# 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 2: An Introduction to the Four Modeling Types

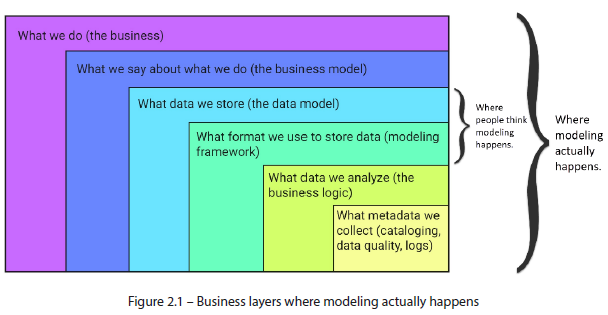
* **A model = a selective simplification of reality**
* There are **variants of models that exist** to describe different aspects databases + the organizations that rely on them to enable their business operations and analytics
* Many think of “modeling” as simply documenting a database (diagramming)
* But **modeling goes way beyond tables + databases by not only helping the developers understand the business but also helping the business understand itself.**
* Organizations use different models + modeling styles from organization charts to network diagrams to navigate their many complexities
* *None of these provide a perfect map, but some serve as the right map for the right job*
* Main Topics
* Breaking modeling down to a **flow**
* Use modeling elements to find a **common language** for communicating with business teams
* Aligning the business model to the data model with **conceptual modeling**
* Expanding on business concepts beyond database objects with **logical modeling**
* Creating the blueprint for database design by translating a logical model into a **physical** **model**
* Creating and managing ELT pipelines using **transformational modeling**

#### Design and Process

* Over the years, many books have been written about modeling + designing database systems.
* However, many of these books fail to **make the connection between technicality and practicality**, which are inextricably linked
* Often, modeling books prioritize *completeness* (such as describing 4 different notations to express a relationship) + ignore **usability** (e.g., which notation would be most accessible to business users?)
* The **most significant complexity of a database system is *not* technical 🡪 It is the business model itself (the interactions w/in a business + the rules that govern them)**
* **If the *business* model is not aligned w/ the *data* model, the database will be repeatedly forced to adjust (while losing organizational trust + resources in the process)**
* What is needed is a ***business-readable* language that developers use to build + document the database landscape**

#### Ubiquitous Modeling

* Before defining data structures, we **need to understand the business model that generates them**
* Such a model will make it possible to build a database system aligned w/ business processes + be able to anticipate change
* Like seeing a forest for the trees, **ubiquitous modeling allows us to see the *business* for the *data***
* A business has many layers: from a mission statement that defines a company ethos, to the sales transactions, to the logistics that support them, to the data, metadata, analysis, + more
* However, when most people think of a “model”, they tend to focus only on the *data* layer, forgetting that, **in *isolation*, the data only tells part of the story**
* In reality, the **modeling process involves teams across the *entire* organization**, including management, business departments, data teams, + analysts
* Everyone in the organization, no matter their technical background or domain expertise, will work with modeling in some capacity (even if it is in the role of a viewer)
* Never lose sight of the variance in skillsets when modeling 🡪 **ensuring they are kept as simple as possible while including all the necessary information for the task at hand**
* The following diagram depicts what is commonly thought of as “data modeling” within the broader business context:



* NOTE: Relying *exclusively* on the data, a cautionary tale:
* The management team asks you to build a model that handles all business data
* Providing sample data {0,3,6,9}, they say they can work with you to ensure you understand what you’re building—ask any (yes/no) questions you like.
* Q. Is 12 in the set? Yes.
* Q. Is 21 in the set? Yes.
* Great, you understand the numbers and build a model to handle *multiples of three*
* The next day, a –1 arrives + breaks your model because the business set is *the set of all integers*
* Starting in the late 1960s and to the present day, modeling, + modeling notation, have changed often: beginning with the first eponymous designs by Peter Chen, Charles Bachman, + Richard Barker, national standards such as Merise, + industry standards such as IDEF1X, **Information Engineering** (**IE**), **Unified Modeling Language** (**UML**), + others
* Many notations, no clear winner, + no standard governing body to arbitrate, so no wonder so many people are confused
* This book is *not* intended as an encyclopedia of modeling but *is* meant to be a practical guide for applying modeling concepts *in Snowflake* to ensure efficiency, consistency, + scalability
* This, + future chapters, will employ the most understandable + concise notation that makes modeling accessible to data engineers, developers, analysts, + business users alike by adhering to the following principles:
* **Keep it simple stupid** (**KISS**): In modeling (as in life), the simplest approach is often the best approach
* **Be concise**
* **Be unambiguous**: Never give 2 meanings to the same convention
* **Embrace industry standards**: *When they facilitate understanding*
* **Break industry standards**: Be flexible + eschew dogmatism *in favor of usability*
* W/ these principles in mind, let’s take a high-level look at the kinds of modeling that exist + where they are used

#### Conceptual Modeling

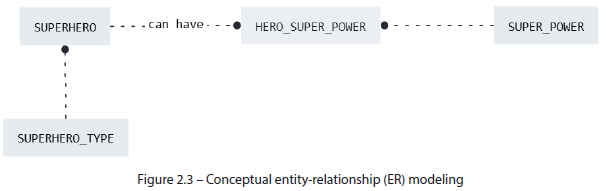
* **Modeling begins *long before* databases (or even data) enter the picture, as it starts w/ the business itself**
* A **conceptual diagram** should be as valid for describing the organization’s operating model as it would be for laying the foundations of its database landscape

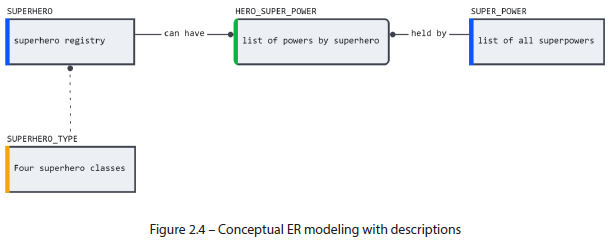
##### What It Is

* **Conceptual modeling = a process of identifying + visually mapping the moving pieces, or entities, of a business operation**
* **Entity** **= A person, object, place, event, or concept relevant to the business for which an organization wants to maintain information**
* Ex: Employee, customer, sale, item, etc.
* **Typically referenced in the singular + represent the *class* or *type* of an object** (more on this + the singular vs. plural naming debate in *Chapter 10, Database Naming and Structure*)
* **Entity instances = occurrences of such a class or type:**
* *Superman is an instance within the SUPERHERO entity*
* **Conceptual modeling** is a ***collaborative* process of discovery between the business and the data teams that produces visual project artifacts**
* **Before data about entities can be collected + stored, the entities themselves must be uncovered + agreed upon**
* **Goal of *conceptual* modeling = the *synthesis* of business + technical understanding between operational + data teams, + the diagram is the by-product**
* The conversations that take place in **conceptual modeling move the organization forward + add value in further stages of design + analysis**
* Because business teams are unlikely to be familiar w/ model design, it is possible that working w/ the data team forces them to think in such terms for the first time
* It is possible that when preliminary conceptual designs are initially drafted, business teams may be confronted w/ a reality that is not universally shared, + *this is a good thing*.
* When business processes are first set down in concrete, unambiguous diagrams, they bring to light misalignments that individuals unknowingly carry, allowing them to converge on the truth
* For instance, are we in the business of serving meals to people in restaurants or selling ingredients wholesale to restaurant kitchens?
* If so, by phone, web, mobile sales force, or all of the above?
* **The data team, *in isolation*, is unlikely to know for sure**
* **Once the domain experts have established the entities, the analysis broadens to uncover their properties + relationships**
* **Properties, or attributes** **= *descriptive pieces of information related to an entity***
* Ex: Superhero entity has a Birthplace attribute w/ the **value** Krypton for the Superman *instance*
* Ex: A customer might have a name, address, height, + driver’s license, **but *which of these are relevant to our business?***
* **Are there pertinent properties that were not included?**
* **The business teams can confirm this information**
* Finally, there is **the *interaction between entities*** = a **relationship**, **expressed as *verb phrases*** such as *is*, *has*, or *attends* w/ relationship names labeled in the diagram on the relationship line
* SUPERHERO *has* a SUPER\_POWER
* So far, we have 2 entities (SUPERHERO and SUPERHERO\_TYPE) + a relationship (SUPERHERO has a SUPERHERO\_TYPE)
* **Business teams must now confirm the granularity of *both* the entities *and* the relationship shared:**
* **Entity granularity**: This is **what a *single instance* of an entity represents**
* The SUPERHERO entity is maintained at the grain of an *individual* hero 🡪 A single row in the SUPERHERO table represents *one* unique hero + their respective attributes
* The SUPERHERO\_POWER entity is maintained at the grain of the hero superpowers 🡪 A single row in SUPERHERO\_POWER represents just 1 of many powers a superhero can have
* **Relationship granularity**: This is the **granularity *between*** **entities**, which consists of 2 parts:
* **Cardinality**: ***maximum* number of instances that 2 entities can share** 🡪 either ***one***or ***many***
* A SUPERHERO has one SUPERHERO\_TYPE
* A SUPERHERO can have more than one SUPERHERO\_POWER
* **Optionality**: **whether a relationship is *mandatory* or *optional***(e.g., *must*, *may*, or *can*)
* A SUPERHERO *must have* a SUPERHERO\_TYPE
* A SUPERHERO may *or* may not have a SUPERHERO\_POWER
* The word “*zero”* is often used as a synonym for “*optional”* in this context (e.g., *zero to many* instead of *optional many to many*) but should NOT be confused with cardinality
* More details on the related nomenclature + notation are in *Chapter 6*
* **When describing the granularity, all this information is condensed into *one* term**
* Suppose the business teams confirm that superheroes must be classified into 1 of 4 types + can have many or no superpowers
* In that case, we can describe the relationship between the entities as follows:
* A SUPERHERO *must have* one and *only one* SUPERHERO\_TYPE
* A SUPERHERO *can* *have* zero or one or more SUPERHERO\_POWER
* Even in this primitive example, the word salad is becoming hard to swallow
* **But if we are to create tables in Snowflake to accurately store this information, we must be able to *systematically capture these nuances*, + a diagram is an ideal way to do so**

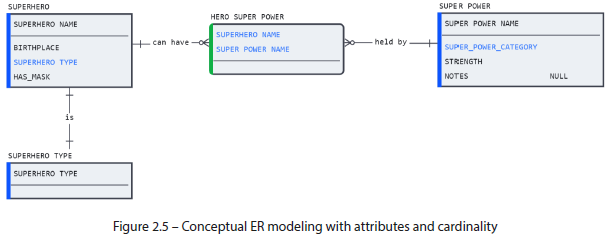
##### What It Looks Like

* It is easier to define a conceptual model by what it is *NOT* expected to contain than by what it does
* Unlike a *physical* model, which must have strict database-specific precision to be deployable, **conceptual models include as much or as little detail as needed to view + understand the details of the model at a high level**
* **At a minimum, a conceptual model must depict entities + their relationships**
* Beyond that, **details such as attributes, granularity, + even some logical features *may* be included**
* Below are 2 examples of the superhero model covered above





* **Even at the highest level of abstraction, conceptual modeling is flexible in its details and presentation to fit the needs of the use case**
* Above, we first see the model at its *lowest* resolution (as it might appear during the discovery phase of a development process), showing *only* entities + relationships:
* In the 2nd model, the same **design can be augmented w/ *descriptions* to give additional context to an audience who is not familiar w/ the domain in question**
* In this early stage, **details such as data types or granular attributes are NOT required**
* Ex: Listing an address attribute + unpacking later into constituent parts, such as street, city, and postal code may suffice:
* Ex: Superheroes:



* **Conceptual models remain useful well after deployment**
* New employees might use a conceptual model to acquaint themselves quickly w/ the business model
* A business user can likewise rely on the simplicity of the conceptual model to explore an adjacent domain to understand how it interacts w/ the one they are familiar w/
* **Conceptual modeling is also used to create mock-ups for planning new designs**
* Like a whiteboard, **conceptual models capture early-stage ideas + allow them to evolve as changes + details are introduced**
* *Unlike* a whiteboard, a conceptual model is **easy to modify + share with team members when maintained online in a collaborative modeling tool**
* Next phase of modeling is **logical**

#### Logical Modeling

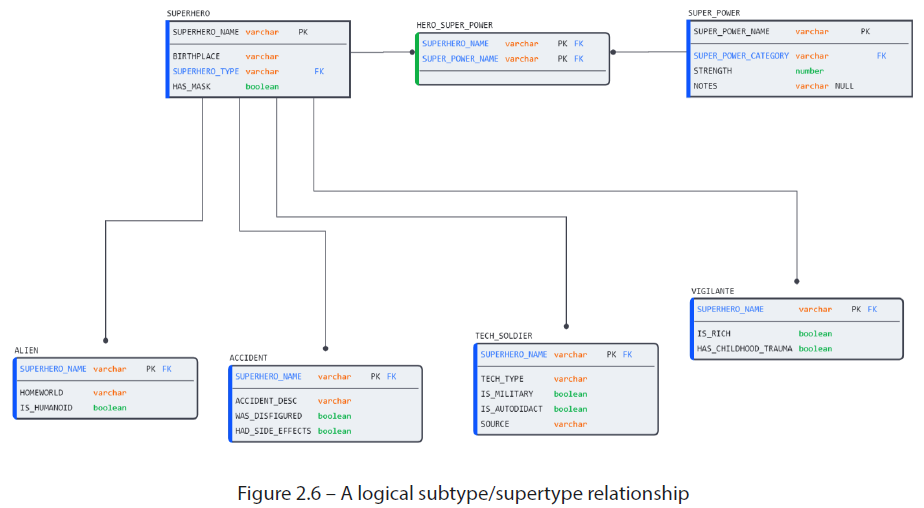
* **Once the building blocks of the organization’s business model have been identified through *conceptual* modeling (entities, attributes, relationships), logical modeling begins**
* However, **there is no strict distinction between the elements used in logical and conceptual designs**
* In fact, many database design textbooks do not differentiate between the two, combining them into a single style + tackling them in a single step
* We **distinguish between conceptual vs. logical modeling**, not due to the elements or their notation, but **due to the natural flow of the design process**
* Because database textbooks are geared towards a technical audience, many lose sight of the less technical participants of database modeling: the *business users*
* **Although a logical model can express everything a conceptual one can, it also includes a great deal of *technical* detail, which may alienate team members who lack a foundation to make sense of it**
* **In the initial stages, domain knowledge is more important than technical details**, so it **may help to pivot between the two modeling styles, depending on the audience.**
* Unlike paper or a whiteboard, a **modeling tool will allow you to switch between different modeling styles + notations seamlessly** w/out affecting the underlying objects, allowing you to choose the appropriate level of abstraction, depending on the audience

##### What It Is

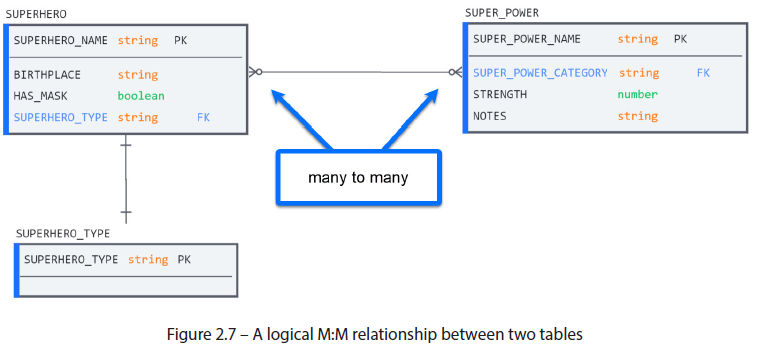
* **Logical modeling = the bridge between the business’s conceptual operating model + the physical structure of the database**
* It **uses conventions that give context to the nature of the data beyond what conceptual or physical models can express**
* It **augments the conceptual by adding notation describing relationship metadata and incorporates elements of physical database design, such as data types + keys**
* **Logical concepts such** as **subtypes** and **supertypes** may already be familiar to those w/ a programming background, but may appear arcane to business users.
* **In the logical modeling phase, technical details required for database implementation, such as data types + keys, are also introduced, *albeit in a simplified form***
* What separates logical modeling from the physical is that **the technical details are NOT specific to any database type**
* Sitting between the physical + conceptual realms allows logical modeling to bridge both worlds and express nuance that would otherwise be lost in the rigidity of *pure* database objects
* In short, **logical modeling helps teams ascertain business nuance on top of a model that closely resembles their existing (or soon-to-be) physical landscape**

##### What It Looks Like

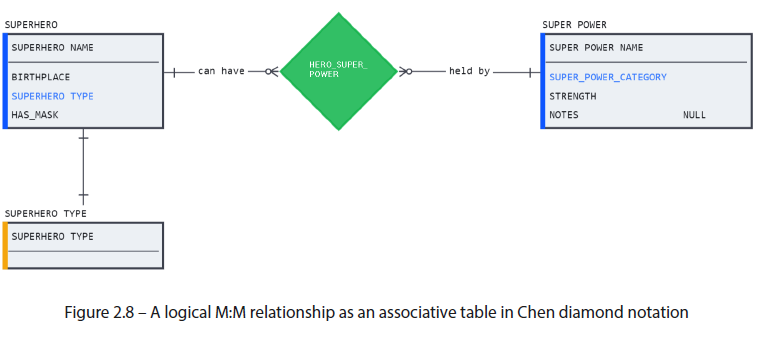
* The **logical modeling phase includes details such as data types, unique identifiers, + relationships**
* ***Contextual notation* for subtypes + supertypes, as well as many-to-many relationships, are also included**
* **Subtypes are used on logical diagrams to represent inheritance**
* In the superhero example, the SUPERHERO\_TYPE *attribute* can be used to **create subtype tables based** **on the** SUPERHERO **supertype**
* **This way, common attributes** such as SUPERHERO\_NAME **can be shared, while attributes *unique* to each subtype can be maintained in the respective subtype tables** (such as the IS\_HUMANOID indicator for alien savior-type heroes)
* Here’s what it looks like for a subtype relationship that is both **complete** (double parallel lines) and **exclusive** (letter **X** in the center):



* **Many-to-many relationships** (**M:M**) **present a unique challenge for logical models**
* In our superhero example, we established that a superhero might have many powers (or none)
* Also, a superpower can be shared by many superheroes (or none)
* In short, *many superheroes with many superpowers*
* The **problem is that in a *physical* database, there is no way to store this information w/out duplicating the data in both tables**:



* **To represent an M:M relationship in a *physical* database, an intermediate associative table is required**
* Although M:M relationships between tables are *allowed* in logical designs, they violate the first 2 modeling principles outlined earlier by using 2 tables to signify the existence of 3
* To avoid such confusion, use the diamond from the Chen notation to label the associative table in an M:M relationship:



* Here, it is clear to the observers that an M:M relationship exists + that a 3rd table will be required (containing its own independent attributes) to capture the required information
* Once the logical context has been defined, work on the physical model can begin

#### Physical Modeling

* *Chapter 1* separated the many meanings + applications of **the word “*model”***, noting how it **is used for the physical structure of database tables *and* the accompanying visual diagram**
* A **physical model contains all the information necessary to construct + deploy the design to a specific database (e.g., Snowflake)**
* This also works in the *opposite direction*, as **many modeling tools + SQL IDEs can reverse engineer diagrams directly from the database DDL**
* **Reverse engineering lets users generate visual representations of the underlying schema on the fly to help them explore relationships.**
* But how exactly does this conversion work?

##### What It Is

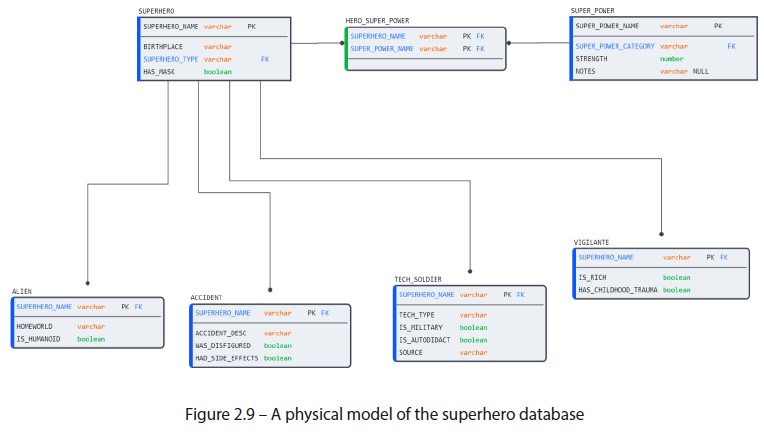
* The **physical model is a blueprint for deploying a database** ***and* the map that holds the technical + operational links that help users navigate it**
* The **transition from logical to physical modeling is largely systematic** (“relationships” become “constraints”, data types are converted to database equivalents, + database properties are defined), but **some design decisions may still be required** (e.g., **rollups** for subtypes (see *Chapter 11*))
* Such decisions will **depend on business factors + future data consumption patterns**.
* Some modeling tools separate logical + physical models into different “projects”
* Others, such as SqlDBM, keep conceptual, logical, + physical models in sync, allowing users to view a model at any level of detail without conversion
* *NOTE: Logical data types and Snowflake conversion*
* To facilitate migrations from other database platforms, Snowflake offers various compatible data types as synonyms for its own
* Ex: Defining a column as either STRING or TEXT is allowed, but would result in the column being created as a Snowflake standard VARCHAR(16777216)
* This means that generic data types often used in logical models, such as STRING or INTEGER, can be used + automatically generate Snowflake equivalents upon deployment
* More info can be found at <https://docs.snowflake.com/en/sql-reference/intro-summary-data-types.html>
* **Physical modeling is the final step before deployment**
* While the diagram displays data types + constraints, **much more detail is embedded w/in the objects as part of this phase**
* **Clustering keys, table types, + identity properties are set where required + supporting objects such as sequences + file formats are also created + assigned**
* All this happens during a new development, but **physical modeling is also used for existing schemas**
* For organizations whose databases lack a formal model document, physical modeling (through reverse engineering) is the most direct way to create one
* Many modeling tools can even suggest relationships in schemas where constraints have not been defined by detecting naming patterns
* Although this lacks the metacontext of a logical model, it is a significant improvement over having no diagram at all

###### What About Views?

* So far, we have been exploring database modeling *in parallel* w/ the business model **by capturing and storing business-critical information as entities, attributes, + relationships**
* **Views, however, do NOT store any data, as they are SELECT statements stored as database objects**
* While it may be impossible to construct the CUSTOMER entity in a *transactional database* as a view w/out an underlying CUSTOMER table, **DWs may use a view to logically unify several CUSTOMER sources into one dimension**
* In such a scenario, is the view not part of the CUSTOMER model? 🡪 **Of course, it is + it should be visible in the physical model**
* However, the *model* used by such a view is *transformational*, NOT physical (covered next)
* It’s unfortunate that the word *logical* has already been taken by a class of modeling + can’t be used to express logic-based transformations such as views and **CREATE TABLE AS SELECT (CTAS)** tables
* Alas, we must adhere to the principle of unambiguity, which states *never give 2 meanings to the same convention*, so that’s that
* So, *how does a physical model differ visually from those discussed previously?*

##### What It Looks Like

* **Physical diagrams look remarkably like their logical counterparts**
* However, they **pack in much more information**
* Although the presentation options are configurable in most modeling tools, **essential elements such as a complete column list, data types, + constraints *must* be defined**
* However, a **physical model has many more details than can be displayed at a given time**
* Each element in the diagram (relationships, tables, + columns) holds its own set of physical attributes, which can be edited separately
* This is an example of the superhero database seen as a physical model:



* **A physical model, therefore, presents the viewer w/ only those properties that can be translated *directly to the database***
* The **relationships**, as previously discussed, **are as much a technical asset as they are clues to the operational nature of a business**
* Even in a DW, where much of the landscape is defined through **transformational modeling**, where the **relationships outlined in the relational model help inform the connections + the JOINs that are made to make the transformations work**

#### Transformational Modeling

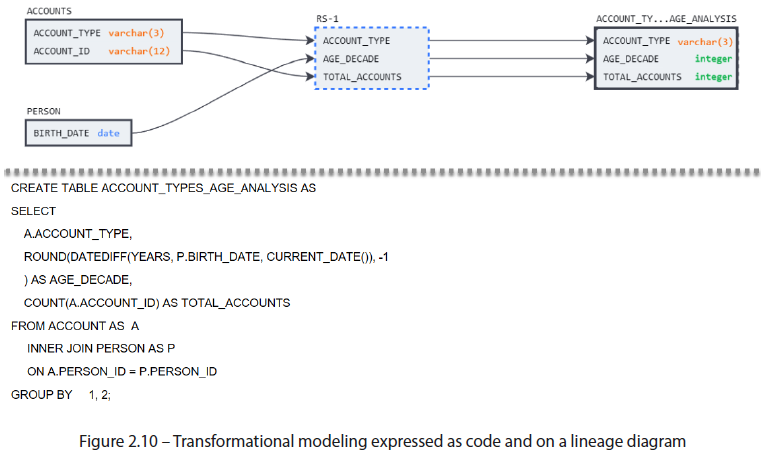
* **Modeling through transformational logic is a powerful + highly maneuverable method for modeling data that comes w/ one serious drawback: *it needs existing data to SELECT from***
* **Transformational modeling is rarely done in *transactional* databases** b/c, in such systems, **data is created + modified through the company’s *operational processes*** (e.g., purchases + profile updates), w/ which **expensive transformational processes should NOT compete for critical system resources**
* **However, in *DWs*, where conformed datasets are extracted + duplicated w/ fresh timestamps for each load, transformational modeling becomes a *necessity***
* **Because transformational modeling SELECTs from *existing, structured* data, the result set is already structured**
* Ex: SELECT-ing the SUPERHERO\_NAME and HAS\_MASK columns and creating a table will preserve their structure (VARCHAR and BOOLEAN, respectively)
* **However, as w/ all modeling, transformations must also be duly documented to allow database users to make sense of the complex data landscape that results**

##### What It Is

* **Transformational modeling, unlike relational, *isn’t* concerned w/ the structure of the data it is meant to hold b/c *it uses existing data as part of the creation process***
* **Transformational modeling creates new objects *by selecting data from existing sources***, **done through views using the CREATE TABLE AS SELECT (CTAS) command, or other DML statements**
* But there is more to transformational modeling; it is also **the MERGE and the bulk INSERT** used to keep data in such tables up to date
* **When it comes to data, OLTP databases generally hold just one source of truth (the latest)**
* **OLAP systems, by design, are *disconnected* from the latest data and must be loaded + synchronized on a schedule**, + a **transformational approach is used to display latest changes**
* **OLTP databases are generally used in *operational* use cases (Ex: sales, reservations, or application data) + operate on the CRUD of individual records**
* In contrast, **DWs rely on transformational modeling to unify data from disparate sources + create versions of the conformed data at various levels of detail + aggregation**
* Transformational logic is also used by BI + dashboarding tools when building visualizations + summarizing information
* Such tools automate the work of writing queries by performing the required joins + aggregations when data is pulled onto the screen in the form of tables and charts.
* Since transformations are expressed as code, *what can we visualize with transformational modeling?*

##### What It Looks Like

* **The best abstraction for the complexity of database logic is *the SQL that produced it***
* **At its core, transformational logic is SQL** 🡪 it is the **CTAS statement** that creates + populates a table, **but also the INSERT or MERGE statements** that may be used to update it, **or the VIEW** that these statements may call to do so
* However, just as **reverse engineering** DDL can produce ER diagrams from DDL, it **can also generate lineage diagrams from transformational SQL logic**
* Unlike relational diagrams, **lineage graphs** are **low-res depictions of data flow from source to target in a SELECT** **statement**
* W/out the accompanying SQL, the details that led to the creation of the table are lost:



* Unlike the other modeling diagrams (where pictures tell a clearer story than DDL), **transformational modeling benefits from having code visible alongside the lineage graph**
* It allows users to acquaint themselves w/ the inputs + outputs of the transformation at a high level before diving into the details of the logic if they so choose.

#### Summary

* **Modeling is a process used to agree, plan, + develop a database design *and* a means to navigate and explore it to gain greater insight into the underlying business context**
* Every new project must pass through the **4 modeling stages**, whether formally recognized or not
* But even for *existing* databases that have *not* been adequately documented, **reverse engineering is an effective mechanism to work backward from a database to uncover its business meaning**
* The design journey starts w/ **conceptual modeling** (a **collaborative process that involves business and data teams working together to understand the core elements that underpin business operations + how they interact**), which **favors simplicity over detail** (making them accessible to team members of all backgrounds + helping steer conversations to converge at a shared understanding of the business model)
* After conceptual modeling, the **data team can *add further business context* to the diagram using logical modeling elements**
* **Logical models** **resemble physical designs but offer business nuances, such as many-to-many relationships, subtype classes, + relationship names, none of which can be represented in the database**
* This **also makes logical models an ideal bridge for transitioning to physical modeling**
* **Physical modeling** is **concerned w/ database-specific details to ensure that business rules are captured correctly + w/ the proper parameters to ensure efficient performance**
* Since **entities, attributes, + relationships have already been nailed down through conceptual + logical collaborations**, the **focus can turn to physical database details, where database-specific properties are defined**
* **Transformational modeling** can begin by ***using* the physical model as an orientative (serving to orient) map, reshaping data into the format (or formats) best suited to answer business questions and drive decision-making**
* Although transformational modeling is performed purely through SQL logic, **accompanying visuals (as in all types of modeling) can help accelerate the understanding + maintenance of complex logic.**
* The next step is learning to apply them the types of modeling that exist

#### Further Reading

* Effective communication between technical + business teams is notoriously challenging
* In *The Rosedata Stone: Achieving a Common Business Language Using the Business Terms Model*. Technics Publications, 2020, Steve Hoberman presents a guide for creating a precise diagram of business terms w/in your projects
* The business terms model is a simple yet powerful communication tool to help bridge the technical knowledge gap and meet business teams on their own *terms*