Wie modelliere ich mein Core Data Warehouse?

Dani Schnider
Trivadis AG
Zürich/Glattbrugg, Schweiz

Schlüsselworte:

Data Warehouse, Datenmodellierung, Historisierung

Einleitung

Das Core dient im Data Warehouse zur Integration und Historisierung von Quelldaten und als Datenbasis für die Data Marts. Während Data Marts fast immer dimensional modelliert und als Star Schema oder multidimensionale Cubes implementiert werden, gibt es für das Core teilweise sehr unterschiedliche Ansätze für die Datenmodellierung. Soll das Core dimensional oder relational modelliert werden? Gibt es Mischformen? Wie weit sollen die Daten normalisiert oder denormalisiert werden? Was ist von Ansätzen wie Data Vault Modeling oder generischen Datenmodellen zu halten? Nachfolgend werden verschiedene Modellierungsansätze und ihre Anwendungsgebiete in der Praxis vorgestellt.

Wie das Core-Datenmodell in einem Data Warehouse aussieht, hängt weitgehend davon ab, was für eine Vorgehensweise für die Erstellung des Datenmodells gewählt wird. Wird das Modell hauptsächlich basierend auf den fachlichen Anforderungen entworfen (Anforderungsgetriebene Modellierung), sieht das Core-Datenmodell typischerweise anders aus als wenn es aus den Datenmodellen der Quellsysteme abgeleitet wird (Quellsystemgetriebene Modellierung).

Anforderungsgetriebene Datenmodellierung

Ein bewährter und empfohlener Ansatz ist die Modellierung basierend auf den konkreten fachlichen Anforderungen, welche Informationen in den BI-Applikationen benötigt werden. Dabei werden in der Regel zuerst die Dimensionen und Fakten der Data Marts festgelegt. Daraus kann dann das Core-Datenmodell abgeleitet werden. Das Resultat ist ein Core mit dimensionalem Modell.

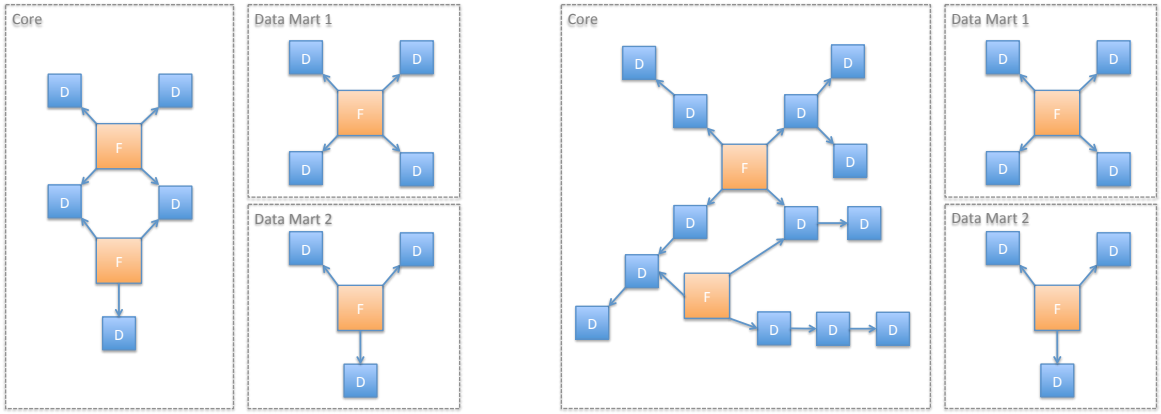

Abbildung 1: Dimensionales Core als Star Schema (links) oder Snowflake Schema (rechts)

Im einfachsten Fall entsteht dabei ein Core-Datenmodell als Star Schema (vgl. Abbildung 1 links), das alle Dimensions- und Faktentabellen der einzelnen Data Marts enthält. Die Data Marts sind dabei nur noch Teilsichten auf die Daten, die im Core zur Verfügung stehen. Teilweise werden die Fakten in den Data Marts auf höhere Hierarchiestufen der Dimensionen aggregiert. Ob bei einem solchen Core-Datenmodell die Data Marts als persistente Schicht mit separaten Dimensions- und Faktentabellen implementiert werden, oder nur als View-Schicht auf das Core-Modell, ist vom Datenumfang und von den Performanceanforderungen an die Data Marts abhängig.

Wenn die Fakten in verschiedenen Data Marts auf unterschiedliche Hierarchiestufen der Dimensionen verweisen, oder wenn die Hierarchiestufen aus verschiedenen Datenquellen geladen werden (z.B. Adressen aus der Kundendatenbank, Bundesländer und Staaten aus einem allgemeinen Master-Data-Management-System), ist es zweckmäßiger, das Core-Modell als Snowflake-Schema zu implementieren (vgl. Abbildung 1 rechts). Dabei wird für jede Hierarchiestufe einer Dimension eine separate Tabelle erstellt. Ob dieses Modell nun noch als „dimensional" bezeichnet werden kann, ist eine Frage für Modellierungs-Puritaner und für die Anwendung in der Praxis nicht relevant. Wichtig ist hingegen, dass die Data Marts weiterhin als Star Schema mit denormalisierten Dimensionstabellen implementiert werden.

Ein wichtiger Aspekt ist die Art der Historisierung der Dimensionsdaten, also welcher Typ von Slowly Changing Dimensions (SCD) verwendet wird. In den Data Marts werden Änderungen an Dimensionsdaten je nach fachlichen Anforderungen überschrieben (SCD Typ 1) oder versioniert (SCD Typ 2). Im Core hingegen ist es empfehlenswert, alle Änderungen zu versionieren. Da es jederzeit möglich ist, aus einer versionierten Dimension den aktuellen Stand zu ermitteln, erlaubt dies eine hohe Flexibilität. Aus einem Core-Datenmodell, das generell SCD Typ 2 verwendet, können Data Marts mit unterschiedlichen Historisierungsanforderungen abgeleitet werden.

In einem Star Schema ist die Versionierung mittels SCD Typ 2 einfach zu implementieren: Wenn sich gegenüber der bisher aktuellen Version eines Dimensionseintrags mindestens ein Attribut ändert, wird eine neue Version erstellt und das Enddatum der letzten Version aktualisiert. In einem Snowflake Schema ist die Situation etwas komplexer. Ändert sich ein Attribut einer übergeordneten Hierarchiestufe, wird in der entsprechenden Tabelle eine neue Version erstellt. Dies hat aber zur Folge, dass auch Fremdschlüsselattribute der darunterliegenden Tabelle angepasst werden müssen. Somit müssen auch in dieser Tabelle – und in allen weiteren Hierarchiestufen –neue Versionen erzeugt werden. Die ETL-Prozesse, die dafür notwendig sind, werden relativ komplex.

Dieses Problem kann vermieden werden, indem die Dimensionstabellen für die einzelnen Hierarchiestufen aufgeteilt werden in einen statischen und einen dynamischen Teil. Für jede Hierarchiestufe wird eine Headtabelle erstellt, welche den Primary Key (künstlicher Schlüssel), den Business Key (fachlicher Schlüssel) sowie optional die statischen Attribute, die sich nie ändern, enthält. Alle dynamischen Attribute, die sich im Laufe der Zeit ändern können, werden in eine Versionstabelle ausgelagert. Diese Tabelle enthält zusätzlich einen Fremdschlüssel auf die Headtabelle sowie ein Gültigkeitsintervall der jeweiligen Version.[1]

Fremdschlüssel auf übergeordnete Hierarchiestufen können entweder in der Head- oder in der Versionstabelle stehen – je nachdem, ob sich die Beziehung im Laufe der Zeit ändern kann. Entscheidend ist aber, dass die Referenz immer auf die Headtabelle der übergeordneten Hierarchiestufe zeigt. Wenn dort nun Änderungen erfolgen, d.h. zusätzliche Versionen in der

[1] Details siehe Buch „Data Warehousing mit Oracle – Business Intelligence in der Praxis", Hanser-Verlag, ISBN 978-3-446-42562-0

Versionstabelle erstellt werden, hat dies keine Auswirkungen auf die Versionierung der untergeordneten Stufen. Die Versionierung der einzelnen Hierarchiestufen ist somit unabhängig.

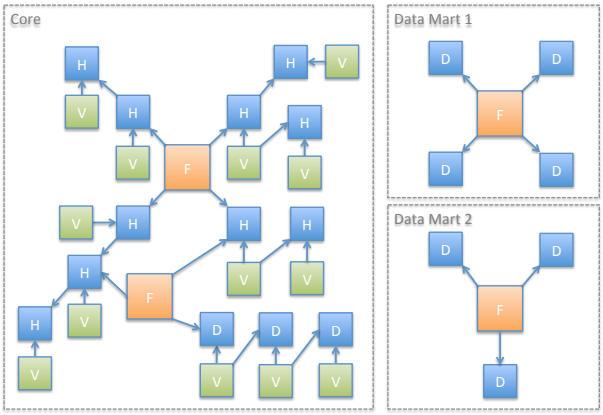

Abbildung 2: Dimensionales Core mit Head- und Versionstabellen

Dieses Prinzip der Versionierung von Dimensionsdaten hat sich in zahlreichen DWH-Projekten bewährt und erlaubt den Aufbau eines flexiblen Core-Datenmodells, aus welchem sowohl Data Marts mit Dimensionstabellen für SCD Typ 1 als auch SCD Typ 2 geladen werden können. Als Vereinfachung für die ETL-Prozesse zum Laden der Data Marts empfiehlt sich die Erstellung bzw. Generierung eines View Layers, der die zusammengehörenden Head- und Versionstabellen in einer View zur Verfügung stellt und auf die relevanten Versionen einschränkt.

Quellsystemgetriebene Datenmodellierung

Eine häufig verwendete Vorgehensweise besteht darin, das Core-Datenmodell aus den Datenmodellen der Quellsysteme abzuleiten. Dieser Ansatz ist vor allem dann von Nutzen, wenn zum Zeitpunkt der Core-Modellierung die konkreten Anforderungen an die Data Marts noch nicht bekannt sind, oder wenn die Quellinformationen für unterschiedliche Anwendungen verwendet werden. Sofern nur ein Quellsystem als Basis für das Core verwendet wird, stellt sich hierbei die berechtigte Frage, ob überhaupt ein Core notwendig ist, da dieses im Extremfall nur noch eine Kopie des operativen Systems sein wird.

In den meisten Fällen wird sich jedoch auch hier der Aufbau eines Core-Datenmodells lohnen, denn früher oder später werden weitere Quellsysteme ins Data Warehouse integriert werden. Außerdem sollen die Daten im Core historisiert abgelegt werden, damit die Data Marts je nach Anforderungen mit Dimensionsdaten von SCD Typ 1 oder Typ 2 versorgt werden können. Das Core erfüllt somit die wichtige Funktion einer versionierten Ablage von Stammdaten aus den Quellsystemen.

Für diesen Zweck kann der bereits beschriebene Ansatz von Head- und Versionstabellen verwendet werden. In diesem Fall wird pro Quelltabelle, die ins Data Warehouse geladen wird, im Core eine Headtabelle mit statischen und eine Versionstabelle mit dynamischen Daten erstellt. Auf diese Weise kann eine lückenlose Historie der relevanten Quelldaten aufgebaut werden, aus denen für jeden Data Mart entweder die aktuelle Sicht oder ein beliebiger Stand in der Vergangenheit extrahiert werden kann.[2]

[2] Siehe auch DOAG-Vortrag „Die generierte Zeitmaschine – Historisierung auf Knopfdruck“ (2010) http://www.trivadis.com/uploads/tx_cabagdownloadarea/DOAG_2010_Historisierung_auf_Knopfdruck.pdf

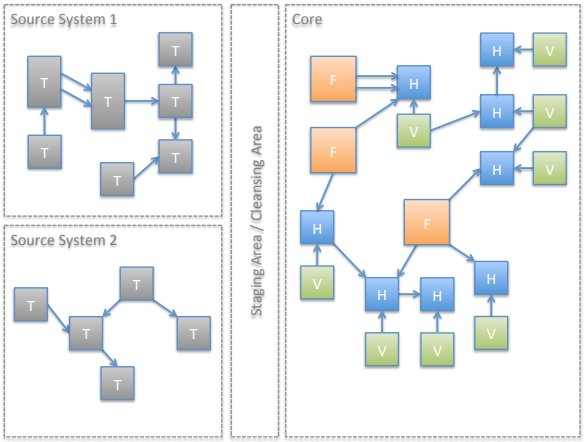

Abbildung 3: Core-Modell zur Historisierung der Quelldaten mit Head- und Versionstabellen

Für Transaktionsdaten, die üblicherweise den Fakten in den dimensionalen Data Marts entsprechen, ist eine Versionierung nicht notwendig, wie in Abbildung 3 angedeutet. Fakten sind normalerweise an einen Ereigniszeitpunkt gebunden und ändern sich nachträglich nicht mehr.

Da sich nicht nur die Daten der Quellsysteme, sondern auch die Tabellenstrukturen im Laufe der Zeit ändern können, stellt sich öfters die Frage, wie solche Strukturänderungen im Core nachgezogen werden. Dies ist vor allem auch dann der Fall, wenn ein zusätzliches Quellsystem ins Core integriert werden soll. Bei jeder Erweiterung und Änderung des Core-Datenmodells müssen unter Umständen historische Daten angepasst werden. Dies kann einen erheblichen Aufwand für Umbau und Datenmigration im Core zur Folge haben.

Hier kommt ein weiterer Modellierungsansatz ins Spiel, der genau diese Probleme entschärfen soll. Das Prinzip von „Data Vault Modeling“ erlaubt den Aufbau eines leicht zu erweiternden Core-Datenmodells und wird vor allem in agilen DWH-Projekten mit vielen Strukturänderungen eingesetzt. Der Vorteil dieser Modellierungsmethode liegt vor allem darin, dass die bestehenden Tabellen nicht angepasst werden müssen, sondern dass nur neue Tabellen hinzugefügt werden.

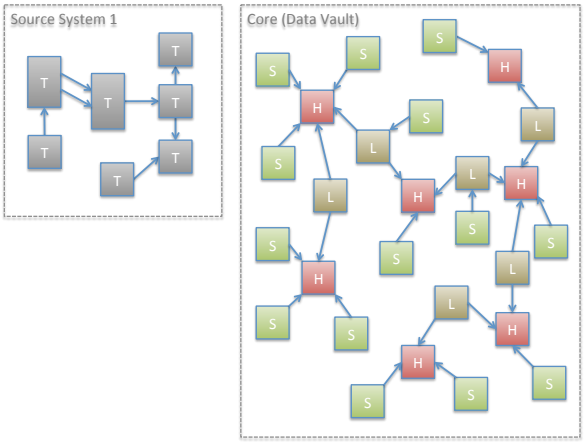

Abbildung 4: Core-Modell mittels Data Vault Modeling

Ein Data Vault Modell besteht aus drei Arten von Entitäten:

- **Hubs** enthalten die Business Keys der Stammdaten-Entitäten aus den Quellsystemen, aber keine beschreibenden Attribute.
- **Satellites** enthalten die beschreibenden Attribute der Entitäten und werden Hubs zugeordnet. Ein Hub kann mehrere Satellites enthalten, die beispielsweise zu unterschiedlichen Zeitpunkten geladen oder auf verschiedene Arten historisiert werden.
- **Links** dienen zur Abbildung von Beziehungen zwischen Hubs und enthalten somit nur Fremdschlüsselattribute auf zwei oder mehr Hubs. Einem Link können ebenfalls Satellites zugeordnet werden.

Wird nun eine bestehende Quelltabelle um zusätzliche Attribute ergänzt, wird einfach dem bereits existierenden Hub ein zusätzlicher Satellite zugewiesen, ohne die bestehenden Satellites zu verändern. Die Unterscheidung zwischen statischen und dynamischen Attributen, wie sie zuvor im Ansatz mit den Head- und Versionstabellen beschrieben wurde, wird in Data Vault mittels separaten Satellites modelliert. Ein Data Vault Modell wird somit im Laufe der Zeit immer wachsen und umfasst eine relativ große Anzahl von Satellites pro Hub.

Ein solches Datenmodell ist natürlich nicht für Abfragen geeignet, da oft Joins über sehr viele Tabellen notwendig sind. Ein Data Vault Modell ist aber auch nicht für diesen Zweck konzipiert, sondern wird ausschließlich für die Modellierung des Cores – bzw. „Data Vaults“ verwendet. Aus dem Data Vault werden dann die Data Marts geladen, die wie üblich ein dimensionales und für Abfragen optimiertes Datenmodell besitzen.

Generische Datenmodelle

Eine typische Eigenschaft der bisher vorgestellten Modellierungsansätze ist die Tatsache, dass Erweiterungen aufgrund von neuen Anforderungen oder Änderungen der Quellsysteme zu Anpassungen des Core-Datenmodells führen. Dies wird teilweise als Nachteil interpretiert, da Datenmodelländerungen aufwendig sind und weitere Anpassungen zur Folge haben (ETL-Prozesse, Datenmigrationen, etc.). Um Erweiterungen des Datenmodells so minimal wie möglich zu halten, kann stattdessen ein möglichst allgemeines generisches Modell verwendet werden, das „die ganze Welt“ abbilden kann. Solche Modelle bestehen typischerweise aus allgemeinen Entitäten (z.B. Partner, Objekt, Vertrag) sowie rollenbasierten Beziehungen zwischen diese Entitäten.

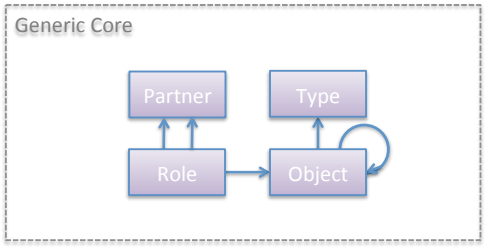

Abbildung 5: Beispiel für ein generisches Core-Datenmodell

Der Vorteil eines generischen Datenmodells besteht darin, dass zusätzliche Informationen ins Core integriert werden können, ohne dass das Datenmodell erweitert werden muss. Stattdessen wird nur zum Beispiel ein neuer Objekttyp oder eine neue Rolle definiert. Was aber auf den ersten Blick einfach erscheint, ist in der Praxis oft komplexer als angenommen. Zwar müssen im Core keine

Tabellen erstellt oder angepasst werden, aber die ETL-Prozesse zum Laden des generischen Modells (inkl. Anpassung der benötigten Metadaten, z.B. neue Rolle) sind trotzdem zu implementieren – und dies oft mit beträchtlichem Aufwand. Außerdem ist es nicht ganz einfach, die Daten effizient aus dem generischen Modell zu lesen, um sie nachher in Dimensions- und Faktentabellen der Data Marts zu laden. Der ursprüngliche Vorteil eines generischen Datenmodells – keine Modellanpassungen bei Erweiterungen – erweist sich somit längerfristig meistens als Nachteil.

Fazit

Den perfekten Ansatz für die Datenmodellierung im Core gibt es nicht. Jede der hier vorgestellten Verfahren – sowie weitere Ansätze, die hier nicht aufgeführt wurden – haben Vor- und Nachteile. Ob nun das Core eher dimensional oder relational modelliert wird, hängt vor allem davon ab, ob das Datenmodell basierend auf den Anforderungen an die Data Marts oder basierend auf den Quellsystemen erstellt wird. In beiden Fällen hat sich der Ansatz mit Head- und Versionstabellen gut bewährt. Vor allem in einem agilen Projektumfeld interessant ist auch der Ansatz von Data Vault Modeling, da er sehr flexibel bezüglich Erweiterbarkeit des Modells ist. Generische Datenmodelle können in gewissen Fällen zweckmäßig sein, erweisen sich aber in der Praxis oft als weniger geeignet.

Welcher Ansatz für die Modellierung des Core am besten geeignet ist, hängt von vielen Faktoren ab – es bleibt also weiterhin eine spannende Aufgabe, ein Core-Datenmodell entwickeln zu dürfen.

Kontaktadresse:

Dani Schnider
Trivadis AG
Europa-Strasse 5
CH-8152 Glattbrugg

Telefon: +41(0)44-808 70 20
Fax: +41(0)44-808 70 21
E-Mail dani.schnider@trivadis.com
Internet: www.trivadis.com