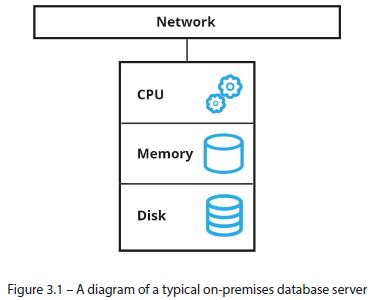
## Part 1: Core Concepts in Data Modeling and Snowflake Architecture

### Chapter 3: Mastering Snowflake’s Architecture

* For as long as databases have existed, they have faced recurring challenges in managing **concurrency** + **scalability** in the face of growing data volume + processing demands
* Many innovative designs have been attempted over the years + have been met w/ varying degrees of success
* However, that success often came w/ fresh drawbacks
* The Snowflake team saw that overcoming the age-old challenges of handling independent consumption demands of data storage + analysis required a radically new approach
* The team decided to **design a database that could operate natively on top of cloud computing platforms + thereby offer near-limitless scalability**
* Their efforts resulted in the creation of what Snowflake calls the **Data Cloud**, **a platform that enables real-time data sharing + on-demand workload sizing through the separation of storage and compute**
* **Main Topics**
* Explore how databases have tried to achieve database scalability in the past
* Discover how Snowflake built a **scalable database to run natively in the cloud**
* See the unique features made possible by Snowflake’s innovative architecture
* Understand the costs associated w/ Snowflake’s variable spend mode
* Learn how to utilize various forms of **cache** to save on costs + improve performance
* To truly appreciate what Snowflake has achieved, it is worth remembering the various traditional architectures that came before it + the limitations they tried to overcome

#### Traditional Architectures

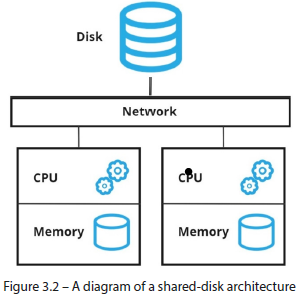
* To appreciate the innovation of the Snowflake **Data Cloud**, take a step back + recall the designs + related limitations associated w. its predecessors
* Long before the advent of the cloud, databases started as physical on-premises appliances and, since their inception, have all faced the same challenge: **scalability**
* In the past, databases were confined to a *physical* server on which they relied for storage and processing power
* As usage increased, memory would fill up, + CPU demand would reach the available limit, forcing the user to add more resources to the server or buy a new one altogether
* As either response involved maintenance + downtime, hardware purchases had to be forward-looking, anticipating database growth several years into the future
* The following figure outlines the structure and key pieces of a traditional database



* Although processing power, memory, + disk space were all customizable to a degree, they came packaged in a physical machine *that could not scale*
* Before cloud architecture unlocked the **software as a service** (**SaaS**) model + introduced **variable-spend pricing**, hardware purchases required considerable **capital expenditure** (**CapEx**)
* Thus, hardware sizing (i.e., estimating current + future computing needs) requires careful planning
* A conservative estimate resulted in a sooner-than-expected upgrade, while overshooting meant paying top dollar for cutting-edge tech + not using it to its full potential for years to come
* While data volume + CPU demands tended to grow synchronously w/ a business, unpredicted usage spikes in storage or compute were also common + led to similar headaches
* That is when **new architectures** such as **shared-disk** and **shared-nothing** emerged to address the limits of scaling physical machines.

##### Shared-Disk Architecture

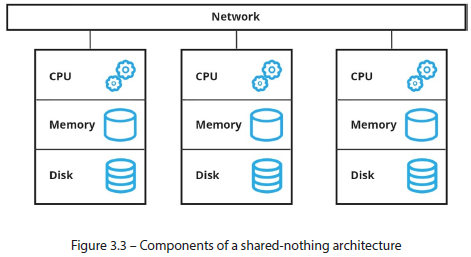
* The **shared-disk approach uses a central storage location + makes it available to various compute clusters in a network**
* It is a simple design that, unfortunately, **suffers from physical implementation issues when locking and accessing the data *concurrently***
* The following figure illustrates how **shared-disk architecture** **externalized data storage** away from individual machines + made it accessible to all computing clusters on the network



* Shared-disk **made it possible to add compute clusters of varying sizes to accommodate different workloads w/in the organization**
* However, this **led to an inevitable bottleneck**, where the **more clusters added to the network in a shared-disk architecture, the worse the contention for the central disk becomes**
* This **design failed to deliver on the promise of scalability b/c it suffered from bottlenecks on the most important resource of a database: the data**
* Where shared-disk failed, shared-nothing *nearly* succeeded

##### Shared-Nothing Architecture

* **Shared-nothing architectures**, like the ones used by AWS Redshift and Greenplum, **avoided the issues of shared-disk by isolating it + making it part of the compute cluster**
* It **also addressed the varying consumption needs of teams across the organization, allowing database clusters to be sized based on demand**
* However, **sharing nothing is not the best strategy for collaboration**
* The following figure shows how shared-nothing architecture allows self-sufficient but isolated database clusters:



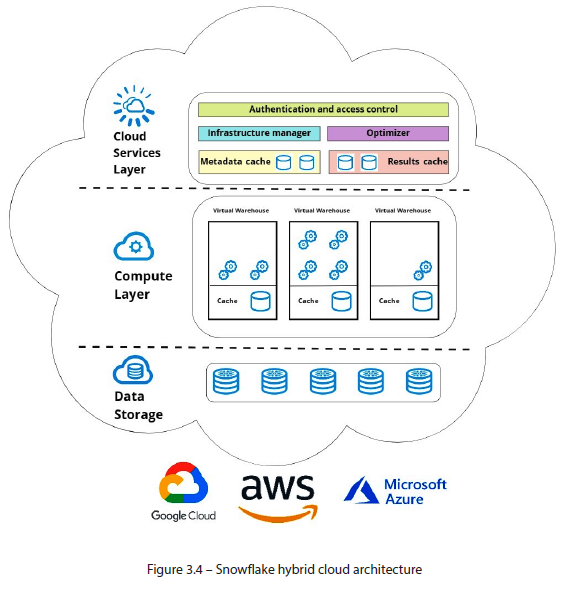
* Creating database clusters in a **shared-nothing architecture**, especially one that runs on virtual cloud resources, **solves the issue of investing up front in physical hardware**
* *However*, **nodes in a shared-nothing architecture require data transfer to share information, which penalizes performance**
* Tying the disk to the overall cluster *also* means **striking the right balance between storage and processing patterns, which can vary independently**
* A heavy data science workload might require lots of compute to be directed at a relatively small dataset, while an ETL bulk-loading process might need the opposite
* However, **shared-nothing platforms did not offer many options for tuning each resource individually, nor were they simple to maintain + administer**
* While shared-nothing architecture did not quite live up to the promise of seamless + easily managed scalability, it paved the way for platforms such as Snowflake to tackle cloud computing challenges with **cloud-native designs**

#### Snowflake’s Solution

* To address the scalability issue that has plagued databases since inception, the Snowflake team decided to formulate a *new* approach that would not be tied down by the limitations of past designs
* They **developed a modern platform built natively for the cloud that uses unique features to enable concurrency, scalability, + real-time collaboration**
* Snowflake’s innovative cloud **architecture** **still relies on physical disks**, but it **integrates them logically to make centralized storage available to its computing clusters *w/out* concurrency bottlenecks or data replication overhead**
* Finally, the best of what shared-disk + shared-nothing promised: **separating the data from compute workloads, which can be independently provisioned + resized**
* **Snowflake runs *entirely* on *virtually-provisioned* resources from cloud platforms** (Amazon, Microsoft, Google Cloud, etc.) **+ handles all interactions w/ a cloud provider transparently, abstracting the underlying virtual resources + letting customers manage their data through a unified 3-layer architecture**

#### Snowflake’s 3-Tiered Architecture

* **Snowflake architecture consists of 3 layers: storage, compute, and cloud services**
* It manages all 3 layers so that **interactions** w/ the underlying cloud architecture **are transparent to the users**
* The following is an illustration of how **Snowflake’s architecture runs on top of cloud data platforms and separates disk from virtual compute clusters while managing a separate operational services layer** so the user doesn’t have to:



##### Storage Layer

* The **storage layer** ***physically* stores data on disks in the cloud provider hosting a Snowflake account**
* As **data is loaded** into Snowflake, it is **compressed**, **encrypted**, **+** **logically organized into tables**, **schemas**, **+** **databases**
* **Users define the logical hierarchy of the database + its objects**, while **Snowflake takes care of the underlying partitioning + storage**
* A customer is **billed only for the data they store, *w/ no provisioning or sizing required***
* Later chapters go into more detail on how Snowflake manages + optimizes data storage + the types of backup + redundancy options it offers
* For now, let us understand how **compute** + **virtual warehouse** **clusters** work in Snowflake

##### Compute Layer

* The **compute layer** (AKA **processing layer** or **virtual warehouse layer**) **provides a cluster of virtually-provisioned CPU + temporary memory resources for executing queries**
* Such **clusters are called warehouses**+ are **provisioned** in t-shirt sizes (from XS to 6XL benchmarks)
* Through warehouses, Snowflake simplifies the consumption of virtual cloud resources using a simple formula: **Each increase in warehouse size doubles the number of virtual servers in the cluster *while also doubling the cost***
* The **packetization of virtual compute clusters into warehouses, w/ simple t-shirt sizes, abstracts the underlying cloud architecture and ensures simple + consistent utilization** no matter the platform on which the Snowflake account is hosted
* This way, **warehouses can scale in a way that’s simple to track**
* **Scaling up (*resizing* a warehouse to the next size) doubles the number of servers in the cluster**
* **Scaling out** ***adds warehouses* of the same size to a compute cluster for increased concurrency**
* Think of scaling up or out as compute *times 2* or *plus 1*, respectively

##### Services Layer

* Snowflake’s **services layer coordinates all account activity in this hybrid-layer model + manages everything from security to encryption to metadata**
* It **handles operations such as query parsing + optimization, data sharing, + caching**
* From login, the services layer is there for the user through every operation they perform in the warehouse
* The amount of **time-saving work + automation** the services layer provides is reflected in Snowflake’s marketing language: **promising users *near-zero maintenance* along w/ low cost + exceptional performance**

#### Snowflake’s Features

* W/ its revolutionary cloud architecture, Snowflake continues to innovate + surprise users w/ game-changing performance enhancements beyond those that made it famous from its inception
* While this is by no means a comprehensive list, the following sections highlight some of the most exciting and relevant features when it comes to data modeling

##### a) Zero-Copy Cloning

* **Zero-copy cloning** allows Snowflake users to **clone data *w/out physically duplicating it***
* **Not having to actually move data** means **cloning happens *instantly*** *(*whether cloning a table or an entire database)
* **Cloned objects =** ***virtual copies*** of their source, so they **do not incur storage costs**
* ***NOTE: Once data changes occur* in the clone *OR* its source, the clone becomes a physical object and begins consuming storage resources**
* **Cloning is an ideal way to create system backups + testing environments**, achieving in seconds what used to take days
* At the *object* level, **cloning is a convenient way to bring data across environments when developing or debugging**

##### b) Time Travel

* This is another enhancement made possible through innovative **columnar storage techniques**
* Imagine the ability to go back in time, to before someone dropped that table in PROD, thinking it was TEST
* With **Time Travel, users can query or clone data from a previous point in time, allowing them to recover from mistakes or compare changes from the present state**
* **Time Travel** is **a built-in backup feature for Snowflake objects and, as such, *uses extra storage + comes w/ the associated storage costs*** compared to objects that have this feature disabled
* You **can configure Time Travel to balance cost + flexibility**

##### c) Hybrid Unistore Travels

* **DWs run OLAP architectures to enable them to efficiently perform the massive aggregations and calculations that analytic workloads require**
* *However*, **use cases that depend on *single-row* operations or indexes perform better in *OLTP* databases**
* Snowflake Summit 2022 introduced the **Hybrid Unistore**, a new kind of table to give users **the best of both worlds**, was announced
* **Hybrid Unistore tables** **are really 2 tables in one**: **an analytical *and* a transactional table working together under the hood to provide users w/ a unified solution that offers the best of both worlds**
* Like all Snowflake objects, Hybrid Tables **live up to the promise of near-zero maintenance by having the services layer automatically route workloads to the underlying table best suited for the task**
* See:
* <https://www.snowflake.com/blog/introducing-unistore/>
* <https://www.snowflake.com/en/data-cloud/workloads/unistore/>

##### d) Beyond Structured Data

* Snowflake is known for its **benchmark-setting performance abilities w/ relational tables + agility in working w/ semi-structured data**
* Even *unstructured* files are supported through **external tables**
* **Users can work w/ any of these file formats from a single place in a way that is as familiar and performant as querying a table**
* **Semi-structured files (such as logs + hierarchies) stored in JSON or similar formats can be loaded into Snowflake + *queried like regular tables w/ almost no impact on performance***
* Snowflake provides an **extensive library of functions that allow users to work with semi-structured records or flatten them into structured columns** (Techniques for doing so are discussed later)
* **Unstructured data** is supported, **(its *metadata*) can be queried through external tables** (**metadata objects created on top of external stages**)
* **External tables are *read-only* but *do* support constraints for relational modeling**
* Additional information for working w/ external tables can be found in the Snowflake documentation: <https://docs.snowflake.com/en/user-guide/tables-external-intro.html>
* W/ **no upfront costs, competitive pricing, + the ability to size warehouses to user needs or suspend them entirely**, users can often spend less on Snowflake than for legacy on-prem hardware
* *The caveat is that to do so,* ***users must understand how Snowflake pricing works + how to manage it effectively***

#### Costs To Consider

* Unlike on-prem databases (purchased upfront + used for the duration of their life cycle), **Snowflake employs a consumption-based model known as variable spend** (or *pay-as-you-go*), which **enables teams to:**
* **Do rapid prototyping or experiment w/ POCs w/out any upfront investment**
* **Control their costs by monitoring + adjusting usage patterns**

##### a) Storage Costs

* **Snowflake bills customers based on the daily average of the data stored in the platform/on the cloud**
* Since Snowflake’s **services layer automatically compresses data for optimal storage**, customers enjoy **lower storage costs w/out sacrificing performance**
* However, it is ***not just the raw data that counts toward storage quotas***
* **Time Travel** + **fail-safe backups** (discussed next chapter) must also be considered

##### b) Compute Costs

* Snowflake **consumption is billed based on the number of virtual warehouses used, how long they run, + their size**
* An ***active* warehouse is billed per second** (w/ a 60-second minimum)
* Snowflake **uses standard units called credits to simplify the math around warehouse pricing**
* *Exact* cost of a credit varies based on factors such as the cloud provider, hosting region, and Snowflake edition (i.e., Standard, Enterprise, Business Critical, or VPS)
* Once these variables are fixed, *a credit equates to one XS warehouse running for 1 hour*
* An important consideration is that **warehouses consume credits for as long as they are active, *regardless of whether they are running queries***
* An **effective cost strategy considers when warehouses are instantiated + shut down** (auto-resume + auto-suspend options are available) **+ the size required for a given workload**

##### c) Services Costs

* For running + managing the platform, **the services layer manages essential operations such as access control + query optimization**
* However, it **also performs automated serverless tasks, such as automatic (re)clustering, loading streaming data (Snowpipe), + replication**
* **Many user-generated actions**, such as querying table metadata (e.g., COUNT, SUM, or MAX), creating objects, + retrieving object metadata through commands such as SHOW and DESCRIBE **are handled by the services layer + *do not require an active warehouse***
* The **cost of the services layer is included in the warehouse credits *as long as it does not exceed 10% of the daily warehouse consumption***
* When the services layer consumption *does* exceed 10% of warehouse spend, its operation is billed at standard credit rates (i.e., one credit per hour of XS or equivalent)

#### Saving Cash by Using Cache

* With on-prem databases, **inefficient operations resulted in longer execution times**
* In Snowflake’s **variable spend model**, that **extra execution time is coupled w/ monetary penalties**
* Besides **writing efficient SQL**, Snowflake **users should also understand the various caches associated w/ the service + virtual compute layers to understand where they can take advantage of pre-calculated results**
* A **firm grasp of Snowflake caching will also inform decisions when modeling + building data pipelines**

##### a) Services Layer

* The **services layer** handles **2 types of cache**: **metadata** **cache** and **query results cache**

###### 1) Metadata Cache

* The services layer manages **object metadata**, like **structure**, **row counts**, + **distinct values by column**
* **Reviewing this metadata** through related SQL functions or the Snowflake UI **will NOT require a running warehouse + does NOT consume credits**
* **Snowflake stores metadata at the *table* level (size in bytes + date created)** **+ keeps stats** (count, min/max, null count, etc.) **for columns in every micro-partition in the table**
* **Micro-partitions** are **groups of rows w/in tables organized in a columnar fashion that allow for the granular pruning of large tables***, which can comprise millions of micro-partitions*
* The **Snowflake services layer automatically creates + manages micro-partitions** as DML operations are performed in tables
* Micro-partitions + caching are explained in detail in a later chapter, but for more information, see *Micro-partitions & Data Clustering* documentation: <https://docs.snowflake.com/en/user-guide/tables-clustering-micropartitions.html>
* The **services layer also stores metadata for object definitions + structure**
* This **includes the DDL for database objects + details** such as the column list, data types (even for views), + constraints
* DDL details can be retrieved through SQL functions such as DESCRIBE and GET\_DDL or by querying INFORMATION\_SCHEMA directly
* The latter will require an active warehouse + consume credits, while the services layer handles the former automatically

###### 2) Query Results Cache

* **All query results in a Snowflake account are persisted for 24 hours by the services layer + can be reused by other users**
* **Referencing a cached result resets the 24-hour clock for a maximum duration of 31 days, after which the result is purged**
* Aside from access privileges, there are **other considerations to ensure that cached results can be reused**, which includes the following:
* The **query** must be **syntactically equivalent**
* **Dynamic functions** such as CURRENT\_DATE()are ***not* used**
* The **data in the underlying tables has not changed**
* **Users have the required permissions** to access the underlying sources
* **Obtaining a result from the query cache is *infinitely* preferable to recalculating it b/c the operation happens instantly + consumes no compute credits**
* Unlike legacy databases, where the cache could be sacrificed to meet memory demands or lost altogether if the cluster was suspended, **Snowflake’s result cache is maintained until purged under the conditions previously mentioned**
* **Understanding the results cache is crucial for efficiently using Snowflake resources + making design decisions**
* However, that is not the *only* cache that users can take advantage of

##### b) Warehouse Cache

* **When a virtual warehouse accesses data from the central storage layer (“*remote disk***”**in underlying cloud storage such as AWS S3 buckets, Azure containers, or GCS buckets), the data is read into the warehouse cache (implemented using SSD storage)**
* The **amount of SSD storage depends on the warehouse size**
* **Reading from SSD storage is faster than reading from a remote disk, + the query optimizer (i.e., the services layer) will attempt to use the warehouse cache before defaulting to remote storage**
* *However*, *UNLIKE* the services layer caches, **warehouse cache comes with 2 significant differences**:
* **1)** The **cache is *specific* to a given warehouse**
* **2) Suspending the warehouse purges its cache**
* W/ this in mind, **users accessing the same data sources would benefit from sharing the same warehouse**
* However, **trade-offs must be considered between the credits required to keep a warehouse active to preserve the cache vs. the I/O required to read the data from scratch from remote storage**
* **When a query is unable to hit any of the previously mentioned caches, Snowflake *must* access the storage layer to perform the operation**

##### c) Storage Layer

* **Snowflake stores data on remote disks in the underlying cloud platform filesystem where the account is hosted**
* **Reading data from a remote disk is less efficient than doing so from the warehouse cache (stored on SSD drives) + *much* less efficient than fetching a pre-calculated result**
* **The storage layer has no cache to speak of, and it is the last resort when no cache exists**

#### Summary

* Snowflake’s **hybrid cloud-native design**, built for the cloud from the ground up, **enables real-time data sharing + on-demand workload sizing that gives its users unparalleled flexibility, overcoming many scalability limitations of previous database architectures**
* Snowflake’s architecture **allows secure data sharing between organizations across regions + cloud providers as quickly as it does between databases in the same account**
* By understanding each of the **layers** that make up Snowflake’s cloud architecture (**storage, compute, services**), we gained insight into how they enable powerful features such as **zero-copy cloning, Time Travel, Hybrid Unistore, +** **hybrid transactional/analytical processing** (**HTAP**) **tables** + open the gates to interacting w/ semi-structured + unstructured data
* We also outlined the **costs of each of the 3 architecture layers** + how to keep them in check
* Furthermore, we discussed how **various caching mechanisms across Snowflake’s architecture could work together to save costs + boost performance**
* With so many revolutionary features, Snowflake does not think of itself as just a database but a **Data Cloud**
* When considering everything its features make possible (services such as a data and app marketplace), it is easy to see why new terminology was needed
* Now it is time to dive deeper to understand the internal workings of Snowflake’s database objects.
* Understanding these objects + their unique features will allow you to model scalable + efficient database designs that take advantage of all the tools available in Snowflake’s arsenal

#### Further Reading

* For the definitive guide on all of Snowflake’s features + object types, beyond the modeling-related content covered in this book, consider Joyce Avila’s *Snowflake: The Definitive Guide: Architecting, Designing, and Deploying on the Snowflake Data Cloud*. O’Reilly Media, 2022