# 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

* Provides you with a comprehensive overview of the power + potential of data modeling w/in the **Snowflake cloud data platform**
* Introduces the fundamental concepts + techniques that underpin effective modeling, including the importance of understanding **data relationships** + the **role of modeling in driving better business outcomes**
* Also includes a detailed examination of the **4 different types of modeling**, highlighting their benefits + use cases
* Finally, focuses specifically on Snowflake architecture + objects, exploring how to master this powerful platform + optimize it for maximum performance + value
* Through a combo of theoretical insights + practical examples, you will gain a deep understanding of how to use modeling to unlock the full potential of Snowflake + transform your approach to data management and analysis

### Chapter 1: Unlocking the Power of Modeling

* *“Modeling”* has come to mean very different things in the half century it’s been practiced in DBMSs
* Book’s overall aim = **demystify modeling, along with its applications, methodologies, + benefits**
* The concept of modeling will unfold into a set of methods + terms that help organizations design + manage data and, more importantly, help them understand themselves
* In its ***broadest interpretation*,** **modeling is a *selective simplification* that aids in navigating or *designing something more complex***
* ***Any* system can be broken down into smaller, more manageable pieces**
* **Manipulating any piece *individually* may be straightforward, but doing so w/out regard to an *overall* strategy is a tenuous proposal that’s sure to encumber scalability + maintenance down the line**
* **While modeling is generally considered database-agnostic, modern cloud data platforms, such as Snowflake, present users w/ many unique features thanks to their innovative architecture and consumption-based pricing**
* A **clear + forward-looking design that takes advantage of the native features of a platform that supports it** **is** **the key to building cost-effective solutions capable of meeting + anticipating business needs**
* As **analytical requirements of a data-driven organization are notoriously complex constantly evolving, modeling must keep pace + accompany data teams from idea to execution**
* To achieve this, **modeling must go *beyond* the structure + relationships of database tables and embrace the *transformational logic* that moves + shapes *the underlying data***
* Only by leaning into the specifics of Snowflake features + architecture can a model be built efficiently from beginning to end
* Main chapter topics:
* Recognizing the **utility of models** in our daily lives
* Getting a glimpse of **modeling conventions** in action
* Getting acquainted with the **tools** in the modeling toolkit
* Uncovering the **benefits** **of** **modeling** for enterprise teams
* **Incorporating modeling** into strategic planning
* **Understanding modeling applications** for transactional and analytical systems

#### Technical Requirements

* *This* book focuses on data modeling *specifically* for the Snowflake Data Cloud
* While **modeling includes many system-agnostic terms + conventions, *this* book leverages unique features of Snowflake architecture, data types, + functions when building physical models + SQL transformations**
* To follow along with the exercises in the following chapters, you will need a Snowflake account w/ access to a **sandbox area for creating schemas, objects, + loading data**
* Can sign up for a 30-day free trial of Snowflake (<https://signup.snowflake.com/>) if you do not already have access.
* This book will frequently use **visual modeling diagrams as part of the modeling process**
* While a diagram can be drawn by hand + constructed in PowerPoint or Lucidchart, **a tool that supports common database modeling features is recommended**
* Exercises in this book take the reader **from conceptual database-agnostic diagrams to deployable + runnable Snowflake code**
* So, a tool that supports various modeling types + can forward-engineer Snowflake syntax is recommended
* Diagrams in this book were generated using the SqlDBM online database modeling tool (<https://sqldbm.com/Home/>), which supports previously-mentioned features w/ a 2-week free trial

#### Modeling with Purpose

* **Models are used to *simplify* complex systems**
* Take a modern city as an example 🡪 see that it consists of intricately linked systems such as highways, electrical grids, + transit systems
* While these systems operate in the same physical territory, they require very different models to help us understand them
* Ex: A subway system snakes + curves below a city’s varied terrain, but our model of it (a subway map) uses straight lines + places stations at nearly equidistant intervals
* The subway map is NOT the city, but **a selective simplification** of the city that makes it easier for passengers to visualize their journey
* The transit map is a model so ubiquitous that it’s hard to imagine doing it any other way, yet it took time to evolve
* As w/ maps, ***various* kinds of modeling exist to help teams w/in an organization make sense of the many layers that make up its operational landscape**
* Also, like maps, **models help organizations prepare for the journey ahead**
* ***But how does one use a model to navigate a database, let alone plan its future?***

#### Leveraging the Modeling Toolkit

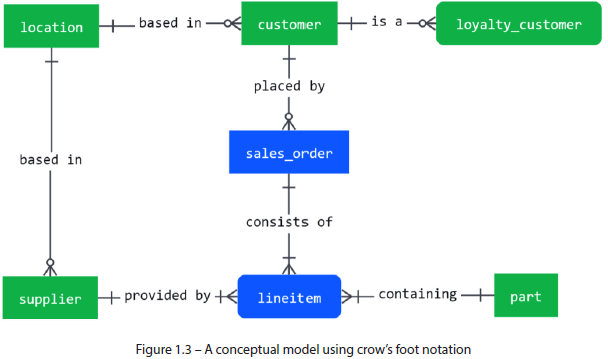
* Before we continue, we need to formally delineate **3 distinct concepts often used together in the service of modeling** to make it simpler to refer to a specific tool in the modeling toolkit later on
* By understanding where each piece fits in the broader domain of database design + management, diving into deeper technical concepts later will become more meaningful + easier to digest
* **The 3 components are**:
* **1) *Natural language* semantics (words)**
* This is **terminology employed in communicating details of a model between *people***
* These are **agreed-upon words that employ pre-defined conventions to encapsulate more complex concepts in simpler terms**
* Ex: When both parties involved in a verbal exchange understand the concept of a **surrogate key**, it saves them from having to explain that it is **a unique identifier for a table record *that holds NO intrinsic business meaning*** (such as an integer sequence or a hash value)
* **To ensure effective technical conversations, it helps to be fluent in the semantics of modeling**
* **Saves time by succinctly communicating a complex concept**, + **also saves even more time by NOT *mis*-communicating it**
* **A properly-modeled database would *never* return different records for the same surrogate key**
* Ex: A waiter would return different foods if ordering chipsin London vs. in LA
* **2) *Technical* semantics (SQL)**
* **SQL** is **a domain-specific language used to manage data in a RDBMSs**
* Unlike a general-purpose language (like YML or Python), **domain-specific languages have a much smaller application but offer much richer nuance + precision**
* While it can’t format a website or send an email, **SQL allows us to create the structure of our database + manipulate its contents**
* **SQL bridges modeling concepts (expressed in words or images) + what is physically defined in the database**
* ***Snowflake* uses an ANSI-compliant SQL syntax, meaning its basic commands (such as SELECT, UPDATE, DELETE, INSERT, and WHERE) are compatible w/ other database vendors who use this standard**
* **Also offers many *extra* functions, clauses, + conventions beyond ANSI-standard SQL to give users added flexibility to manage the database**
* **Unfortunately, due to its domain-specific nature, SQL presents a significant limitation in that it *can only express what the database explicitly understands***
* While SQL can define table structure + precisely manipulate data, it is **too detailed to easily articulate the underlying business requirements**
* **3) *Visual* semantics (diagrams)**
* **Through their simplicity, images can convey a density of information that other forms of language simply cannot**
* **In modeling, diagrams combine the domain-specific precision of SQL w/ the nuance of natural language**
* This gives diagrams a lot to work w/ to capture a data model’s business meaning *and* technical specifics
* To start, diagrams vary in the level of detail they present, giving the observer exactly what they’re looking for w/out overwhelming (or underwhelming) them w/ information
* **Most importantly, the semantic conventions used in diagrams are universal + can be understood by people besides data analysts + engineers**
* Yes, modeling diagrams are considered “technical drawings” (they represent strict technical concepts through agreed-upon visual conventions)
* **However, in their simplest form, models can be understood almost intuitively w/ no prior knowledge**
* Even at more advanced levels, such as **logical** and **physical**, learning to read a model is much simpler than learning SQL
* **When all these semantics come together and are understood by the entire organization, they form a ubiquitous language**, a concept first described by Eric Evans in *Domain-Driven Design*
* Modeling then forms a part of the vocabulary that is understood universally throughout the organization to describe its business + store the data assets that support it
* *But that is just one of the many benefits that modeling provides*

#### The Benefits of Database Modeling

* For many people, database modeling brings to mind stale diagrams, arcane symbols, or extra work at the end of a project
* Only a decade ago, fueled by the rise of distributed computing in the early 2000s (which popularized the concept of “big data”), the notion that “modeling is dead”gained notoriety
* **More precisely, it was thought that cheap + near-limitless computing power had made planning and designing a thing of the past**
* **It was said that flexible, semi-structured data formats + the ability to parse them on the fly (known as schema-on-read) had made modeling obsolete**
* **Eventually, operating + maintenance costs caught up with reality + revealed 2 great shortcomings of the schema-on-read approach**
* **1) No matter how data is structured, it *must* be functionally bound to the business that it helps support**
* i.e., Semi-structured formats are **neither a panacea nor an excuse to forgo the process of business validation**
* **2) *Most importantly*, a model is NOT simply the shape that data takes once uploaded to a database, but rather, the blueprint for business operations, w/out which it is impossible to build sustainable architectures**
* ***Sustainable* solutions require a long-term strategy to ensure their design matches the underlying business model**
* ***Without this* *strategy*,** schema-on-read (Chapter 15, Modeling Semi-Structured Data), star schema (Chapter 17, Scaling Data Models through Modern Techniques), or **any other schema are narrow-sighted tactics that lead nowhere**
* **But *done right*, modeling makes developing database architectures more agile + helps projects evolve from the idea phase to implementation**
* **At every stage of development, the model serves as a guide for supporting the conversations necessary to propel a design into the next phase + provide additional business context**
* **Once *implemented*, the model becomes a living document that helps users understand, navigate, + evolve the system it helped create**
* While every organization models in the *technical* sense (creating tables, transforming data, etc.) **not everyone models *strategically*, end to end, in the broad sense of the word**, + **thereby foregoing the long-term benefits** thatinclude the following:
* **Consensus** and **visibility** of the broader business model
* **More productive conversations** w/ business teams
* **Better quality of requirements**
* **Higher signal, lower noise** in technical conversations
* **Cross-platform, cross-domain**, + **widely understood conventions**
* **Big-picture visual overview of the business** + its database footprint
* Preliminary designs become **implementation blueprints**
* Accelerating onboarding of new team members
* **Making data more accessible + unlocking self-service w/in organizations**
* Keeping the **database landscape manageable at scale**
* Getting **a handle on complex data pipelines**
* To demonstrate the difficulties of working *w/out* formal modeling, we can take a simple schema based on Snowflake’s shared TPC-H dataset (available in the shared database called SNOWFLAKE\_ SAMPLE\_DATA), which, at first glance, looks like this:



* While these tables have been “modeled” in the strict sense of the word, + even contain data, we **get very little information on what that data represents, how it relates to data in other tables, or where it fits in the broad context of business operations**
* Intuition suggests that SALES\_ORDER and CUSTOMER share a relationship, but **this assertion needs to be tested**
* Even in this trivial example of only 8 tables, **it will take considerable time to thoroughly sift through the data to understand its context**
* The irony is that **many of the details we’re looking for are already baked into the design of the physical tables, having been modeled at some point in the past, + *we just can’t see them***
* *Without a map, the terrain is lost from view*
* Here is the same set of tables visualized through the **Entity-Relationship Diagram** (**ERD**) modeling convention, using crow’s foot notation



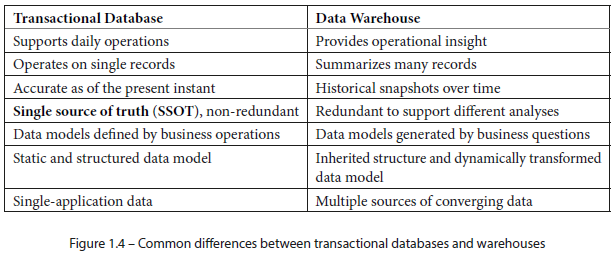
* At a glance, the big picture comes into focus
* **Diagrams such as this allow us to understand the business concepts behind the data + ensure they are aligned**
* Having a visual model **also lets us zoom out from individual tables + understand the semantics of our business: What are the individual pieces involved + how do they interact?**
* This global perspective gives everyone in the organization a means of finding + making sense of data assets w/out requiring a technical background
* Thus, business analysts or new hires can unlock the value of the information without any help from the data team
* As the organization grows, expanding in personnel + data assets, it will inevitably become too big for any person, or even a team of people, to coordinate
* Here, **organizations that have embraced data modeling will stand out from those that did not**
* **Modeling can be the thing that helps organizations scale their data landscape, or it can be the technical debt that holds them back**
* **Yet, for all its benefits, modeling is NOT a cookie-cutter solution that *guarantees* success**
* There are **many approaches to modeling + various modeling methodologies that are suited for different workloads**
* We will tackle the fundamentals of modeling that will allow you to understand these differences and apply the best solution using a **first-principles approach**
* First, we begin by breaking down the 2 main database use cases + observing the role modeling plays in each of them

#### Operational and Analytics Modeling Scenarios

* The RDB as we know it today emerged in the ‘70s, allowing organizations to store their data in a centralized repository instead of on individual tapes
* Later that decade, **Online Transaction Processing (OLTP)** emerged, **enabling faster access to data + unlocking new uses for databases** such as booking and bank teller systems
* This was a paradigm shift for **databases**, which **evolved from data archives to *operational systems***
* **Due to limited resources, data analysis could not be performed on the same database that ran operational processes**
* **The need to analyze operational data gave rise to Management Information Systems (MIS) in the ‘80s, or Decision Support Systems (DSS)** as they later became known
* **Data would be extracted from the operational database to the DSS, where it could be analyzed according to business needs**
* **OLTP architecture is NOT best suited for the latter case, so Online Analytical Processing (OLAP) emerged to enable users to analyze multidimensional data from multiple perspectives using complex queries** ***(the same paradigm used today by modern data platforms such as Snowflake)***
* **The approach to storing + managing data in OLAP systems fundamentally differs from the operational or transactional database**
* **Data in OLAP systems is generally stored in data warehouses (DWs), centralized repositories that store *structured* data from *various sources* for the *purpose of analysis + decision-making***
* The ***transactional* system keeps the up-to-date version of the truth + is generally concerned with *individual records***
* But a **DW snapshots many *historical* versions + aggregates volumes of data to satisfy various analytical needs**
* **Data *originates* in the transactional database when daily business operations (Ex: bookings, sales, withdrawals) are recorded**
* **DW’s do not “create” but rather load extracted information from 1+ various source systems**
* **Functional differences between transactional databases + DWs present some modeling challenges**
* A **transactional system *must* be modeled to fit the nature of the data *it is expected to process***
* i.e., Knowing the format, relationships, + attributes required for a transaction
* **NOTE: The main concern of a *transactional* database model = the *structure* + *relationships* between its tables**
* **By contrast**, the **DW *loads* existing data from source systems + *isn’t* concerned w/ defining a single transaction, but instead w/ analyzing *multitudes* of transactions *across various dimensions* to answer business questions**
* To do this, **a DW must transform the source data to satisfy multiple business analyses, which often means creating copies w/ *varying* granularity + detail**
* **NOTE: Modeling in a DW *builds* *upon* the relational models of its source systems by conforming common elements + transforming the data using logic**
* So, **if transformational logic is a core concept in DW modeling, why is it so consistently absent from modeling discussions?**
* *Because* **in order to do transformational modeling justice, one must forgo the universality of general modeling principles + venture into the realm of *platform specifics* (i.e., syntax, storage, + memory utilization)**
* This book, in contrast, will embrace Snowflake specifics + go *beyond* physical modeling by diving into the transformation logic behind the physical tables
* This approach provides a fuller understanding of the underlying modeling concepts + equips the reader w/ the required SQL recipes to not only **build models but to load + automate them in the most efficient way possible**
* This is **where Snowflake truly shines and confers performance + cost-saving benefits**
* *Is Snowflake limited to OLAP?*
* **Snowflake’s primary use case is that of a DW, w/ all the OLAP properties to enable multidimensional analysis at scale over massive datasets**
* However, since 2022, there’s **a new table type called** **Hybrid Unistore**, which **features both an OLTP-storage table *and* an OLAP analysis table under *one* semantic object**
* This means Snowflake users can now **design transactional OLTP database schemas while leveraging the analytical performance Snowflake is known for**
* **Although OLAP + OLTP systems are optimized for different kinds of database operations, they are still databases at heart + operate on the same set of objects (ex: tables, constraints, + views) using SQL**
* However, **each use case requires very different approaches to modeling the data w/hn**

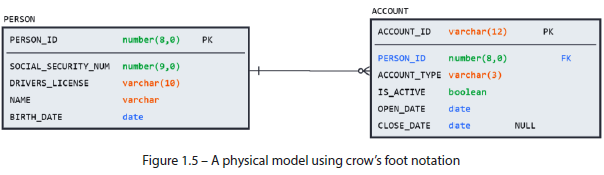
#### A Look at Relational and Transformational Modeling

* Before exploring the modeling process in detail, it’s helpful to understand the look + feel of relational and transformational modeling + what we’re working toward
* Before proceeding, we **summarize the main differences between transactional databases and DWs**



##### What Modeling Looks Like in *Operational* Systems

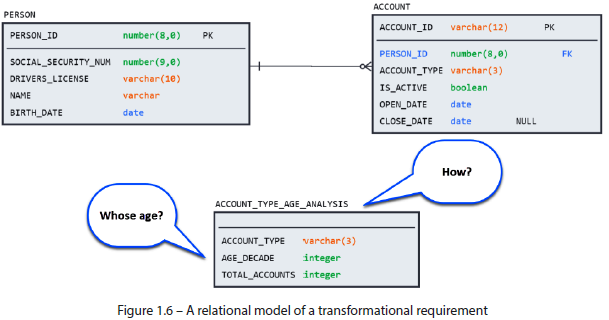
* Completely ignoring the modeling workflow that got us here (covered later), we can observe an example of the **type of modeling most commonly seen in transactional systems**
* The physical diagram below serves both as a blueprint for declaring the required tables + a guide to understanding their business context



* Following modeling conventions, **we can infer a lot of information from this simple diagram**
* Ex: A person is uniquely identified by an 8-digit identifier (the PK) + *must* have a SSN, driver’s license, name, + birth date.
* The **one-to-many relationship** between the 2 tables establishes that while a person does not necessarily need to have an account created, an account must belong to just one person
* **These details, combined w/ the list of attributes, data types, + constraints, not only dictate what kinds of data can be written to these tables but also provide an idea of how the business operates**.
* *So, how does this differ in analytical databases?*

##### What Modeling Looks Like in *Analytical* Systems

* In a **DW scenario**, the PERSON and ACCOUNT **tables** above **would NOT be defined from scratch** + instead **would be extracted from the source in which they exist + then loaded, bringing both structure *and* data into the process**
* Then, the ***analytical* transformations begin in answer to the organization’s business questions**
* This is a process known as **ETL** (Although **ELT has become the preferred processing order**, the original term stuck)
* Suppose the management team wanted to analyze which age groups (by decade) were opening which account types + they wanted to store the result in a separate table for independent analysis
* The following diagram shows the **resulting relational model of an object obtained through transformational analysis *but provides no business context***:



* Although physical modeling *could* describe such a table (as seen above, containing the account type w/ age and count of accounts as integers), **such a model would fail to communicate the most relevant details**, presented here:
* The **logic used to perform the analysis**
* The **relationship between the source tables and the output**
* The **business requirement for ACCOUNT\_TYPE\_AGE\_ANALYSIS in this example *purposely* excludes the source key fields from the target table, preventing the possibility of establishing any relational links**
* *However*, **the relational model still serves a vital role: it tells us how the sources are related + how to JOIN them correctly to produce the required analysis**
* The logic could then be constructed by joining PERSON and ACCOUNT, as shown here:
* CREATE TABLE account\_types\_age\_analysis AS

SELECT

a.account\_type,

ROUND(DATEDIFF(years, p.birth\_date, CURRENT\_DATE()), -1) AS age\_decade,

COUNT(a.account\_id) AS total\_accounts

FROM account AS a

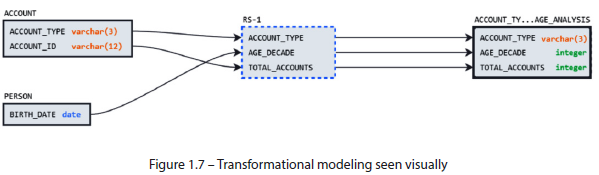
INNER JOIN person AS p

ON a.person\_id = p.person\_id

GROUP BY 1, 2

;

* **Although there is no *relational* connection between ACCOUNT\_TYPE\_AGE\_ANALYSIS and its sources, there is still a clear *dependency* on them + their columns**
* **Instead of using ERDs, which convey entities + relationships, transformational pipelines are visualized through a lineage diagram**
* This type of diagram **gives a column-level mapping from source to target, including all intermediate steps**, as shown here:



* ***Paired w/ the SQL logic used to construct it,* the lineage graph gives a complete picture of the transformational relationship between sources + targets in an analytical/DW scenario**
* Having witnessed both **relational + analytical approaches** to modeling, it is clear that **both play a vital role in navigating the complex dynamic environments that one is liable to encounter in an enterprise-scale Snowflake environment**
* Although we have only skimmed the surface of what modeling entails + the unique features of the Snowflake platform that can be leveraged to this end, this chapter has hopefully given you an idea of the **vital role that modeling plays in building, maintaining, + documenting database systems**

#### Summary

* Whether thinking about a company business model or sharing a finalized design w/ team members, we all engage in modeling to varying degrees
* **Embracing database modeling + learning to speak in a commonly understood language unlocks many time-saving + collaborative benefits for an entire organization**
* **Thinking *long-term* + modeling *strategically*, as opposed to reacting tactically, aligns database designs to the business that they underpin, ensuring their viability**
* Having seen the advantages that modeling uncovers + where it can be implemented, we can begin to analyze its components to understand precisely where they should be used + how they form a natural design progression
* Next, we explore the 4 modeling types used in database design + discuss where they excel + how they build on one another to help take an idea + evolve it into a technical system design while generating living project artifacts to navigate and maintain the final product

#### Further Reading

* Evans, Eric. *Domain-Driven Design: Tackling Complexity in the Heart of Software*. Addison-Wesley Professional, 2004
* Explores how to create effective models by going beyond the surface + getting to the intention of the system itself
* A recommended read for those wishing to go deeper into the realm of effective communication through models, unrestricted by specific technical domains, methods, or conventions