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

## Part 3: Solving Real-World Problems with Transformational Modeling

### Chapter 17: Scaling Data Models Through Modern Techniques

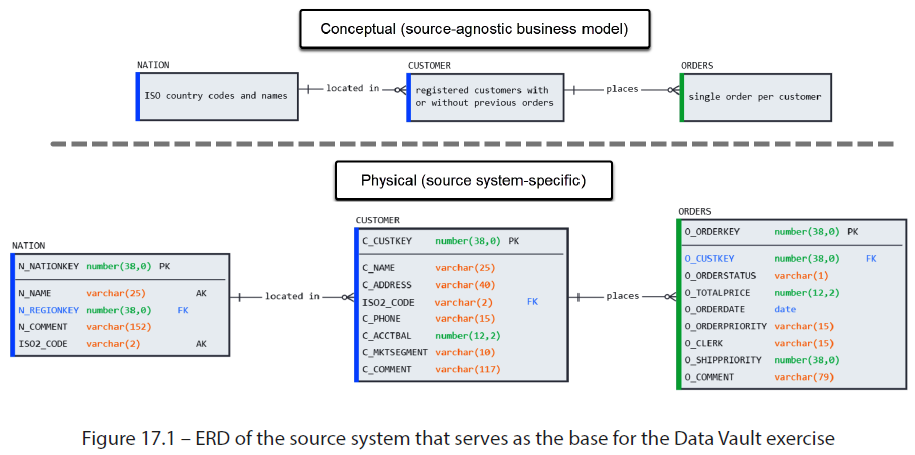
* After covering theory, architecture, terminology, methodology, + Snowflake-centered transformation strategies, this chapter builds upon that foundational knowledge to address common data management challenges in large, complex environments
* Specifically, we will explore **Data Vault 2.0** and **Data Mesh** methodologies, popular solutions that have emerged in response to some of the biggest challenges facing large organizations today
* Despite their similar naming, Data Vault and Data Mesh attempt to tackle very different challenges, and are often used together
* **Data Vault** is a methodology that **focuses on the efficient + flexible storage of data, w/ a primary focus on auditing + effortless scalability**
* It is made up of **3 pillars**: **modeling**, **methodology**, + **architecture**
* Its **standardized, repeatable design patterns** can be applied regardless of the complexity of the data or how many source systems are used
* **Data Mesh** exists for a very different purpose: **to facilitate data discovery + sharing among distributed enterprise teams**
* Data Mesh is *not* a specific technology or tool, but rather **a set of principles and best practices that can be applied to *any* architecture or platform**
* It **emphasizes the importance of domain-driven design, self-service, federated data governance, and the use of data products + APIs to facilitate data discovery + sharing**
* Data Mesh consists of **principles + technical practices that enable effective communication and collaboration between domains**
* **Many of these principles should be considered general best practices whether or not an organization decides to take up the Data Mesh banner wholeheartedly**
* W/ this in mind, let’s dive in and familiarize ourselves w/ Data Vault + how it can help organizations overcome some of the challenges associated w/ traditional DWs + analytics
* Main Topics:
* An introduction to Data Vault + its utility in modern data platforms
* Addressing the challenges of managing large, complex, + rapidly changing data environments with Data Vault
* The functional layers of the Data Vault architecture and core elements of the Raw Vault
* Efficiently loading Data Vault with multi-table inserts
* Modeling techniques for data marts
* An introduction to Data Mesh and managing data in large, complex organizations
* Reviewing Data Mesh and modeling best practices mentioned throughout the book

#### Technical Requirements for Local Snowflake Work

* The scripts used to instantiate + load the examples in this chapter are available in the following GitHub repo: <https://github.com/PacktPublishing/Data-Modeling-with-Snowflake/tree/main/ch17>

#### Demystifying Data Vault 2.0

* **Data Vault** emerged in the early ‘00s as **a response to the extensibility limitations of DWs built using 3NF + star schema models**
* Data Vault **overcame these limitations while retaining the strengths of 3NF + star schema architectures by using a methodology especially suited to meet the needs of large enterprises**
* Around 2013, Data Vault was **expanded to accommodate the growing demand for distributed computing and NoSQL databases, giving rise to its current iteration, Data Vault 2.0**
* Data Vault uses a **pattern-based design methodology to build an auditable and extensible DW**
* When most people refer to “Data Vault”, they are referring to the **Raw Vault**, which **consists of Link, Hub, + Satellite tables**
* ***Atop* the Raw Vault, sits the Business Vault, designed to be a business-centric layer that abstracts the technical complexities of the underlying data sources and uses constructs such as Point-in-Time (PIT) and Bridge tables**
* **Data cleansing + additional business rules are performed in *separate* data mart (or information mart) layers based on department + organizational reporting requirements**
* The **Business Vault + data/information mart layers are then *exposed* to end users for curated and self-service analytics**
* **Auditability** is **a major focus in Data Vault**
* DW’s are often more than just sources of historical data
* **As source systems are updated or decommissioned, a DW becomes the system of record**
* **Depending on where + how data is modified in a DW, *traceability to the source system may be permanently severed***
* **To overcome this problem, Data Vault uses an insert-only strategy** (no updates, no data cleansing, + no business rules in the base layers (this is handled *downstream* of the data marts and information delivery layers))
* As Data Vault practitioners often say: **“*Data Vault is a source of facts, not a source of truth”***
* Since “truth”, in the data world, can vary based on whose business rules you follow
* Architectural alignment with Snowflake
* ***Under the hood*, Snowflake architecture is insert-only**
* **Data stored in Snowflake micro partitions is *never physically modified***
* **Rather, modifications such as deletions or updates are performed *logically*, based on the record ID**
* This design makes Snowflake architecture an ideal fit for the insert-only methodology of Data Vault
* While the **Data Vault methodology** is simple to understand, it **requires training + experience to execute correctly**
* This chapter will examine the core elements of the **Raw Vault** + highlight Snowflake-specific features that aid in efficient design + data loading
* ***However, before undertaking a Data Vault implementation, consider obtaining a certification or enlisting the help of an expert to help decide whether + how Data Vault should be implemented in your organization***
* Remember, **Data Vault is scalable b/c it is source-system-agnostic and business-focused**
* To ensure the success of your Data Vault endeavor, **start the same way as you would w/ any design: by understanding business semantics + building a conceptual business model as**
* By doing so, you will **preemptively address many of the design questions to** **come**
* This exercise will use only *one* source system + a small subset of tables to demonstrate the core elements of the Raw Vault
* The conceptual model belowwill guide the design



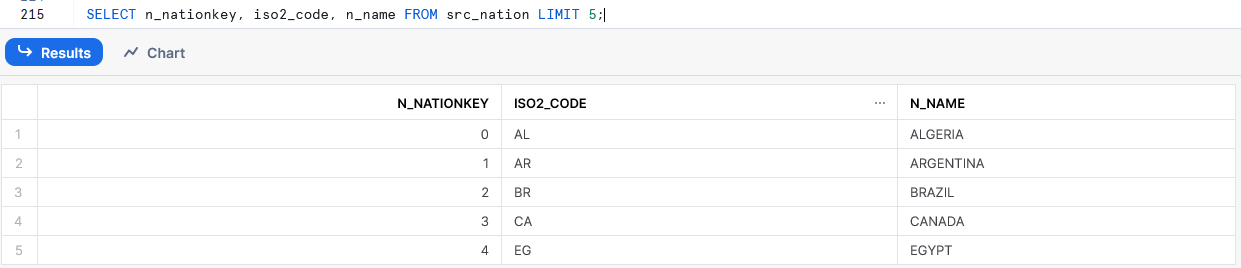
* Even though only one source system is used in this example, **an accurate conceptual model will ensure that business entities + their relationships are modeled correctly** (no matter where sourced from, referred to as *country* or *nation*, or identified by columns labeled *key* or *id*)
* **If the business model is *not* well understood in advance, the Data Vault model will be built on a shaky foundation + will fail to deliver the expected outcomes**
* The conceptual business model + the physical source system model above consist of two dimension tables + a fact table
* For this exercise, we will again source data from Snowflake’s TPCH sample data
* Use the accompanying code to instantiate the environments and create the landing area tables and associated objects via:
* Setting up environments
* Setting up the landing area
* Simulating data loads from the source system
* Starting w/ this foundation, let’s get to know the **core elements of the Raw Vault: links, hubs, and satellites**

##### Building the Raw Vault

* In this section, we will familiarize ourselves w/ the building blocks of the Data Vault by constructing a basic **Raw Vault**
* **Data Vault is a rule-based methodology + also highlights the importance of understanding general modeling standards like the ones covered so far**
* **Data Vault standards mandate the structure, transformation rules, + naming conventions**
* Naming conventions can vary as long as it is **consistent across the entire implementation + follows suggested guidelines**
* A list of suggested naming standards can be found on the Data Vault Alliance website:
* <https://datavaultalliance.com/news/dv/dv-standards/data-vault-2-0-suggested-object-naming-conventions/>
* To begin the exercise, we will create a **landing area for the source tables**
* Once the source data is loaded, we can begin modeling the **core of the Data Vault design: hubs**

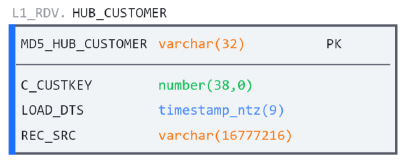
###### Hubs

* The first building block of Data Vault are **hubs**, which are **a collection of business keys belonging to a business entity**
* **In modeling terms, this would be a natural key, and *not* a surrogate w/ no intrinsic meaning**
* Take our LOCATION table as an example:



* N\_NATIONKEY, the PK in the source system, is a ***surrogate* key that’s *unrelated* to any business concept**
* **ISO2\_CODE, which is also unique + contains meaningful values, is better suited to act as the hub key**
* A **hub table** is a simple structure **consisting of the following**:
* **Hash key**, a **calculated field consisting of a hash** (e.g., SHA Binary or MD5) **of the Business Key columns that serves as the PK for the hub**
* This technique has been used to generate unique IDs in previous chapters
* This example will use the naming format <dv\_object>\_<entity\_name>\_hk
* e.g., hub\_customer\_hk
* **Business key column(s)**, which are **natural keys** **(NOT surrogates) that the business understands and commonly refers to**
* This can be a single *or* compound key, + uses the same column names as the source table
* **Load date**, or the **timestamp of when the record was *first* loaded into the Data Vault**
* This example will use the naming format load\_dts
* **Source system**, an **identifier for the source system from where the data is being loaded**
* This identifier should be **consistent across the entire Data Vault**
* This example will use the naming format rec\_src
* **Hub table naming convention:** Hub tables should **follow a uniform naming convention that *identifies them as hubs +* points to the source business entity** (not necessarily the table name)
* This example will use the naming format hub\_<entity> (e.g., hub\_customer)

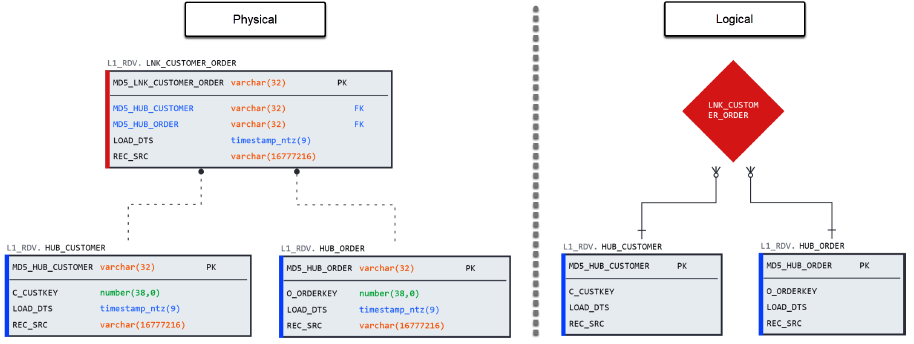
* **NOTE on repeating keys and multiple source** **systems**:
* **Hubs, like all Raw Vault objects, record only the *first* instance + source system that introduced a business key to the Data Vault**
* **Once a business key exists in the hub, subsequent loads, even those originating in *other* source systems, are not inserted or updated**
* **By incorporating a timestamp into their PK, satellites track historical changes to business attributes,** but, like other Data Vault objects, **only do so the *first* time a record enters the Vault**
* An example of a hub based on the CUSTOMER source table would look as follows:



* **While a hub may be simple in nature, it can already answer business questions, such as counting the number of entity instances + the sources that provide data on each**
* Now, we will learn how to **record transactions *between* entities using link tables**

###### Links

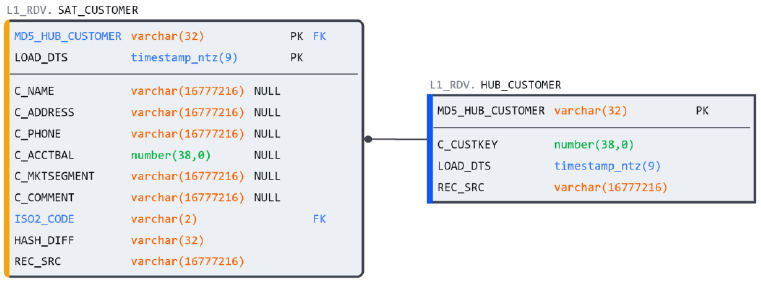
* Data Vault owes much of its versatility to **links**, which **store the intersection of business keys** **(or the FKs) from *related* hubs**
* In this regard, **links can be thought of as fact tables, but *w/out facts or attributes***, as they **contain nothing but business keys and *must* be connected to 2 or more hubs**
* A link structure resembles that of a hub in its simplicity, + it **contains the following**:
* **A link PK**, a **calculated field consisting of a hash of the Business Key columns**
* This example will use the naming format <dv\_object>\_<entity\_names>\_hk
* e.g., lnk\_customer\_order\_hk
* ***Associated hub* PKs**, or the **hashed PK columns used in associated hubs, declared here as *FKs***
* **Load date**, the **timestamp of when the record was first loaded into the Data Vault**
* **Source system**, an **identifier for the source system from where the data is being loaded**
* **Link table naming convention: Link tables should follow a uniform naming convention that identifies them as such**
* This example will use the naming format <dv\_object>\_<entity\_names>
* e.g., lnk\_customer\_order
* ***Just like hubs*, links record only the *first* instance of a relationship loaded into the Data Vault**
* An example of a link between CUSTOMER and ORDER entities would look as follows:



* **Notice that, *logically*, a link is identical to an M:M associative table**
* Even though CUSTOMER happens to have a one-to-many (optional) relationship w/ ORDER, **links are designed w/ flexibility in mind + can store relationships of *any* cardinality w/ no changes required**
* If business rules changed to allow joint orders from multiple customers, the Data Vault could continue loading w/out changing its structure or ETL processes
* Now that we have a way to store entity business keys + interactions between them, we can learn how to **record attributes + their changes using satellites**

###### Satellites

* **Satellites (sats)** **store the attributes in a Data Vault and provide change history**
* **Sats function very much like Type-2 SCDs *except***, like all Data Vault tables, they are **insert-only**
* B/c **sats can be *split* to accommodate scenarios such as multiple source systems, rate of change, and security**, it is a **good practice to include the scenario abbreviation in the sat table name**
* In this basic example, we use a single source system called *system 1*
* A **sat has the following structure**:
* **A hub PK column**, b/c **a sat uses the associated hub or link PK column (name + value) as its PK**, like hub\_customer\_hk
* **Load date**, the timestamp of when each set of attribute values was first loaded into the Vault
* **Hash Diff**, a **hash of all *attribute* values to allow for easy comparison of changes on new loads**
* This technique was used for SCDs for its convenience, as it is **much faster to compare one column than write out comparisons for every attribute in the table**
* Due to the name, beginners often mistake this column as a diff between multiple records, but **it is only the concatenation of the attributes in one record**
* **The diff occurs (or doesn’t) when two hash diffs are compared**
* **Source system**, an **identifier for the source system from where the data is being loaded**
* This value ***may not match that of the hub*** (e.g., if the business key was first loaded from Sys A but later modified in Sys B)
* **Non-PK source attribute columns**, which means **all the non-PK columns that did not make it into the hub (or link) are included in the sat**
* **Sat table naming convention**: Sat tables should follow a **uniform naming convention that identifies them as sats and *points to the source business entity*** (NOT necessarily the table name)
* This example will use the naming format sat\_<src system>\_<entity>
* e.g., sat\_sys1\_customer
* **Sats use a 2-part PK consisting of the associated hub key + load date**
* On every load, the records are compared using the business key + hash diff
* When changes are detected, the new records are inserted + given a timestamp (this implies that Data Vault is real-time compatible)
* An example of a sat based on the CUSTOMER source table would look as follows:



* **Every column that didn’t make it into the hub or link ends up as an *attribute* in the sat**
* Although hubs, links, + sats comprise the Raw Vault’s core, **an additional object type can also be found at this level**

###### Reference Tables

* **Reference (ref) tables** **store descriptive data about information in satellites or other Data Vault objects but do NOT warrant a business key**
* Common examples of ref tables include date dimensions (month, year, + quarter attributes) or descriptive information about the source systems in the REC\_SRC field
* Rather than creating a link table for every satellite w/ a date dimension, you can use a reference table for direct joins instead
* **While Data Vault rules prohibit sat-to-sat joins, a sat may reference a ref table through an FK**
* In our example, the NATION table contains ISO 3166 values that are *not* source system-dependent
* Instead of decomposing NATION into a hub and sat and *link*ing it to CUSTOMER, a ref table can instead be used
* This gives us a **completed Data Vault structure** based on the original 3 source tables we started w/:



* W/ the structure in place, all that remains is to **create an efficient pipeline to load records into the vault**

##### Loading with Multi-Table Inserts

* Although the loads in a Data Vault are insert-only, they are NOT trivial, as **validations for new records and CDC checks must be performed to keep the data consistent**
* As we saw w/ the ORDER table, which generated 3 objects (a hub, link, + sat) in the Raw Vault, **the challenge of loading multiple objects from a single source also exists**
* Snowflake provides a **multi-table insert operator** to allow users to perform such an operation efficiently
* In its basic (unconditional) form, the multi-table insert loads data from a single *subquery* into a list of target tables:
* INSERT ALL

INTO t1

INTO t1 (c1, c2, c3) VALUES (n2, n1, DEFAULT)

INTO t2 (c1, c2, c3)

INTO t2 VALUES (n3, n2, n1)

SELECT n1, n2, n3 from src;

* **However, *in Data Vault*, we must define *conditions* that only *insert* records (business keys, transactions, + attributes) when the Data Vault *sees* them for the *first* time**
* For this, we will use the ***conditional* multi-table insert**
* **Besides the WHEN condition, the multi-table insert allows users to specify an operator that determines whether all WHEN clauses are executed for a record or only the first that evaluates to TRUE:**
* -- Conditional multi-table insert

INSERT [ OVERWRITE ] { FIRST | ALL }

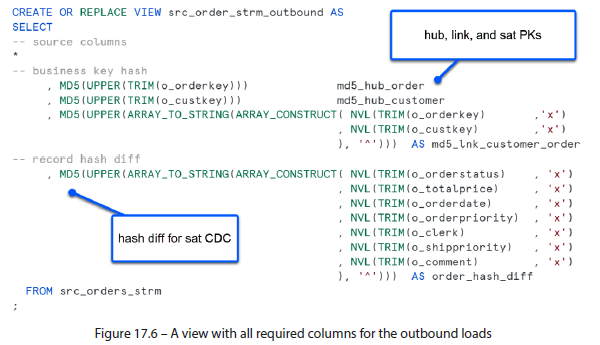
{ WHEN <condition> THEN intoClause [ ... ] }

[ ... ]

[ ELSE intoClause ]

<subquery>

* Pull over to the side of the load
* Outside of the Data Vault framework, the multi-table insert is a tool that can aid in tactical data quality enforcement
* When the type of data anomaly is known in advance (e.g., out of range or not permitted values), the multi-table insert can divert anomalous records to a staging table for review while allowing the rest of the load to proceed as planned to the intended target
* **Before writing the multi-table insert, we must define the outbound views that contain all the source fields + Data Vault-required columns to simplify downstream loads**
* The **view is constructed over the Stream object** created in the previous exercise b/c it **allows us to identify new records w/out added logic**
* To follow along with the accompanying code, execute the scripts in provided code:
* *Creating views for loading the Raw Vault*
* *Setting up the Raw Vault*
* The outbound view for the ORDER table would look as follows:



* Using these views, which contain the columns needed to load the Raw Vault objects, we can **configure the multi-table insert statement, allowing us to take a single subquery that selects from the source table + load the associated links, hubs, and satellites *in parallel***
* The multi-table insert would have the following structure:
* INSERT ALL

-- condition to check if BKEY exists in the hub

WHEN (SELECT COUNT(1) FROM hub\_customer tgt WHERE tgt.hub\_customer\_hk = src\_hub\_customer\_hk) = 0

-- if it's a new BKEY, insert into the hub table

THEN INTO hub\_customer

( < hub columns> )

VALUES

( < hub columns> )

-- condition to check if BKEY exists in the sat

WHEN (SELECT COUNT(1) FROM sat\_sys1\_customer tgt WHERE tgt.hub\_ customer\_hk = src\_hub\_customer\_hk

-- and only insert if changes based on

-- hash diff are detected

AND tgt.hash\_diff = src\_customer\_hash\_diff) = 0

-- if it's a new BKEY, or changes to attribute values

-- are detected, insert into the hub table

THEN INTO sat\_sys1\_customer

( < sat columns> )

VALUES

( < sat columns> )

-- subquery

SELECT < columns AS src\_columns> --aliased source columns

FROM l0\_src.src\_customer\_strm\_outbound src;

* Complete (and optionally repeat) the Raw Vault load by running the script in the section titled *Loading the Raw Vault using multi-table insert*.
* Each simulated load will process 1,000 customer records + approximately 10X as many related order records
* Due to the new-records-only insert nature of Data Vault, repeating the same batch of customer records into the landing area tables will result in no records being loaded into the Vault
* Now that we are familiar w/ the Raw Vault and the fundamentals + applications of the Data Vault methodology in conjunction w/ relevant Snowflake features, it is worth spending some time on the modeling patterns often used in **information marts** (or **data marts**, as they are called outside of Data Vault) to serve as a springboard for architecting the reporting layer

#### Modeling the Data Marts

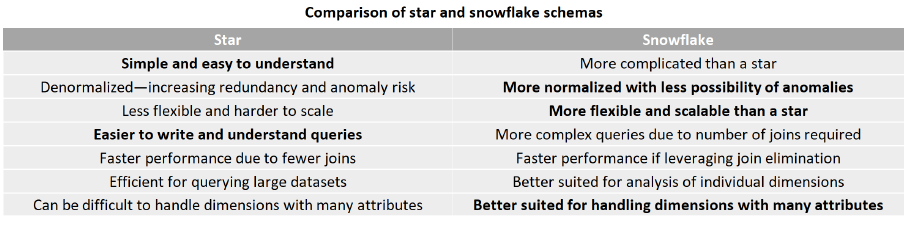
* This section will explore the **Star + Snowflake schemas**, popular options for architecting user-facing self-service schemas + data marts due to their efficiency + ease of understanding
* Both approaches are designed to optimize the performance of data analysis by organizing data into a structure that makes it easy to query + analyze
* **Data Mart versus Data Warehouse**
* A **data warehouse** and a **data mart** are **repositories for storing and managing data**, but they **differ in scope, purpose, and design**
* A **data warehouse** is a **large, centralized repository of integrated data used to support decision-making and analysis across an entire organization**
* DWs are **optimized for complex queries** + **often use Kimball’s dimensional modeling technique or Inmon’s 3NF approach**
* On the other hand, a **data mart** is a ***subset* of a DW designed to serve a *specific* department or function w/in an organization**
* Data marts are **typically designed using a star or snowflake schema model**

##### Star Schema

* A **star schema** is a database architecture that consists of a **central fact table surrounded by several dimension tables radiating out in a star-like pattern**
* The **fact table contains the measures** (e.g., sales or revenue), while the **dimension tables represent the attributes** (e.g., time, geography, + product)
* A star schema is ***highly* denormalized**, making it **efficient for querying large datasets and allowing for fast aggregations + drill-downs**
* The **advantage of a star schema** is that it is **simple to understand and easy to query**
* The **downside**, as w/ any denormalized design, is that **integrity issues + data anomalies can occur**
* While the star schema may be **simple to create**, it becomes **difficult to maintain as the schema grows due to redundancy concerns**, + the **inability to create entity hierarchies or many-to-many relationships**
* Such concerns are addressed by the snowflake schema approach

##### Snowflake Schema

* A **snowflake schema** is a **variation of the star schema** where the **dimension tables are *further normalized* into sub-dimension tables**
* For example, a product dimension might be split into product, product category, + product sub-category dimensions, forming a **hierarchy**
* In a snowflake schema, the **dimension tables are connected through a series of one-to-many relationships, creating a snowflake-like pattern**
* While this approach **increases data normalization**, it **makes writing queries more complex and harder to understand than a star schema**
* The snowflake schema is **generally considered less performant than a star because of the number of joins required to query it**
* However, that **may not be true once Snowflake’s performance enhancements, such as join elimination through RELY** are considered (Ch. 12)
* The following table summarizes the differences between the star and snowflake schemas, highlighting their respective benefits in **bold**:



* **Choosing between a star + snowflake schema will ultimately depend on the organization’s specific needs and goals**
* A **star schema is simpler + more efficient for querying large datasets** but **may not be as flexible for complex table relationships**
* A **snowflake schema** offers **greater data normalization + flexibility** but at the **cost of increased complexity**
* As a modeler, it’s important to understand these differences + choose the appropriate approach for each data mart
* **When organizations become large enough to warrant *multiple* data marts, or even warehouses, coordinating them w/out creating data siloes becomes challenging**
* In the last few years, **a new data management framework, Data Mesh, has emerged to facilitate sharing and collaboration among organizational data domains**

#### Discovering Data Mesh

* **Data Mesh** is an approach to **organizing + managing data in large, complex organizations,** introduced in 2019 by Zhamak Dehghani, a thought leader in the field of data architecture
* This approach **advocates for decentralized data ownership + governance, w/ data treated as a product owned + managed *by the teams using it***
* This contrasts with the traditional centralized (or *monolithic*) approach to data management, where a single team or department is responsible for *all* data-related activities
* In a Data Mesh architecture, **data is organized into self-contained domains, each responsible for its own data curation + sharing**
* These domains are **often organized around business capabilities or processes + are staffed by cross-functional teams that include technical + business experts**
* Data Mesh consists of **4** **principles that aim to enable effective communication + collaboration between domains: domain-driven design, self-service, + data product thinking**
* These **practices help ensure that each domain can operate independently while still being able to share data + insights with other domains as needed**
* The **four principles of Data Mesh are as follows**:
* **1) Domain-oriented decentralized data ownership** – In Data Mesh, **data is owned by the domain that produces it**, + th**e domain is responsible for making it available to the rest of the organization**
* This means **each domain has the autonomy to choose its tech stack, data models, + data storage**
* **2) Data as a product** – Data is treated as a product **designed + built for consumption by other teams**
* Data is managed w/ the same rigor + discipline as software products, **focusing on delivering customer value**
* **3) Self-serve data infrastructure as a platform** – Data Mesh promotes the idea of building self-serve data infrastructure that **provides a set of core data services that different domains can consume**
* This **helps reduce the complexity of data integration + allows domains to focus on their core competencies**
* **4)** **Federated governance** – Data Mesh recognizes that governance is **important for ensuring data quality, compliance, + security**
* However, instead of *centralized* governance, Data Mesh promotes **federated governance, where each domain has the autonomy to govern its own data while adhering to organization-wide standards + policies**
* While Data Mesh attempts to tackle the biggest data challenges that enterprises face, it has also been **criticized for its complexity + the skillset required to execute it correctly** b/c, **when mismanaged, the decentralization of Data Mesh can lead to a proliferation of data siloes** (the very thing it intends to mitigate)
* But, on the face of it, Data Mesh is a collection of best practices that are hard to argue w/
* Best of all, **Data Mesh guidelines are perfectly aligned + reinforced by Snowflake architecture and innate functionality**
* Next is a rundown of best practices for Data Mesh and the broader modeling context

##### Start with the Business

* **A successful data model must accurately reflect the business model**
* As the understanding of business rules + processes does not sit w/ the data team, **attempting to build any system w/out the guidance + express approval of business teams is folly**
* In larger organizations w/ **multiple business lines, identifying domain boundaries is key to segmenting the modeling task into smaller independent chunks**
* Depending on the depth + breadth of the organizational hierarchy, **a segmented data model can be separated logically into schemas, databases, or even Snowflake accounts**
* Snowflake sharing + securitization features are then used to establish appropriate access rights for anyone within the organization

##### Adopt Governance Guidelines

* **Governance must be woven throughout the development process to ensure end-to-end consistency**
* Define a **development workflow that starts w/ modeling** (iterated through *all* its stages) + **provides the appropriate levels of checks + approvals throughout**
* Ensure that **naming conventions are clearly defined + enforced at every stage of the process**
* Be sure that **all the work w/in a team, from the model to the transformational logic, is versioned and source controlled**
* Most people are familiar w/ code repositories such as Git + should also be aware that modern modeling tools allow for version control + parallel development of the same kind
* Version control **allows teams to compare + recover prior states of their modeling structure or transformational logic**
* Once checked into a repository + validated, **physical deployments must also be accompanied by the requisite documentation and metadata collateral to ensure usability by others in the organization**, including other departments
* Such materials include ERDs, table and column-level descriptions, + table constraints
* Following these guidelines will accelerate self-service and data discovery

##### Emphasize Data Quality

* **Data quality is driven by the design of the systems that generate it, + good design starts w/ an accurate model**
* When the model + the business it supports are in alignment, anomalous data, such as values that fall outside the accepted bounds or violate established cardinality, will be less likely to occur
* However, **if the data is inconsistent by nature, integrity checks must be built as far upstream in the loading process as possible**
* Some anomalies, such as null values, can be caught by Snowflake table constraints
* However, **integrity checks on FKs, PKs, + AKs must be performed manually during loading**
* As Data Vault rightly suggests, truth is subjective, while facts are not
* Transitive business rules (e.g., active customers) may change over time or vary by department, but a customer entity remains a cornerstone of the business model
* **Pushing transitive business rules downstream to mutable reporting layers while keeping data in its original form at the lower levels will ensure the model can handle changes or re-interpretations of business rules w/out requiring an integral redesign**
* **One of the pillars of Data Mesh + a best practice that all data teams stand to benefit from is the concept of data as a product**
* This simple change of perspective helps domain + centralized data teams embrace data quality holistically + take responsibility for the data they generate or maintain

##### Encourage a Culture of Data Sharing

* Whether facilitating sharing or reviewing data assets w/in or across teams, **data sharing is the foundation of Snowflake’s cloud architecture**, **allowing organizations to convene on a single source of truth w/out moving or copying data through cloning + time travel**
* The documentation + governance guidelines mentioned previously will enable everyone in the organization to use + interpret the data assets in a straightforward manner
* The **same principles of sharing + collaboration that Snowflake has baked into its architecture also apply to the data modeling assets**
* **Data modeling is an essential tool for engineering + deploying to a physical database, but its utility extends much further when shared w/ the broader organization**
* Once a data model is deployed + operational, the diagrams, definitions, + functional details must be made available to the organization through a collaborative interface that keeps everyone aligned + allows for iterative enhancement
* Modern modeling solutions provide the previously mentioned features + integrations w/ other tools in the BI stack

#### Summary

* **Data Vault 2.0 is designed to address the challenges of managing large, complex, + rapidly-changing data environments**
* It is a **hybrid approach that combines elements of 3NF + star schema** and uses a **standardized, repeatable design pattern that can be applied to any dataset, regardless of size or complexity**
* Data Vault **design begins by defining the business model and constructing the base layer, known as the Raw Vault**, which contains the following elements:
* **Hubs** – **natural keys** that **identify business entities**
* **Links** – store the **interactions between business entities**
* **Satellites** – store the **descriptions + attributes of business entities**
* **Reference tables** – include **descriptive information + metadata**
* ***On top of the Raw Vault*, a** **Business Vault** **is constructed to meet changing business needs and requirements w/out disrupting the overall data architecture**
* Next, **domain-oriented information marts are built to meet organizational reporting demands**
* **All of these features working in unison provide agility, scalability, change history, and full auditability/ traceability, given any number of source systems while absorbing source and business rule changes without requiring redesign**
* Whether on top of Data Vault or other DW architectures, **reporting + self-service layers are often modeled using star + snowflake schema designs**
* A **star schema** consists of a **central fact table connected to multiple dimension tables**, while a **snowflake schema** expands on this by **further normalizing the dimension tables to reduce redundancy**
* Business users prefer these architectures over fully normalized schemas b/c they are **more intuitive and easier to query**
* **Some organizations are large enough to contain a mix of modeling architectures + data platforms**
* When an organization is **large enough to warrant *multiple* data domains**, the **Data Mesh framework**, introduced in 2019, has been **instrumental in establishing best practice guidelines to ensure cross-domain data access and self-service**
* By pushing the responsibility of **data stewardship to the domain teams + treating data as a product, data assets are held to standards similar to those of software products**
* Whether embracing Data Mesh in full or applying its most effective practices to an existing data platform, **establishing and solidifying standards + governance rules will ensure data quality, usability, and easy maintenance of the data models you produce**
* **Starting w/ the business model + ensuring a conceptual alignment between functional + data teams provides a solid foundation for building the technical solution**
* **Governance guidelines + standards must then be set in place to ensure a ubiquitous language is understood by everyone in the organization in the technical + semantic domains**
* **A data model built w/ business domain consensus will provide a scalable foundation that ensures data quality + consistency**
* However, **treating data as a product is the responsibility of *every* data domain, not just the central BI team**
* **When multiple data domains exist w/in an organization, a mix of architectures + data platforms will pose a barrier to effective sharing**
* To overcome this, **teams must leverage the native sharing features of the Snowflake Data Cloud + other tools that comprise their BI stack to equip everyone w/ the technical means and functional documents to discover + consume cross-domain information**
* **No matter the platform or methodology, an accurate + accessible data model is the key to simplifying and making sense of complex systems**