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

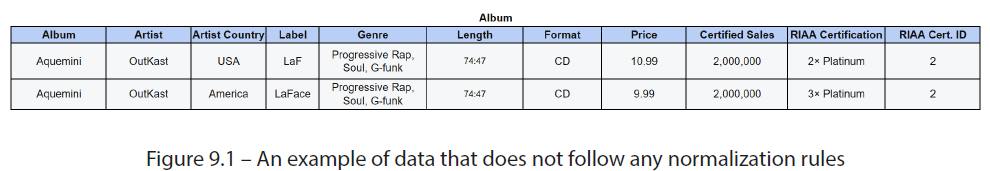
## Part 2: Applied Modeling from Idea to Deployment

### Chapter 9: Database Normalization

* Previous chapters explored the method of **capturing the real-world business workings of an organization + modeling them using visual semantics**
* The **resulting model + accompanying diagrams make it easy for the domain + data teams to reach a consensus about the business’s fundamental entities + the interactions between them**
* However, as the modeling process approaches the physical stage, we should understand that **many ways exist to structure the data at the database level**
* The **process of dividing the data into smaller, modularized segments is known as normalization**
* **Normalization is *NOT* a binary classification but a *spectrum* of ever-increasing rules a design must satisfy to achieve a stated level**
* Despite containing the root word *normal*, along w/ obvious positive connotations, ***more* normalization does NOT necessarily imply a *better* design**
* **Greater levels of normalization impose more rigor on the database’s structure**, reducing data anomalies at the **cost of increased maintenance + analytical complexity**
* We explore normalization through its various levels to understand rules, advantages, + trade-offs
* After a thorough overview, we focus on the most common types of normalization + understand the aspects that make them popular choices for database design
* Main topics:
* **Understanding** normalization on a **spectrum**
* The **pros and cons** of normalization
* **How much is too much?** Finding the sweet spot
* Reviewing the **data anomalies** that normalization protects from
* Taking an in-depth look at normalization from the **1NF to 6NF**

#### Overview of Database Normalization

* Database **normalization = the process of organizing a database in a way that *reduces redundancy and dependency* w/in its tables**
* **It is achieved by breaking a large table into smaller ones + linking them through FK relationships**
* Doing so **leads to fewer data inconsistencies + improved data integrity**
* A **normalized database results in a modular design that is easy to scale + modify**
* **It occurs through escalating stages of formal rules called** **normal forms**, ranging from the **1NF** to **6NF** (although the **1NF-3NF are most commonly used + are sufficient for most use cases**)
* Each normal form builds on the requirements of its predecessor + adds additional criteria that every database table *must* satisfy
* A **normal form is considered satisfied when every table in the database meets the criteria laid out for it (and, by extension, its predecessors)**
* **Neglecting normalization rules may result in erroneous self-contradicting data that’s difficult to manage**
* The following screenshot shows a table that violates nearly every consistency principle there is:



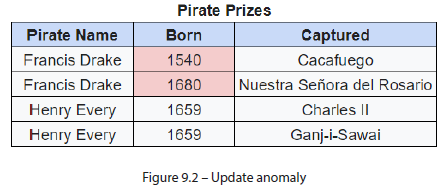
* How many data issues could you spot in just 2 rows?
* No unique identifier makes knowing what a single row represents difficult
* Multiple values stored in the Genre column hinder the analysis
* The 2 rows appear to be duplicated but there is no way to tell without a PK
* *USA* and *America* are two values describe the same entity
* Duplicate data entry in the LABEL column results in misspelled values
* The PRICE value is different between the 2 records w/ no way to know which is correct
* The RIAA CERTIFICATION value is a function of CERTIFIED SALES, so “3x Platinum”with 2,000,000 copies sold is impossible
* RIAA CERTIFICATION is at odds with RIAA CERT. ID
* Adding a new FORMAT value would require duplicating all ALBUM values
* If the store in question wants to stop selling the CDformat, deleting both rows would also remove album information
* **Normalization helps prevent data anomalies such as the preceding ones through schema design and database rules**
* As **more normalization** is applied, the **possibility of anomalous information drops while the overall number of tables grows**
* **Data quality safeguards obtained through normalization are offset by the complexity of querying information across multiple tables instead of having it in a single source**

#### Data Anomalies

* Bad data comes in many flavors, + from misspellings to improper encoding, + **some data quality issues cannot be avoided**
* However, **DE-normalized designs make it possible to walk headlong into several well-known and preventable blunders**
* To understand how normalization prevents data anomalies, we need to unpack the **dual dangers it mitigates: redundancy and dependency:**
* **Redundancy** = **Repeated data, whether w/in one or across multiple tables**
* When data values are duplicated, **synchronizing everything through DML operations becomes harder**
* **Dependency** = **When the value of 1 attribute depends on the value of another**
* Dependencies can be **functional** (Ex: a person’s age attribute depending on their name) or **multivalued** (Ex: name, age, + hobby stored in a single table would make it impossible to delete a hobby w/out deleting all the people who practice it)
* With these dangers in mind, let’s review the kinds of data anomalies that have the potential to creep into a denormalized database design

##### UPDATE Anomaly

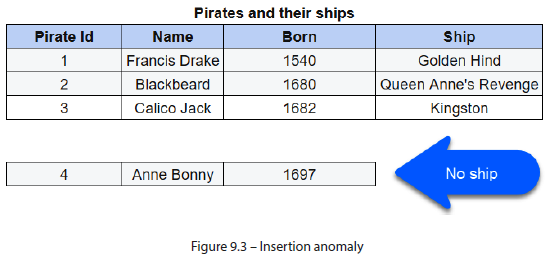
* An **update anomaly** results from a ***partial* update of *redundant* information**
* **Ex: When information is duplicated over multiple records or tables, it becomes possible to update *some* but not *all* occurrences, resulting in conflicting or contradictory instances**
* In the following example, the birth year, an attribute of the PIRATE dimension, is included in the PIRATE PRIZES fact table
* Birth year detail provides extra context for consumers of the fact table *but results in duplicate data + opens the door to an update anomaly*, as displayed below:



* Because the birth year value is *repeated* instead of *stored once as an attribute* of the PIRATE entity, anomalous + contradicting records can occur
* However, that is not the only anomaly that can result from placing details from various dimensions in a single table

##### INSERT Anomaly

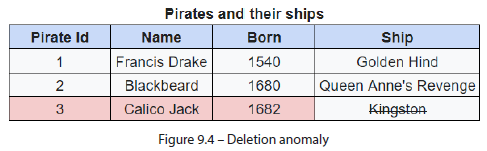
* An **insertion anomaly** occurs when it is **difficult or impossible to insert new data into the database consistently + accurately**
* When **too many details of *varying* granularity are bunched into a *single* table, inserting information that does not simultaneously tick *ALL* the boxes becomes difficult**
* In the following example, pirates and ships are stored in the *same* table, + both values are mandated through NOT NULL constraints
* Here, “Anne Bonny” presents a challenge since she hung around “Calico Jack” +never captained a ship of her own:



* In this example, the table constraints make it impossible to insert a pirate’s information w/out specifying a ship
* **This dependency does not just encumber insertions, it affects *deletions* as well**

##### DELETE Anomaly

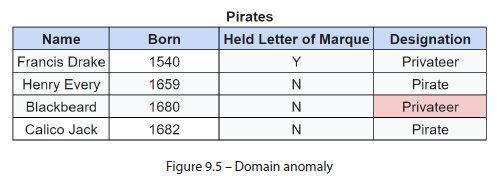
* A **deletion anomaly** occurs when **deleting data from the database results in the unintended loss of related information**
* Like an insertion anomaly, a deletion anomaly **results from too many dimensions being bunched into one table** (such as PIRATE ID and SHIP, from the previous example)
* “Calico Jack” never could hold on to a ship for very long, + the “Kingston”was repossessed (w/ cargo intact) less than 2 months after being taken
* However, as we can see below, deleting “Kingston” from our table *also means wiping away all mention of “Calico Jack”*:



* **One thing all these anomalies have in common is that they are self-evident**
* However, **some anomalies only reveal themselves *through business knowledge***

##### Domain Anomaly

* A **domain anomaly** occurs when a **database fails to incorporate business rules into its design, thereby making it possible to create values that violate them**
* This anomaly is **harder to catch b/c it requires functional knowledge of the data in question**
* The following example shows a list of PIRATE attributes:



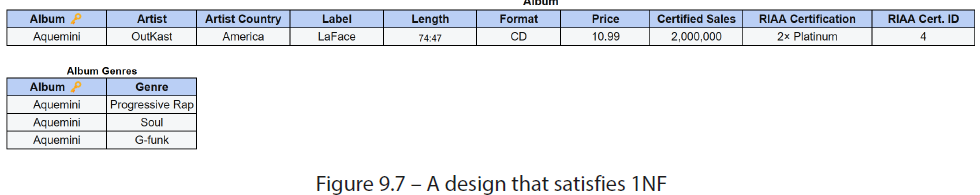
* Those unaware of the distinction between “pirate” and “privateer” (the latter performs piracy on behalf of a sovereign who authorizes the activity through a letter of marque) may miss the discrepancy in Blackbeard’s record
* **Normalization can prevent the above data anomalies by subdividing tables to reduce dependency and redundancy and by encoding business rules into the design**

#### Database Normalization Through Examples

* **Normalization** = first proposed by Edgar F. Codd, the inventor of the relational model for database management
* Introduced normalization in 1NF form and later extended it to 2NF + 3NF
* Later, Codd, working w/ Raymond F. Boyce, developed **Boyce-Codd NF (BCNF)**, or **3.5NF**
* As database theory continues to develop, subsequent normal forms have been proposed up to 6NF, following the progression from least to most restrictive
* While it is important to understand normalization in all its forms, it is also **essential to recognize that typical business scenarios do not require going past 3NF**
* To understand how normal forms organize a database + prevent data anomalies, we will run through an exercise of taking a wholly denormalized dataset using examples of data from the music industry + running it through the normalization rules required to satisfy each form
* But, unlike the wide world of music, the *examples assume that an album name is unique + that no two artists can have identically named albums*

##### 1NF

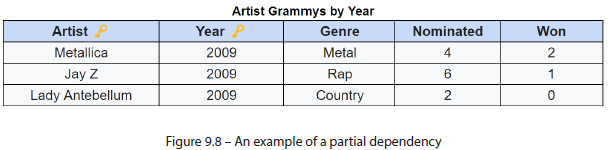
* **1NF is satisfied when the following are in place:**
* **Every record is unique**
* **Every *cell* contains a single value**
* **Ensuring unique records is one of the fundamental requirements of a relational database design**
* In our music example from before + working w/ the premise that ALBUM is a unique identifier, we begin by eliminating the duplicate 2nd row + assigning a PK constraint on the ALBUM column
* Next, we must **ensure that every *cell* contains single (atomic) values**
* **Storing multivalued data w/in a single cell presents a challenge for data analysis looking for meaningful insights**
* Ex: W/out atomic values, calculations like GENRE count or checking for existence of a specific GENRE value require parsing multi-valued contents of each cell before they can be analyzed
* **Pivoting these values into individual rows + creating a *separate dimension*** for GENRES satisfies 1NF
* The resulting design allows efficient analysis of GENRES w/out violating the ALBUM table’s PK or duplicating its attributes:



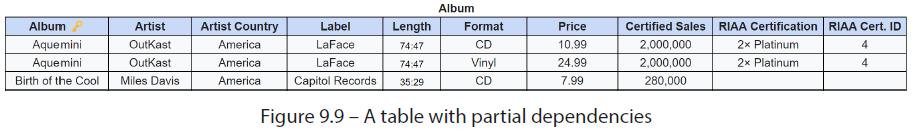
* Now that we have unique identifiers + no multi-valued data, let’s move on to the 2NF

##### 2NF

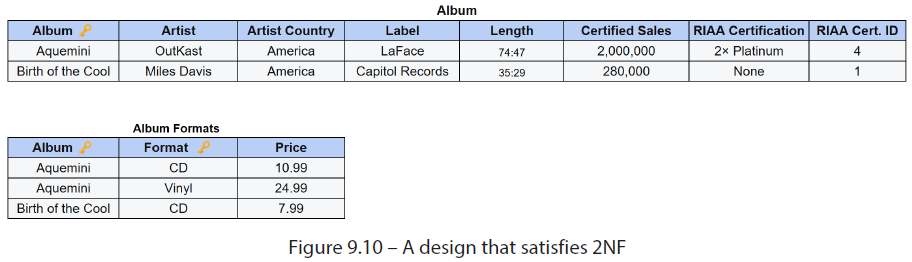
* **2NF** is satisfied when the following are in place:
* **1NF rules are satisfied**
* **Every *non*-candidate-key attribute must depend on the *whole* candidate key (i.e., there are no *partial dependencies*)**
* A **candidate key = a unique identifier that *cannot be further simplified and still remain unique***
* The following example shows an instance of a **partial dependency**:



* While ARTIST, YEAR, + GENRE together identify a unique record, *only ARTIST + YEAR are required (this is the candidate key)*
* Number of Grammys nominated + won depends *entirely* on the candidate key (ARTIST and YEAR) + would not make sense if all or part of the key were missing
* *However*, GENRE depends on ARTIST *only* (and *not* YEAR), forming a **partial dependency and thus violating 2NF**
* In our original example, the ALBUM and FORMAT columns form the **candidate key** in the table



* However, nearly *all* the attributes besides PRICE (such as ARTIST and LENGTH) depend *only* on ALBUM
* That is to say**, they *partially* depend on the candidate key and do *not* satisfy 2NF**
* The present design requires us to **duplicate all ALBUM attributes for every available FORMAT**
* In 2NF, we **can achieve a much cleaner design by decomposing the preceding table into 2 separate tables w/ no partial dependencies**
* Every attribute now depends entirely on the *complete* candidate key (Album’s ALBUM and then ALBUM + FORMAT, respectively):



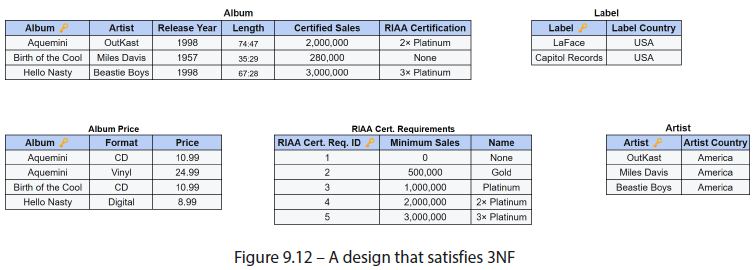
* Now that every attribute in our tables depends on a candidate key, let’s see what’s required to satisfy the 3NF

##### 3NF

* **3NF** is satisfied when the following are in place:
* **2NF rules are satisfied**
* **There are no transitive functional dependencies (TFDs)**, which are **formed when a *non*-PK column depends on another non-primary column**
* For example, LABEL COUNTRY, which depends on LABEL
* **TFDs are a problem b/c they allow update anomalies**, such as those in the following table, to occur:



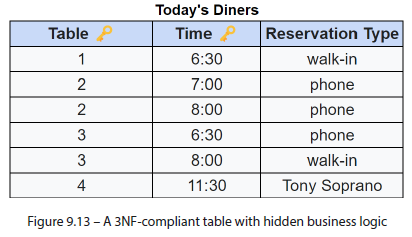
* To get around this issue, we **must resolve all TFDs by creating *separate dimension tables* for *each* of them**, where **all the attributes depend *entirely* on the *complete* PK**
* The resulting schema satisfies 3NF:



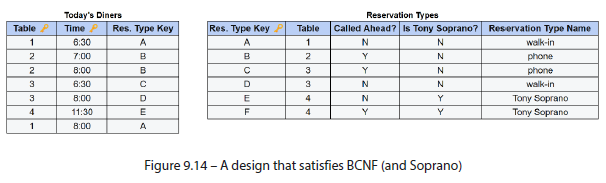
* **A mnemonic for remembering the first 3 normal forms:**
* In court, sworn testimony is often accompanied by a commitment, to **tell the truth, the whole truth, + nothing but the truth**
* In database design, **normalization rules stipulate that records should depend on the key (1NF), the whole key (2NF), + nothing but the key (3NF)**
* We have removed TFDs from our design, but before moving on to 4NF, let’s examine a slightly more restrictive version of 3NF

##### BCNF

* **BCNF** (also referred to as **3.5NF**) is satisfied when the following are in place:
* **3NF rules are satisfied**
* **Every non-trivial functional dependency is *itself* a key**
* In 1974, Raymond Boyce + Edgar Codd got together to invent a stricter version of 3NF to address additional potential anomalies, and they **aimed to reduce redundancy while having the structure reflect the underlying business rules**
* Before going into detail, it should be said that **it is rare to encounter scenarios that satisfy 3NF but *not* BCNF**
* To invent such a scenario, let us abandon music for a moment + look at restaurants:
* Take the following table of dining reservations from Nuovo Vesuvio, Artie Bucco’s upscale Italian eatery:



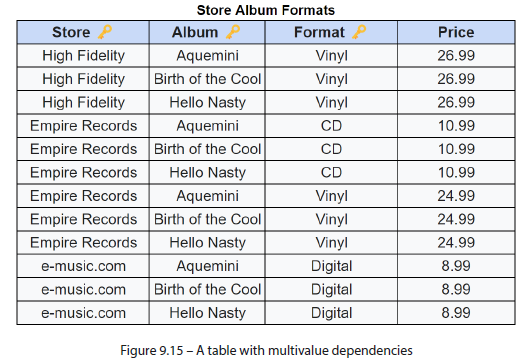
* Although the table *is* in 3NF + free from data anomalies *today*, the design fails to reflect or enforce some important business restrictions
* Namely, table 2is available only to those who’ve phoned to reserve in advance, table 1forwalk-insonly, +, most importantly, table 4is exclusively reserved for Artie’s friend, Tony Soprano
* To satisfy Boyce + Codd a RESERVATION TYPES *table* must be created to maintain the set of allowable possibilities
* The resulting layout would be as follows:



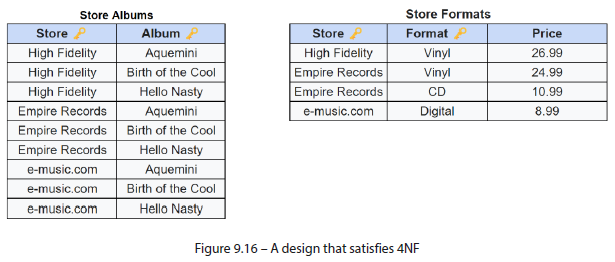
* Nuovo Vesuvio follows stringent rules about reservations that vary from table to table, *but what if every table accepted every known reservation method*?
* Could the existing design be *improved* to reduce redundancy?
* Ronald Fagin certainly thinks so, + it’s why he created 4NF

##### 4NF

* **4NF** is satisfied when the following are in place:
* **BCNF rules are satisfied**
* **Every non-trivial functional and multivalued functional dependency begins w/ a superkey**
* Recall that a **superkey is a combination of columns that uniquely identifies a row (but, *unlike a candidate key, it can also include extra columns*)**
* A **multivalued dependency** is **a relationship between 2+ attributes in a database, where the value of one attribute depends on the values of *multiple*, *other* attributes**
* To understand this in practice, observe this table of album inventory by format across 3 stores:



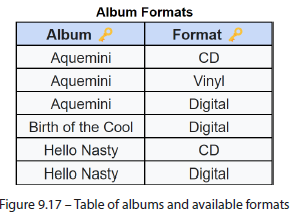
* High Fidelity and e-music.comonly sell one format (vinyl+ digital only, respectively), while Empire Recordssells all albums on CD *and* vinyl
* Storing everything in a single table means that ALBUM depends on FORMAT *and* STORE, which are *not* superkeys
* **Separating the information into 2 tables removes the multi-valued dependency and satisfies 4NF**



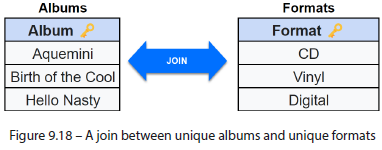
* However, this is ***ONLY true if the rules*** *in the STORE FORMATS table* ***hold***
* If *any* store could sell *any* format, the restaurant table from earlier, which satisfies BCNF, would *also* satisfy 4NF
* **As business knowledge is required to determine 4NF, *no systematic heuristics can guarantee it***
* The **same holds true for the 5NF**

##### 5NF

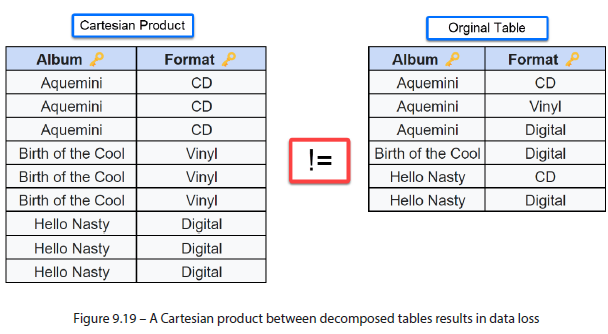
* **5NF** is satisfied when the following is in place:
* **4NF rules are satisfied**
* **Tables cannot be decomposed further *w/out loss of data***
* **After removing the functional + multi-valued dependencies, the only way to reduce further is by analyzing the data *as a function of existing business rules***
* Let us look at the following table, which shows albums in their available formats:



* **Decomposing this table *further* will yield tables (seen in the subsequent diagram) that *cannot* be joined to recreate the data in the table above**:



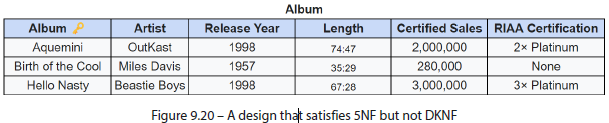
* **Attempting to join the smaller tables produces a Cartesian product with every combination of possible values, thereby losing the original detail**



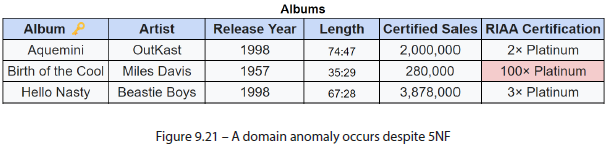
* **5NF is *not* commonly used in practice, as it is difficult to achieve + is unnecessary for most databases**
* **Rarely do tables that satisfy 5NF *not* satisfy 4NF (and, by extension, BCNF)**
* However, **even 5NF fails to guard against domain anomalies, which are common enough to consider avoiding**

##### DKNF

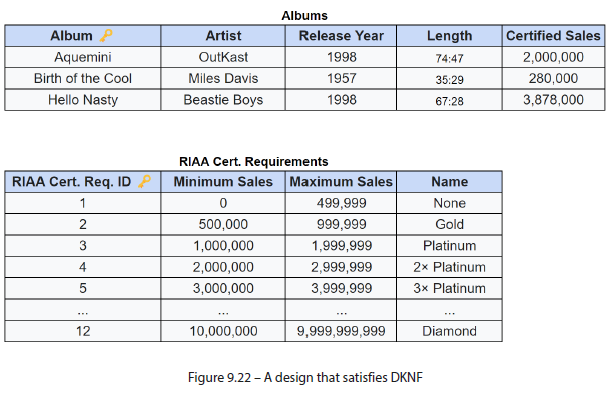
* The **domain-key normal form (DKNF)** is satisfied when the following are in place:
* **5NF rules are satisfied**
* **Domain constraints do not exist in any table**
* Various examples throughout this chapter have included an album’s RIAA CERTIFICATION and CERTIFIED SALES
* In the music industry, RIAA certifications are assigned based on the number of albums sold:



* CERTIFIED SALES and RIAA CERTIFICATION are both attributes that *depend* on the Album’s PK, but *there is an* ***implicit domain constraint*** *between them as* ***record sales dictate the RIAA status***
* As we can see in the following table, it is **subject to domain anomalies that may result in nonsensical data, which nevertheless conforms to 5NF**:



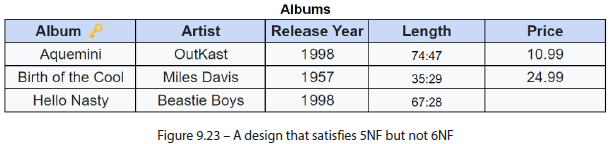
* To avoid the possibility of a domain anomaly, RIAA CERTIFICATION must be separated into *another* table, w/ clearly-defined bounds for each certification
* This way, the correct certification can be derived logically from an album’s CERTIFIED SALES column:



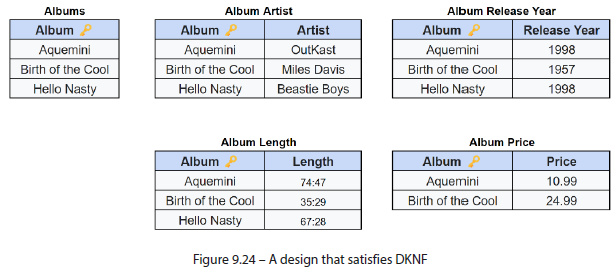
* **The final form we will cover is still under debate in the modeling community + is only practicable in rare cases**,however, it is not without merit

##### 6NF

* **6NF** is satisfied when the following are in place:
* **5NF rules are satisfied**
* **Every table contains a PK and, *at most*, *one* attribute**
* **6NF takes reducing dependencies to an extreme by making them a physical impossibility**
* **In 6NF, tables are allowed no more than one attribute per table**
* The following example shows a table in 5NF consisting of a PK - 4 attributes (Notice the last record has no value for PRICE):



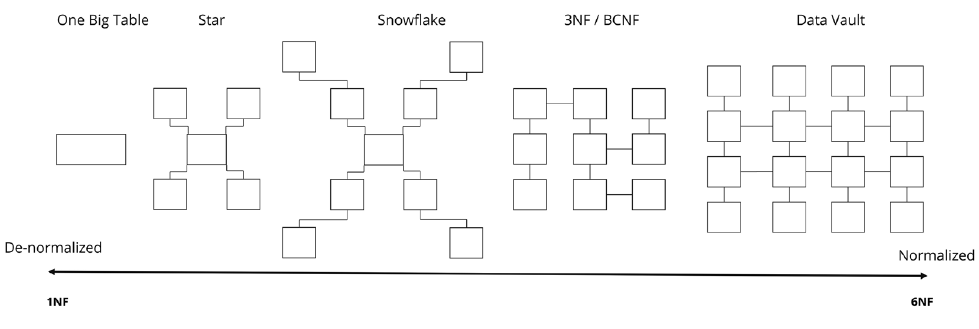
* To satisfy 6NF, this table must be split into *five* smaller tables
* Notice how the resulting tables conform to the original PK column (ALBUM), but the ALBUM PRICE table has fewer records due to “*Hello Nasty”* missing that attribute:



* **The proliferation of tables under 6NF (+ the effort required to query + maintain them) makes it *impractical* for most business purposes**
* 6NF has gained some notoriety through ensemble modeling frameworks such as Anchor Modeling,
* However, it is worth mentioning the database features that Anchor Modeling depends on, both of which are supported in Snowflake (+ covered in more detail in later chapters):
* **1) W/ the many tables that 6NF requires, joins become exceedingly difficult to write**
* However, **query performance can be significantly increased in 6NF through features such as join elimination** **compared to larger denormalized tables**
* With **join elimination**, the **database optimizer can intelligently avoid performing certain joins if attributes from the related tables are not *explicitly* requested in the query** (More on join elimination in Chapter 12)
* **2) Multi-table insert is another Snowflake feature w/ broad applications in many modeling scenarios, particularly in Data Vault** (Chapter 17) **and Anchor Modeling**
* In the figure above, we created 5 tables that share the same PK
* **The multi-table insert feature would allow us to load all 5 in a single load operation + even include conditional logic dictating which tables should be updated + when**

#### Data Models on a Spectrum of Normalization

* The prior sections demonstrate how, **as we move up the normal form scale, redundancy + the potential for data anomalies *decrease* while *raising* the number of tables + associated complexity**
* With this in mind, **we can visualize where various data modeling patterns fall on the normal form spectrum to help understand their suitability in design scenarios**
* While dimensional modeling was discussed in Chapter 7, the associated schema patterns (star and snowflake) are explained in further detail in Chapter 17, along w/ Data Vault (where they are often implemented at the information mart layer)
* However, **visualizing such schema patterns in the context of normal forms will help lay the groundwork for later understanding their design**
* *Also, bear in mind that the following assertions speak to general tendencies, not hard-set rules*
* The following image displays the various modeling patterns + methodologies mentioned throughout this book on the normalization spectrum:



* Arguably, **1NF (*the key*) is the *minimum* requirement for working w/ relational data**
* For its ease of understanding, **1NF is often seen in the** **One Big Table (OBT) approach**, where **facts + dimension attributes are all stored in a single table**
* While **difficult to maintain**, OBTs are **dead-simple to query** (i.e., no joins are required)
* Business users are comfortable w/ OBTs, provided they don’t get *too* big (by size or number of columns) to where usability begins to degrade
* **Star** and **snowflake schemas** **are patterns often seen when following Kimball’s DM approach + are good candidates for the data platform’s analytical self-service or reporting layers**
* In **both models**, the **central fact table is typically normalized to 3NF**
* In a **star schema,** **dimension tables are connected *directly* to the fact table + denormalized to 2NF** (a **good trade-off between maintenance + ease of use**)
* In a **snowflake schema, dimension tables are generally normalized to 3NF, forming a hierarchy + trading on analytical complexity in exchange for easier maintenance w/ less redundancy**
* At the far end of the normalization spectrum are **Data Vault models**
* **While Data Vault does not satisfy 6NF in the *strict* sense, it does incorporate many of its principles in its design**

#### Summary

* Through various examples in this chapter, we saw how **the process of normalization organizes a database in a way that reduces redundancy + dependency w/in its tables**
* **Dependency + redundancy in database tables increase the likelihood of data anomalies**, which come in many forms
* **Update anomalies** occur due to redundancy, which makes it possible to update some, but not all, of the associated records
* Physical (as opposed to logical) dependencies are the root cause of **insertion + deletion anomalies**
* When too many details of varying granularity are bunched into a single table, inserting or deleting records that do not match all the criteria becomes difficult
* The **domain anomaly** is the **hardest to spot because it requires functional knowledge of the data in question**
* Database normalization can be applied through **escalating stages of formal rules called normal forms, ranging from 1NF to 6NF** to avoid such anomalies
* The most commonly used normal forms are **1NF through 3NF, which are sufficient for most databases**
* The mnemonic, ***the key, the whole key, + nothing but the key***aids in recalling the considerations for each of the first 3 forms
* Through practical examples that took data from a completely denormalized format + transformed it through each normal form, a common theme emerged: **W/** **each normal form, the possibility for data anomalies decreased while the number of tables grew, encumbering maintenance + analysis**
* The examples in this chapter demonstrate that **more normalization does not imply a better design**, **it simply imposes stricter rules on the database**
* Having seen the specific data issues that each of the **normal forms** addresses, **applying their principles broadly or selectively will help modelers achieve an optimal design w/ the right balance of rigor + usability**