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

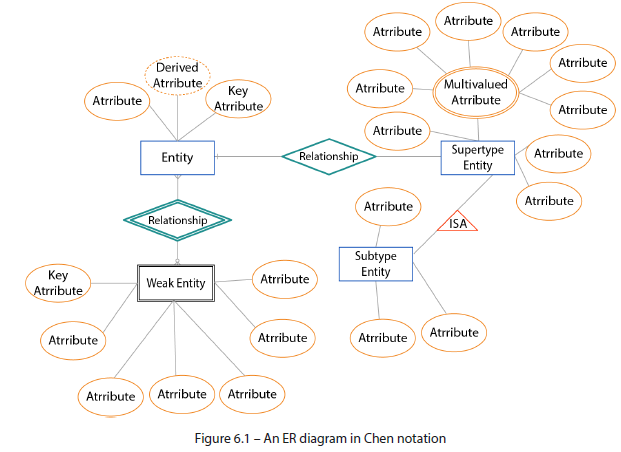
## Part 1: Core Concepts in Data Modeling and Snowflake Architecture

### Chapter 6: Seeing Snowflake’s Architecture Through Modeling Notation

* Throughout this book, **relational diagrams** have been used to support examples + illustrate ideas in ways that words could not
* Although various modeling styles + notations have been introduced, a thorough overview of the **visual semantics of modeling + the various elements + their properties has not been covered**
* In this chapter, we run through the complete visual toolkit that will enable users to accelerate the design + understanding of a physical Snowflake database + add functional context through conceptual conventions
* Using a simplified + pragmatic approach, users of any background can view + explore a database schema at the level of detail that best suits their needs
* Main topics:
* Recall the history of **modeling styles + notations**
* Understand the connection between the relational model + its visual depictions
* Learn how to depict the various types of **entities**
* Explore how to represent **relationships** at the physical and conceptual levels
* Understand how to add **functional context** to go beyond physical tables
* Discover how to simplify various modeling conventions to arrive at a **synchronized modeling practice**
* Do it all in a manner that is **accessible to technical and business users** alike

#### A History of Relational Modeling

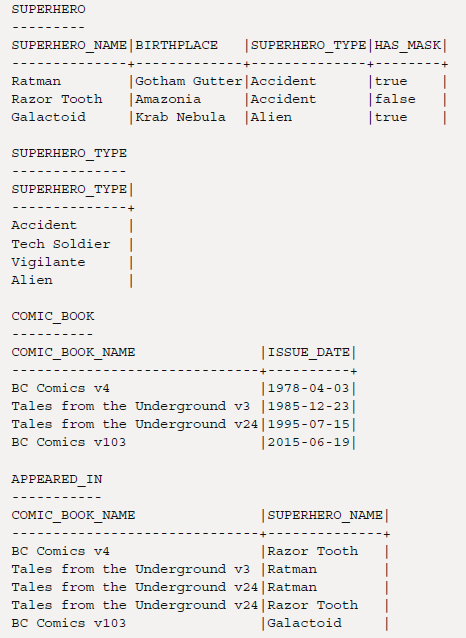
* The diagrams + examples in this book rely on **relational modeling** to illustrate database concepts + constructs
* Learning how to parse + communicate through **relational diagrams** provides a dual-faceted benefit to database users by allowing them **to rapidly visualize + bring to life complex database landscapes as well as design them from scratch using visual cognitive aids**
* Yet, despite these benefits, many database users consider modeling an arcane practice w/out relevance in a modern DW + mistake it for a chore instead of the time-saving boon it really is
* The practice of **data modeling** + its associated notations have passed through many changes and diverging trends w/out a final standard ever emerging, encumbering universal adoption
* The **conceptual data model** dates back to the ‘60s when Charles Bachman first used rectangles to denote record types + arrows to form relationships between them
* Toward the end of the decade, Edgar F. Codd formalized the **relational model (RM)**, which gave **rise to the RDB as we know it today**
* Using the RM, Codd managed to tie mathematical theory, consistent w/ the structure + language of **first-order predicate logic**, to database systems
* Unfortunately, Codd was less concerned w/ the visual aspect of the model + offered no diagramming guidelines
* In the ‘70s, computer scientists such as Peter Chen + Richard Barker expanded Bachman’s simplistic modeling conventions to include more context for the underlying **entities** **+** **relationships**
* However, none of these designs were convincing enough to warrant widespread adoption
* While singular elements of these designs gained prevalence (such as the use of crow’s foot notation to describe cardinality) wholesale adherence to any one diagramming style proved too restrictive for widespread adoption by failing to strike a balance between readability and technical detail
* *Some, including Chen’s original ER format, arguably, achieve neither*
* The following diagram in Chen notation attempts to describe a simple RM consisting of 4 entities and their attributes
* *Yet, it fails to do so effectively due to an overwhelming number of visual elements.*



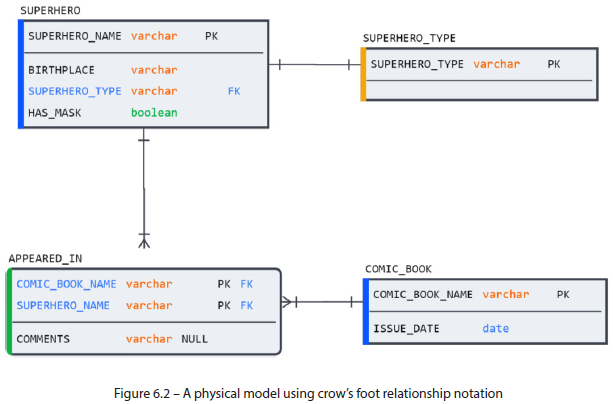
* **To effectively use visual diagrams, it is essential to separate the model (structure) from its presentation (diagram)**
* A data model’s visual representation may vary depending on the level of abstraction, but **the elements it represents correspond to real-world business entities that are NOT subject to reinterpretation**

#### Relational Modeling Vs. Entity-Relationship Diagram

* Recall that **an RM can be identified from business-relevant concepts + mapped onto a database schema that includes their attributes, identifiers, + relationships**
* An **entity-relationship diagram** (**ERD**) is **a *visual rendering* of the RM, using simple + easy-to-understand semantics**
* In the following examples, we will revisit a set of relational **entities** from Chapter 1
* In the RM, each of the 3 entities corresponds to a table identified by a PK + related to other entities through FKs
* Just observing the table names or even a data preview does not make it easy to determine these relationships, as can be seen in the following RM:



* However, **an ERD brings the associative context of the tables into rapid focus by highlighting the entities + relationships behind the data being presented**



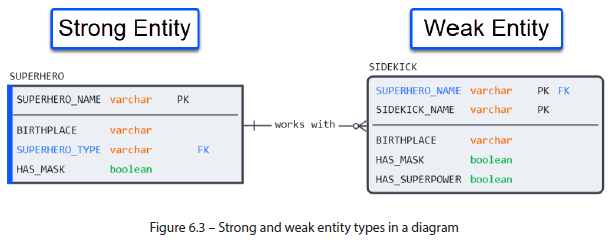
* Thanks to visual clues in the ERD, the context *behind the data*, such as the **many-to-many (M:M)** relationship between COMIC\_BOOK and SUPERHERO, instantly comes into focus (*that is, if the viewer is familiar w/ the visual semantics that the ERD is using*)
* Let us review these elements in detail + demonstrate how we can represent a wide array of technical and conceptual modeling conventions using a few visual conventions

#### Visual Modeling Conventions

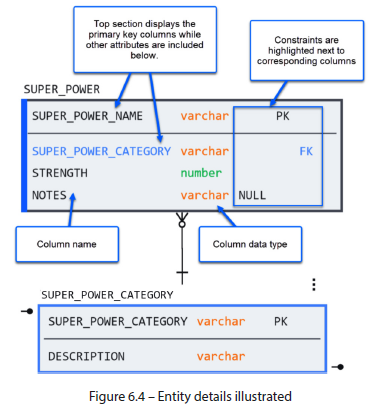
* This chapter highlights all **semantic elements** that will be used throughout the rest of the book
* Recall: ***No* fixed + universally prescribed standard for drawing modeling diagrams exists**
* Conventions used here, along w/ their accompanying visual semantics, are the author’s attempt at simplifying the broad spectrum of modeling concepts and notations into a core set of elements, using the most widely used symbols to represent them
* **The visual semantics in this book do *not* conform to any single modeling notation but borrow freely from several leading industry standards**
* The aim is to provide a m**odeling language that is both precise enough to accurately reflect technical database details + simple enough for business users to understand the business context that lies beyond the technical specifications**
* Unlike textbooks that separate conceptual, logical, + physical models and force users to convert and adapt one to the other, this book will use conventions that make it possible to keep all 3 in sync throughout the modeling process
* This approach minimizes the effort involved in maintaining the models consistently as development unfolds and offers a consistent view at various levels of abstraction.

##### Depicting Entities

* Since the early days of modeling, **entities have been depicted using *rectangles***
* Whether displayed on a conceptual diagram w/ no additional detail besides the entity name or on a physical diagram containing a list of attributes, data types, + other information, entities are almost *always* represented as rectangles
* However, modeling semantics *do* distinguish between **strong entities** and **weak entities**
* When we think of an “entity”, we generally think of a **strong entity (identified by its PK)**
* However, **weak entities *cannot* stand independently + *depend on attributes from another table to help identify them***
* They **must use a FK as *part of their PK* because the information they contain does not make sense in isolation**
* This dynamic can be observed in the SIDEKICK weak entity table below, which can only exist in relation to the strong entity, SUPERHERO



* On a diagram, **entities are depicted as rectangles**, w/ **corners depicting their strong or weak nature**
* **Strong entities have straight, perpendicular corners**, while **weak entities have rounded corners**
* Now, let’s zoom in on the details *inside* the entity
* **In logical + physical modeling, entities include details about their attributes, constraints, + data types**
* The following figure explains what the details inside the entity represent:



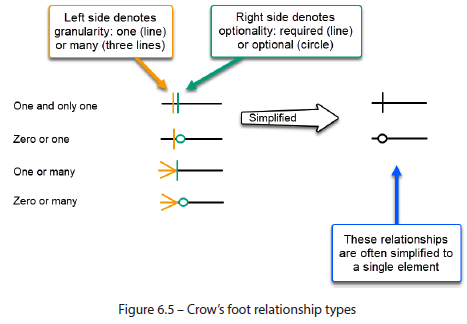
* Now that we know how to draw entities, let us learn how to represent their **relationships**

##### Depicting Relationships

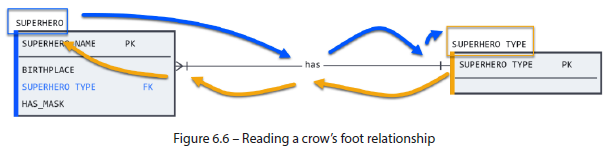
* **Relationships between entities are represented as connecting lines**
* However, **a relationship can convey more context than a solid line can capture**
* **Relationship lines can *also* relay information about cardinality + dependency between associated entities**
* Some of these details map perfectly onto the physical database, while others are conceptual and orientative
* **Crow’s foot (more common) depicts granularity**

###### a) Crow’s Foot

* **Crow’s foot notation derives its name from the 3-pronged *many* symbol**
* **Each side of a crow’s foot connector displays information about the cardinality + optionality of the relationship**
* There are **2 cardinality options** (**one** or **many**) and **2 optionality indicators** (**required** or **optional**)
* This gives us **4 possible combinations,** as described in the following figure:



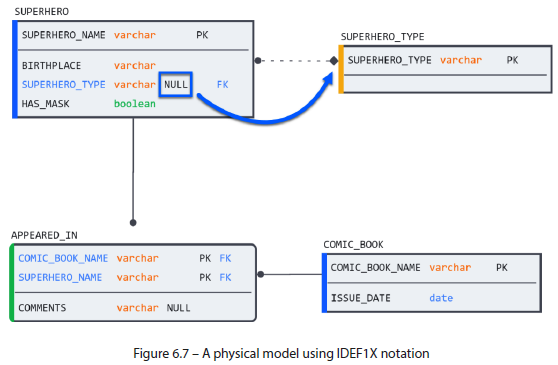
* In **logical/conceptual modeling**, a **relationship name** (also known as a **role name**) **can be specified to add context**
* **When specifying cardinality, remember that it applies to BOTH edges of the relationship (meaning, a relationship can be read in either direction)**, as shown in the following example:



* The diagram is read **from the entity to the role name**, **to the connector on its far side**, **to the other entity**
* Take the following example:
* A superhero has one and only one superhero type
* A superhero type is had by one or many superheroes
* **Using crow’s foot notation, database modelers can capture functional details of the minimum + maximum granularities in entity relationships**
* **However, crow’s foot cardinalities do NOT map one to one** onto physical designs
* **When technical precision is needed for physical models, IDEF1X notation is preferred**.

###### b) IDEF1X

* **IDEF1X** (correctly pronounced *eye-def-one-ex* but often shortened to *eye-dee-fix*) notation was invented in the ‘70s by the US Air Force
* **The IDEF1X appeal is how closely it maps onto physical database structures**
* The **entities + relationships defined in IDEF1X notation can be unambiguously translated into SQL and vice versa**
* **NOTE:** While the IDEF1X standard *does* include a specific connection relationship syntax (e.g., *P, Z, n, n-m*), this syntax is NOT easily readable and is NOT widely used
* *Instead* of *cardinality*, **IDEF1X notation focuses on the strong/weak entity nature of a relationship**
* **Strong entity relationships are depicted by dotted lines** and are referred to as **non-identifying**
* **Relationships to weak entities use a solid line** and are called **identifying** b/c **they require a PK from a strong entity to identify a unique instance**
* Because IDEF1X notation is tied to physical database properties, it is **limited to optionality options directly supported by the database**
* When a non-identifying FK is NULL-able in a child entity, the parent edge of the connector is displayed as a diamond instead of a circle
* In the following diagram, the weak entity, APPEARED\_IN (displayed w/ corresponding rounded corners) connects to strong entities using solid lines
* For demonstration, the SUPERHERO\_TYPE FK in SUPERHERO has been changed to accept NULLs therefore causes the SUPERHERO\_TYPE edge to take a diamond symbol.

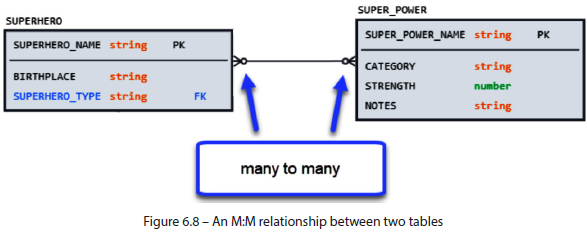


##### Adding Contextual Context to a Snowflake Architecture

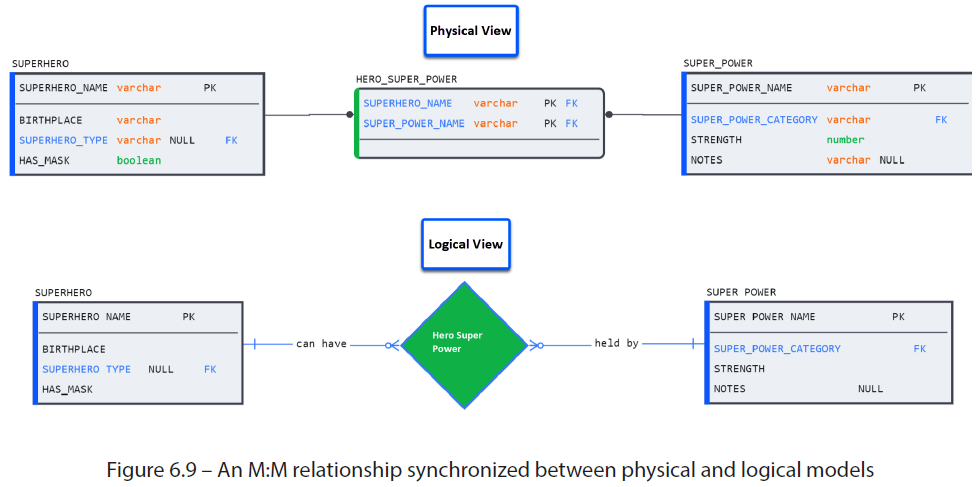
* In the overview of logical modeling in Chapter 2, we introduced concepts such as the **M:M relationship + subtypes**
* **M:M relationships** are peculiar b/c they **cannot be created in a physical database w/out an intermediary associative table**
* **Subtypes** also hold a special significance beyond mere child tables as they **embody the principles of inheritance in entity classes**
* To ensure that this context is not lost in a physical database, **logical modeling conventions exist to describe them**

###### a) Many-to-Many

* Although **crow’s foot notation allows for M:M relationships in theory,** it turns out that **in practice,** the approach suffers from one significant limitation: **it prevents modelers from keeping their conceptual/logical model in sync w/ their physical model**
* Recall that the following relationship is impossible to create in a physical database w/out using an **associative table**

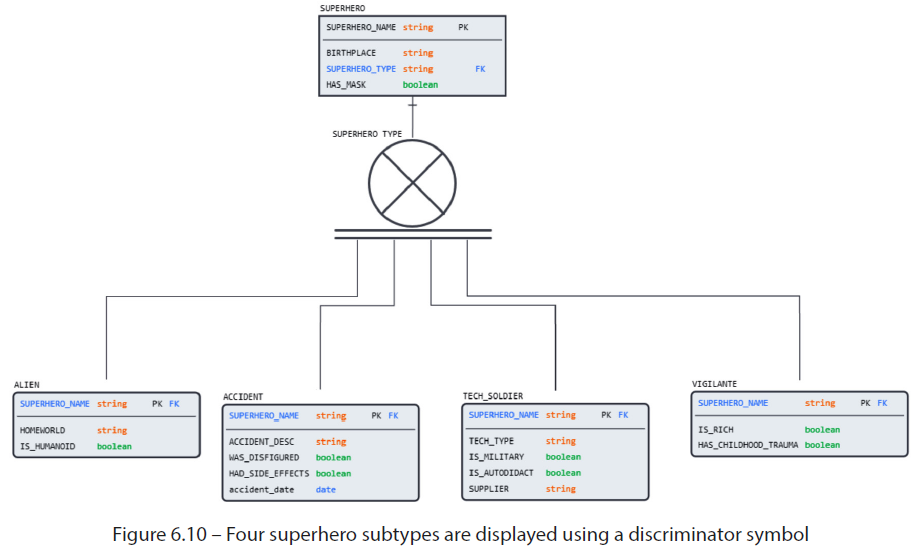


* **However, by coopting the relationship diamond from Chen notation to represent the associative table, we can keep the physical + conceptual models in sync while keeping the necessary context within the model**
* The following figure shows the same M:M relationship in both physical and logical models

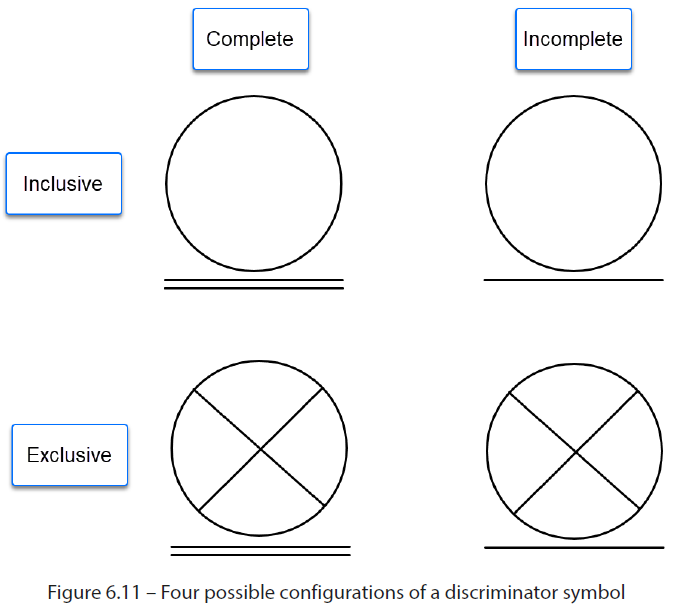


###### b) Representing Subtypes and Supertypes

* In Chapter 2, we saw how **subtypes** **were used on logical diagrams to represent inheritance**
* Recall the example in the following figure showing how a SUPERHERO entity can have 4 subtypes:



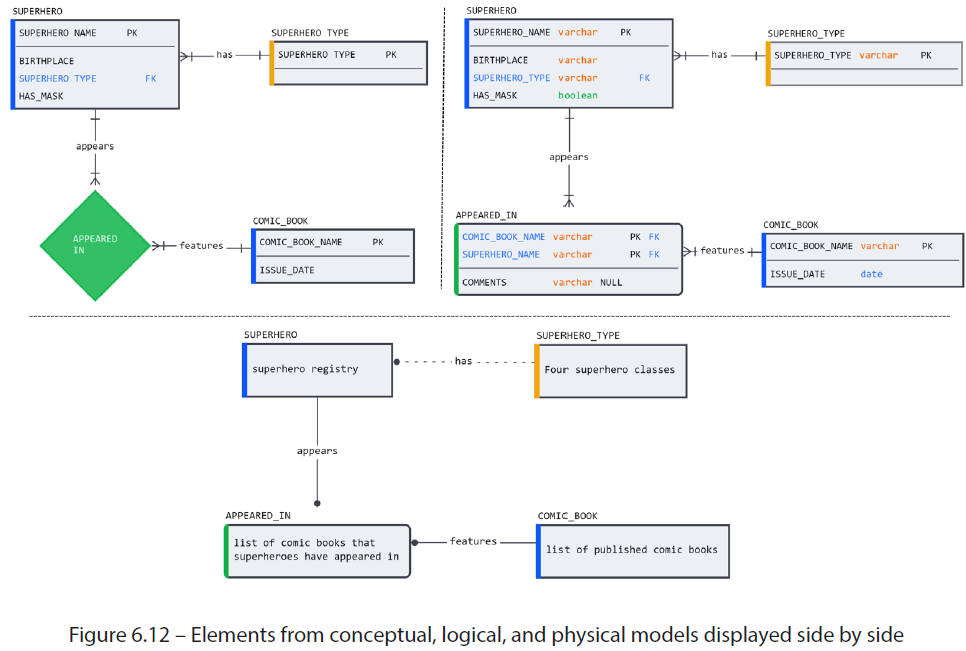
* The figure not only identifies the 4 tables as subtypes of SUPERHERO but **also gives us context into the *nature* of the subtype (that it is both complete (double parallel lines) and exclusive (letter *X* in the center)**
* Let’s understand these properties and how to denote them.
* **Subtype relationships have 2 binary properties to describe their nature:**
* **Complete/incomplete**: **Indicates whether all subtypes are known or whether more can be added in the future, respectively**
* The above diagram is an example of a complete relationship, as all 4 superhero types are present
* **Inclusive/exclusive: Indicates whether the supertype can exist as multiple subtypes**
* Since a superhero can only belong to one of the 4 subtypes, the relationship is exclusive
* **These properties can be displayed using a mix of IE and IDEF1X notation**
* The central discriminator icon can be displayed as a circle or the letter D rotated 90 degrees counterclockwise
* The defining elements are the letter *X* through the middle to denote an exclusive relationship + single or double lines at the bottom to describe complete + incomplete, respectively
* The 4 configurations can be seen in the following figure:



* Now that we’ve covered how to depict a physical database on a diagram + extend the visual context beyond what the physical database is capable of storing, consider how we can take advantage of the primary challenge we laid out at the start of the chapter: **keeping the models in sync**

#### The Benefit of Synchronized Modeling

* Throughout this chapter, we have seen examples of diagrams describing data models + capturing details that may not have a strict corollary in SQL
* **By simplifying modeling to its core components** (many of which are likely to be understood even by people without a formal background in database design), we **ensure that modeling is not confined solely to the database team**
* As you recall, **a model describes not only the database but also the business itself**
* Keeping the **physical + conceptual models in sync provides another benefit**: **the model can be viewed at any time at various levels of abstraction because the underlying physical elements remain the same**
* Users can **view the model at a physical level when more detail is needed + flip to a conceptual view if they wish to see it more abstractly**
* To illustrate this, the various diagrams in beloware displayed side by side, demonstrating how physical + conceptual elements can be mixed + matched as necessary to perform an analysis



* **Keeping the entities consistent across models carries an additional benefit in that no other maintenance is required to keep the models consistent**
* **By changing the detail level, no conversion or rework is needed to adapt one style to the other**

#### Summary

* In this chapter, we looked back at the history of relational modeling + the various notation styles that have been used to help visualize the designs
* Focusing on the most straightforward + practical semantics of these modeling styles, we could zero in on a pragmatic approach that preserves technical precision while being simple enough to be understood even by people without a formal background in database design
* **The modeling technique proposed in this book keeps the number of entities synchronized across conceptual, logical, + physical models, thus saving manual effort + ensuring that anyone at any level of detail can view the same model.**
* Using the consolidated list of modeling conventions, we looked at how to display an entity on a relational diagram in the simplest way possible
* Next, we covered relationship notation + the additional context that can be conveyed beyond a simple line using 2 of the most popular conventions in database modeling: crow’s foot and IDEF1X
* The former is ideal for communicating granularity details between entities, the latter uses a simplified style that maps exactly into the physical model.
* We also learned how to go beyond what the physical model can display + added functional context through conceptual/logical modeling conventions such as subtypes and M:M relationships