Partnering for Success

Data Quality and The Decision Model: Advice from Practitioners

By Steven Worsham and Kenneth von Halle

Previously, this column introduced the idea of using The Decision Model to specify and enforce Data Quality logic (i.e., DQ logic) [1] . That column covered DQ logic from the perspective of an organizational Data Quality Framework. The organizational DQ Framework depicted various dimensions of Data Quality and how a particular project decided to represent each dimension using The Decision Model (TDM).

Today, it is very common for organizations to use The Decision Model for managing DQ logic. The results are impressive and also deliver unique advantages over other approaches. In some cases, organizations represent DQ logic in The Decision Model as part of requirements deliverables. In other cases, organizations create DQ logic in TDM-compliant software which validates the logic against TDM principles, generates and executes test cases, and sometimes deploys to target technology.

The information in this column comes from two decision analysts: Steven Worsham with Sapiens International Corporation [2] and Ken von Halle with Knowledge Partners International LLC [3]. They have created various kinds of DQ decision models for different organizations and industries. They recommend dividing the DQ logic into four basic DQ categories. This month's column describes these DQ categories along with examples of how they fit into process models and decision models.

Background

How would you define a Data Quality Decision Model (i.e., DQ decision model)?

A *DQ decision model* contains the logic that makes sure the data required by a business decision model is of acceptable quality. This means, it contains logic to test the quality of input data. On the other hand, *a business decision model* contains logic that uses that data to arrive at a business decision.

For example, a *DQ decision model* determines if an address is, in fact, a **valid** address by testing its individual data fields (called fact types in The Decision Model) against postal requirements. Subsequently, a *business decision model* uses that **valid** address to determine if it is within an organization's marketing territory.

This separation means DQ decision models make sure the input data is valid **before** using that data in business decision models.

Do you need experience in data management or data architecture to create DQ decision models?

No, a decision analyst does not need knowledge of an organization's stored data structures to create decision models, even when dealing with data quality. Although DQ decisions must execute prior business decisions in a run-time environment, we typically model the DQ decisions **after** we model the business decisions, or at least after thorough analysis of the business process. The reason is that the business decisions drive the DQ requirements (e.g., what data do we need? Is any of the data conditionally required? What should the data look like?). The good news is that the physical nature of the data has no bearing on the data quality requirements when working with decision models.

Do you still need different skills to create DQ decision models than those skills needed to create business logic decision models?

You need the same skills to create DQ decision models as to create business decision models. These skills are a disciplined analytical approach to problem solving and attention to detail. Particularly, it is the attention to detail that distinguishes excellence in decision modelers.

Why use The Decision Model for DQ logic?

In your experience, why do companies choose to use The Decision Model for DQ logic in the first place?

There are at least three reasons. First, The Decision Model provides business control over the logic in a form understandable to non-technical audiences, ensuring that the logic is written accurately and is clearly understood throughout the enterprise.

Second, The Decision Model can be deployed into any (and multiple) technologies for execution. This provides the ability to maintain a single source of record for DQ logic and ensures that the same logic is applied in the same way within the various execution technologies.

Third, creating, maintaining, and governing a decision model repository creates appropriate transparency throughout an organization. A central decision model repository provides the ability to communicate DQ decision models across functional areas.

Why not include DQ logic within the business decision model that uses that input data?

First, executing DQ logic in its own decision model prior to executing a decision model that uses that data, avoids executing large decision models only to have them break due to low quality or missing data. The separation guarantees that the business logic is behaving as expected and that false conclusions are not reached due to poor data.

Second, if DQ logic is maintained independently, it becomes reusable across multiple processes.

Third, when stored as a separate decision model, the management and governance over DQ logic is substantially simpler, allowing both business and technical users to clearly comprehend the logic.

Experience has uncovered yet a fourth and powerful reason to create separate DQ models - more sophisticated options become available. For example, a DQ decision model can provide a complete error count for all data quality infractions along with corresponding error messages. This is quite powerful and is explored in greater detail below.

How do DQ decision models provide business control over the logic?

Most organizations perform DQ directly against actual data sources. Therefore, the DQ "code" executes against data as that data physically exists in electronic form. This leads to a dependence on technical professionals to translate DQ requirements from business people into technical DQ code.

In contrast (and by design), decision models represent the business person's DQ requirements in a form they understand. That's because, in The Decision Model, the input data is defined by business people as business-friendly fact types. Business-friendly fact types represent the data in the way business people think of it and want to see it. This means that there is not necessarily a one-to-one correspondence between input source data and the fact types referenced in a decision model. This is a very unique and important characteristic of The Decision Model. Let's explore it more.

Fact types in decision models have business-friendly names chosen by business Subject Matter Experts (SMEs), rather than names from a database or file. Fact types also have business-friendly domain values also chosen by business SMEs, not necessarily the values actually found in the data source. It is best to illustrate with an example.

Consider that a decision model needs the birth date of a driver because it uses the birth date of a driver as a condition to come to a decision about that driver. To a business SME, this is simply Driver Birth date. However, in a database, this birth date may not be stored as Driver Birth date. It may be stored as a Person Birth date (in a Person table) where that person plays the role of driver (a data relationship) on a Driver License (in a Driver License table). Presenting this data to a SME as a fact type called Driver Birth date involves joining those two tables. However, a business SME really has no need to know that it involves multiple tables. Therefore, a corresponding DQ decision model can represent this as one fact type, Driver Birth date, along with DQ logic to test the quality of that birth date. [4]

Unlike traditional DQ code, decision model DQ logic also applies to business friendly interim fact types created from a combination of persisted data inputs as opposed to the individual inputs themselves. For example, a fact type of Driver Age has its own domain restrictions (e.g., greater than 21 in some places if the driver is supervising a new driver), but usually only birth date is stored. Using The Decision Model, the business user can define an interim fact type for Driver Age and represent the DQ restrictions in the business-friendly representation.

So business SMEs define fact types for decision models, regardless of their actual storage, along with logic to be sure it is of high quality. This is very unlike other approaches to DQ.

Creating DQ Decision Models

What do DQ decision models look like and how are they different from business decision models?

DQ decision models can be constructed in many fashions, but their structure is often much larger than business decision models in terms of the quantity of Rule Families. This is because DQ logic typically requires a separate Rule Family for each fact type in an entire input data set and addresses all DQ categories. Yet, the individual DQ Rule Family tables tend to be smaller and less complex, usually requiring only one or two conditions. As with many decision models there are usually a variety of acceptable structures that the DQ decision models can take. Four DQ categories play a role in determining the acceptable structures.

Can you describe the four DQ categories?

The four primary DQ categories test for:

(1) DQ Complete: this logic tests for populated versus unpopulated fact types, including conditionally required Fact Types

(2) DQ Domain Validity: this logic determines whether fact type values are within the allowed values

(3) DQ Consistency: this logic tests for required relationships among fact types

(4) DQ Value Validity: this logic determines whether fact type values are reasonable and accurate.

Can you describe the best way to model the four DQ categories?

There are two important considerations: how many DQ decision models and what exactly do those DQ decision models do.

How Many DQ Decision Models?

At one extreme, a single DQ decision model can test all four DQ categories for all input fact types. Such a decision model tends to consist of many Rule Families and is generally not recommended for a variety of reasons [5]. This one DQ decision model executes completely within one process task. [6]

On the other hand, there can be distinct DQ decision models for each DQ category [7]. In this case, these decision models execute in their own process task and therefore execute in the sequence enforced in the process flow.

Let's look at an example.

Assume you are working with a process carried out by a police officer when he or she stops a car due to suspicious or careless driving. One business decision the officer makes is to determine if the driver is compliant with respect to the use of interactive electronic devices. Let's call this business decision Driver Device Compliance and its logic tests whether the driver is using a cell phone, video game, or other hands-free device. However, before executing that business decision model, a DQ decision model must test the quality of the input data, specifically whether all required data is populated and whether its values are within the allowable domain values [8].

Let's explore the creation of two DQ decision models. The first one, called Driver Device Compliance DQ Complete, tests whether or not required fact types are populated. The second one, called Driver Device Compliance DQ Domain Validity, tests for domain values. Figure 1 illustrates the process model for this option.

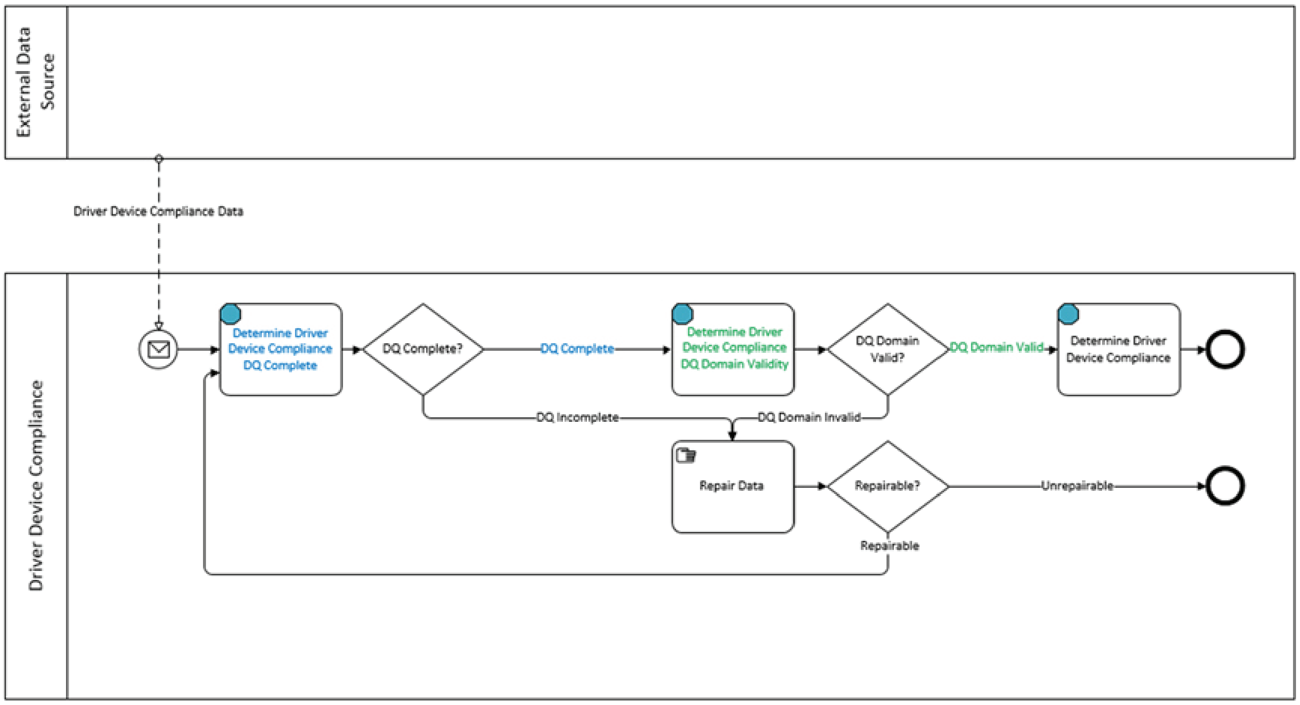


Figure 1: Process Model with Two DQ Decision Models

The first process task in Figure 1 contains the DQ decision model only for checking whether required (or conditionally required) fact types are populated. Figure 2 contains this Decision Model Diagram (Decision View) while Figure 3 contains the corresponding Rule Family.

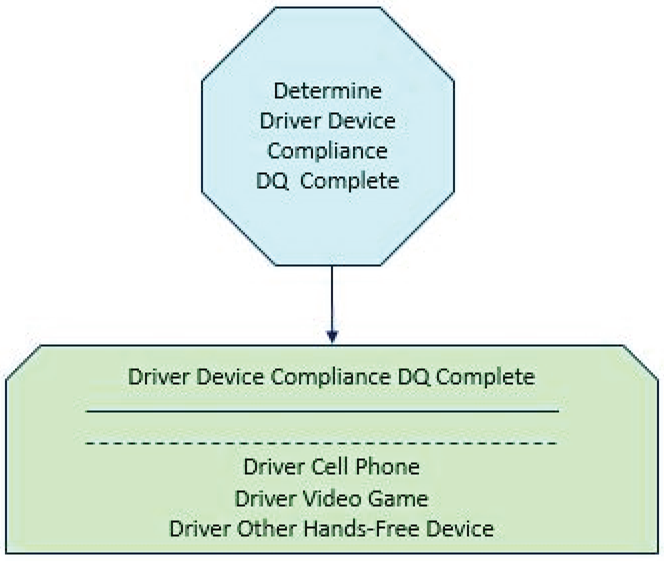
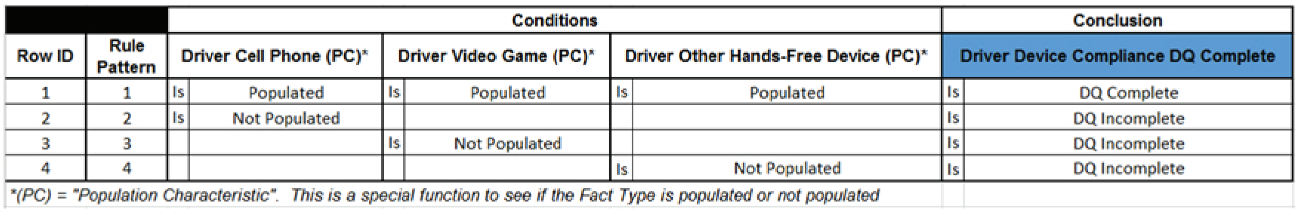

Determine Driver Device Compliance DQ Complete
Driver Device Compliance DQ Complete
Driver Cell Phone
Driver Video Game
Driver Other Hands-Free Device

Figure 2: Decision Model Diagram (Decision View) for Driver Device Compliance DQ Complete



		Conditions						Conclusion	
Row ID	Rule Pattern	Driver Cell Phone (PC)*		Driver Video Game (PC)*		Driver Other Hands-Free Device (PC)*		Driver Device Compliance DQ Complete	
1	1	Is	Populated	Is	Populated	Is	Populated	Is	DQ Complete
2	2	Is	Not Populated					Is	DQ Incomplete
3	3			Is	Not Populated			Is	DQ Incomplete
4	4					Is	Not Populated	Is	DQ Incomplete

*****(PC) = "Population Characteristic". This is a special function to see if the Fact Type is populated or not populated*

Figure 3: Rule Family for Driver Device Compliance DQ Complete

If the first DQ decision model is successful, the second process task in Figure 1 proceeds to the DQ decision model for checking fact type values. The Decision Model for this is in Figure 4 while its Rule Family is in Figure 5.

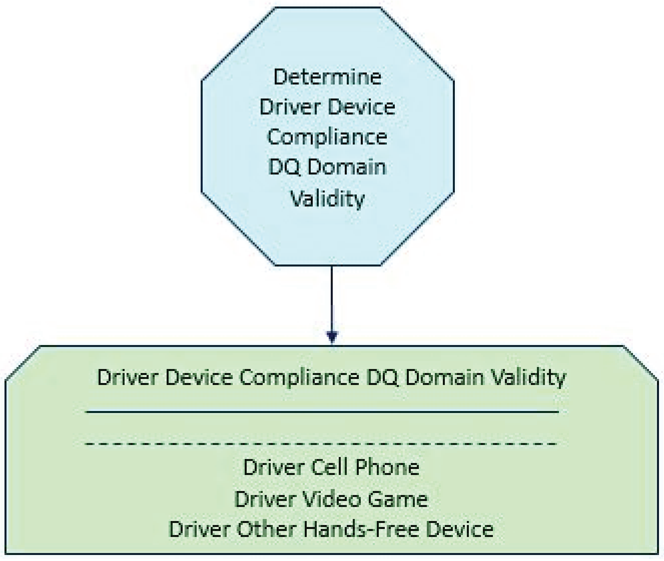
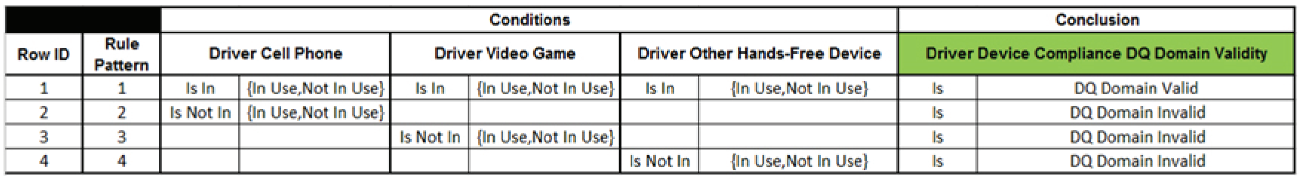

Determine Driver Device Compliance DQ Domain Validity
Driver Device Compliance DQ Domain Validity
Driver Cell Phone
Driver Video Game
Driver Other Hands-Free Device

Figure 4: Decision Model Diagram (Decision View) for Driver Device Compliance DQ Domain Validity



		Conditions						Conclusion	
Row ID	Rule Pattern	Driver Cell Phone		Driver Video Game		Driver Other Hands-Free Device		Driver Device Compliance DQ Domain Validity	
1	1	Is In	{In Use,Not In Use}	Is In	{In Use,Not In Use}	Is In	{In Use,Not In Use}	Is	DQ Domain Valid
2	2	Is Not In	{In Use,Not In Use}					Is	DQ Domain Invalid
3	3			Is Not In	{In Use,Not In Use}			Is	DQ Domain Invalid
4	4					Is Not In	{In Use,Not In Use}	Is	DQ Domain Invalid

Figure 5: Rule Family for Driver Device Compliance DQ Domain Validity

The sequence between these DQ decision models is enforced by the process flow.

Can you clarify when to create separate decision models for different DQ categories and when to bundle them all into one DQ decision model?

Determining how many DQ decision models depends on the size and complexity of the logic. If there aren't many fact types (e.g., fewer than ten data elements) and there aren't complex relationships among them, one DQ decision model may suffice, but is rare. Most often, separate decision models for different DQ categories works best.

What Exactly Should DQ Decision Models Do?

In general, DQ decision models can be re-active or pro-active. Re-active DQ decision models simply report on the DQ errors while pro-active DQ decision models are more sophisticated and actually work around the errors.

Re-Active DQ Decision Models

Re-active decision models function like a "glass screen." They are passive in their evaluation of each fact type against DQ logic by simply detecting and reporting data quality errors.

Re-active DQ decision models are useful in processes that require absolute assurance of data quality due to their ability to catch all data quality errors along with specifications about the errors (e.g., unpopulated values, invalid fact type values, etc.). This allows the process to stop at the occurrence of DQ errors and attempt to remedy the data prior to executing business decision logic.

Pro-Active DQ Decision Models

Pro-active DQ decision models correct or enhance input fact type values that do not meet data quality standards. These decision models replace missing or incorrect data values with reasonable, assumed values. To accomplish this, assumed values are set as the conclusion to a representative interim fact type when the true data value does not meet data quality standards. The representative interim fact type can then be used in a subsequent business decision model, allowing the process to continue despite the lapse in absolute data quality. It is recommended that any decision models that use the assumed values include specific messages detailing the assumptions and stating that the resulting business conclusion is based on those assumptions.

An example is a DQ decision model executing against a student loan application. For example, it can supply missing data for income. It will base its conclusion on the supplied data and conclude that the student will be eligible (if all other input data leads to an eligible conclusion) if, when the true income is provided, it conforms to the DQ-added value.

Mixing Pro-Active and Re-Active Decision Models

It is interesting to note that a single DQ decision model may be a combination of Pro-Active and Re-Active data quality checks. In these instances, some fact types are "show stoppers" because the business decision requires highest quality data. In a combination DQ model, the business decision may be flexible enough to allow for supplied values. A decision model of this nature results in a negative conclusion if any of the "show stopper" fact types do not pass data quality. It results in a positive conclusion if all "show stopper" fact types pass all DQ tests and flexible fact types may be reset to supplied values. A negative conclusion from this decision would most likely prevent the business process from continuing with the business decisions.

Some people would be surprised to learn that DQ for valid values for fields is handled in decision models. Isn't this usually handled by the DBMS? Why use decision models?

It depends on the business's needs. Most of the time, the business side of an organization wants to be able to specify the DQ validation logic. For example, business people often want the ability to change, as needed, the allowable values for a fact type. Using decision models not only provides a business-friendly translation of the DQ validation logic, but also allows non-technical people to build and manage the model to their requirements, providing the business the ability to manipulate the logic as they see fit.

Wrap Up

What was the biggest surprise in creating DQ decision models?

The biggest surprise was how complex the DQ decision models can become. When modeling a DQ decision model for three fact types with relatively simple relationships, the DQ decision model quickly turned into eleven Rule Families. This proved ***how The Decision Model is capable of exposing the complexities hidden within DQ logic, but does so in an understandable, non-technical way.***

What were the most sophisticated (or interesting) DQ decision models?

The most sophisticated example was the one mentioned above about a student loan application. [9] It provides assumed values for unpopulated data, carries out the decision, and then informs the user which data values needed to be populated, along with a warning that the use of actual data values may result in a different conclusion. T***his is a lot of functionality to execute in a technology-independent, truly declarative representation that can deploy to any target environment.***

Do you have any words of advice to people just starting out doing DQ decision models?

Consider the following approach to start with:

- Break up the DQ logic into 3-4 different, process-connected high-level decisions based on the DQ categories discussed in this article.
- Begin with the DQ Complete decision model. This decision model should check that every fact type that needs to be populated is populated. It may include population logic for conditionally required fact types.
- Next focus on the DQ Domain Validity decision model. This decision model validates that each populated value is an acceptable value.
- Then, create the DQ Consistency decision model to validate the relationships among the fact types (e.g., if STATE is NJ, then ZIP has to be a valid ZIP for NJ).
- If any additional data quality considerations remain, consider creating one or more DQ Value Validity decision models to address logic for accuracy, reasonableness, and other DQ dimensions.

What is the most valuable but subtle advantage to using decision models for both business logic and DQ logic?

The benefit of having these separate DQ decision models is that, by the time the business logic decision model executes, it has all the data it needs in the highest quality it needs. The DQ decision models ensure that every required fact type value coming in is populated, its value is acceptable, its relationship to other fact types are valid, and other DQ logic is correct. This separation makes it easier to manage the DQ logic separate from the business logic.

In addition, organizations managing a business decision repository (or BDMS for Business Decision Management System) are able to deliver a central repository used by both business people and IT for managing all logic models across an enterprise, from business logic to DQ logic.

1. See Better, Faster, Cheaper Part II - The Decision Model Meets Data Quality Head On

2. For more information on Sapiens International Corporation, specifically DECISION software, see http://www.sapiensdecision.com/about-sapiens-decision.html.

3. For more information on Knowledge Partners International LLC, see www.kpiusa.com

4. In reality, behind the scenes and not visible to the business SME, the Driver Birthdate fact type is materialized as the product of a table join and presented in its business-friendly form. The specification for this join, however, can happen later; it need not slow down the creation, validation, and testing of decision models. More importantly, it need not be visible to the business SME.

5. The option of one DQ decision model for all DQ categories and fact types is usually unmanageable when input data sets are large.

6. This decision model can execute in any sequence because decision models are declarative meaning that sequence is irrelevant to arriving at the conclusion.

7. There is the option of dividing a single DQ category decision into multiple decision models due to the size of the data set. Typically, best practice is to base the division on business concept (entity) or other characteristic. For example, DQ logic for DQ complete may divide naturally into Borrower Information DQ Complete, Loan Information DQ Complete, Property Information DQ complete, etc.

8. In this simple example, there are no relationships among fact types to test.

9. And others that are similar to the student loan application are equally sophisticated

Steven Worsham, a Senior Business Analyst at Sapiens, uses The Decision Model in a variety of different project environments within the mortgage and financial industries. His experience leading end-to-end decision modeling projects incorporates both data Quality and Business Decisions within the Sapiens DECISION software application. Steven graduated from the University of North Carolina in Chapel Hill with a Bachelor's Degree in Business Administration and is a certified Decision Model Practitioner. He currently resides in Raleigh, North Carolina

Ken von Halle is a Decision Analyst with Knowledge Partners International (KPI), helping clients to become proficient in decision modeling. He has experience in creating decision models for various purposes such as: banking, mortgage eligibility, student loan eligibility, wealth management, data quality, and data migration. Ken currently resides in Mendham, NJ.

Contact us

For more information please visit us:

www.sapiensdecision.com

info@sapiensdecision.com

+ 1-800-858-9473 (US)

+ 44 1895 464 000 (UK)

Published in: www.modernanalyst.com, JJanuary 26, 2014