Das Data Warehouse

Planung, Aufbau, Auswertungen und Statistiken mit SSIS und Delphi

Delphi Tage 2013, Leipzig

Daniela Seyfzig, 23.09.13

Version 1.0
Montag, 23. September 2013

Daniela Sefzig
www.alien.at, www.playbox.at
daniela.sefzig@alien.at

Inhaltsverzeichnis

Vorwort

In einem Unternehmen begegnen uns Daten aus unterschiedlichen Quellen und in den verschiedensten Formaten. Um Daten für Auswertungen und Analysen zu sammeln, müssen diese aufbereitet werden. Zur Realisierung gibt es den Ansatz des Data Warehouses, kurz DWH genannt.

Dieses Dokument gibt Antwort auf die Frage, was man unter einem DWH versteht und in wie weit es uns bei der Auswertung von komplexen Daten behilflich sein kann. Tools, die bei der Erstellung eines DHWs behilflich sind, werden in diesem Dokument vorgestellt und kurz beschrieben.

Ein gutes DWH Design ist unumgänglich für eine erfolgreiche Auswertung und Analyse von betriebsrelevanten Daten. Die nötigen Schritte, die für das Design benötigt werden, werden im folgenden vorgestellt. Der Leser ist danach in der Lage, ein Aufbau und die Funktionsweise eines DWH von Grund auf zu verstehen, oder bei der Unterstützung der Implementierung eines DWHs durch ein externes Unternehmen die nötigen Begriffe und Schritte nachzuvollziehen und aktiv an Diskussionen teilnehmen zu können.

Da ein DWH aus Datenbanken liest und auch schreibt, sind Grundlagen der SQL Sprache dringend vorausgesetzt. Auf weiterführende Literatur wird im Anhang verwiesen.

Nachdem dieses Dokument begleitend für die Delphi-Tage erstellt wird, soll natürlich Delphi nicht zu kurz kommen. Für die Erstellung des DWHs blicken wir über den Tellerrand und werfen einen Blick auf Microsofts Integrated Services. Hier gibt es vorgefertigte Komponenten, die bei der Entwicklung behilflich sind. Für die Auswertung der Daten bietet sich auf jeden Fall Delphi an. Vor allem wird auch gezeigt, wie man die vorhandenen Daten durch Implementierung eines Statistik-Tools aufwertet. Zu diesem Zweck werden wir das Statistik-Tool R an Delphi anbinden und Daten austauschen.

Bei diesem Dokument handelt es sich um die erste Version. Ich bin dankbar für Informationen bezüglich Fehler, oder wenn Beschreibungen unklar formuliert sind.

Was ist ein Data Warehouse?

Ein Data Warehouse ist eine Datenbank, die integrierte Daten für bestimmte Entitäten beinhaltet, die Informationen über aktuelle und historische Geschäftsabläufe speichert und dessen Werte für zukünftige, strategische Entscheidungen herangezogen werden können.

Definition nach William H. Inmon:

> A data warehouse is a *subject oriented, integrated, time variant, non-volatile collection* of data *in support of management's decision making process.* [Inmon92]

- ***subject oriented***: für bestimmte Entitätentypen zugeschnitten, z.B. Verkäufe, Produkte, geographische Bereiche.
- ***integrated:*** die Daten im Data Warehouse stammen i.d.R. aus verschiedenen Quelldatenbanken, z.B., aus mehreren Verlagskatalogen, Lagerbeständen einzelner Lager, Einnahmen einzelner Läden, usw.
- ***time-variant:*** Data Warehouse zeigt die zeitliche Evolution der betrachteten Entitäten.
- ***non-volatile:*** Daten werden nicht gelöscht oder nachträglich geändert, Änderungen im Datenbestand sind allein auf das Laden neuer Daten zurückzuführen.
- ***support decision making:*** nur wichtige Daten für solche Entscheidungen speichern.

Wichtig: Ein Data Warehouse ist **kein** weiteres IT Projekt, es ist ein strategisches Projekt!

Der Unterschied zu einer herkömmlichen Datenbank besteht darin, dass beim Design der Datenbank keine Normalisierung erfolgt. Es werden bewusst auch redundante Daten verwendet, da die Geschwindigkeit, mit der die Daten gelesen werden, im Vordergrund steht. Ein Data Warehouse ist ungleich größer als eine herkömmliche Datenbank, da darin auch historische Daten, mit deren Hilfe Veränderungen nachvollzogen werden können, gespeichert werden.

Unterschiedliche Ansätze

	Relationale Datenbank OLTP (Online Transaction Processing)	Data Warehouse OLAP (Online Analytical Processing)
Normalisierung	Normalisierung meist bis zur 3. oder 4. Ebene	Denormalisiertes Datenmodell um das Laufzeitverhalten zu verbessern
Redundante Daten	werden vermieden	werden bewusst implementiert
Optimiert auf	Daten schreiben	Daten lesen (analytische Abfragen)
Historische Daten	nur für einen gewissen Zeitraum	werden für einen definierten Zeitraum gespeichert
Aggregierte Daten	üblicherweise nicht	ja
Datenvolumen	klein	sehr groß
Inhalt der Daten	Anwendungs- und Funktionsbezogen	Themenbezogen
Modellierung	Entity Relationship Modell (ERM) Relationales Datenbank Model (RDM)	Application Design for Analytical Techniques (ADAPT) Dimensional Fact Model (DFM)

Unterschiedliche Sicht auf das Data Warehouse

Die Entwickler und das Management bzw. Analysten haben eine unterschiedliche Sicht auf das Data Warehouse. Für den Entwickler ist es jedoch wichtig, auch die Sicht des Managements zu kennen und zu verstehen. Das Management interessiert sich in der Regel nicht für die technische Umsetzung. In der Kommunikation sollte der Entwickler daher technische Begriffe vermeiden.

Entwickler	Management
Backup Datendurchsatz Loadblancing	Portfolio Umsatz Werbung

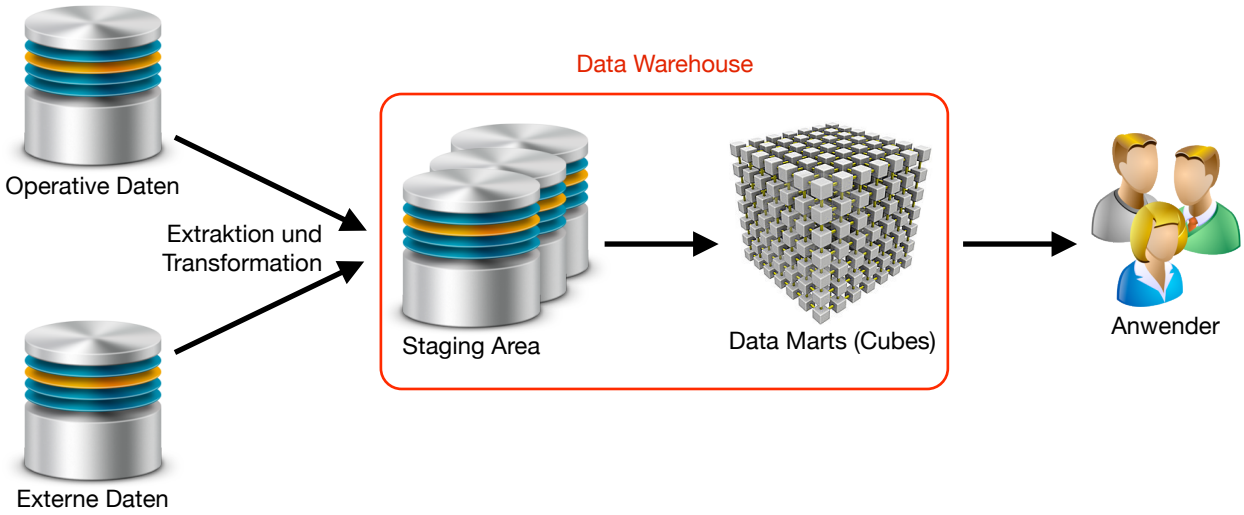


Abbildung: Das Data Warehouse

Was muss ein Data Warehouse beinhalten?

Ein Data Warehouse muss...

- ...organisatorische Informationen einfach zugänglich bereitstellen
- ...konsistente, im Unternehmen definierte Bezeichnungen verwenden
- ...anpassungsfähig und belastbar auf Änderungen reagieren können
- ...Informationen schützen, damit unberechtigte Personen nicht darauf zugreifen können
- ...die richtigen Daten beinhalten, die als Entscheidungsgrundlage für strategische Ziele im Unternehmen dienen
- ...von den Endanwendern akzeptiert werden

Das erfolgreiche Data Warehouse

Damit ein Data Warehouse nachhaltig von Erfolg geprägt ist, sollten folgende Punkte berücksichtigt werden:

- Unterstützung des Managements
- Einbeziehen der einzelnen Abteilungen bzw. Analysten und erfassen ihrer Bedürfnisse
- Ein Mitarbeiter ist dezidiert für das DWH zuständig und bleibt dies auch
- Schrittweise Implementierung, beginnend mit dem Geschäftsprozess, der dem Unternehmen den meisten Nutzen bringt
- Die Bedürfnisse der Benutzer müssen befriedigt werden, nicht die der IT-Mitarbeiter
- Zum Testen bereits Echt-Daten verwenden, damit testende Personen die Richtigkeit der Werte überprüfen können

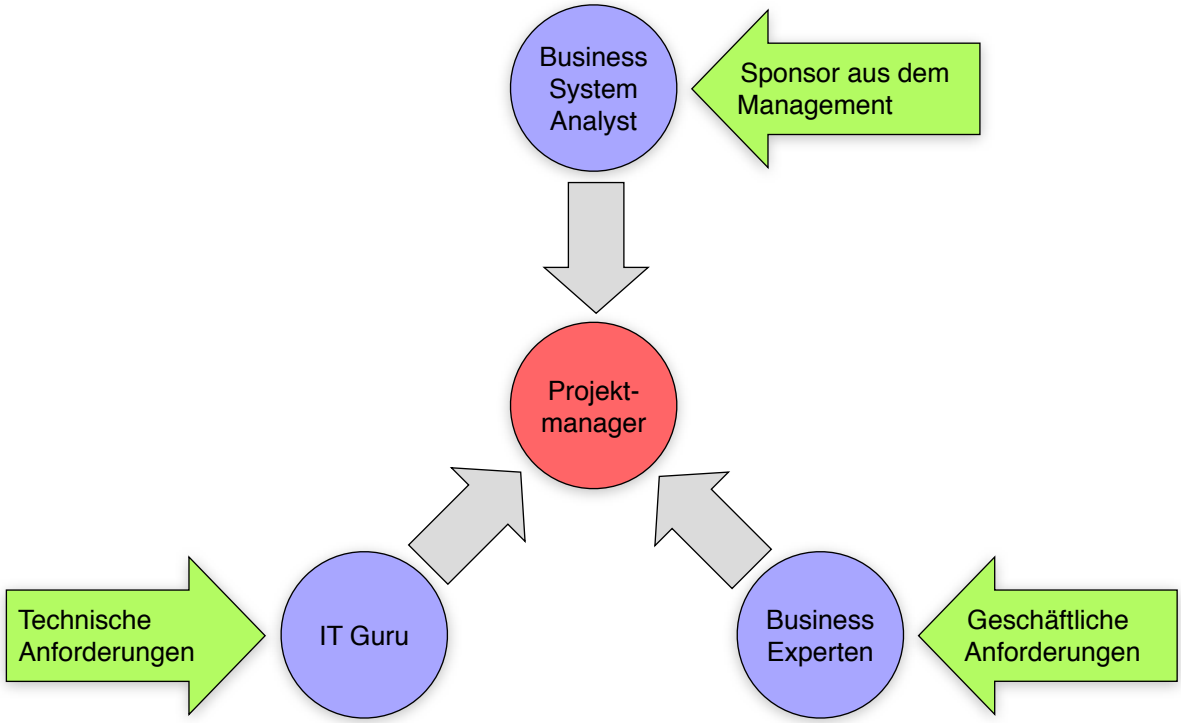


Abbildung: Das erfolgreiche Data Warehouse

Aufbau

Master Plan

Der oft vernachlässigte, sogenannte Master Plan wird zu Beginn erstellt. Er ist aber für den Erfolg und die Übersicht des Data Warehouses unverzichtbar.

Im ersten Schritt wird damit begonnen, die Geschäftsprozesse und deren zugehörige Daten zu erfassen. Im Idealfall richtet man sich nach der Reihenfolge des Supply Chain Managements. Verschiedene Abteilungen sind in die unterschiedlichen Bereiche der Supply Chain involviert. Jede Abteilung stellt unterschiedliche Anforderungen an das Data Warehouse, um an Daten zur Optimierung ihrer Prozesse zu gelangen.

Sind die Geschäftsprozesse identifiziert, dann wird damit begonnen, die Dimensionen zu identifizieren. Hierzu definiert man die bei dem jeweiligen Geschäftsprozess benötigten Daten und trägt diese in die Tabelle ein.

Ist der Master Plan erstellt, so kann man anhand der Dringlichkeit entscheiden welche Auswertung als erstes umgesetzt werden kann. Ist es zum Beispiel erforderlich Daten für die Beschaffung schnell zu erfassen, um den Nachschub zu optimieren, so kann mit diesem Schritt unabhängig von den anderen Prozessen begonnen werden. Dadurch wird einerseits die Akzeptanz des Data Warehouses erhöht als auch erste Erfahrungen gesammelt.

Wenn keine Dringlichkeit besteht, ist es ein guter Schritt mit dem Geschäftsprozess zu beginnen, der für das Unternehmen den größten Mehrwert darstellt.

Dimension →

	Date	Product	Store	Promotion	Warehouse	Vendor	Contract	Shipper
Retail Sales	X	X	X	X				
Retail Inventory	X	X	X		X			
Retail Deliveries	X	X	X					
Warehouse Inventory	X	X			X	X		
Warehouse Deliveries	X	X			X	X		
Purchase Orders	X	X			X	X	X	X

↑ **Geschäftsprozess**

Abbildung: Master Plan

Multidimensionale Ansicht - Data Marts

Die multidimensionale Ansicht (Data Mart) wird mit Hilfe eines Würfels (Cube) dargestellt. Die Kanten des Würfels entsprechen den jeweiligen Dimensionen. Der Inhalt an einer bestimmten Stelle stellt einen Wert dar.

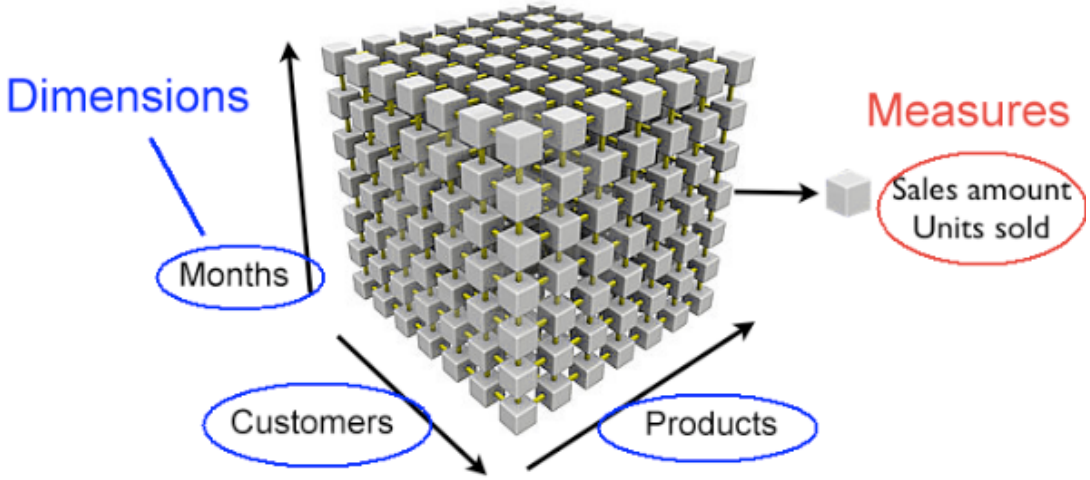


Abbildung: multidimensionale Ansicht aus [12]

Die Dimensionen bilden die Grundlage für die Dimension-Tabellen. Sie beinhalten Spalten, die die Dimension beschreiben. Um die einzelnen Werte und die Relationen zu den Dimensionen und deren Spalten zu speichern, wird eine Fact-Tabelle verwendet. Im Kapitel „Aufbau des Data Warehouses“ kommen wir auf diese Tabellen noch zu sprechen.

Zuerst können wir jedoch festhalten, dass ein Würfel aus folgenden Elementen besteht:

- Einer Fact Tabelle für die Zelleneinträge
- Verschiedene Dimension Tabellen für die Level-Einträge
- 1 zu N Beziehungen von den Dimension Tabellen zur Fact Tabelle

Mythen von Dimensions-Modellen und Data-Marts

Kimbal identifiziert in seinem Data Warehouse Toolkit [9] fünf Mythen beim Modellieren von Dimensions-Modellen. Bei der Planung oder bei Besprechungen werden diese Mythen immer wieder angesprochen. Die Erklärung soll helfen richtig zu Argumentieren und Ängste oder Befürchtungen abzubauen.

Dimensions-Modelle und Data-Marts...

...sind nur für Summenwerte

Der erste Mythos ist der Ausgangspunkt für jedes schlecht geplante Data Warehouse. Man kann nicht alle Fragen der Anwender berücksichtigen, aber man kann ihnen Daten zur Abfrage bereitstellen, damit sie diese selbstständig ermitteln können. Dazu werden Daten auf dem kleinstmöglichen Level gespeichert. Die Anwender bilden die benötigten Summenwerte selbstständig.

Ebenso sollte nur eine limitierte Anzahl an historischen Daten in einer Dimension gespeichert werden. Es ist nicht verboten, historische Werte in einem Dimension-Modell zu speichern, aber die Anzahl der Werte muss sich nach den Anforderungen des Geschäftsprozesses richten.

...sind keine Lösung für das Unternehmen, sondern für die Abteilung

Data-Marts sind so organisiert, dass sie Geschäftsprozesse abbilden. Ergebnisse aus den Prozessen dienen zur Auswertung für viele verschiedene Abteilungen, denn oft benötigen Abteilungen die gleichen Werte, um ihre Prozesse zu optimieren. Wenn die Geschäftsprozesse optimiert werden, wirkt sich das in der Regel positiv auf das Unternehmen aus. Ebenso stellen diese Daten die Grundlage für strategische Entscheidungen für das Unternehmen.

...sind nicht skalierbar

Fact Tabellen beinhalten heutzutage Millionen von Datensätzen. Anbieter von Datenbanken haben die Möglichkeiten eines Data Warehouses längst erkannt und verschiedene Funktionen in deren Produkte integriert, um beste Skalierbarkeit und Performance zu bieten.

...sind nur anwendbar wenn ein Anwendungsmuster erkennbar ist

Die dimensionalen Strukturen der dimensionalen Modelle sind sehr flexibel. Das Geheimnis dieser Flexibilität ist es, die Auflösung der Fact-Tabellen so detailliert wie möglich zu halten. Werden Daten allerdings zu grob aggregiert, dann ist die Flexibilität nicht mehr gewährleistet.

...können nicht integriert werden und führen daher zu Flaschenhals-Lösungen

Dimensionale Modelle können integriert werden, wenn sie konform zur Data Warehouse Bus-Architektur sind.

Fehlerquellen

Es gibt einige Fehlerquellen, die bei der Entwicklung oft missachtet werden und die Akzeptanz des Data Warehouses negativ beeinflussen (mit dem Wissen über diese Fehlerquellen kann man dem Problem natürlich entgegenwirken).

- Der Entwickler ist in die technische Lösung verliebt und verliert dabei die Unternehmens-anforderungen aus den Augen
- Es wird zu viel Energie in eine normalisierte Datenstruktur gesteckt und dabei das Budget überschritten, anstatt eine denormalisierte Datenstruktur für ein multidimensionales Modell zu erstellen
- Es wird viel Aufmerksamkeit in die Performance für die Aufbereitung der Daten gesteckt und dabei die Performance der Abfrage vernachlässigt
- Die Abfrage für den Endanwender wird zu kompliziert gestaltet
- Dem Endanwender werden nur aufsummierte Daten zur Verfügung gestellt
- Es wird angenommen, die Anforderungen, die Analysen und die zugrundeliegenden Daten sind statisch
- NULL anstatt 0 Werte werden verwendet. Werte müssen immer definiert sein. Wenn sie nicht vorhanden oder verfügbar sind, sind sie mit 0 anzugeben
- Das Data Warehouse wird von den Endanwendern nicht akzeptiert

Aufbau des Data Warehouses

Bevor auf das Design eines Data Warehouses eingegangen wird, werden des besseren Verständnisses halber, noch zwei wichtige Stichwörter, Dimension Table und Fact Table, beschrieben.

Dimension Table

Eine Dimension Tabelle beinhaltet Informationen, die einen bestimmten Bereich beschreiben. Beispielsweise würde eine Dimension Tabelle für Kunden den Kundennamen, dessen Adresse, die Telefonnummer usw. beinhalten.

Fact Table

Die Fact Tabelle beinhaltet die Werte, die für die Analyse benötigt werden. Dies können Rohdaten als auch aufbereitete Daten sein. Über einen Fremdschlüssel sind die Dimension Tabellen mit der Fact Tabelle verknüpft.

Eine Zeile in einer Fact Tabelle entspricht einem Wert. Die Werte in dieser Zeile müssen den selben Detaillierungsgrad aufweisen. Es darf nicht sein, dass ein Wert für einen Tag und ein Wert, der für eine Woche gültig ist, in der gleichen Zeile vorkommen. Es darf auch nicht vorkommen, dass ein Wert NULL ist. Ist der Wert nicht verfügbar oder vorhanden, so muss eine 0 eingetragen werden.

Der Primärschlüssel einer Fact-Tabelle wird aus den Fremdschlüsseln zu den Dimension-Tabellen gebildet. Es wird kein eigenes Feld für den Primärschlüssel implementiert.

Design des Data Warehouses

Für das Erstellen und Identifizieren von Dimension Tabellen sollten folgende vier Schritte berücksichtigt werden:

4 Schritte zum Design

1. Auswahl des Geschäftsprozesses

Im ersten Schritt betrachtet man den Geschäftsprozess, den man im Master Plan definiert hat. Hier wurden schon Gruppen von Informationen ermittelt, die jetzt die Basis für die Dimension Tabelle bilden.

Für das erste Modell, das man erstellt, wählt man den Geschäftsprozess mit dem größten Wert für das Unternehmen. Er soll Antworten auf die am häufigsten gestellten Fragen liefern.

2. Detaillierungsgrad definieren

Der Detaillierungsgrad beschreibt, was eine Zeile in der Dimension Tabelle beinhalten soll. Man kann sich die Frage stellen *„Wie beschreibe ich eine einzelne Zeile in der Fact-Tabelle?“*. Dies kann jede einzelne Transaktionen sein oder aber auch zusammengefasst.

Den Detaillierungsgrad zu Beginn richtig zu dimensionieren ist sehr wichtig. Ist er zu grob definiert, so wird zwar zur Erfassung der Daten Speicherplatz gespart, aber eine zeitliche Auswahl der Analyse kann dadurch verhindert werden. Ist der Detaillierungsgrad feiner, als er

für Analysen benötigt wird, so wird viel Speicherplatz benötigt und die Beladung der DHWs dauert verhältnismäßig lange.

Eine Anforderung kann zum Beispiel sein, dass die Anforderung aus der Controlling Abteilung besagt, dass eingehende Waren nur wöchentlich erfasst werden. Der Detailierungsgrad ist somit mit einer Woche gewählt. Eine Auswertung, welche Artikel an welchen Tag angeliefert wurden, ist aber nicht mehr möglich.

Wichtig: Der Detaillierungsgrad kann im Nachhinein nur noch sehr schwer, bis gar nicht mehr, geändert werden. Daher wird eine genaue Abklärung unbedingt angeraten!

3. Dimensionen wählen

Dimension Tabellen werden an verschiedenen Stellen im DWH immer wieder verwendet. Man sollte der Frage nachgehen *„Wie beschreiben Anwender die Daten, die aus dem Geschäftsprozess kommen?“*.

Beispiele wären etwa das Datum, das Produkt, die Kunden, verschiedene Transaktionstypen und der Status.

4. Fakten wählen

Im letzten Schritt werden die numerischen Fakten ermittelt, die in einer Fact-Tabelle als Zeile eingefügt werden. Auch hier kann man sich mit einer Frage helfen *„Was messen wir?“*. Analysten sind daran interessiert die Performance eines Geschäftsprozesses zu messen.

Typische Beispiele sind Kosten, Einnahmen, verkaufte Stückzahlen, Fehlerraten usw.

Fact und Dimension Tabellen zusammenführen

Die multidimensionale Ansicht muss in eine für den Computer lesbare Form gebracht werden und wird daher in Dimension- und Fact-Tabellen übernommen. Als Beispiel nehmen wir den Würfel, den wir im Kapitel der multidimensionalen Ansichten bereits vorgestellt haben.

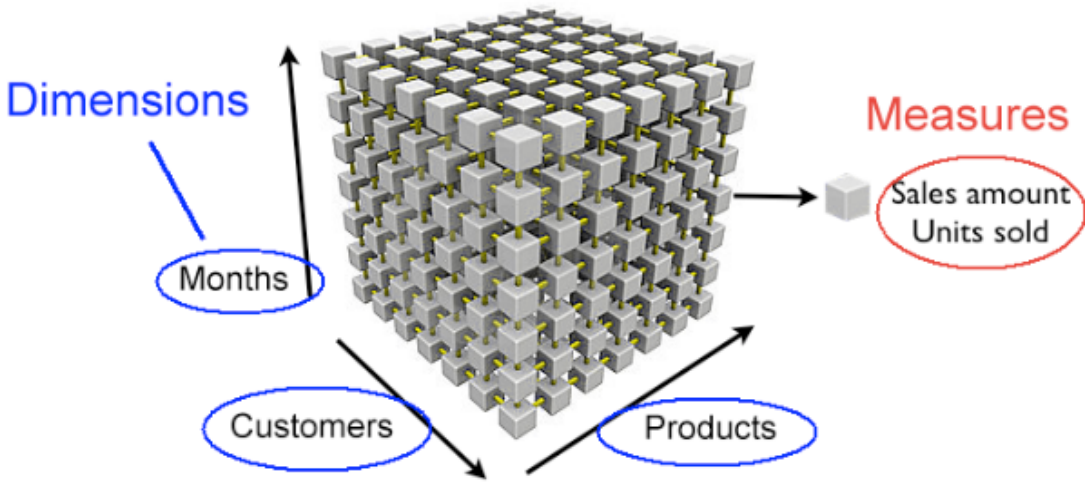


Abbildung: multidimensionale Ansicht aus [12]

Die Dimensionen sind die Kanten des Würfels, und man kann diese in die Dimension Tabellen übernehmen. Dazu werden alle benötigten Spalten verwendet, die einen Datensatz in der Dimension beschreiben. Danach muss noch die Auflösung definiert werden. Im speziellen betrifft das die Dimension des Datums. Wie aus dem Würfel ersichtlich, könnte dies das Monat sein. Wollen wir aber flexibler sein und dem Endanwender detailliertere Informationen zukommen lassen, so kann zum Beispiel die Auflösung auf den Tag festgelegt werden. Wie bereits erwähnt (man kann es nicht oft genug ansprechen) ist es wichtig, diesen Detaillierungsgrad zuvor zu definieren, da er nachträglich nur schwer geändert werden kann.

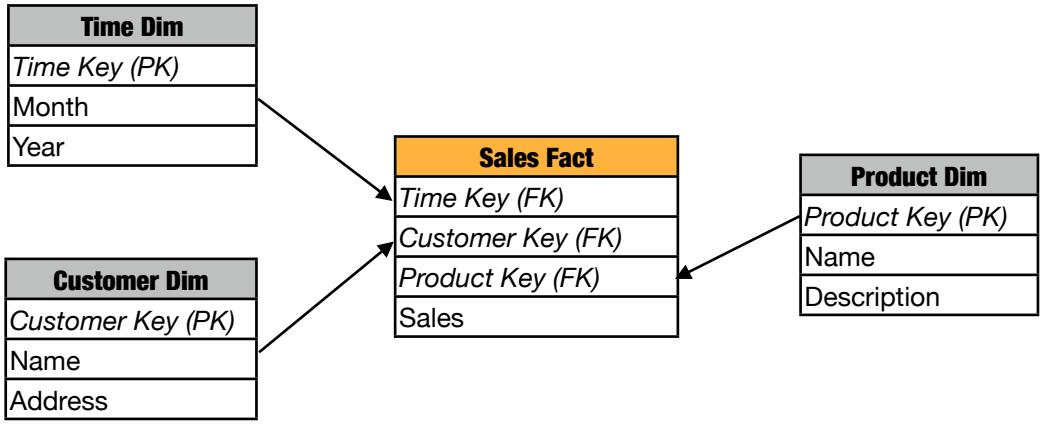


Abbildung: Dimension- und Fact-Tabellen aus der multidimensionalen Ansicht

Aus der dimensionalen Ansicht wurde nun das sogenannte Star-Schema erstellt. Das Modell ist aber noch nicht ganz vollständig. Zumeist beinhalten die Spezifikationen auch nicht die Ansicht eines Würfel, vielmehr wird eine Modellierungssprache verwendet. Zu diesen Sprachen kommen wir etwas später. Zuerst gilt es noch die verschiedenen Eigenheiten der Dimension- und Fact-Tabellen kennen zu lernen, um im Anschluss auch ein komplettes Modell des Data Warehouses zu erstellen.

Star-Schema

Wie bereits erwähnt, werden Dimension Tabellen mit den Fact Tabellen verknüpft. Durch das Muster, das dadurch entsteht, wird dieses Schema auch „Star-Schema“ genannt.

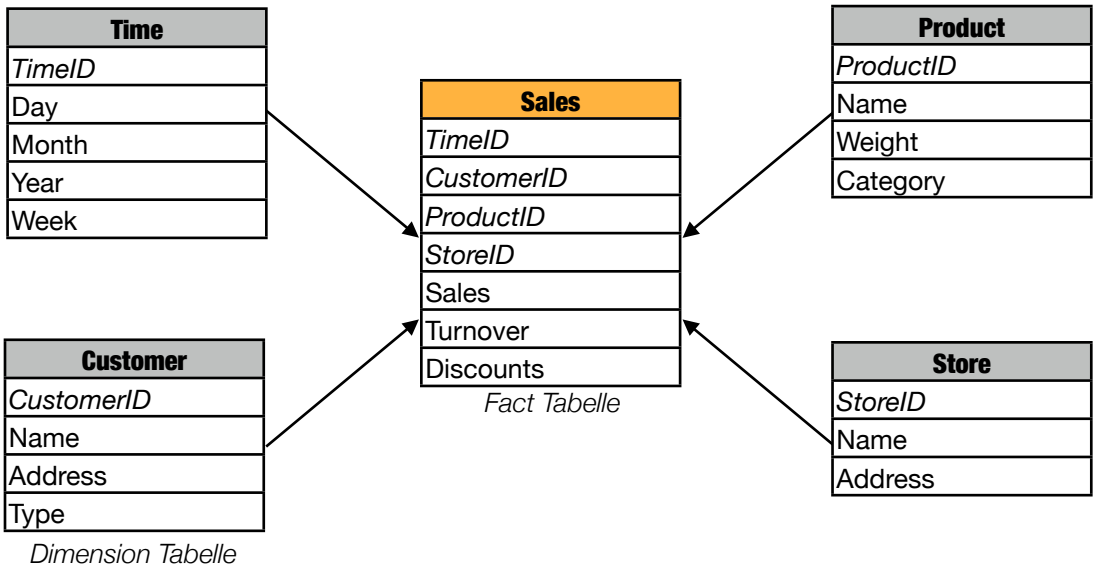


Abbildung: Star-Schema

Snowflake-Schema

Wenn über das Star-Schema gesprochen wird, soll ein Begriff vorweg genommen werden. Das sogenannte Snowflake-Schema ist eines Sonderheit, die beim Design eines Data Warehouse entstehen kann. Es entsteht, wenn Dimension Tabellen mit anderen Dimension Tabellen verknüpft werden. Dies ist in der Regel nicht verboten, jedoch deutet eine übermäßige Anzahl von Snowflakes auf ein schlechtes Design hin.

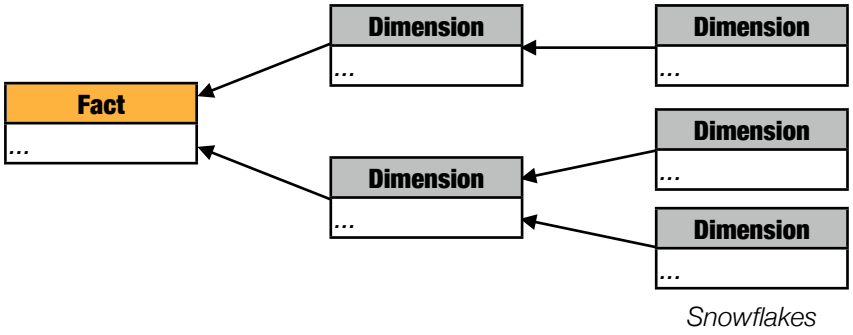


Abbildung: Snowflake-Schema

Slowly Changing Dimensions (SCD)

Bei Dimension Tabellen können Änderungen der gespeicherten Daten auftreten. Man bedenke ein Außendienstmitarbeiter ist für ein bestimmtes Bundesland zuständig. Im Laufe seiner Karriere wird er befördert und ein anderer Mitarbeiter übernimmt seinen Bereich. Würde dieser Umstand nicht berücksichtigt werden, so würden dem neuen Mitarbeiter auch die Umsätze und Verkäufe des Vorgängers zugerechnet werden und somit ein falsches Bild entstehen. Um diesen Umstand zu berücksichtigen, gibt es verschiedene Techniken.

Wir nehmen das zuvor gegebene Beispiel zum Anlass die Unterschiede der einzelnen Typen zu beschreiben. Der Manager „Homer Simpson“ verkauft Duff-Bier und gibt seinen aktuellen Verkaufsbereich „Ost“ auf und wechselt am 1.1.2014 zu dem Verkaufsbereich „West“.

Vorab, Type I und II werden in der Praxis bevorzugt verwendet. Je nach Anwendungsbereich kann zwischen den einzelnen Methoden gewählt werden.

SCD Type I

Bei Type I werden die vorhandenen Daten einfach überschrieben. Es kann anschließend nicht mehr festgestellt werden, wann eine Änderung erfolgt und welcher Bereich dem Mitarbeiter vorher zugewiesen war.

ID	Region	Account Manager
381	West	Homer Simpson

Dieser Typ ist naturgemäß einfach zu implementieren, lässt jedoch keinerlei Rückschlüsse auf vergangene Ereignisse zu.

SCD Type II

Bei Type II wird für die Änderung ein neuer Datensatz angelegt. Optional dazu kann das Datum der Änderung gespeichert werden. Alternativ kann auch ein Zeitbereich für die Gültigkeit des Wertes (von-bis), oder ein „Historical-Flag“ gesetzt werden.

ID	Region	Account Manager	Date
381	Ost	Homer Simpson	1.6.2011
382	West	Homer Simpson	1.1.2014

SCD Type III

Bei Type III wird der alte Wert in eine entsprechende Spalte verschoben und der neue Wert eingesetzt. Optional kann auch das Datum der Änderung eingetragen werden.

ID	Region	Old Region	Account Manager	Date
381	West	Ost	Homer Simpson	1.1.2014

SCD Type VI

Type VI ist eine Kombination aus den drei zuvor vorgestellten Methoden. Es ist eine sehr aufwändige Methode und wird in der Praxis nicht oft angewendet.

ID	Region	Old Region	Account Manager	Date
381	West	Ost	Homer Simpson	1.1.2014
382	West	West	Homer Simpson	1.1.2014

Spezialfälle

In der Praxis gibt es einige Spezialfälle, wenn es um Dimension-Tabellen und die Abbildung von Daten geht.

Junk Dimensions

In sogenannten Junk Dimensions werden Daten gespeichert, die miteinander in keinen Zusammenhang stehen. Es werden einfache Attribute kombiniert, um zusätzliche Dimension-Tables einzusparen. Üblicherweise werden Werte in diesen Junk Dimensions zum Filtern und Gruppieren verwendet.

OrderID	Payment Type	Order Indicator	Commission Credit Indicator
1	Cash	Inbound	Commissionable
2	Cash	Inbound	Non-Commissionable
3	Cash	Outbound	Commissionable
4	Cash	Outbound	Non-Commissionable
5	Credit	Inbound	Commissionable
6	Credit	Inbound	Non-Commissionable
7	Credit	Outbound	Commissionable
8	Credit	Outbound	Non-Commissionable

Man beachte, dass jedmögliche Kombination der Werte durch die Tabelle abgedeckt wird. Auf die ID, die die richtige Kombination der Werte beinhaltet, wird in der Fact-Tabelle verwiesen.

Bridges

Normalerweise sind Fact-Tabellen die Verbindung zwischen den einzelnen Dimension Tabellen. Es kann aber auch vorkommen, dass zwei Fact-Tabellen miteinander verknüpft werden müssen, die über keine Fact-Tabelle miteinander in Beziehung gebracht werden können. Für diesen Fall wird eine sogenannte Bridge-Tabelle erstellt.

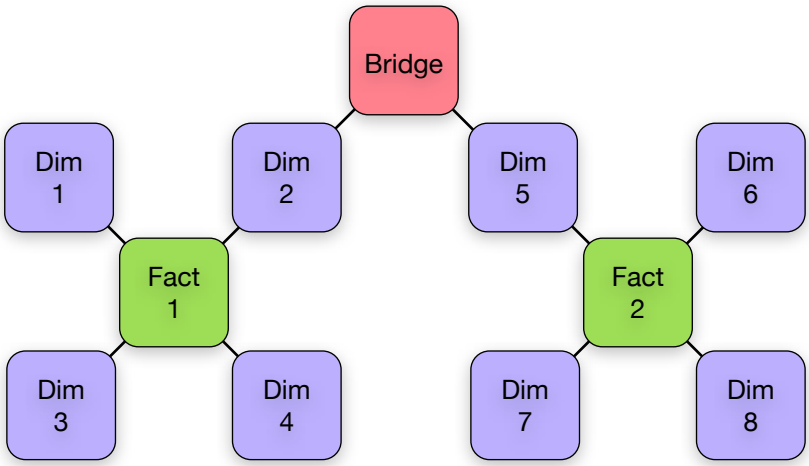


Abbildung: Bridge

Ein weiterer Anwendungsfall von Bridge-Tabellen ist die hierarchische Darstellung von Werten durch optionale Verwendung zwischen Fact- und Dimension-Tabelle.

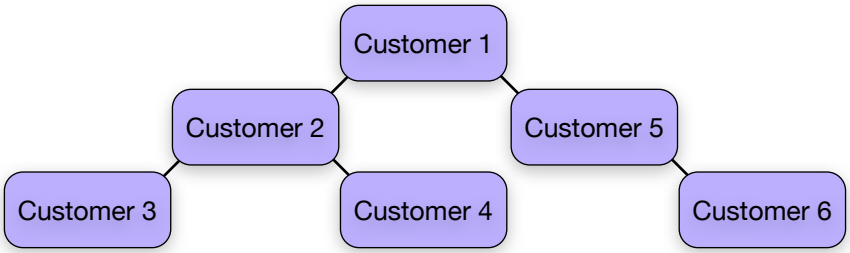


Abbildung: hierachische Darstellung

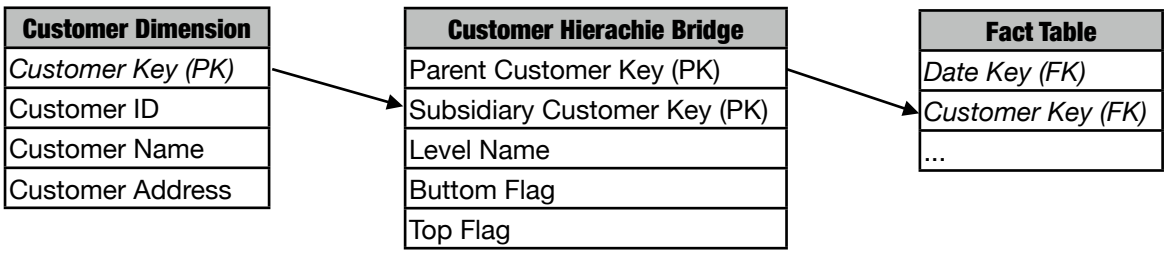


Abbildung: hierachische Darstellung

Hinweis: Ein schlecht geplantes Data Warehouse erkennt man daran, dass zu viele Bridge Tabellen verwendet wurden. Bridge Tabellen müssen also wohl überlegt eingesetzt werden. Sind es zu viele, dann muss das Design des Data Warehouses überdacht werden.

Outtriggers

Manche Dimensionen tendieren dazu, sehr lange zu werden oder sich inhaltlich sehr häufig zu ändern. Bestimmte Daten können auch von mehreren Datensätzen verwendet werden und somit redundant vorliegen. Obwohl Redundanz im Data Warehouse durchaus akzeptiert wird, kann versucht werden, bestimmte Daten zu optimieren.

Für diese Fälle gilt es einen Weg zu finden, um Speicherplatz zu sparen. Dies wird durch Absplittung der relevanten Daten in eine eigene Dimension erreicht. Als Beispiel nehmen wir an, dass zu einem Kunden auch demographische Daten zu seinem Land gespeichert werden sollen. Ein Eintrag in der Dimension Tabelle für die demographischen Daten wird von verschiedenen Kunden verwendet. Zusätzlich ändern sich diese Daten auch sehr oft bzw. müssen ständig aktuell gehalten werden. Es bietet sich also an diese demographischen Daten in eine Outtrigger Dimension zu verlagern.

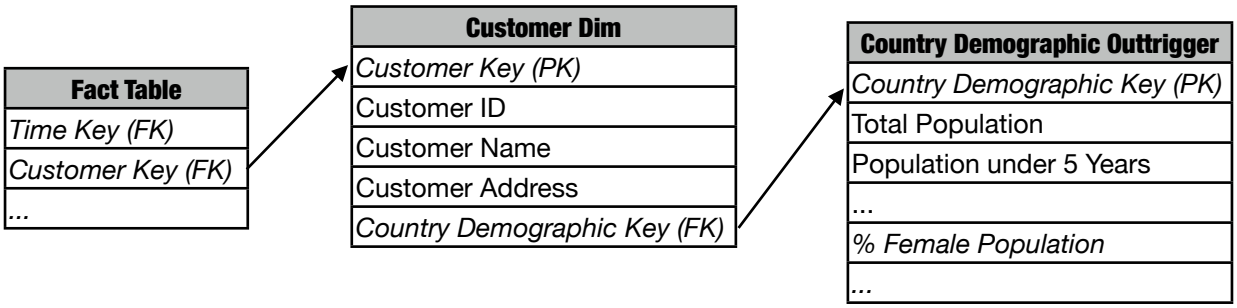


Abbildung: Outtrigger

Doch nicht immer eignen sich Daten, um sie in eine Outtrigger Dimension zu verschieben. Das folgende Beispiel zeigt, wie man es nicht machen sollte! Das Datum des ersten Einkaufs ändert sich nur einmal, nämlich dann, wenn der Kunde seine erste Bestellung tätigt. Durch diesen Outtrigger wurde ein sogennantes Snowflake Schema erstellt.

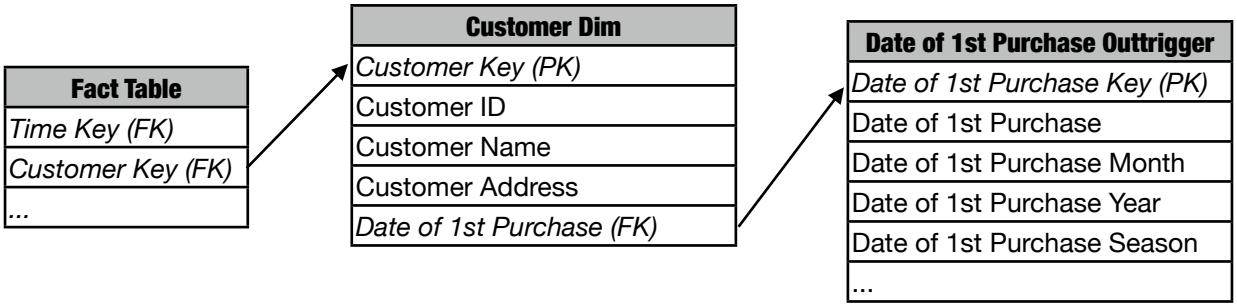


Abbildung: Schlechtes Beispiel für einen Outtrigger

Hinweis: Outtrigger Tabellen führen zu einem Snowflake Schema, welches auf ein schlecht geplantes Data Warehouse hinweist! Daher sollten Outtriggers nur begrenzt verwendet werden. Sind es zu viele, dann muss das Design des Data Warehouses überdacht werden.

Minidimensions

Minidimensions sind den Outtriggern sehr ähnlich. Während Outtrigger Dimension Tabellen Joins auf andere Dimension Tabellen darstellen, stellen Minidimension Tabellen eine Extraktion einer Dimension Tabelle dar und sind über Joins zu Fact-Tabellen integriert.

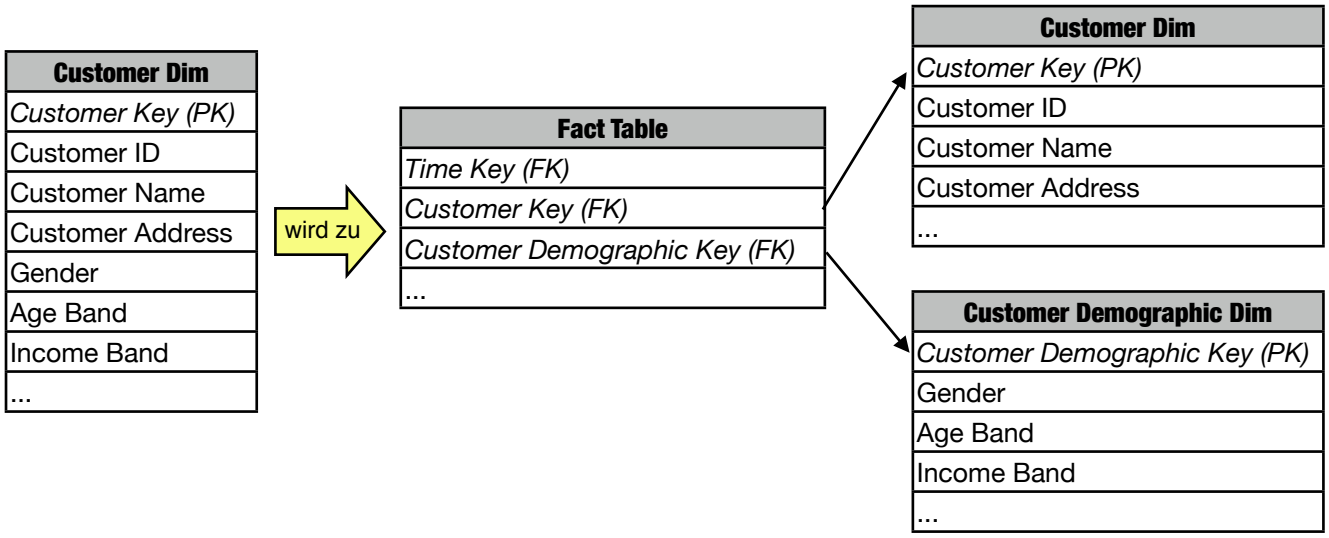


Abbildung: Minidimension

Multivalued Dimensions

Mittels Multivalued Dimensions können 1:N Beziehungen realisiert werden. Es werden also Gruppierungen erzeugt. Wenn die Werte der Dimension nur Bezeichnungen sind, so können Multivalued Dimensions über Outtriggers realisiert werden.

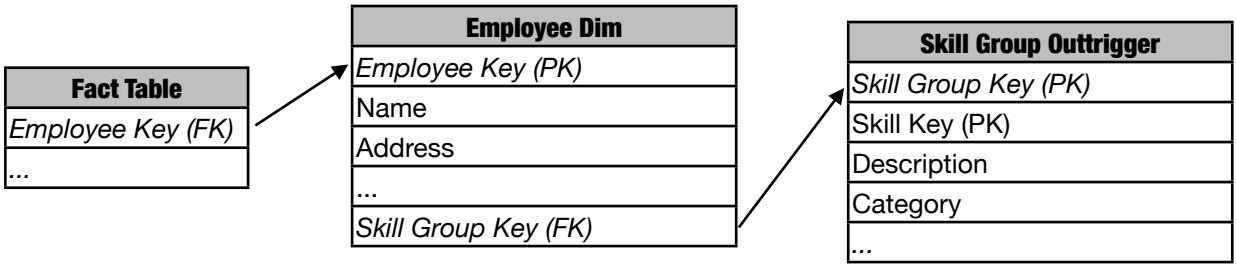


Abbildung: Multivalued Dimension mit Outtrigger

Beispiele für die Skill Group wäre etwa die gesprochene Sprache, die Ausbildung oder vorhandene Programmierkenntnisse. Sind die Werte jedoch Einträge einer anderen Fact Tabelle, so kann eine Multivalued Dimension über Bridges implementiert werden.

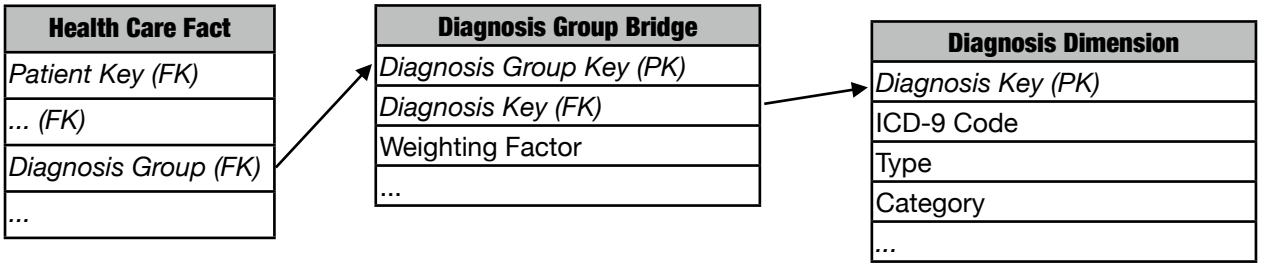


Abbildung: Multivalued Dimension mit Bridge

Unterschiedliche Fact-Tabellen

Fact-Tabellen werden in drei Bereiche eingeteilt. Diese drei Bereiche werden auch später bei der Befüllung mit Daten unterschiedlich behandelt. Daher ist es wichtig, den Bereich, für den die Fact Tabelle bestimmt ist, zu identifizieren.

Transactional Fact Tables

...bilden einen präzisen Moment ab. Sie werden, wie der Name schon sagt, für die Abbildung von Transaktionen verwendet. Das können zum Beispiel Bestellungen von Kunden oder Lieferungen an Kunden sein.

Eine Zeile in einer Transactional Fact Tabelle beinhaltet dabei genau eine Transaktion.

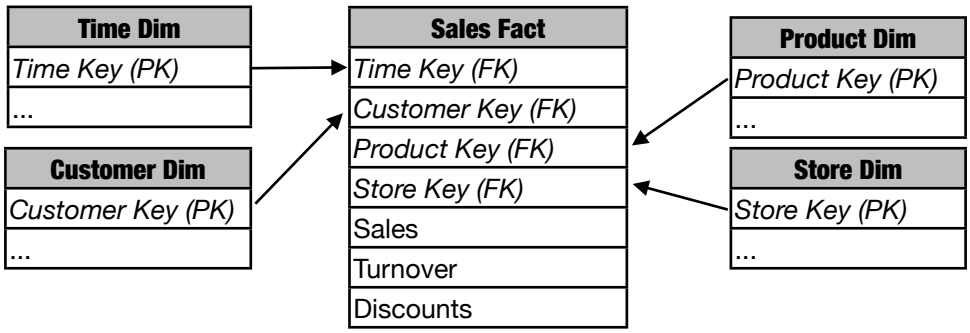


Abbildung: Transactional Fact Table

Snapshot Tables

...bilden ein Bild eines Momentes ab. Snapshot Tabellen sind typischerweise semi-additiv. Beispielsweise sind Lagerbestände nicht-additiv über die Zeit, aber additiv über Niederlassungen für einen bestimmten Zeitpunkt.

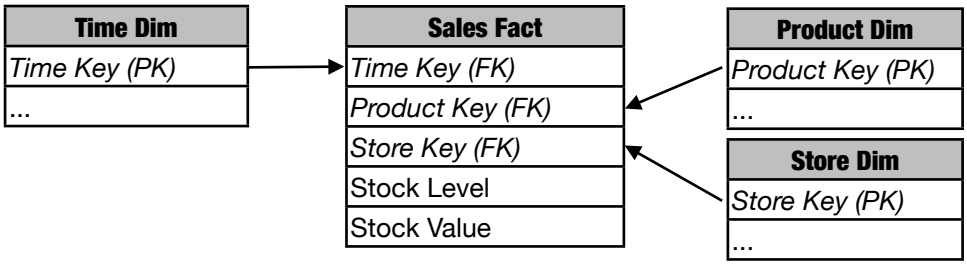


Abbildung: Snapshot Fact Table

Accumulated Tables

...bilden einen Prozess ab, der einen definierten Anfang und auch ein definiertes Ende hat, zum Beispiel der Prozess einer Bestellung. Mit einer Fact-Tabelle wird der Status über die Zeit erfasst.

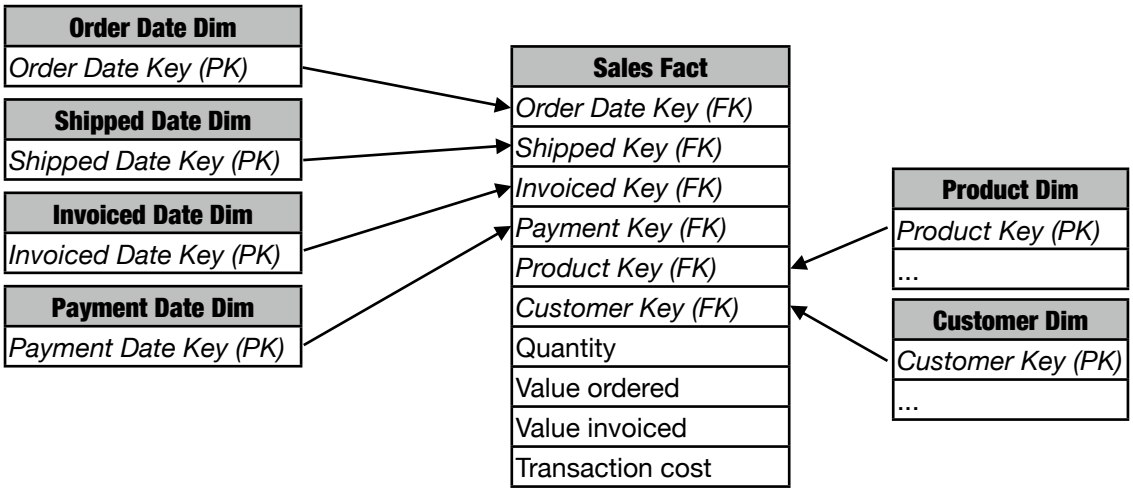


Abbildung: Accumulated Fact Table

Factless Fact Tables

Es gibt nichts, was es nicht gibt, daher gibt es auch Fact-Tabellen die keine weiteren Werte enthalten. Sie dienen zum Beispiel zum Zählen von Ereignissen.

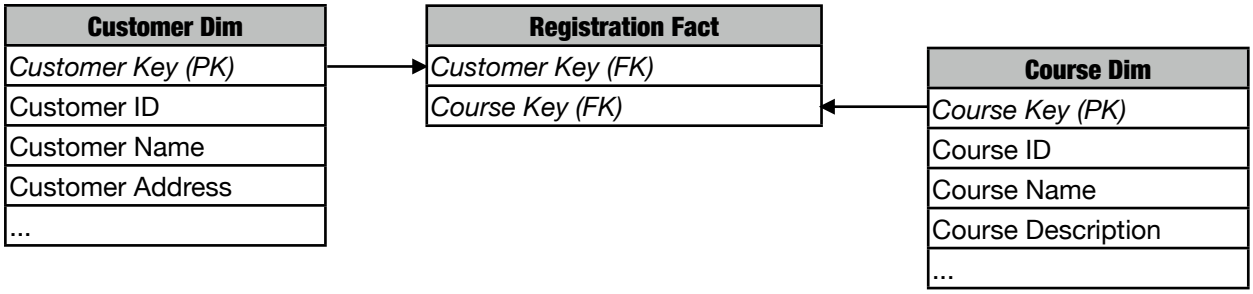


Abbildung: Factless Fact Table

```
select [Course Description], Count(*) from ... group by [Course Description]
```

Modellierung

Die Modellierung hilft, die unterschiedliche Sprache der Mitarbeiter auf eine verständliche Ebene zu bringen. Mitarbeiter der IT haben eine unterschiedliche Herangehensweise als Mitarbeiter aus dem Management, die Geschäftsprozesse mit einem anderen Hintergrund-verständnis modellieren.

Von Analysten, die die Anforderungen an das Data Warehouse stellen, wird üblicherweise die Modellierung in der Adapt (Application Design for Analytical Techniques) verwendet. Diese Anwender befassen sich nicht mit den technischen Aspekten, sondern haben eine Analyse-Sicht auf das Data Warehouse.

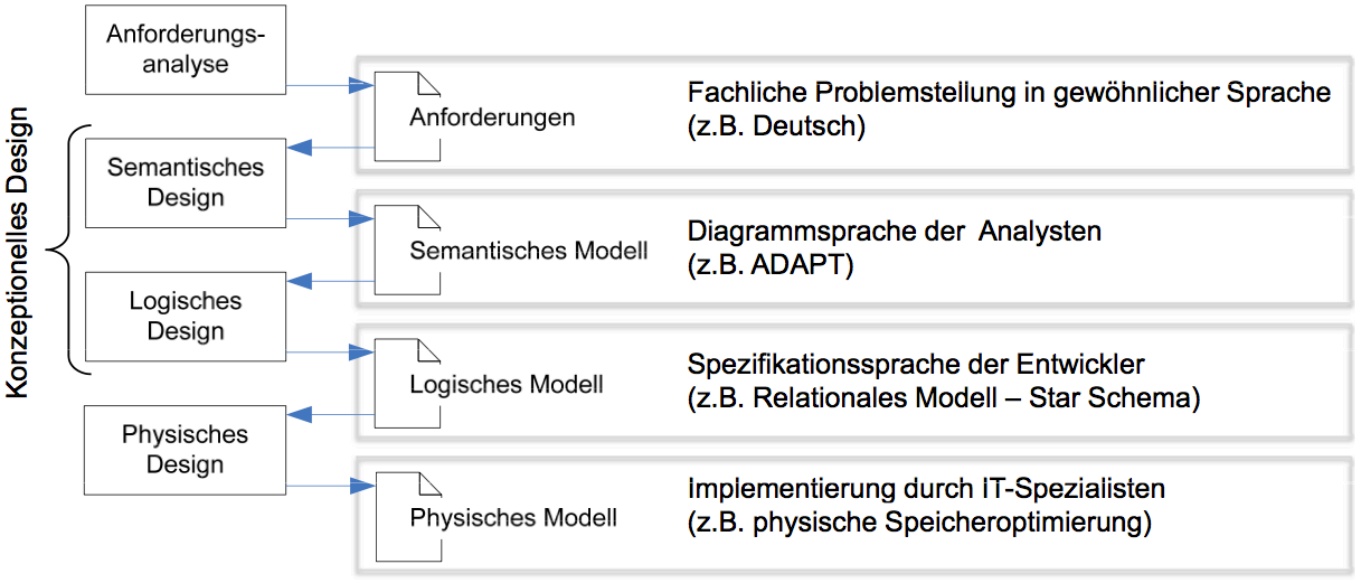


Abbildung: Konzeptionelles Design, aus [11]

In der IT wird das DFM (Dimensional Fact Modeling) für das richtige Design der Datenbank verwendet. Die Aufgabe der IT ist es, das ADAPT wieder zurück auf die Multidimensionale Ansicht zurückzuführen und anschliessend das DFM für die Modellierung der Datenbank zu erstellen.

Im folgenden wird kurz auf die unterschiedlichen Modellierungsarten eingegangen. Auf eine detailliertere Beschreibung wird unter [13] und [x] verwiesen.

Datenwürfel (Cube)

Die Visualisierung von Data-Marts kann mittels Würfel (Cube) erfolgen.

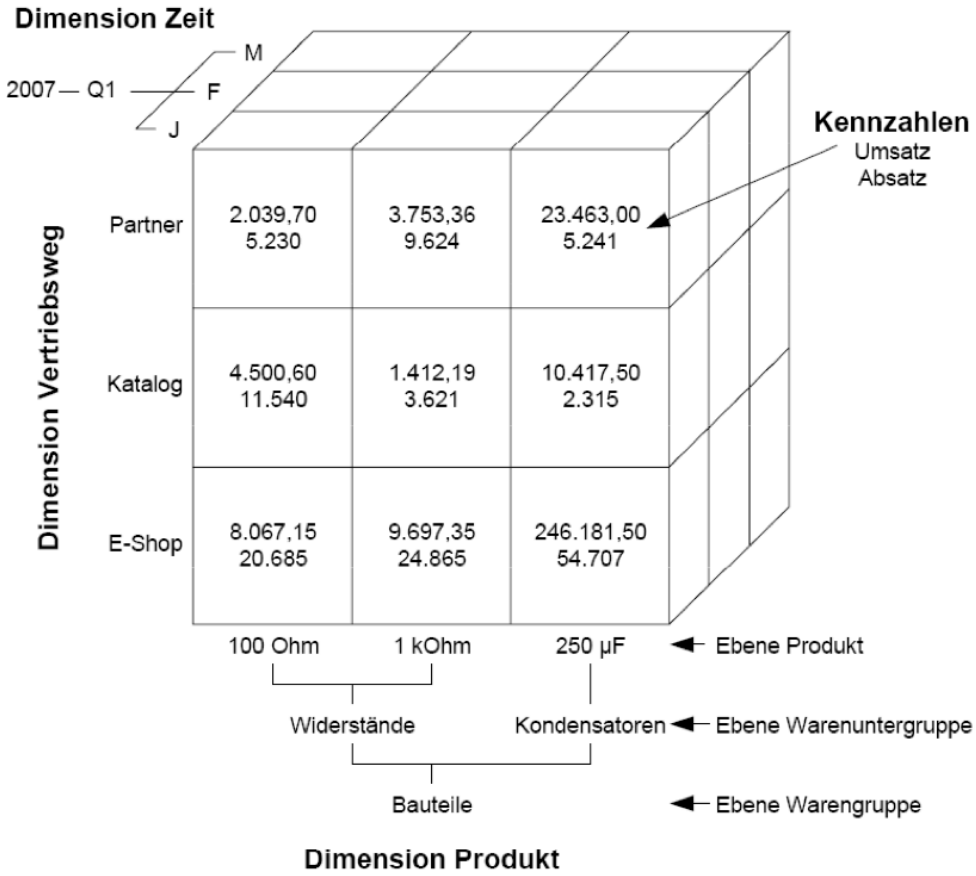


Abbildung: Cube, aus [11]

Bei zunehmender Anzahl und Ebenen der Dimensionen wird die Ansicht der Cubes sehr kompliziert. In dieser Ansicht können Operationen durchgeführt werden, die man sich räumlich darstellen kann. Allerdings besteht die Gefahr, im beruflichen Umfeld auf Personen zu treffen, die mit dieser räumlichen Sicht Schwierigkeiten haben. Bei den Operationen handelt es sich um herausschneiden, rotieren, Teilung und um Drill-Down handeln.

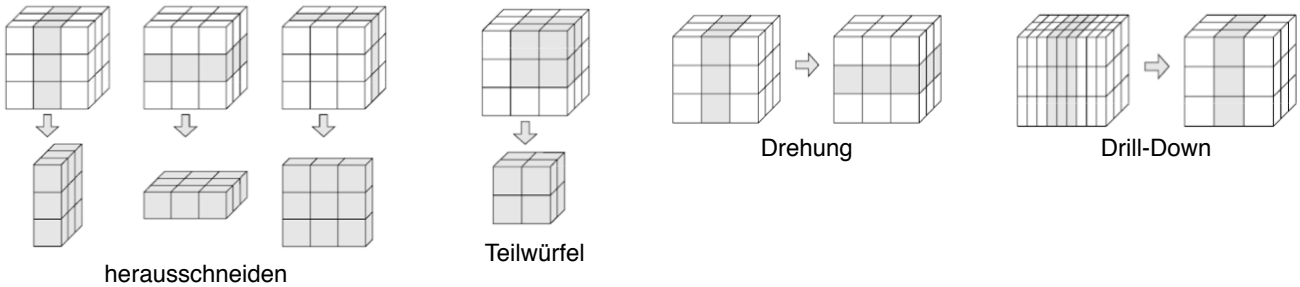


Abbildung: Cube-Operationen, aus [11]

ADAPT (Application Design for Analytical Techniques)

Der Vollständigkeit halber seien ADAPT Modelle, die gerne von Analysten verwendet werden, beschrieben. Die Aufgabe des Entwicklers ist es, das Model auf einen Würfel im Data-Mart zurückzuführen und daraus das Konzept mittels Dimensional Fact Model zu entwerfen.

Um den Rahmen für dieses Dokument und den Vortrag nicht zu sprengen, sei auf [13] verwiesen. Das downloadbare Dokument gibt eine gute Übersicht über die Adapt Modellierung.

Grundlegende Modellierungselemente

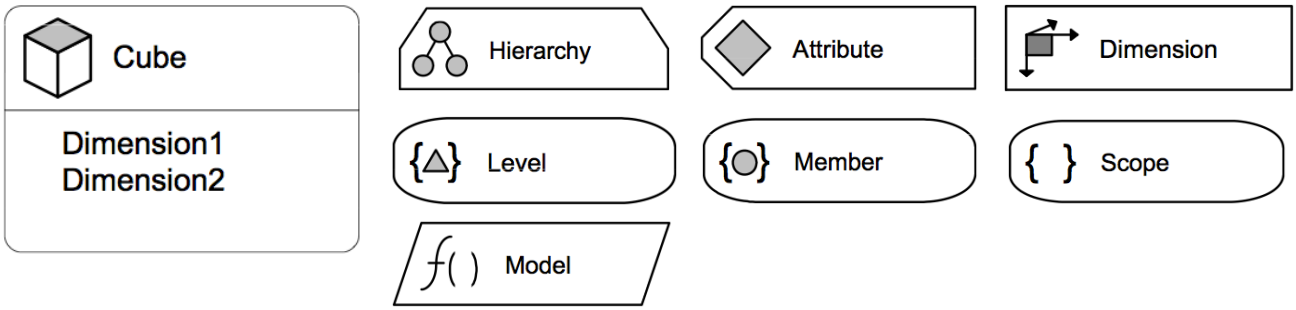


Abbildung Grundlegende Modellierungselemente, aus [11]

Verbindungselemente

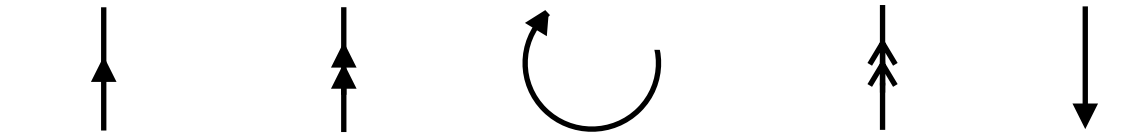


Abbildung: Verbindungselemente, aus [11]

Operatoren für Dimensionsausschnitte

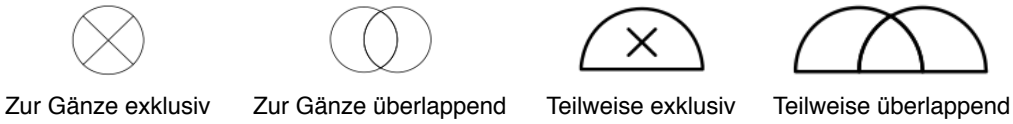


Abbildung: Operatoren für Dimensionsausschnitte, aus [11]

Beispiel

Der Übersicht halber wird zuerst die Fact-Tabelle mit den zugehörigen Dimensionen modelliert.

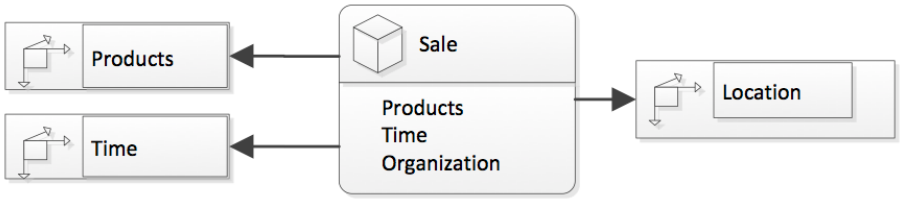


Abbildung: Beispiel Fact Tabelle modelliert mit Adapt

Erst im nächsten Schritt werden die Dimensionen modelliert.

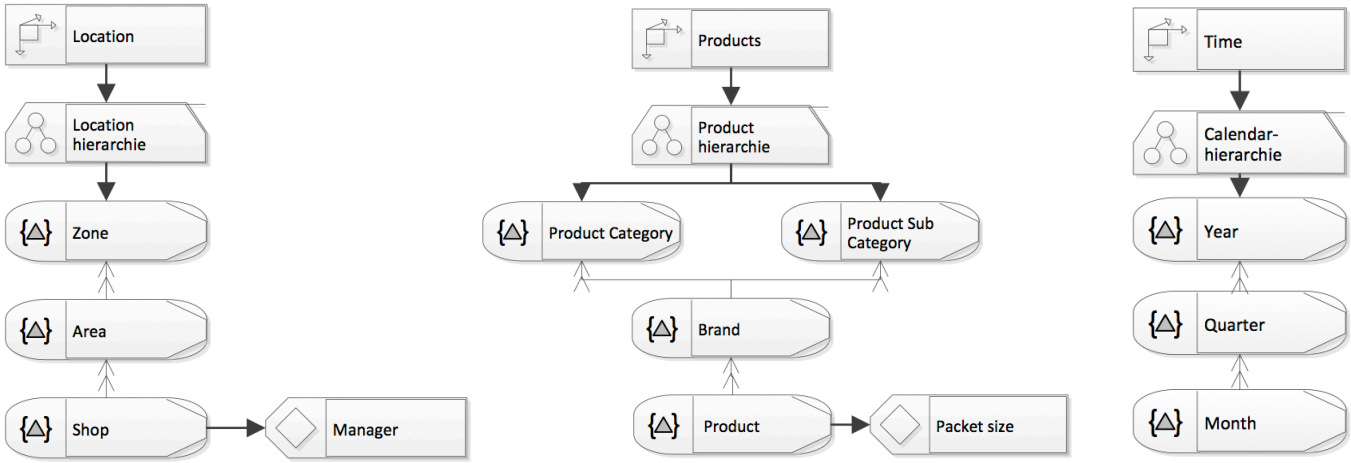


Abbildung: Dimensionen modelliert mit Adapt

Dimensional Fact Modeling (DFM)

Das Dimensional Fact Model wurde 1998 von Golfarelli, Maio and Rizzi entwickelt und beschreibt das DWH als ein in einer Baumansicht strukturiertes Modell, welches die multidimensionale Ansicht (Data-Marts) beschreibt. Dabei wird die Fact-Tabelle zentriert als Stamm eingezeichnet. Sie beinhaltet deren Namen und die zugehörigen Attribute. Die abzweigenden Äste stellen die einzelnen Dimensionen dar, wobei die Summe der Dimension nicht angezeigt wird (im Beispiel nur ein „*Item*“ statt „*Items*“).

Anhand des folgenden Beispiels* sei die Modellierung erklärt:

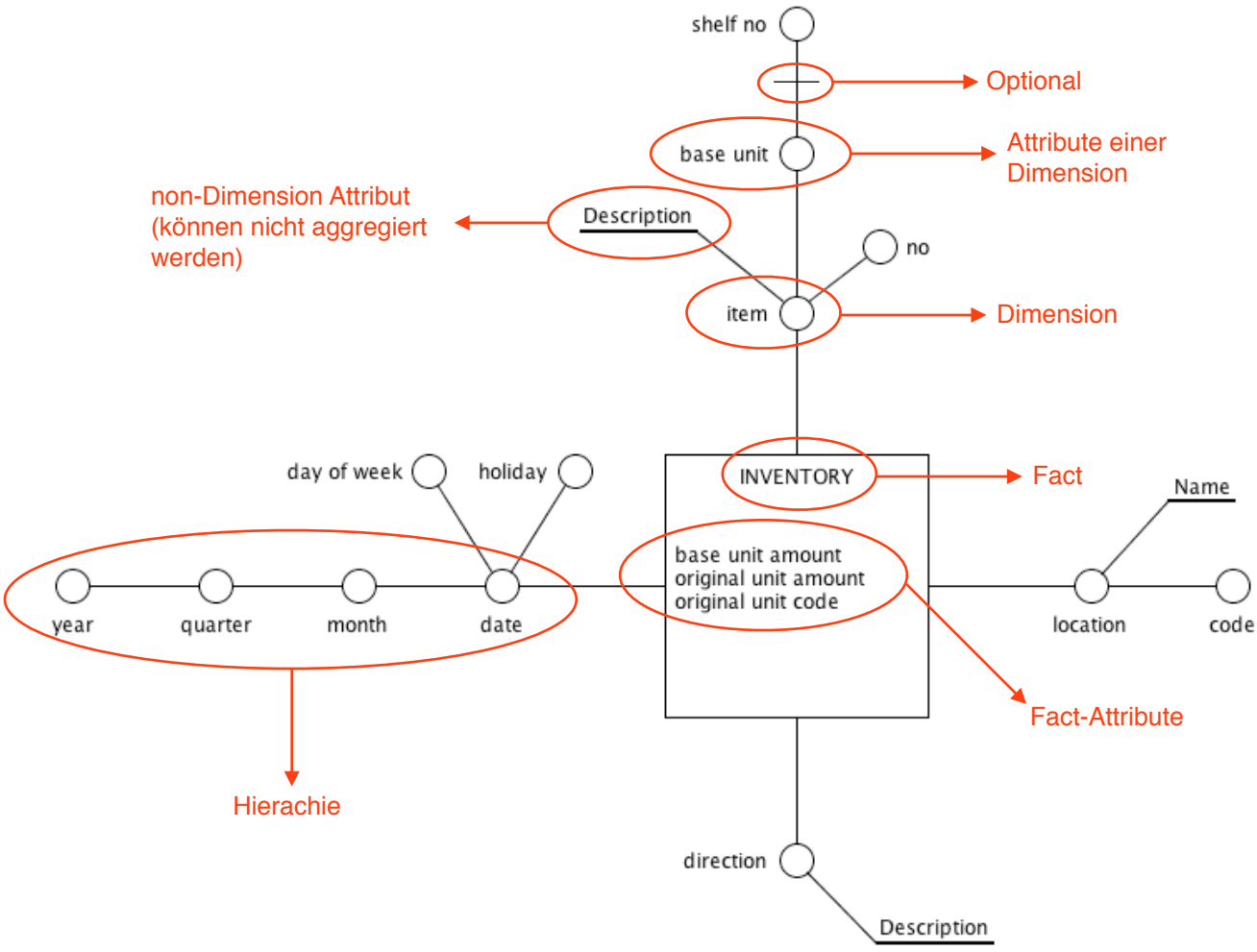


Abbildung: Dimensional Fact Modeling

dieses DFM dient im Anschluss für eine Beispiel-Implementierung

Der Entwickler führt das DFM über in ein Star-Schema, das dem Aufbau der Datenbank entspricht.

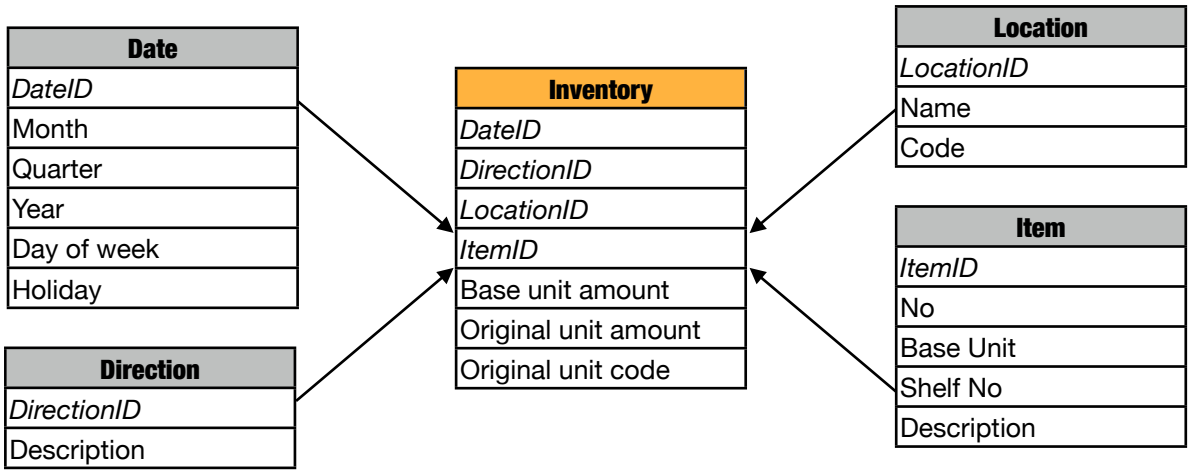


Abbildung: Star-Schema zur vorherigen Abbildung

Das DFM bietet eine datenbankunabhängige, konzeptionelle Betrachtungsweise zwischen Endanwender und Entwickler. Der Nachteil ist die mangelhafte Beschreibung von Aggregationen. Außerdem kann das Diagramm bei wachsender Anzahl an Dimensionen unübersichtlich werden.

Ein Vorteil ist jedoch die Einfachheit der Symbolik, die auch von Mitarbeitern verstanden wird, die mit dem Thema Data Warehouse bislang nicht in Berührung gekommen sind.

Der ETL-Prozess

Nach dem Design des Data Warehouses ist nun das Befüllen mit den Daten an der Reihe. Man bezeichnet diesen Prozess als ETL-Prozess. Die Abkürzung bezeichnet die drei Hauptschritte beim Befüllen des DWH mit Daten:

Extract	Extrahieren der benötigten Daten aus den Source-Systemen
Transform	Transformieren der Daten um die Qualität der Daten sicherzustellen.
Load	Laden der Daten in das DWH

Kimbal [9] vergleicht in seinem Data Warehouse Toolkit das Data Warehouse mit einem Restaurant. Im „*Back Room*“, also in der Küche, wird das Essen zubereitet und kein Gast hat Zutritt zu diesem Bereich. Im „*Front Room*“ sitzen die Gäste und genießen ihre Mahlzeit.

Der ETL-Prozess beschreibt, wie das Essen serviert wird. Er bringt das Essen zu den Gäste,n vergleichbar mit einem Kellner.

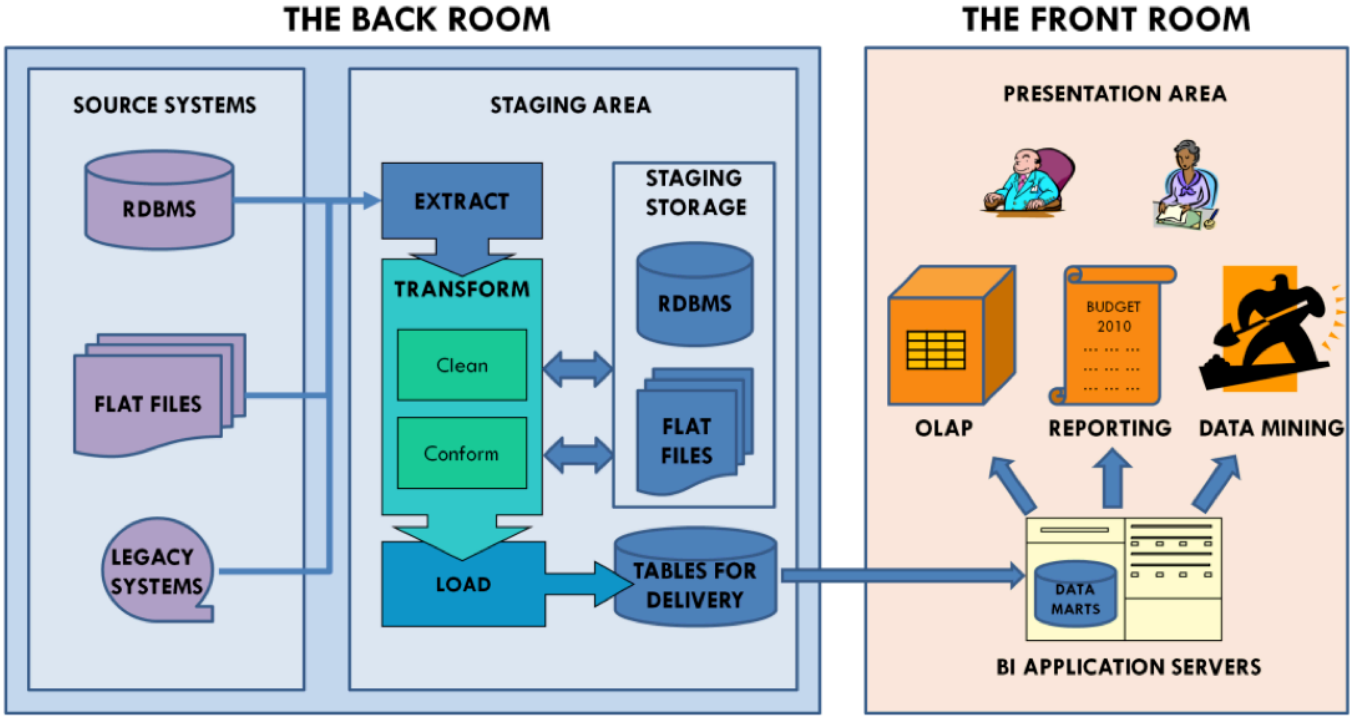


Abbildung: ETL-Prozess, aus [10]

Staging?

Staging bedeutet, die Werte schrittweise und nachvollziehbar zu übernehmen und immer wieder Tabellen der einzelnen Schritte zu erstellen. Warum sollte Staging verwendet werden?

Wiederherstellbarkeit

Der ETL Prozess kann jederzeit fehlschlagen. Je mehr Staging Tabellen es gibt, um so mehr Recovery-Punkte gibt es. Werden Tabellen im ETL-Prozess mehrfach benötigt, so werden die Source-Systeme, von denen die Daten kommen, nicht nochmals belastet, da auf die Tabellen im Staging Bereich zugegriffen werden kann.

Sicherung

Daten, die fertig für die Auslieferung sind, können komprimiert und als Backup gespeichert werden.

Auditing

Staging erlaubt es die Daten zu überprüfen, wenn es zu um die Frage der Datenqualität, beziehungsweise um die Fehlersuche geht.

Extract

Der Extract-Prozess ist der heikelste und aufwändigste Prozess. In der Praxis dient nicht nur eine relationale Datenbank als Quelle für das DWH. Daten kommen aus den unterschiedlichsten Quellen und in unterschiedlichen Formaten. Manche Systeme sind alt, keine aktuellen Treiber mehr verfügbar und es müssen einige Hürden für die Abfrage der Daten bewältigt werden.

Der Extract-Prozess erfolgt nach folgendem Schema:

1. Data Map erstellen
2. Zu den Quellen verbinden
3. Aktualisierungszeitraum festlegen (täglich, wöchentlich)
4. Änderungen ermitteln (neue Daten, Änderungen, gelöschte Daten)
5. Staging der Daten

Logical Data Map (LDM)

Wichtig im Extract-Prozess ist es, die Herkunft der Daten zu beschreiben. Speziell bei der Fehlersuche oder der nachträglichen Erweiterung ist die LDM von immenser Bedeutung.

Wichtig: Das Logical Data Map wird erstellt, **bevor** mit der Implementierung des Data Warehouses begonnen wird!

Wird während der Implementierung festgestellt, dass die Herkunft der Werte nicht korrekt ist, so muss auch das LDM geändert werden und immer am aktuellen Stand bleiben. Man bedenke, schlimmer als gar keine Dokumentation ist eine falsche Dokumentation. Auch wenn dieser Schritt viel Arbeit bedeutet, so kann er im späteren Verlauf sehr viel mehr Arbeit ersparen. Ebenso hilft der Plan anderen Mitarbeitern, die sich selbstständig in das Projekt einarbeiten müssen, als Nachschlagewerk.

Vorlage eines Logical Data Map

Target					Source				Transformation
Table Name	Column Name	Data Type	Table Type	SCD Type	Database Name	Table Name	Column Name	Data Type	

Abbildung: Vorlage Logical Data Map

Die Herkunft der Daten wird in den Source-Bereich der Tabelle eingetragen. Der Target-Bereich gibt an, wohin die Daten gespeichert werden. Hier wird auch der Type der Slowly Changing Dimension (SCD) des Feldes eingetragen.

Die Spalte „Transformation“ beinhaltet Informationen, inwiefern die Daten transformiert werden. Im Anhang A befindet sich ein Beispiel eines LDMs, in dem Daten aus der Microsoft Navision Demo-Datenbank für den Inventurprozess extrahiert wurden.

Änderungen erfassen

Ziel jeder Methode ist es, nur die Datensätze, die sich seit der letzten Beladung geändert haben, für die neuerliche Beladung zu erfassen. Dadurch werden weniger Daten übertragen und die Source-Systeme nicht unnötig belastet. Grundsätzlich gibt es zwei verschiedene Methoden, um Änderungen zu ermitteln.

Methode A: Verwenden von Timestamps

Wenn zu jedem Datensatz ein Timestamp der letzten Beladung gespeichert wurde, so können über diesen Änderungen nachvollzogen werden. Natürlich funktioniert dies nur, wenn auch im Source-System ein Timestamp zu jedem Datensatz vorhanden ist. In ERP Anwendungen ist dies oft der Fall.

Methode B: Aktuelle Daten mit letzter Ladung vergleichen

Etwas aufwendiger wird es, wenn die aktuellen Daten mit der letzten Beladung verglichen werden müssen, um Änderungen zu erfassen. Verschiedene Tools stellen bereits eine diesbezügliche Funktion zur Verfügung (Microsoft SSIS wird im Anschluss an dieses Kapitel vorgestellt).

Transform

Der Transform Prozess besteht aus der Bereinigung und der Anpassung von Daten. Dazu sind die Qualität und Überprüfung der Daten wichtig.

Qualität der Daten

Daten in einem Data Warehouse müssen eine definierte Qualität haben. Es erschwert die Arbeit, wenn beispielsweise ein Datumsfeld unterschiedliches Format hat oder reine Zahlenwerte in Textfelder gespeichert werden.

Datenqualität bezieht sich auf korrekte, unmissverständliche, gültig, konsistente und komplette Daten. Je nach Fehler in den Daten kann man vier Bereiche behandeln, wo Daten korrigiert werden können.

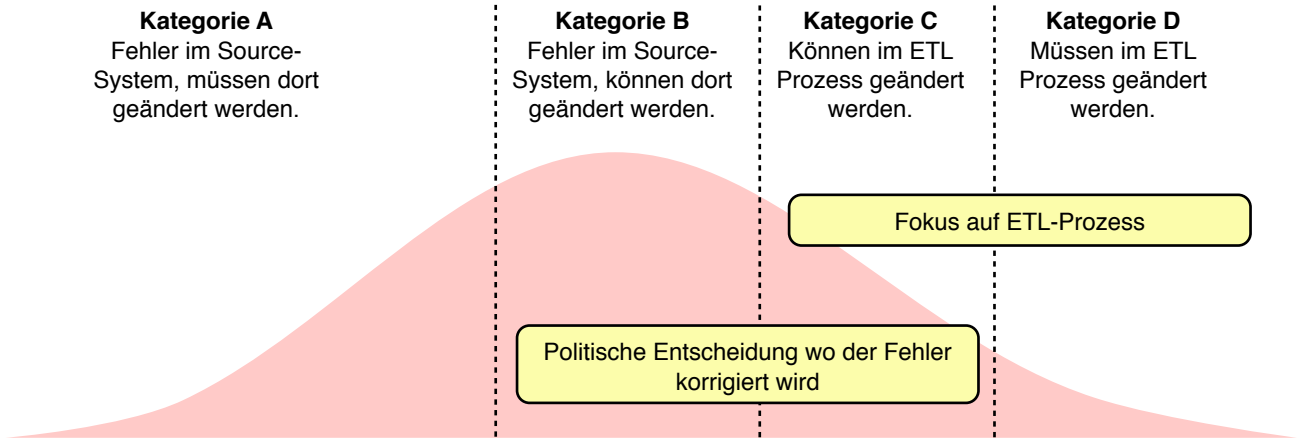


Abbildung: Einteilung der Fehler. Adaptiert aus [9]

Wie die Einteilung zeigt, müssen einige Fehler im Source System, andere wiederum im ETL Prozess korrigiert werden. Bei den Kategorien B und C muss eine Entscheidung getroffen werden, wo diese Daten korrigiert werden sollen.

Kategorie A

Daten müssen im Source System geändert werden. Es handelt sich zum Beispiel um fehlende Daten. Im DWH müssen diese Daten als unvollständig markiert werden.

Kategorie B

Daten, die im Source System fehlerhaft sind, aber im ETL Prozess korrigiert werden könnten. Zum Beispiel der Rückschluss von der Postleitzahl auf den Ort.

Kategorie C

Fehler, die beim ETL Prozess auftreten, und auf das Source System zurückzuführen sind. Das können beispielsweise fehlerhafte Datumswerte sein.

Kategorie D

Löst ein Problem, das im Source-System nicht korrigiert werden kann. Zum Beispiel Anpassung der Daten aus verschiedenen Systemen.

Cleaning Deliverables

Hauptaugenmerk im Cleaning-Teil des Transform-Schrittes wird auf die Bereinigung und Korrektur von fehlenden und falschen Daten gelegt.

Screens

Mit sogenannten Screens werden Daten auf Fehler überprüft. Es geht darum, Listen von Werten zu erstellen und solche Werte herauszufiltern, die sich nicht an bestimmte Regeln halten. Häufig handelt es sich dabei um Tippfehler, wie das Vergessen eines Kommas bei einem Preis oder dadurch entstandene unterschiedliche Namensbezeichnungen.

Das folgende Beispiel zeigt, dass in einer Liste die Stadt München zwar vorhanden ist, aber ein ähnlicher Beitrag mit einem Tippfehler vorhanden ist. Hier handelt es sich um einen Klassischen Kategorie B Fehler. Er kann (bzw. sollte) nicht im ETL Prozess behoben werden, sondern muss im Source-System ausgebessert werden.

```
select [City], Count(*) from ... group by [City] order by Count(*)
```

City	Count(*)
München	124245
Münhen	2

Auch können Geschäftsbedingungen überprüft werden. Angenommen der Status „Key Customer“ darf nur ab einem gewissen Mindestumsatz vergeben werden, so kann über eine einfache Abfrage fehlerhafte Zuweisungen ermittelt werden.

```
select Name, Status, sum(Turnover) from Sales ...
having (sum(Turnover) < 150000) and (Status = „Key Customer“)
```

Es ist auch wichtig diesen Screening Prozess zu dokumentieren. Welche Daten wurden geprüft? Worauf wurde geprüft?

Diese Informationen können über eine Multidimension Tabelle in das DWH integriert werden. Über Audit-Dimensions können die Ergebnisse auch an den Anwender weitergegeben werden. Diese können dann selbstständig die Qualität der Daten überprüfen und gegebenenfalls weitere Schritte zu deren Verbesserung unternehmen.

Back Room

Bezugnehmend auf das Restaurant-Beispiel könnte der interne Aufbau folgendermaßen implementiert werden:

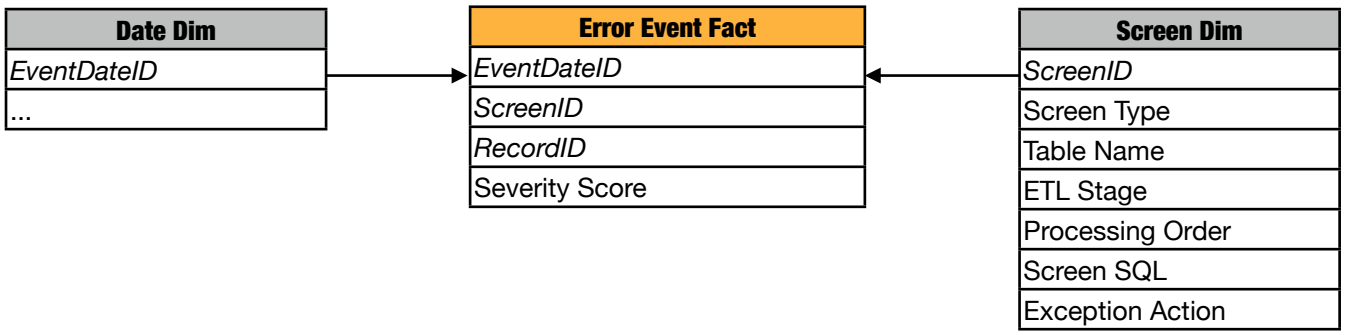


Abbildung: Back Room

Front Room

Wie zuvor erwähnt sind Informationen im Front Room auch für Endanwender ersichtlich. Für diese werden sogenannte Audit Dimensions erstellt, die einfacher und übersichtlicher sind. Die blau markierten Spalten sind nur befüllt, wenn Fehler aufgetreten sind.

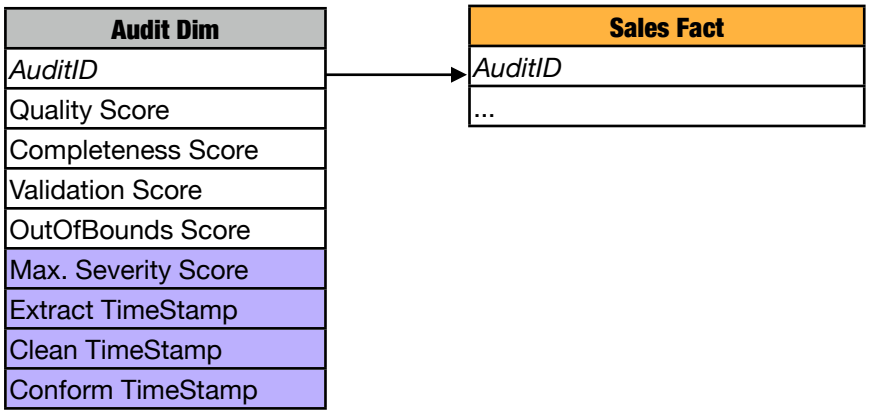


Abbildung: Audit Dimensions, Front Room

Conforming Data

Ziel im Conforming-Teil des Transform-Schrittes ist es inkonsistente Daten zu entfernen oder aufzulösen. Dieser Vorgang kann speziell bei redundanten Daten sehr problematisch und kompliziert sein.

- Vereinheitlichen von Bezeichnungen („Januar“ - „Jänner“)
- Redundante Daten aus einer Dimension entfernen („M. Müller“ - „Max Müller“)
- Bei den Fact- und Dimension-Tabellen an das Logical Data Map halten
- Auseinandergehende Bezeichnungen durch festgelegte Regeln zusammenführen (Artikelbezeichnung in Verkauf und Produktion)

Load

Der Load-Schritt ist im Gegensatz zu den vorherigen Schritten relativ einfach. Es befasst sich damit, die Daten in die vorbereiteten Dimension- und Fact-Tabellen zu kopieren. Bei der Beladung der Dimension-Tabellen müssen dabei die zuvor definierten Slowly Changing Dimension Methoden beachtet werden. Fact-Tabellen müssen beladen und die vorhandenen Business-Keys (eindeutige IDs der Source Systeme) werden durch eigene, eindeutige Schlüssel (Surrogate Keys) ersetzt.

Beide Funktionen werden bei der Verwendung von verschiedenen Tools unterstützt. Im Kapitel „Enwicklung“ wird eines dieser Tools vorgestellt.

Dimension: Neue Werte

Um einen neuen Wert in eine Dimension-Tabelle einzutragen muss ein neuer Surrogate Key gefunden werden. Normalerweise sind diese Schlüssel durch Autoincrement festgelegt und man muss sich nicht weiter darum kümmern.

Der Business Key wird dabei zu einem herkömmlichen Datensatz in der Dimension Tabelle.

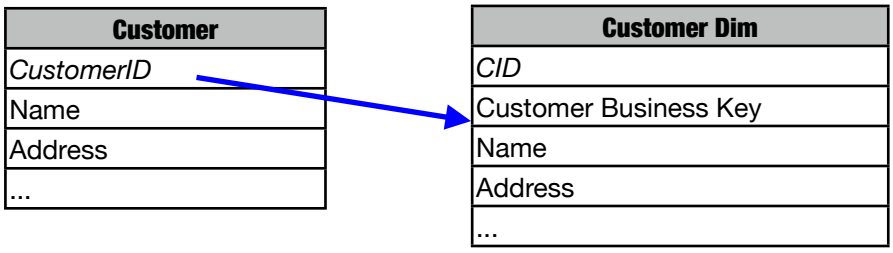


Abbildung: Primary Key der Source Tabelle wird zu Business Key

Dimension: Aktualisierte Werte

Anhand des Business Keys der aktuellen Dimension-Tabelle wird auf aktualisierte/geänderte Werte überprüft.

Für SCD Type III wird der Wert, der ersetzt werden soll, in die „old“ Spalte kopiert.
Für SCD Type I und Type III wird der aktuelle Wert in die Spalte geschrieben und ersetzt somit den alten Wert.

Für SCD Type II wird ein neuer Datensatz mit dem Änderungsdatum gespeichert. Wenn vorhanden, so kann auch ein Historical-Flag gesetzt beim alten Eintrag gespeichert werden. Alternativ kann auch ein Zeitbereich für die Gültigkeit des Wertes gesetzt werden (von-bis).

Dimension: Gelöschte Werte

Im Normalfall werden gelöschte Werte im DWH beibehalten, denn sie sind nicht falsch. Wird zum Beispiel ein Aussendienstmitarbeiter im Source System gelöscht, so ist er zwar dort nicht mehr vorhanden, seine Daten müssen aber im DWH erhalten bleiben.

Um wirklich falsche Werte zu erfassen, sollte periodisch auf verwaiste Werte geprüft werden.

Fact: Zeilen vorbereiten

Nun benötigen wir die zuvor festgelegte Art der unterschiedlichen Fact Tabellen auch eine besondere Vorbereitung der Zeilen.

Transactional Facts

Für jede Transaktion wir eine neue Zeile erstellt.

Snapshot Facts

Für jeden Moment, der erfasst werden soll, wird eine neue Zeile erstellt. Auch wenn der Moment leer ist und nichts beinhaltet, wird der Wert 0 gespeichert.

Accumulated Snapshot Tabellen

Diese Art der Fact-Tabellen ist die komplizierteste. Wenn der Prozess beginnt, müssen die Datensätze hinzugefügt werden. Sobald der Prozess in seiner Bearbeitung voranschreitet, müssen die Werte (Status, Zeitschlüssel) aktualisiert werden.

Im letzten Schritt werden degenerierte Fact-Tabellen erstellt. Wenn Preise auf Währungen abzielen, dann werden zuletzt auch die Umrechnungsfaktoren eingefügt, oder der Verweis auf eine Umrechnungstabelle hinterlegt.

Fact: Schlüssel ersetzen

Nun ist der ETL-Prozess fast fertig. Es müssen nur noch die Business Keys durch die Surrogate Keys ersetzt werden. Dies geschieht durch ein Lookup auf die Business Keys in den Tabellen. Dieser Vorgang kann sehr viele Ressourcen benötigen. Es ist daher darauf zu achten, dass der Cache des Systems ausreichend dimensioniert ist.

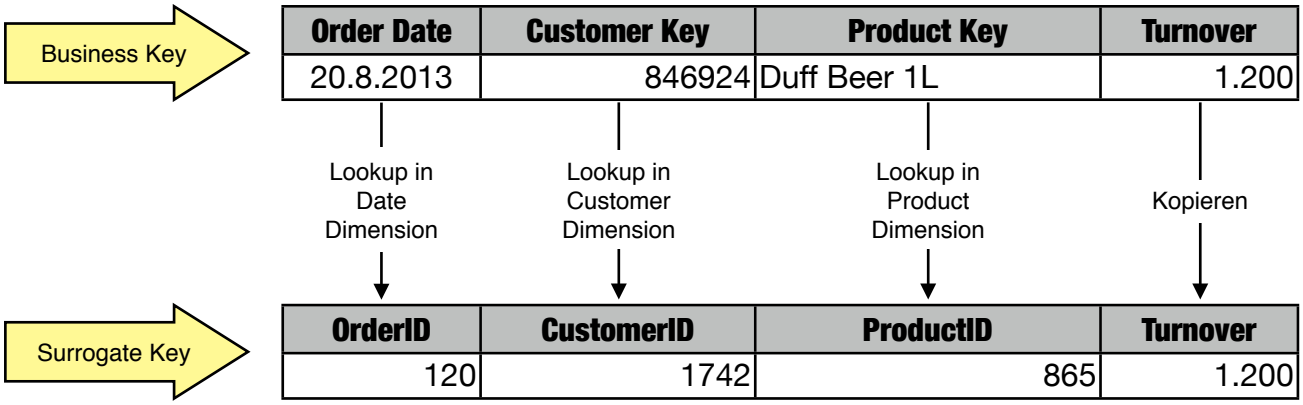


Abbildung: Fact-Tabelle Schlüssel ersetzen

Entwicklung

Für das Beladen eines Data Warehouses kann natürlich auch Delphi verwendet werden. Das Ergebnis ist aber hauptsächlich eine Aneinanderreihung von SQL-Befehlen und daher nicht sonderlich übersichtlich. Es lohnt sich, über den Tellerrand zu sehen und zum Befüllen ein anderes Tool zu verwenden. Folgende Hersteller bieten diesbezügliche Produkte an:

Oracle mit dem Warehouse Builder, SAP mit dem Data Integrator, IBM mit Decision Stream, Microsoft mit Integration Service und die Open Source Lösung Pentaho mit Kettle [15].

Im folgenden wird die Entwicklung anhand der Integration Service von Microsoft (SSIS) vorgestellt. Diese Lösung verwendet die IDE von Visual Studio, benötigt daher dessen Installation und Lizenzierung.

Installation

Datenbank

Als Datenbank wird Microsoft SQL Server 2012 verwendet.
Die Gratis Express Version kann unter folgendem Link geladen werden:
http://www.microsoft.com/de-de/download/details.aspx?id=29062

Entwicklungsumgebung

Als Entwicklungsumgebung für das Data Warehouse kommt Visual Studio 2012 zum Einsatz. Es gibt verschiedene Versionen, die je nach Bedarf installiert werden kann. Auf dem Testsystem befindet sich die Ultimate Version.
http://www.microsoft.com/visualstudio/deu/products/compare

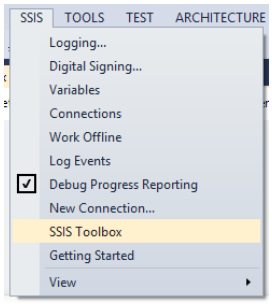
Visual Studio benötigt noch das Business Intelligence Paket, um die notwendigen Funktionen bereit zu stellen
http://www.microsoft.com/de-de/download/details.aspx?id=36843

Da es sich im die Delphi Tage handelt, soll Delphi nicht zu kurz kommen. Wer möchte, kann auch mit Delphi das Beladen des Data Warehouses realisieren. Da es sich aber zumeist nur um eine aneinandergereihte Ansammlung von SQL Befehlen handelt und vor allem für den Extract Prozess verschiedene Treiber zu Datenbanken, sowie möglichen Dateien (Excel, XML, CSV, usw.) benötigt werden, ist der SSIS die bessere Wahl.

Bei der Auswertung und der Statistik kann Delphi seine ganze Stärke ausspielen.
http://www.embarcadero.com/de/products/delphi

Komponenten

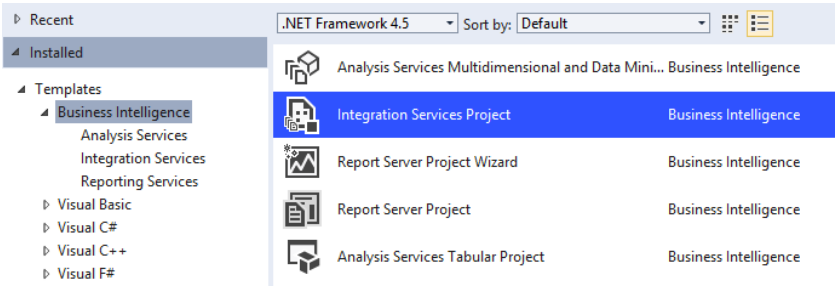
Vergleichbar mit der Komponentenliste in Delphi finden sich die Komponenten in der SSIS Toolbox. Wird die Toolbox nicht angezeigt, so kann diese in Visual Studio im Menü „SSIS“ unter „SSIS Toolbox“ aktiviert werden.



Die wichtigsten Komponenten aus dieser Toolbox werden im folgenden kurz erklärt und sollen einen Überblick über die Funktionsweise geben. Für detaillierte Beschreibungen sei auf [14] verwiesen.

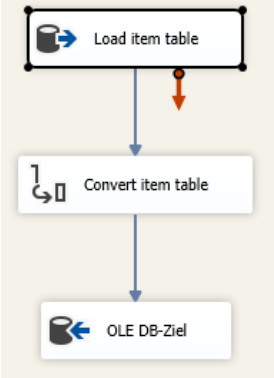
Projekt erstellen

Um ein neues Projekt für ein Data Warehouse zu erstellen muss das passende Projekt erstellt werden. Unter File/Project öffnet sich der Menüpunkt zum Erstellen von Projekten. In diesem Dialog sucht man nach „Business Intelligence“ und wählt „Integration Services Project“ aus.



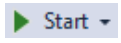
Funktionsweise

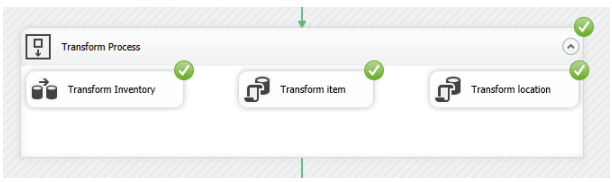
Mittels Drag&Drop werden die Komponenten auf die Arbeitsfläche geschoben. Durch Pfeile an den Komponenten können die Ausgänge mit Eingängen anderer Komponenten zugewiesen werden. Dabei stehen blaue Pfeile für den normalen Datenfluss und rote Pfeile für einen Datenfluss, der im Fehlerfall ausgeführt wird.



Um die Funktionsweise einzelner Komponenten zu testen, kann mit der rechten Maustaste über der jeweiligen Komponente mit „Execute Task“ der betreffende Teil ausgeführt werden. Sobald sich eine Komponente in einem Container befindet, ist diese Option jedoch nicht mehr vorhanden. Es kann nur der gesamte Container ausgeführt werden.

Starten

Wie in Delphi kann über den Play Button () die Anwendung gestartet werden. Die Abarbeitung erfolgt gemäß der Reihenfolge der verbundenen Datenflüsse.



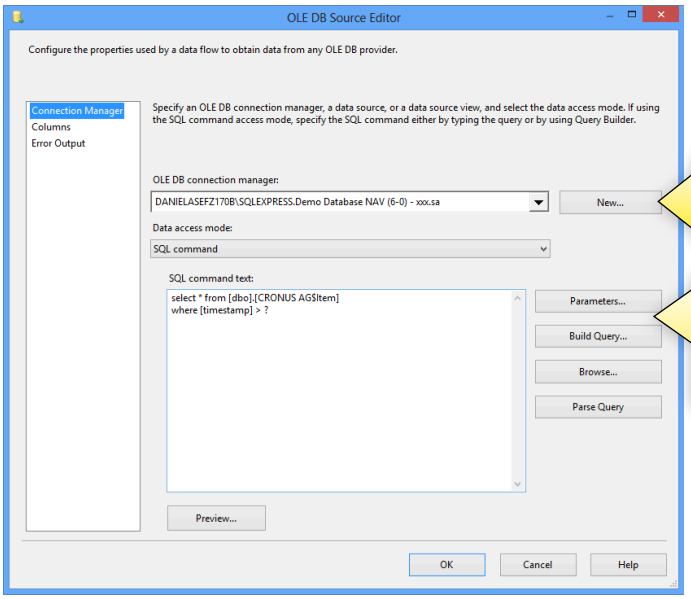
Datenflusstask

Der Datenflusstask stellt einen Bereich dar, mit dem ein bestimmter Datenfluss festgelegt werden kann. Das kann das Importieren einer Datei sein oder aber die Verbindung mit einer anderen Datenbank. Die Daten können Konvertiert und anschliessend wieder in eine Datei oder Datenbank ausgegeben werden.

Sobald man in den Datenflusstask schaltet, gibt es eine Reihe neuer Symbole in der SSIS Toolbox:

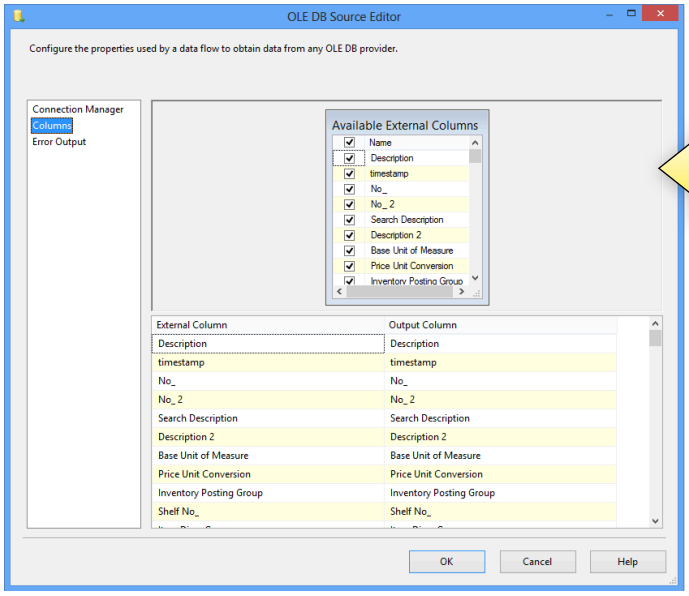
Source

Für das Einlesen von Daten aus einer Datenbank gibt es verschiedene Source-Komponenten. Das kann zum Beispiel ADO NET oder OLE DB sein.



Verbindung von der die Daten gelesen werden sollen

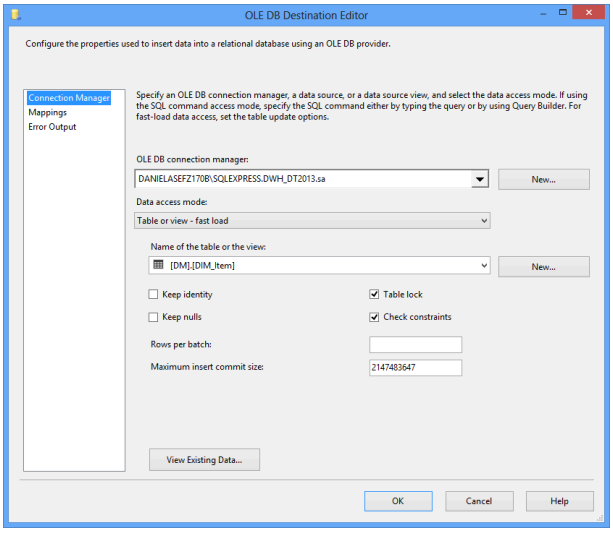
Es kann angegeben werden, ob die gesamte Tabelle ausgewählt werden soll oder ob über einen Befehl bestimmte Daten eingelesen werden sollen.



Auf der Seite „Columns“ können die Spalten ausgewählt werden, die benötigt werden

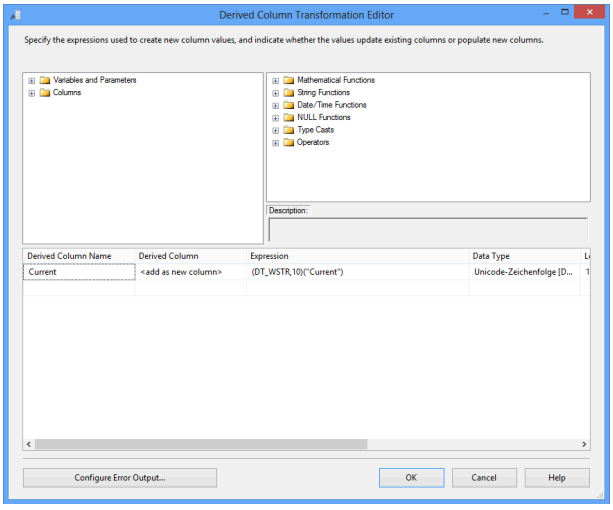
Destination

Die Komponente speichert Daten in eine Tabelle. Ja nach Datenbank gibt es hier verschiedene Destination-Komponenten. So gibt es zum Beispiel eine Komponente für ADO NET und eine für OLE DB. Das Mapping der Felder des eingehenden Datenflusses und der Felder der Datenbank erfolgt auf der Seite „*Mappings*“



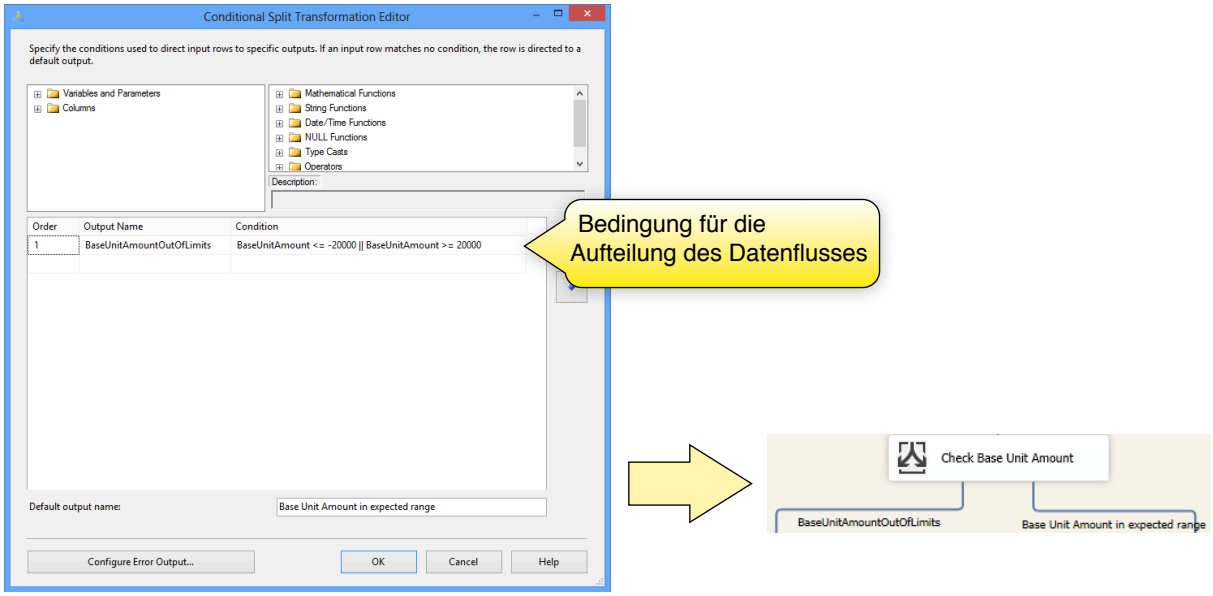
Abgeleitete Spalten

Wenn eine Spalte von einer anderen Spalte abgeleitet wird, so kann auch der Wert der Tabelle ersetzt werden. Ebenso kann ebenso eine neue Spalte hinzugefügt werden.



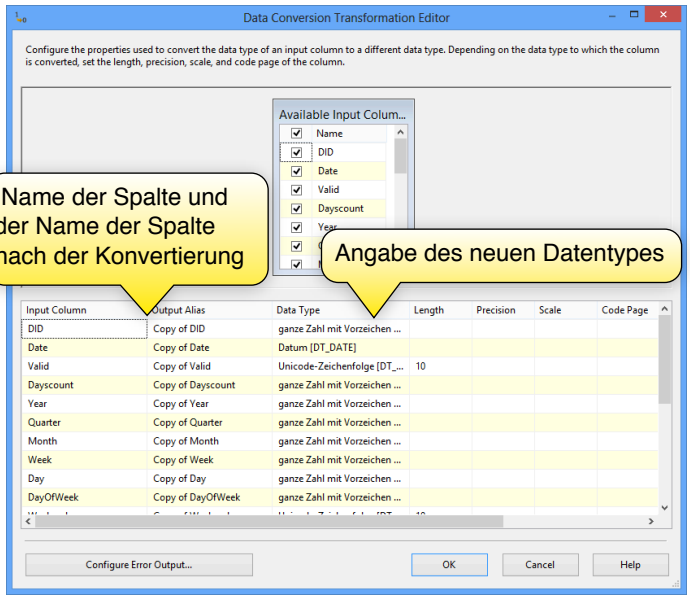
Bedingtes Teilen

Der Datenfluss kann bei bestimmten Ereignissen aufgeteilt werden. Der „*Default output name*“ gibt den Ausgang an, der Daten beinhaltet, wenn keine der Bedingungen auftritt. Bei der jeweiligen Bedingung wird ebenfalls ein „*Output name*“ angegeben. Die Komponente erstellt für jede Bedingung einen Ausgang, der dann weiter verarbeitet werden kann.



Datenkonvertierung

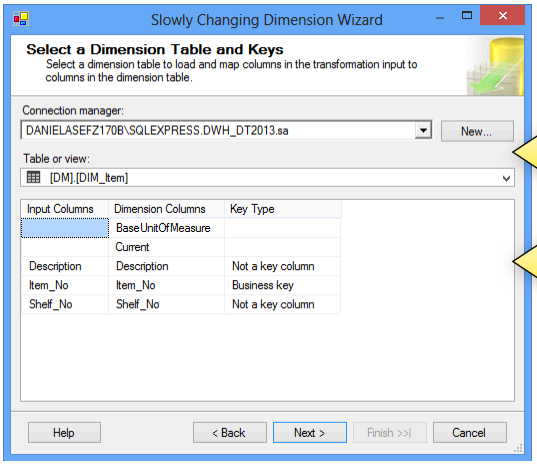
Wenn Daten aus verschiedenen System geladen werden, so ist zumeist auch das Format der einzelnen Felder unterschiedlich. Man denke zum Beispiel an Datumsfelder. Hier gibt es unterschiedliche Typen, die von Datenbank zu Datenbank und von System zu System verschieden sind. Um die Datentypen zu vereinheitlichen, kann die Komponente „*Datenkonvertierung*“ verwendet werden.



Langsam veränderliche Dimensionen

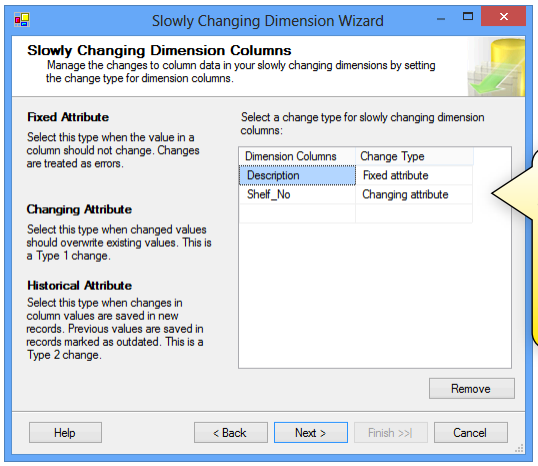
SSIS stellt Funktionen für die Implementierung von Slowly Changing Dimensions für Type I und Type II zur Verfügung. Es entfällt also eine umständliche Implementierung mittels SQL Befehlen.

Das Menü stellt einen Assistenten dar, der durch die Konfiguration folgt.



Verbindungsparameter und Tabelle, auf die die SCD angewandt werden.

Es muss ein Business-Key festgelegt werden

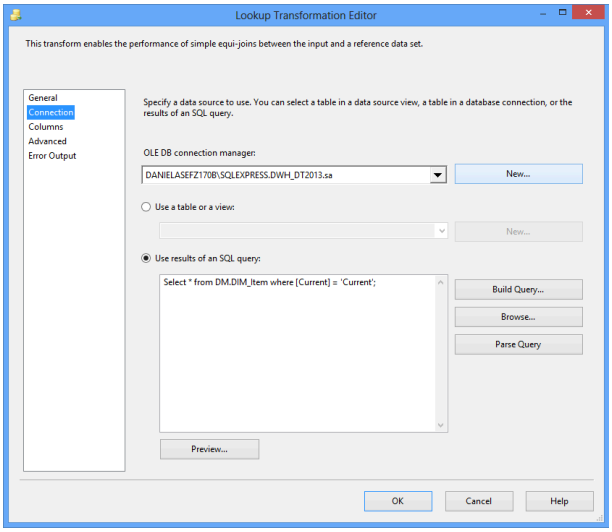


Im nächsten Schritt wird angegeben, ob es sich um eine SCD handelt und wenn ja, um welchen Type.
Dabei steht „Fixed Attribute“ für ein normales Feld, „Changing Attribute“ repräsentiert SCD Type I und „Historical Attribute“ SCD II

Die weiteren Seiten in diesem Menü bieten zusätzliche, selbsterklärende Optionen für die SCD. Die Ausgabemöglichkeiten ändern sich je nach verwendeter SCD Typen. Bei „*Neue Ausgabe*“ werden neue Werte in die Datenbank geschrieben. Bei „*Ausgabe der Updates abgeleiteter Elemente*“ erfolgt eine Aktualisierung bereits vorhandener Datensätze. Die Funktionen werden vom SSIS automatisch angelegt.

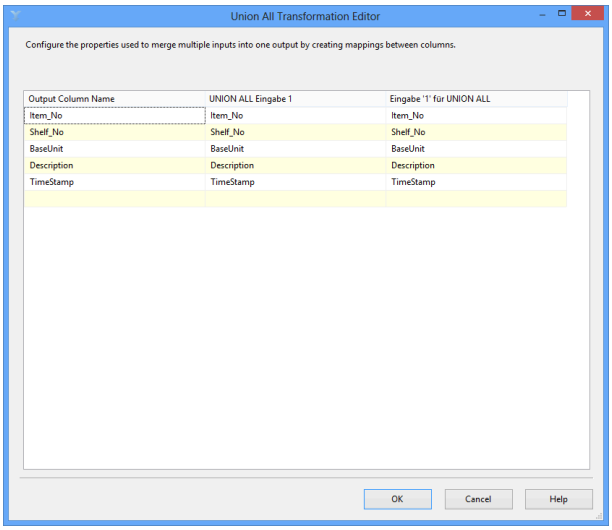
Suche

Natürlich ist es auch möglich, nach bestimmten Datensätzen zu suchen. Unter „*General*“ kann ein Cache Modus gewählt werden. Mittels „*Connection*“ wird der Verbindungsparameter und ggf. ein SQL Befehl für die Suche hinterlegt. Auf der Seite „*Columns*“ kann man nun die Spalten wählen, die nach der Suche verwendet werden sollen.



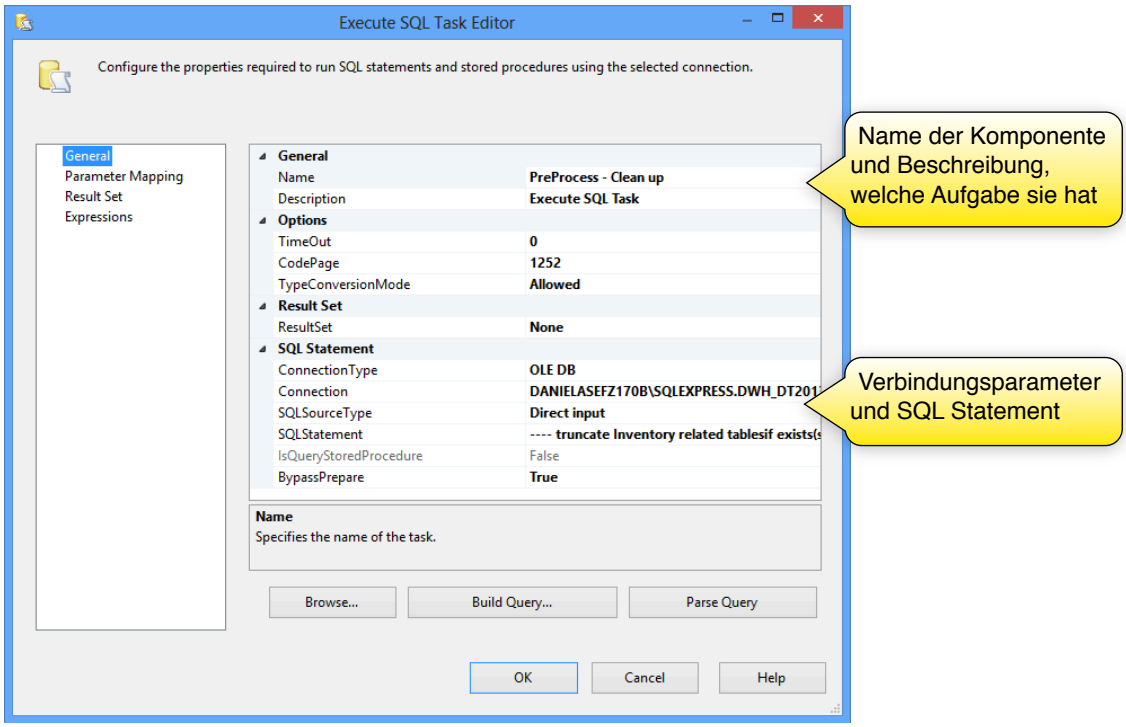
Union All

Die Union All Komponente wird verwendet, um zwei Datenflüsse wieder miteinander zu vereinen. Dabei werden die Spaltennamen der Eingangstabellen angezeigt. Die Feldtypen der Tabellen müssen natürlich zueinander passen. Die Spaltennamen der Ausgangstabellen können, wenn notwendig, ebenfalls angepasst werden.



Execute SQL Task

Der Datenflusstask kann ein oder mehrere SQL Befehlssequenzen speichern und ausführen.

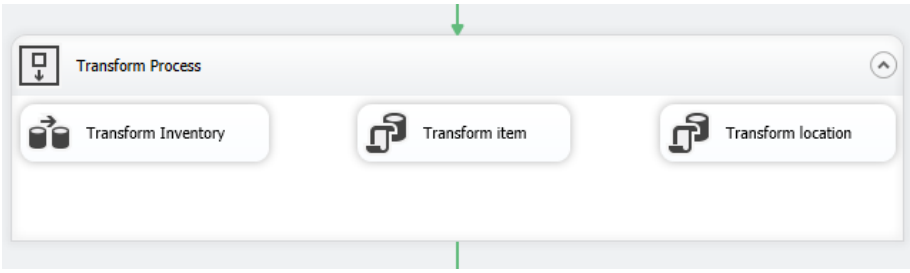


Sequence Container

Der Sequence Container ist ein Sammelcontainer, in dem weitere Komponenten platziert werden können. Möchte man zum Testen einen Task ausführen, so lässt sich hierbei aber nur noch der gesamte Container und nicht eine einzige Komponente ausführen.

Es bietet sich an die Komponenten der einzelnen Stufen des ETL-Prozesses in jeweils einen eigenen Container abzulegen.

Die Komponente schafft nicht nur Übersichtlichkeit, vielmehr werden auch alle Tasks darin parallel gestartet und abgearbeitet.



Beispiel-Implementierung

Nachdem nun die einzelnen Komponenten und deren Funktionsweise beschrieben wurde, kann die Anwendung anhand eines Beispiels näher gebracht werden. Aufgrund des Umfanges werden nur Teile eines Projektes beschrieben, es stellt aber kein komplettes lauffähiges Programm bereit. Das Beispiel wird als Gesamtlösung zum Download bereitgestellt, welche auch kompiliert und gestartet werden kann. Wir beschränken uns auf die Erläuterung der Tabellen für „*Inventory*“ und „*Item*“. Alle anderen Tabellen werden in ähnlicher Art und Weise abgehandelt.

Vorwort

Für diese Anwendung bedienen wir uns der Demo-Datenbank aus Microsofts Navision. Es sollen Informationen zu dem Inventar aus Navision für eine Analyse erfasst werden.

Es gibt in der DWH Datenbank zwei verschiedene Schemas. STA für die Staging Area und DM für das Datenmodell. Auf das STA Schema haben nur die Entwickler Zugang. Endbenutzer können auf das DM Schema zugreifen. Es beinhaltet das fertige Datenmodell.

Initialisierung

Als ersten Schritt empfiehlt es sich, einen SQL Task einzuplanen, mit dessen Hilfe der Staging Bereich aufgeräumt wird. Für unser Beispiel nehmen wir an, wir haben im Staging Bereich eine Tabelle „*Inventory*“, die Daten zum Inventar beinhaltet, und eine „*Inventory_transformed*“ Tabelle, die Daten aus „*Inventory*“ transformiert und vorbereitet. Diese beiden Tabellen sollen zu Beginn geleert werden. Die gleiche Struktur hat auch die Tabelle, die den Artikelstammbeinhaltet, die „*Item*“ Tabelle.

```
---- truncate Inventory related tables
if exists(select * from sys.tables t inner join sys.schemas s on t.schema_id=s.schema_id where
s.name='STA' and t.name='Inventory')
 truncate table STA.Inventory;
if exists(select * from sys.tables t inner join sys.schemas s on t.schema_id=s.schema_id where
s.name='STA' and t.name='Inventory_transformed')
 truncate table STA.Inventory_transformed;
```

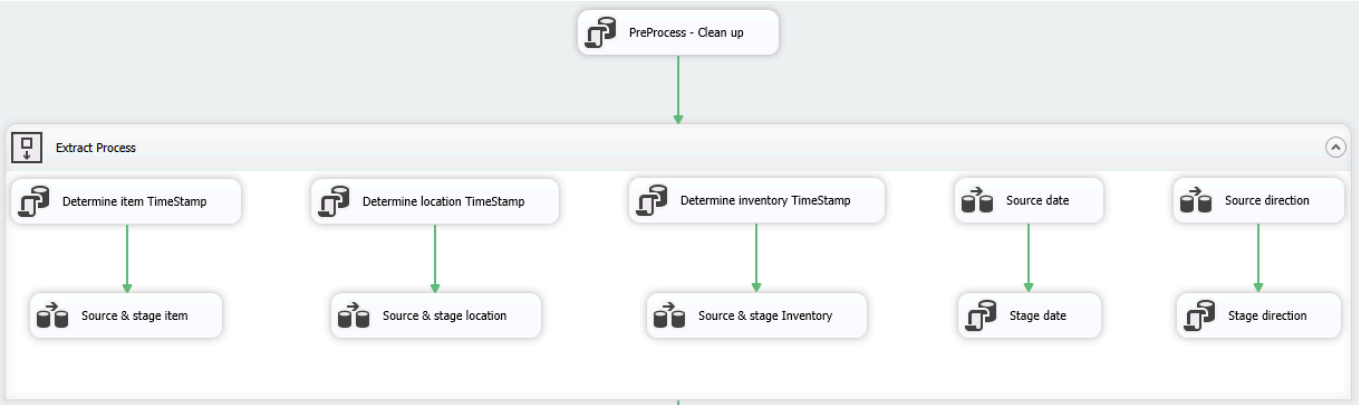
Selbiges gilt für die „*Item*“ Tabelle.

```
---- truncate Item related tables
if exists(select * from sys.tables t inner join sys.schemas s on t.schema_id=s.schema_id where
s.name='STA' and t.name='Item')
 truncate table STA.Item;
if exists(select * from sys.tables t inner join sys.schemas s on t.schema_id=s.schema_id where
s.name='STA' and t.name='Item_transformed')
 truncate table STA.Item_transformed;
```

Nachdem die jeweiligen Tabellen der Datenbank vorbereitet wurden, kann mit der Implementierung des ETL-Prozesses begonnen werden.

Extract

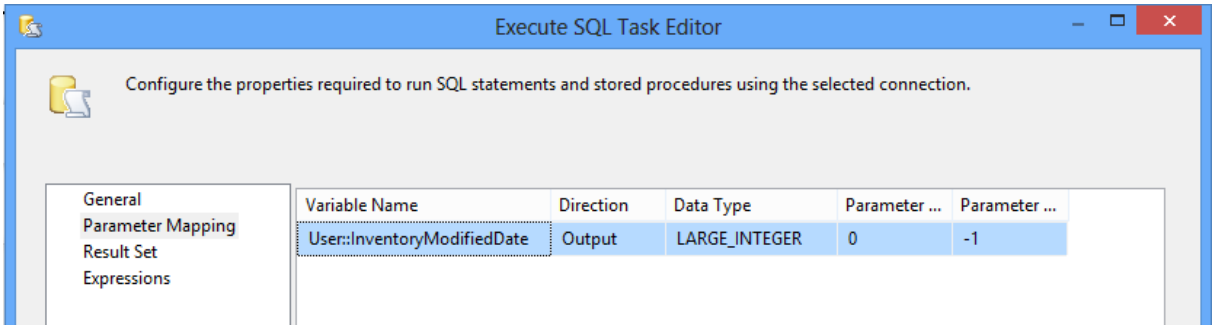
Den Extract Prozess packen wir in einen Container. Den Ausgang unseres „*PreProcess - Clean up*“ Tasks verbinden wir mit dem Eingang des Containers. Der Container wird also direkt nach dem ersten Task ausgeführt. Sobald die Anwendung zur Ausführung des Containers kommt, werden alle darin befindlichen Tasks gleichzeitig ausgeführt.



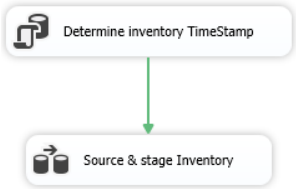
Für die „*Inventory*“ Tabelle wollen wir nur Änderungen seit der letzten Befüllung abfragen. Navision stellt hier glücklicherweise einen Timestamp zu jedem Datensatz bereit. Es gilt also nur den Timestamp der letzten Beladung zu ermitteln und für die Abfrage zu verwenden. Wir fügen daher einen SQL Task hinzu und benennen ihn „*Determine inventory TimeStamp*“. In dem SQL Statement fragen wir den letzten Timestamp ab. Dafür haben wir im Staging Bereich eine Tabelle „OLD_Inventory“ vorgesehen, die den Inhalt der letzten Beladung speichert.

```
if exists(select * from sys.tables t inner join sys.schemas s on t.schema_id=s.schema_id where
s.name='STA' and t.name='OLD_Inventory') begin
 declare @max_TS timestamp;
 select @max_TS = MAX([TimeStamp]) from STA.OLD_Inventory;
 set ? = @max_TS
 end;
```

Mit dem ? setzen wir einen Platzhalten für einen Parameter. Auf der Seite „*Parameter Mapping*“ kann diesem Platzhalter nun ein Name zugewiesen werden. Wir nennen ihn „*InventoryModifiedDate*“, der Typ ist „*Output*“ und „*LARGE_INTEGER*“. Wird der Inhalt des SQL Statements ausgeführt, dann wird nun dieser Parameter den höchsten Timestamp der letzten Beladung beinhalten.



Im Datenfluss geht es nun weiter und wir fügen einen Datenfluss-Task danach ein. Wir benennen diesen Datenflusstask mit „*Source & Stage Inventory*“ und verbinden den Ausgang der vorherigen Abfrage mit dessen Eingang.

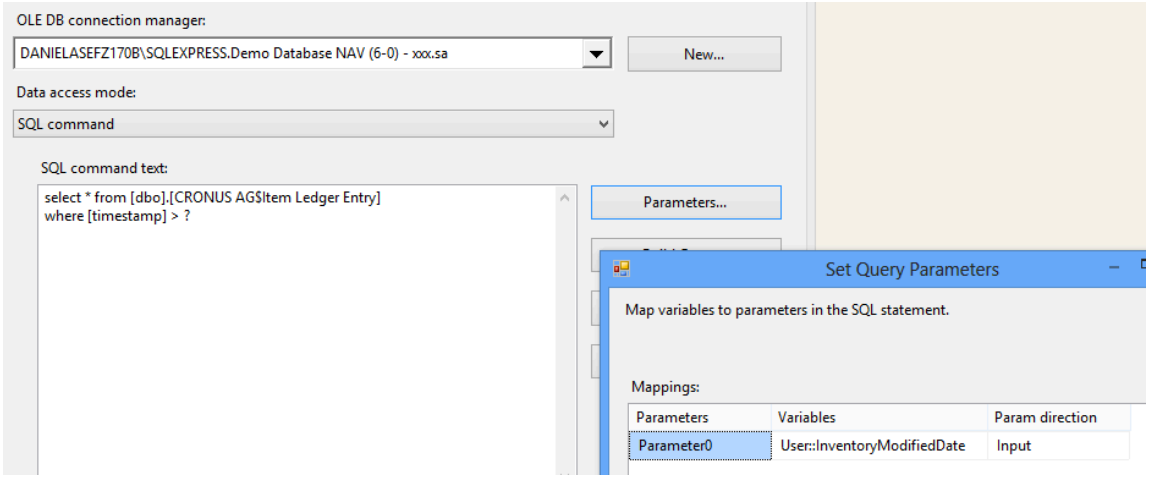


Durch einen Doppelklick auf den Task wird die Arbeitsfläche des neuen Datenflusses geöffnet.

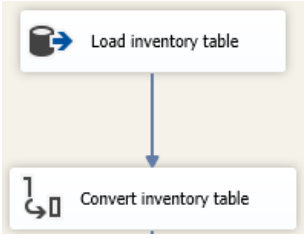
Wir müssen nun aus der Navision Datenbank die Datensätze filtern, die seit der letzten Beladung geändert wurden. Hierfür verwenden wir den zuvor ermittelten Timestamp und fügen den Befehl als SQL Statement ein.

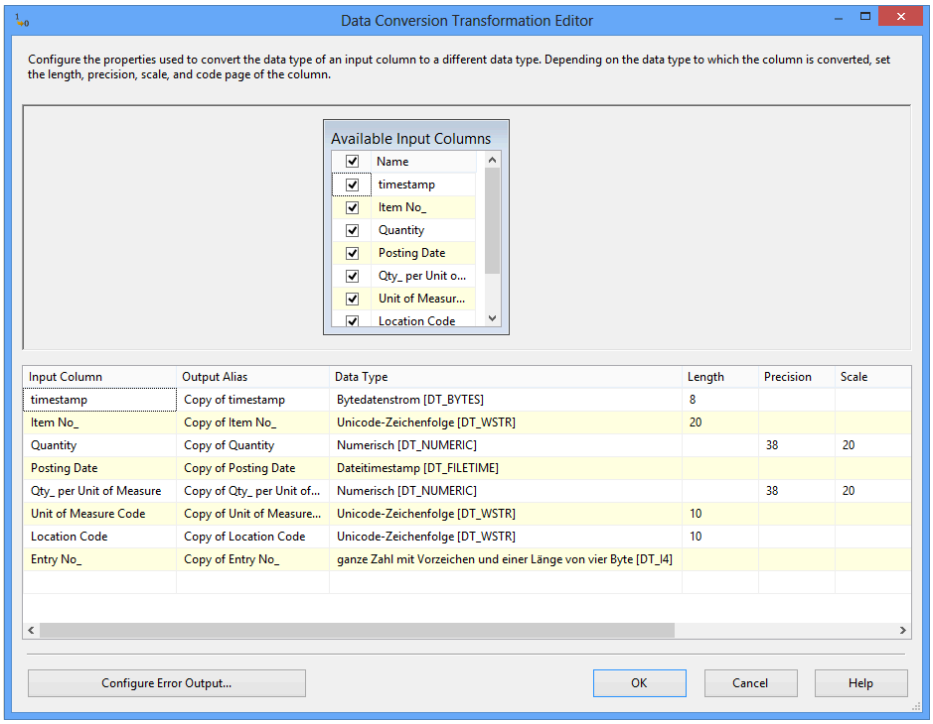
```
select * from [dbo].[CRONUS AG$Item Ledger Entry] where [timestamp] > ?
```

Dem Platzhalter ? wird nun der Parameter zugewiesen. Hierzu klickt man rechts auf „Parameter“ und gibt als ersten Parameter (Parameter0) die zuvor erstellte Variable „*InventoryModifiedDate*“ als Input-Parameter an.

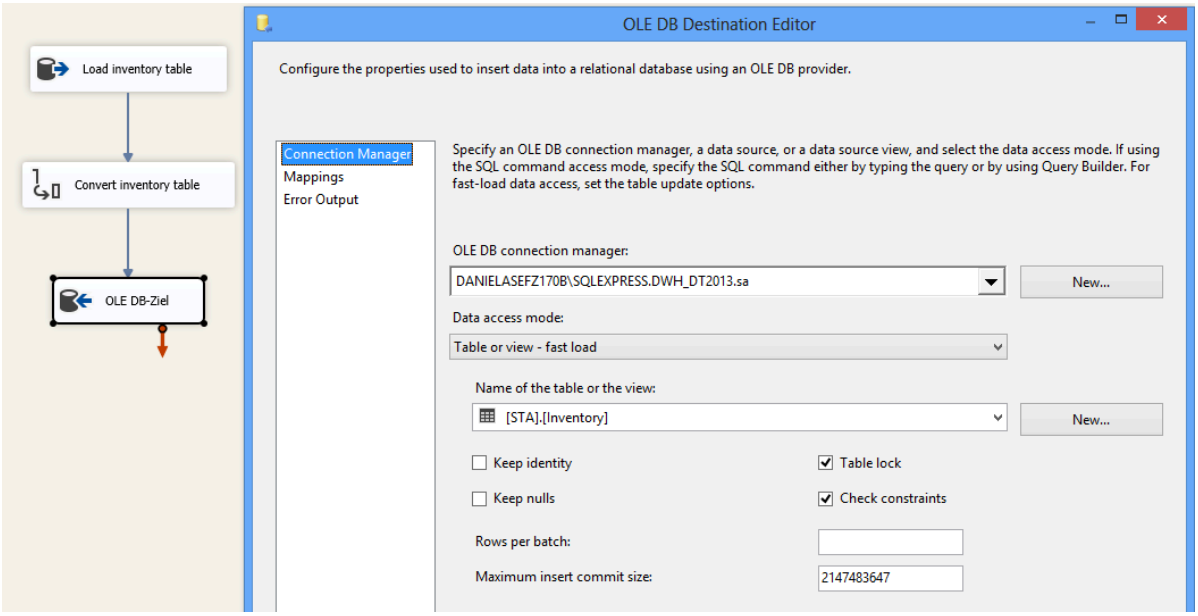


Am Ende des Task bekommen wir nun eine Liste mit den aktuellen Änderungen. In der Regel stimmen die Formate der Felder nicht mit denen der Zielfelder überein. Daher fügen wir eine Datenkonvertierungskompenente hinzu, benennen sie „*Convert inventory table*“ und verbinden deren Eingang mit dem Ausgang des vorherigen Tasks. Durch einen Doppelklick auf die Komponente wird ein Menü zur Konvertierung der Datenfelder geöffnet.





Das Ergebnis speichern wir nun unter „*STA.Inventory*“ ab. Dies kann durch eine entsprechende „*Destination*“ Komponente erfolgen.

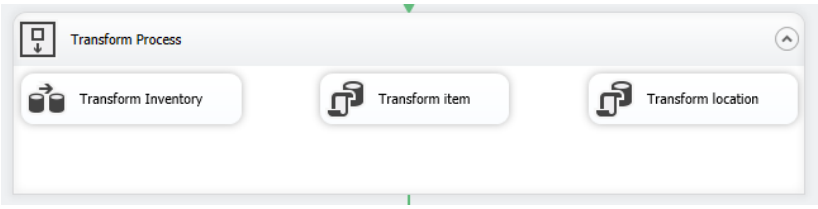


Die Invertar-Daten haben wir nun erfolgreich im Extract-Prozess behandelt.

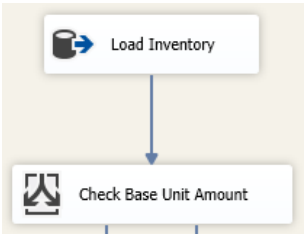
Transform

Nun geht es daran, die Inventur-Daten zu transformieren. Wir beschränken uns darauf zu überprüfen, ob die Daten „*BaseUnitAmount*“ in einem gültigen Bereich liegen oder nicht, d.h. wir erstellen eine Audit-Datei, in die Datensätze gespeichert werden, die möglicherweise nicht richtig sind und vom Endanwender überprüft werden sollen.

Wiederum erstellen wir einen Container für den Transform Prozess und fügen einen Datenfluss-Task mit der Bezeichnung „*Transform Inventory*“ hinzu. Den Eingang des Containers verbinden wir mit dem Ausgang des „*Extract*“ Containers.



In „*Transform Inventory*“ laden wir nun den Inhalt der zuvor erstellten Tabelle „*STA.Inventory*“ mit Hilfe einer Source Komponente. Nun möchten wir den Datenfluss in richtige und möglicherweise falsche Werte aufteilen. Dazu fügen wir die Komponente „*Bedingtes Teilen*“ hinzu und benennen diese „*Check Base Unit Amount*“.

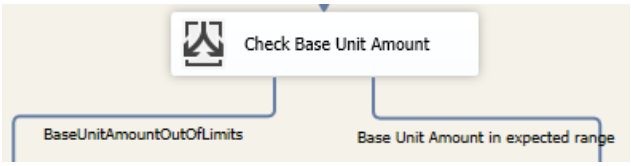


Durch einen Doppelklick öffnet sich wiederum ein Menü, indem wir unsere Parameter angeben können.

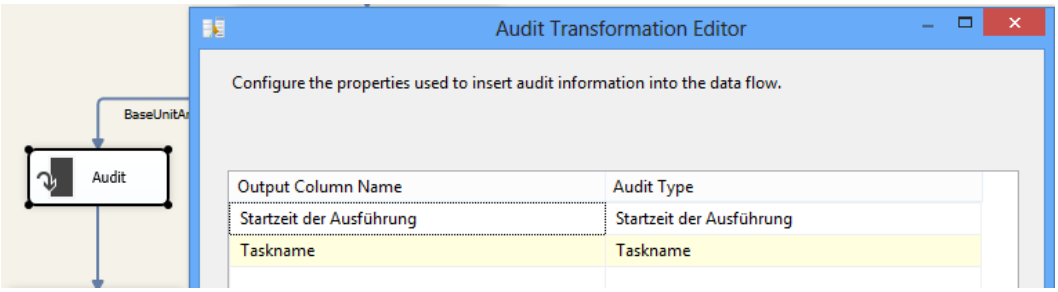
Wir nehmen an, dass „*BaseUnitAmount*“-Werte größer als 20.000 und kleiner als -20.000 als ungültig erkannt werden sollen. (Hinweis: Navision beinhaltet hier positive als auch negative Werte für Ein- und Ausgänge) Wir geben die Bedingung an und benennen einen Ausgangsnamen. Dieser Ausgangsname wird als Pfeil, der den zugehörigen Datenfluss bezeichnet, am Ausgang der Komponente angezeigt.

Order	Output Name	Condition
1	BaseUnitAmountOutOfLimits	BaseUnitAmount <= -20000 || BaseUnitAmount >= 20000

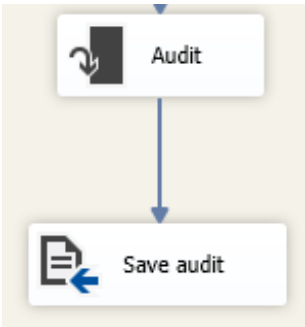
Jetzt müssen wir auch noch einen Standard-Ausgang benennen, der den Datenfluss bezeichnet, der alle Inhalte auf dessen die Bedingung nicht zutrifft, beinhaltet.



Werte, die als falsch erkannt wurden, sollen nun in eine Datei gespeichert werden. Hierfür wird die „*Audit*“ Komponente verwendet. Hier können wir einige Parameter zu dem Audit angeben.



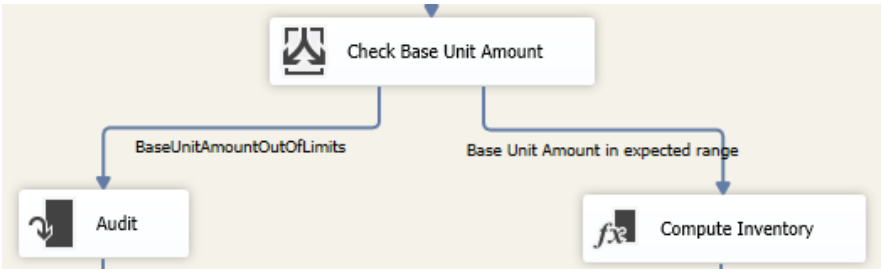
Der letzte Schritt ist es, den Inhalt des Datenflusses zu speichern. Hierzu bietet sich die Komponente „*Flatfileziel*“ an. Wir fügen sie hinzu, verbinden den Datenfluss und benennen sie als „*Save audit*“. Alternativ könnten wir die Daten auch in eine Tabelle in der Datenbank speichern.



Nach einem Doppelklick auf „*Save audit*“ öffnet sich der Connection Manager. Hier kann das Format der Datei, das Verzeichnis und der Dateiname angegeben werden.

Nun können wir uns an den Datenfluss für die korrekten Daten annehmen. Wir haben vorhin bemerkt, dass Navision für ein- und ausgehende Artikel positive und negative Zahlen verwendet. In der Fact Tabelle wäre diese Eigenschaft äußert umständlich und verwirrend. Wir ändern diese Eigenschaft also. Wir fügen ein Feld für „*IN*“ und „*OUT*“ hinzu und korrigieren den „*BaseUnitAmount*“ Wert, indem wir den Betrag davon speichern und so uns des Vorzeichen entledigen.

Dazu fügen wir eine Komponente für „*Abgeleitete Spalten*“ ein und benennen sie „*Compute Inventory*“.



Durch einen Doppelklick auf die Komponente öffnen sich wieder das Menü.

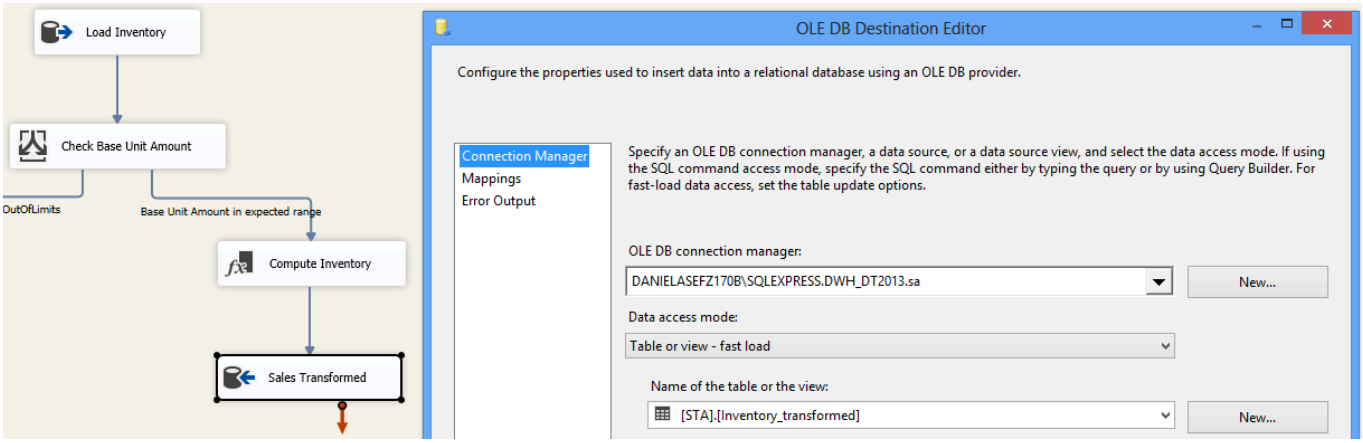
Zuerst wird der Ein- bzw. Ausgang ermittelt und ein Feld „*Direction*“ mit den Werten „*IN*“ bzw. „*OUT*“ eingefügt. Der entsprechende Ausdruck lautet *BaseUnitAmount < 0 ? „OUT“ : „IN“*.

Von „*Item_No*“ und „*LocationCode*“ entfernen wir sicherheitshalber Leerzeichen von deren Anfang und Ende und wandeln sie in Unicode-Zeichen um, da diese in Navision anders gespeichert werden.

Zuletzt wird noch der Betrag von „*BaseUnitAmount*“ erstellt und damit das Vorzeichen entfernt. Wir wissen ja nun, ob es sich um einen Ein- bzw. Ausgang handelt.

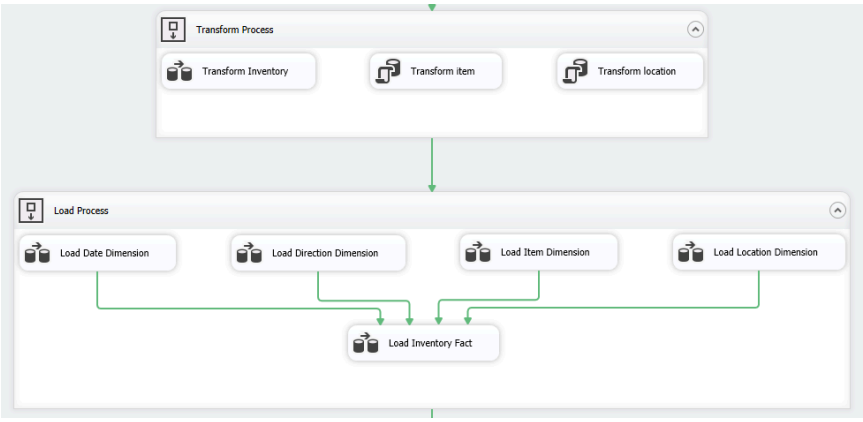
Derived Column Name	Derived Column	Expression	Data Type	Length	Precision	Scale
Direction	<add as new column>	BaseUnitAmount < 0 ? "OUT" : "IN"	Unicode-Zeichenfolge [DT_WSTR]	3		
Item_No	Replace 'Item_No'	TRIM(Item_No)	Unicode-Zeichenfolge [DT_WSTR]	20		
BaseUnitAmount	Replace 'BaseUnitAmount'	ABS(BaseUnitAmount)	Numerisch [DT_NUMERIC]		38	20
LocationCode	Replace 'LocationCode'	TRIM(LocationCode)	Unicode-Zeichenfolge [DT_WSTR]	10		

Als letzten Schritt speichern wir nun wieder den Datenfluss. Nach der Transformierung der Daten möchte wir diese in die Tabelle „*STA.Inventory_transformed*“ speichern.



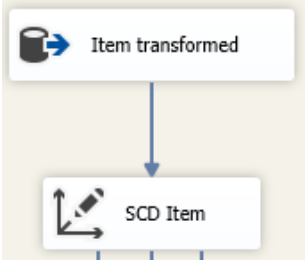
Load

Wenn alle Daten vorbereitet sind, kann die Beladung in das Datenmodell erfolgen. Wieder erstellen wir einen Container für den Load-Prozess und verbinden ihn mit dem Ausgang des „*Transform*“ Containers.



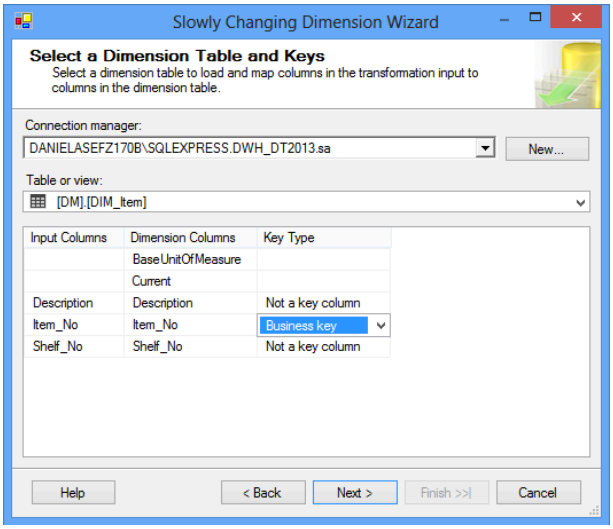
Um die Fact Tabelle zu befüllen, müssen erst alle Dimension Tabellen zur Verfügung stellen. Wir sehen uns dies am Beispiel der Artikel an. Zuvor wurde im Programm die Tabelle „*STA.Item_transformed*“ auf ähnliche Weise, wie die beschriebene „*STA.Inventory_transformed*“ vorbereitet.

Es wird wiederum eine Datenfluss Komponente verwendet. Sie wird mit „*Load Item Dimension*“ bezeichnet. Geladen wird die Tabelle „*STA.Item_transformed*“.

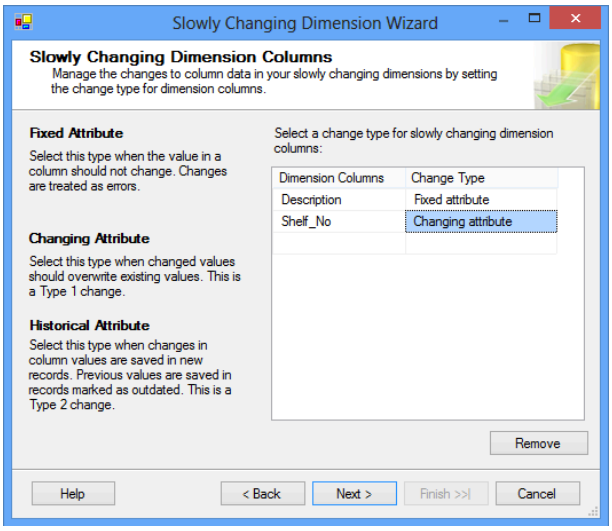


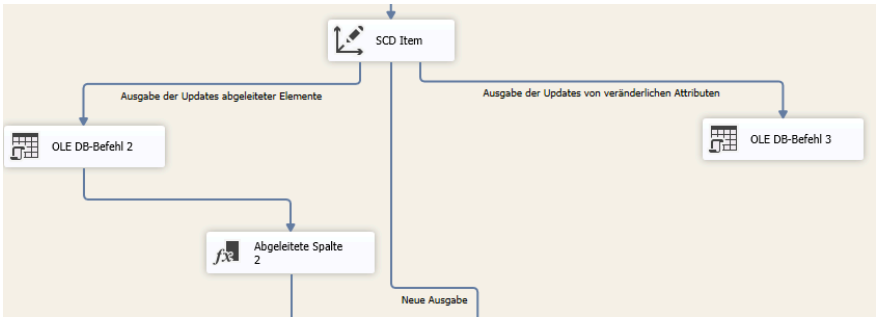
Nun kommen die Slowly Changing Dimension zum Einsatz, denn es sollen auch Veränderungen (in unserem Fall Lagerplatzänderungen) erfasst werden. Nach hinzufügen der entsprechenden Komponente können die SCD Einstellungen vorgenommen werden.

Das Menü für die SCD kommt als eine Art Wizzard daher. Auf der erste Seit werden die Zieltabelle, also die Dimension Tabelle des Datenmodells (*DM.DIM_Item*) und der zugehörige Business Key festgelegt. Es ist der Primary Key der Tabelle in Navision, die „*Item_No*“. Wir erinnern uns, dass dieser Business Key nicht mehr als Primärschlüssel verwendet wird, sondern durch einen Surrogate Key als Primärschlüssel ersetzt wird.

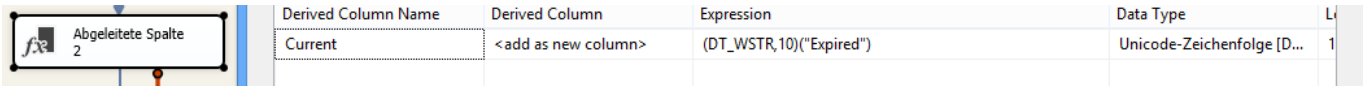


Im weiteren werden die Spalten und deren Typ angegeben. Wir nehmen an, „*Description*“ ist unveränderbar und bei einer Änderung wird ein Fehler ausgegeben. Daher nehmen wir es als „*Fixed Attribute*“. Bei „*Shelf_No*“ sollen alle Datensätze geändert werden. Daher wählen wir für „Shelf_No“ den Typ SCD I. Bei den weiteren Fenstern des Menü belassen wir die Default Werte. Nach Beenden werden die zugehörigen Ausgabepfade erstellt.





Den Datenfluss von „*Ausgabe der Updates abgeleiteter Elemente*“ wollen wir uns näher ansehen. Er beinhaltet die Datensätze, die ersetzt werden. Wir möchten zu diesen Datensätzen eine Spalte „*Current*“ hinzufügen, deren Inhalt „*Expired*“ beträgt. Dies kann wieder mit der „*Abgeleitete Spalte*“ Komponente realisiert werden.

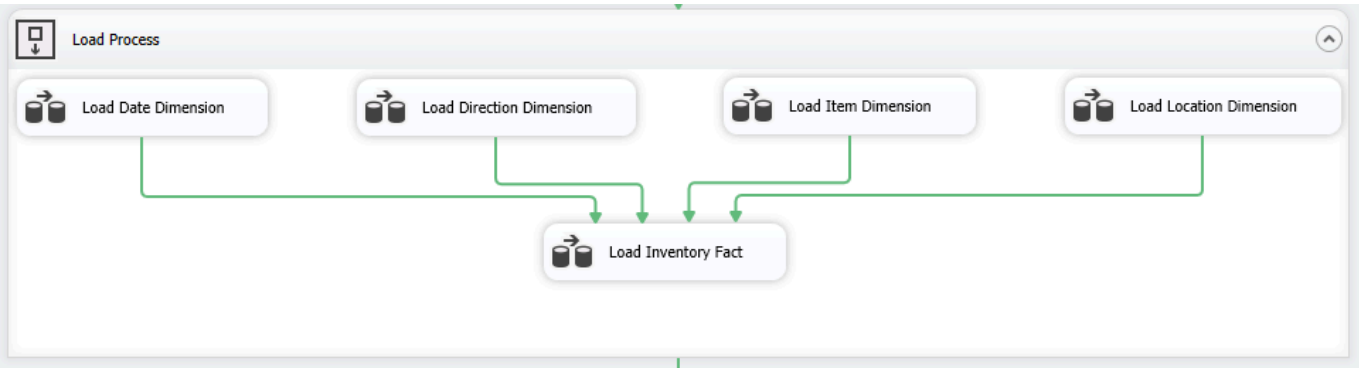


Der ausgehende Datenfluss dieser Änderung muss nun mit der neuen Ausgabe verbunden werden. Über die „*Union all*“ Komponente können die Datenflüsse zusammengeführt werden. Die Spalten der beiden Eingänge werden zu den Spalten der Ausgängen zugewiesen.

Output Column Name	UNION ALL Eingabe 1	Eingabe '1' für UNION ALL
Item_No	Item_No	Item_No
Shelf_No	Shelf_No	Shelf_No
BaseUnit	BaseUnit	BaseUnit
Description	Description	Description
TimeStamp	TimeStamp	TimeStamp

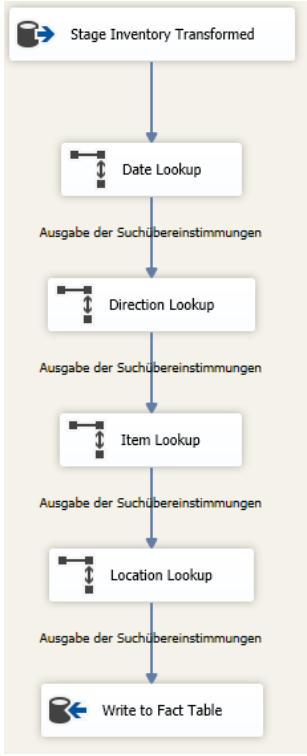
Das Ergebnis wird wiederum in „*DM.DIM_Item*“ gespeichert. Unsere Dimension Tabelle ist nun fertig und wir nehmen auch an, dass alle anderen Dimension Tabellen ebenfalls fertig sind. Der letzte Schritt ist es, die Surrogate Keys der Dimension Tabellen in die Fact Tabelle einzufügen.

Damit dieser Schritt erst geschieht, wenn die Bearbeitung an allen Dimension Tabellen abgeschlossen ist, werden die Ausgänge aller Dimension Tabellen zum Eingang einer „*Datenfluss*“ Komponente der Fact Tabelle geführt.



Für den Datenfluss der Fact Tabelle laden wir die „*STA.Inventory_transformed*“ Tabelle. In diese Tabelle müssen wir nun die Surrogate Keys der Dimension Tabelle laden.

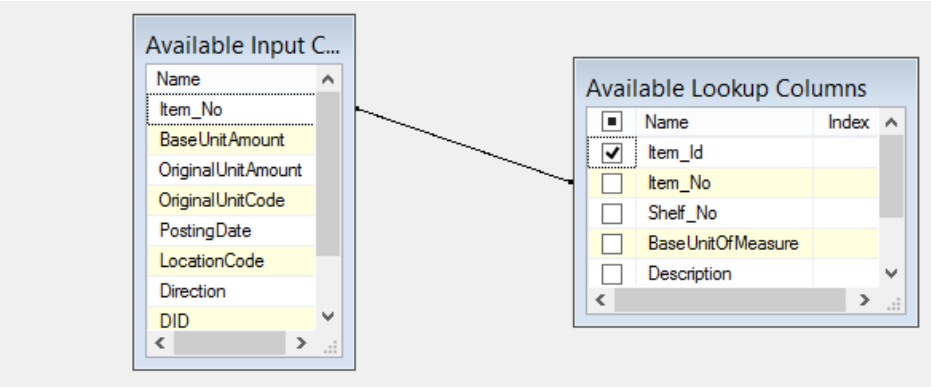
Mittels Lockup Komponenten werden diese Surrogate Keys geladen.



Am Beispiel von „*Item_Lookup*“ betrachten wir die Arbeitsweise dieser Komponente. Auf der Seite Verbindung geben wir an, dass wir nur Datensätze mit dem „Current“ in unsere Auswahl einschliessen wollen. Geänderte Tabellen wurden ja mit „*Expired*“ vermerkt.

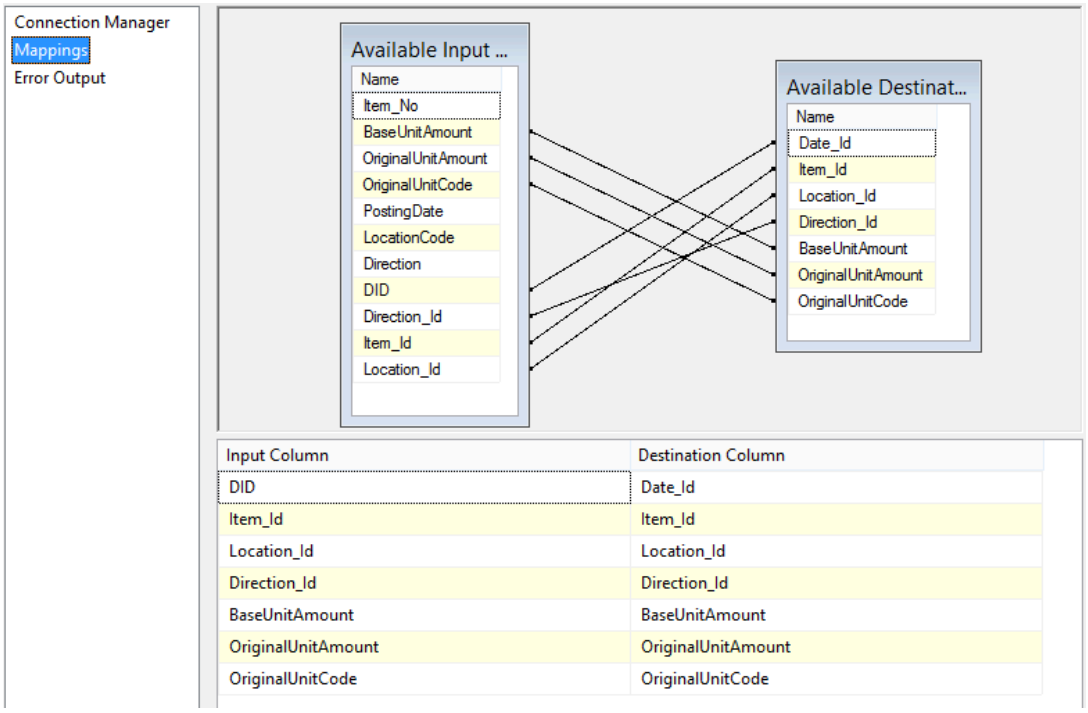
```
Select * from DM.DIM_Item where [Current] = 'Current';
```

Auf der Seite „*Columns*“ weist man nun die Spalten zu. Dafür ziehen wir die „*Item_No*“ zu dem Equivalent der Lookup Tabelle. Über eine Linie und Textzeile wird diese Verbindung visualisiert. Da der Surrogate Key in der Spalte „*Item_Id*“ gespeichert ist und dieser eingefügt werden soll, wird das Feld durch ein Häckchen aktiviert. Mit allen Dimension-Tabellen wird auf gleiche Weise verfahren.



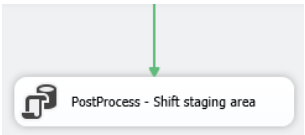
Lookup Column	Lookup Operation	Output Alias
Item_Id	<add as new column>	Item_Id

Zum Schluss werden die Daten in die Fact Tabelle des Dimension Modells geschrieben. Die Spalten müssen noch zugeordnet werden. Dies geschieht auf der „*Mappings*“ Seite der Destination-Komponente.



Finalisierung

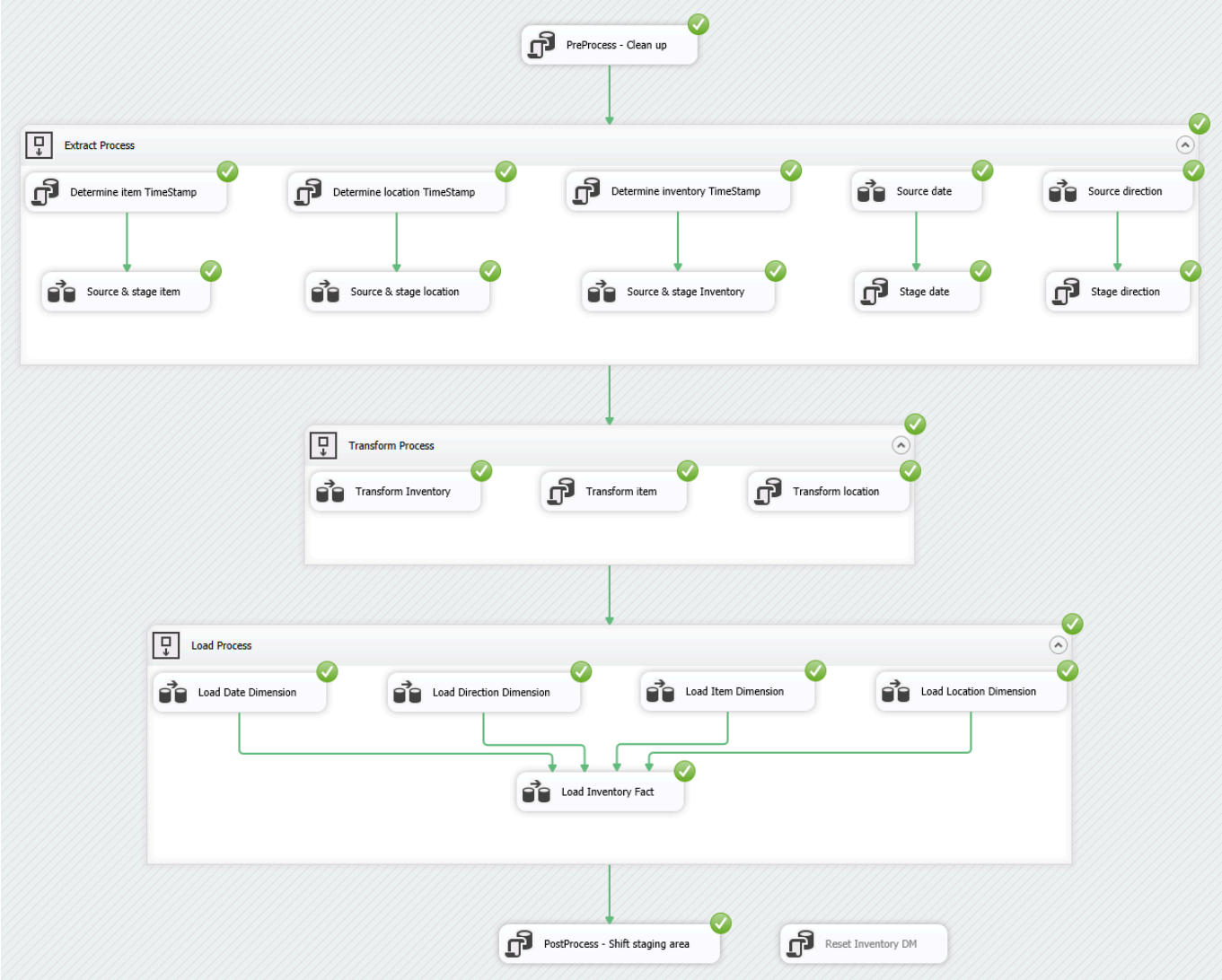
Am Ende müssen wir wieder einige Aufräumungsarbeiten durchführen. Die *STA.Inventory* Tabelle soll zu *STA.OLD_Inventory* gespeichert werden, da wir die letzte Beladung für die Berechnung des Timestamps benötigen. Hier hilft uns wieder die Execute SQL Task Komponente. Die Kopier- und Löschoperationen werden im SQL Statement eingetragen. Die anderen Tabellen der Staging Area werden ggf. ebenso behandelt.




```
----- Update Old Inventory Table to store the latest timestamp only if there are changes
if exists(select * from sys.tables t inner join sys.schemas s on t.schema_id=s.schema_id where
s.name='STA' and t.name='OLD_Inventory') begin
 if (select count(*) from STA.Inventory) > 0 begin
  truncate table STA.OLD_Inventory
  insert into STA.OLD_Inventory select * from STA.Inventory;
  end;
 end;
----- Update Old Item Table to store the latest timestamp only if there are changes
if exists(select * from sys.tables t inner join sys.schemas s on t.schema_id=s.schema_id where
s.name='STA' and t.name='OLD_Item') begin
 if (select count(*) from STA.OLD_Item) > 0 begin
  truncate table STA.OLD_Item
  insert into STA.OLD_Item select * from STA.Item;
  end;
 end;
```

Ausführen

Durch klicken auf „*Start*“ wird das Beispiel ausgeführt. Wenn alles geklappt hat, wird zu den jeweiligen Komponenten ein grünes Häckchen angezeigt. Andernfalls erscheint ein rotes Kreuz.



Auswertungen mit Delphi

Die Auswertungen der Daten können ganz einfach mit Delphi dargestellt werden. Eine gute Vorgehensweise ist es, die SQL-Abfragen als Sichten (Views) direkt in der Datenbank zu speichern. So können diese Abfragen optimiert bzw. korrigiert werden, ohne den Entwickler für eine Neukompilation der Software in Anspruch zu nehmen. Auch sind alle SQL-Abfragen auf einen Blick ersichtlich und lassen sich einfach testen.

Für Delphi gibt es verschiedene Reporting Komponenten, zum Beispiel Fast Report, Virtual Print Engine, Rave Reports, Quickreport und viele mehr. Leider wurden in der Vergangenheit öfters die Reporting Tools geändert, so dass viele Entwickler unterschiedliche Tools präferieren. Deshalb soll auch nicht näher darauf eingegangen und nur wichtige Eigenschaften, die von den Tools benötigt werden, angeführt werden.

Folgende Eigenschaften sollten die Reporting Komponenten beinhalten:

- Export in ein Format, das dem Unternehmen, respektive dem Analysten, dienlich ist (Excel, Word, PDF)
- Eine Anbindung an ein TDataSet erleichtert die Übernahme der Daten
- Überlegen, ob ein Bildexport für Web-Seiten benötigt wird
- Überlegen, ob die Auswertung durch statistische Funktionen erweitert werden soll

Beispiel

Wir nehmen an, wir möchten für unser Beispiel einfach nur die Lagerplätze mit der Anzahl der ausgehenden Artikel als Diagramm anzeigen. So können wir ermitteln, über welchen Lagerplatz die meisten Artikel unser Unternehmen verlassen.

Die Implementierung ist dank Delphi denkbar einfach. Wir nehmen eine TADOConnection Komponenten und verbinden sie mit der Datenbank.

Der Verbindungsparameter im Beispiel lautet:

```
Provider=SQLOLEDB.1;Persist Security Info=True;User ID=sa;Initial
Catalog=DWH_DT2013;Data Source=DANIELASEFZ170B\SQLEXPRESS;Use Procedure for
Prepare=1;Auto Translate=True;Packet Size=4096;Workstation ID=DANIELASEFZ170B;Use
Encryption for Data=False;Tag with column collation when possible=False
```

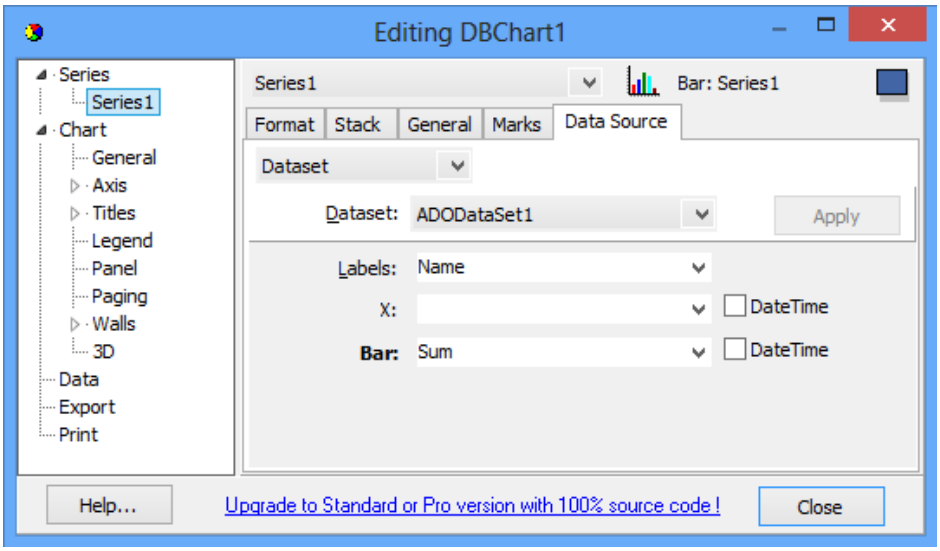
Dazu kommt eine TADODataSet Komponenten, deren Connection Property mit der TADOConnection Komponente verbunden wird. Als CommandText geben wir die View an, die in der Datenbank gespeichert ist:

```
select * from DM.View_Items_Inventory_Movement_OUT
```

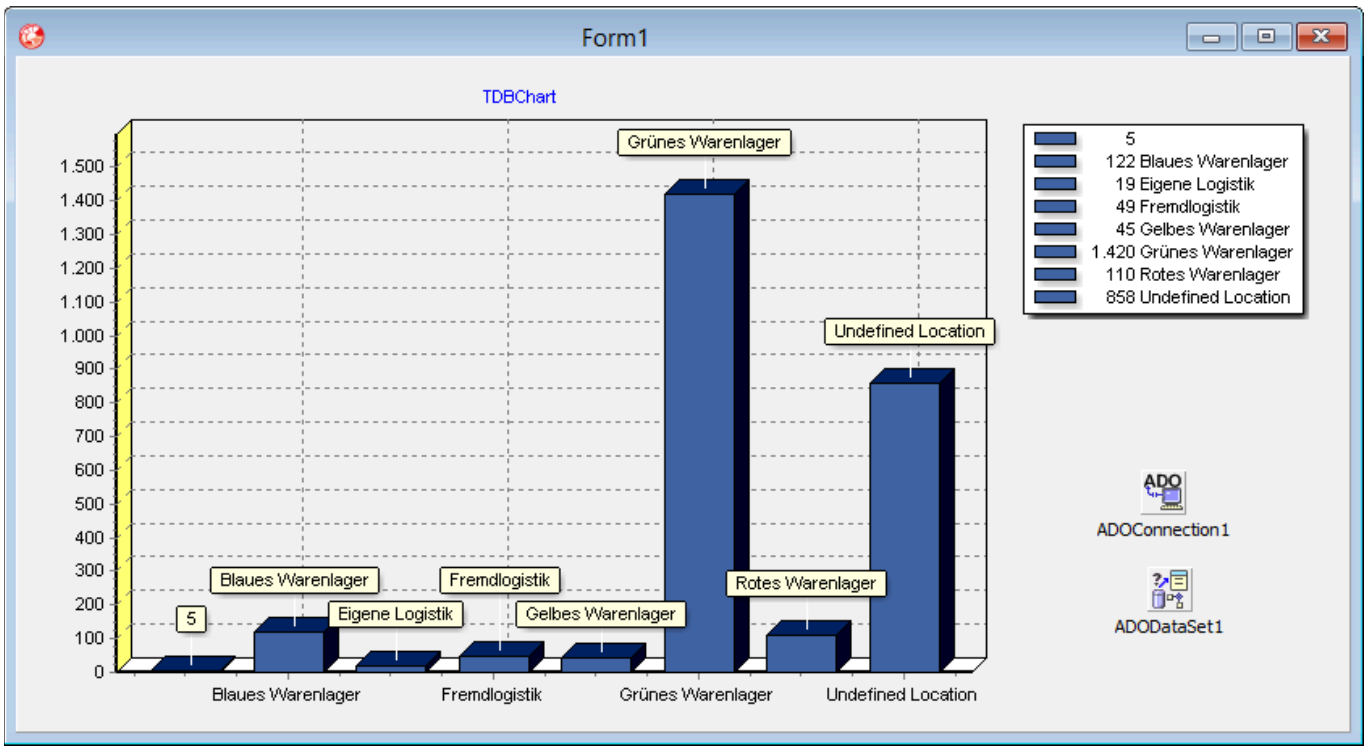
In der Datenbank ist die View gespeichert und sie beinhaltet die nachfolgende SQL Abfrage. Es empfiehlt sich, immer die SQL Abfragen in der Datenbank zu hinterlegen, da für eine Änderungen am SQL-Befehl nicht die Anwendung neu kompiliert werden muss.

```
SELECT        l.Code, l.Name, f.Direction_Id, SUM(f.BaseUnitAmount) AS Sum
FROM            DM.Fact_InventoryAmount AS f INNER JOIN
                         DM.DIM_Date AS d ON f.Date_Id = d.DID INNER JOIN
                         DM.DIM_Location AS l ON l.Location_Id = f.Location_Id INNER JOIN
                         DM.DIM_Item AS i ON i.Item_Id = f.Item_Id
GROUP BY l.Code, l.Name, f.Direction_Id
HAVING        (f.Direction_Id = 2)
```

Anschließend platzieren wir ein TChart auf das Formular und fügen eine Bar-Serie hinzu, wobei wir als DataSource unser DataSet angeben.



Im Anschluss daran wird bereits unser Diagramm angezeigt.



Wir erkennen in diesem Beispiel gleich, dass die Datenqualität nicht stimmt. Zum einen gibt es scheinbar einen Eintrag, bei dem der Lagerplatz oder bessen Bezeichnung leer ist, zum anderen gibt es einige Einträge mit dem Lagerplatz „*Undefined Location*“. Diese können im ETL Prozess nicht korrigiert werden, da nicht bekannt ist, welches Problem sich dahinter verbirgt und zu welchen Lagerplatz diese Artikel gehören. Es handelt sich also um Kategorie A Fehler, die wir im System selbst beheben müssen.

Statistikauswertungen

Zusätzlich bietet es sich bei einem Data Warehouse an, mit Hilfe verschiedener Statistik-Tools die Auswertungen aufzuwerten. Sei es die Aufarbeitung und Analyse des Geschäftsjahres, oder der Blick in die Zukunft mit Trendanalysen, die Anwendung der Statistik bietet einen erheblichen Mehrwert für ein Unternehmen.

Statistik ist ein sehr umfassendes Thema und würde den Rahmen dieses Dokuments sprengen. Im folgenden wird das Open Source Tool R vorgestellt und beschrieben, wie dieses in eigene Delphi Programme implementiert werden kann.

Statistiken mit R

R ist eine Programmiersprache und Entwicklungsumgebung für statistische Auswertungen und Grafiken. Das Open Source Projekt wird unter der GNU General Public License veröffentlicht und kann unter [1] heruntergeladen werden. Die Software ist für Windows, Mac und Linux verfügbar. Trotz der Möglichkeit mit Hilfe von Delphi Mac Anwendungen zu erstellen, beschränkt sich die Anbindung von R auf Windows Anwendungen.

Um R in eigene Anwendungen zu integrieren steht ein DCOM Server zur Verfügung. Ein Installationspaket, das alle wichtigen Programmteile beinhaltet, kann über den Server der Universität Wien, unter „*RAndFriends*“ [2], geladen werden. „*RAndFriends*“ kann für nicht-kommerzielle Anwendungen kostenlos verwendet werden. Für kommerzielle Anwendungen wird auf [3] verwiesen.

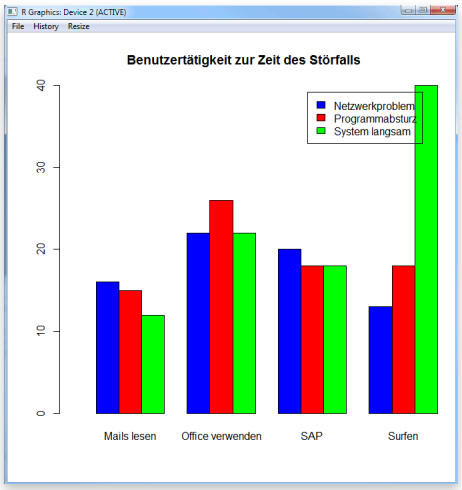
Installation

Für die Installation von R und dem benötigten DCOM-Server wird das Installationspaket von „*RAndFriends*“ [2] empfohlen. Die Installation läuft automatisch. Zusätzlich bietet das Paket auch die Möglichkeit der Anbindung von R an Excel. Wird nur der DCOM-Server installiert, so findet sich unter [4] eine Schritt für Schritt Anleitung.

Der modulare Aufbau von R bietet die Möglichkeit, weitere Pakete für verschiedene statistische Auswertungen nachzuladen. Hierzu muss R direkt geöffnet und unter dem Menüpunkt „*Pakete* / *Lade Paket*“ nachgeladen werden.

Erste Schritte

Die folgende Anleitung bezieht sich auf die von Dr. Dieter Menne unter [7] vorgestellte Unit. Sie erleichtert die Arbeit und nimmt uns wesentliche Arbeit bei der Implementierung und Kommunikation mit R ab. Somit können wir uns ganz auf die Anwendung von R konzentrieren.



Der einfachste Anwendungsfall ergibt sich, wenn in Delphi Befehle direkt an die Eingabe von R gesendet werden. Dieser Vorgang kann durch Verwendung der Methode `EvaluateNoReturn`, oder deren kurzem Derivat `ENR`, realisiert werden.

Das folgende Beispiel zeigt eine Auswertung der Benutzertätigkeit zur Zeit eines fiktiven Störfalls. Der Zugehörige Plot wird ein einem Fenster ausgegeben.

```
begin
  RCon.Clear;
  RCon.ENR('problem = rbind(c(16,22,20,13), c(15,26,18,18), c(12,22,18,40))');
  RCon.ENR('colnames(problem) = c("Mails lesen", "Office verwenden", "SAP",
           "Surfen")');
  RCon.ENR('rownames(problem) = c("Netzwerkproblem", "Programmabsturz", "System
           langsam")');
  RCon.ENR('names(dimnames(problem))=c("Problem", "Aktivität")');
  RCon.ENR('barplot(problem, beside=T, legend=T, main="Benutzertätigkeit zur
           Zeit des Störfalls", xlim=c(0,15), col=c("blue", "red", "green"))');
end;
```

Für die Beschreibung der Befehle in R sei auf die sehr gute Lektüre „*R - R-Einführung: Einführung durch angewandte Statistik*“ [5] verwiesen.

Datenübergabe

Nun ist die zuvor vorgestellte Art der Datenübergabe nur bedingt hilfreich. Daten werden in unserem Fall aus der Datenbank des DHWs geladen und stehen in Datasets zur Verfügung. Die Unit bietet auch hier eine Erleichterung um diese Daten mit Hilfe von Arrays an R zu übergeben. Es muss also ein Zwischenschritt, in dem die Daten aus der Datenbank gelesen und in ein Array gespeichert werden, durchgeführt werden.

```
SetDoubleVector
SetStringVector
SetIntVector
SetDoubleMatrix

GetDoubleVector
GetStringVector
GetIntVector
GetDoubleMatrix
```

Mit Hilfe der `Get` Befehle können Ergebnisse aus den Berechnungen wieder in die Delphi-Anwendung geladen werden. Mit dem Befehl `Sym` können einzelne Symbole bzw. Variablen aus R gelesen werden.

```
var
  input : array[0..1] of integer;
  len   : Integer;
begin
  // Daten laden
  input[0] := 10;
  input[1] := 5;

  // Symbole in R anlegen
  RCon.SetIntVector('input', 2, input);

  // Befehl ausführen, Summe bilden
  RCon.EvaluateNoReturn('output=sum(input)');

  // Symbol aus R laden und weiterverarbeiten...
  ShowMessage('Ergebnis: ' + IntToStr(RCon.Sym['output']));
end;
```

Die Methode dieses Beispiels entspricht folgendem R-Code:

```
> input = c(10,5)
> output = sum(input)
> output
[1] 15
```

Tipp: Eventuell kommt es vor, dass aus der Auswertung heraus NULL-Werte in den Daten vorhanden sind. Diese Werte sind bei einer Auswertung hinderlich. Über eine einfache Zeile in R können diese Werte entfernt werden. Es ist also nicht notwendig, die Arrays in Delphi neu zu ordnen.

```
> umsatz = umsatz[which(umsatz != is.na(umsatz))]
```

Tabellen übergeben

Über einen Umweg können auch ganze Tabellen an R übergeben werden. Hierzu ist es notwendig, die Daten spaltenweise zu übergeben. Je ein Symbol wird mit den Werten einer Spalte gefüllt. Im Anschluss daran wird in R eine Tabelle erstellt.

```
> name = c("Maria", "Tim", "Christine", "Claudia")
> b = c("angestellt", "selbstständig", "Student", "selbstständig")
> beruf = factor(b)
> f = c("dünn", "dünn", "muskulös", "hager")
> figur = ordered( f, levels = c("hager", "dünn", "schlank", "normal", "mollig",
"muskulös"))
> groesse = c(1.73, 1.99, 1.88, 1.79)
> gewicht = c(81, 83, 93, 88)
> trinkt = c(NA, TRUE, TRUE, TRUE)

> freunde = data.frame("Name" = name, "Figur" = figur, "Groesse" = groesse,
"Gewicht" = gewicht, "Trinkt" = trinkt, "Beruf" = beruf)
> freunde
       Name    Figur Groesse Gewicht Trinkt         Beruf
1     Maria     dünn    1.73       81     NA    angestellt
2       Tim     dünn    1.99       83   TRUE selbstständig
3 Christine muskulös    1.88       93   TRUE       Student
4   Claudia    hager    1.79       88   TRUE selbstständig
```

Um die Funktion besser zu veranschaulichen, wurden die Werte direkt in R eingetragen. Wichtig in diesem Zusammenhang ist die Methode `data.frame`, mit deren Hilfe die Tabelle durch Angabe der einzelnen Spaltennamen und Spaltenwerte erstellt werden kann.

Daten importieren / exportieren

Für den direkten Import und Export von Daten stellt R eine Reihe von Funktionen zur Verfügung. Die Funktion `read.table` kann dazu verwendet werden, Daten in Tabellenformat einzulesen. Abgeleitet von `read.table` sind die Funktionen `read.csv`, `read.csv2`, `read.delim` und `read.delim2`. Das Gegenstück zu den `read.*` Funktionen stellen die Funktionen `write.table`, `write.csv` und `write.csv2` dar.

Für weitere Informationen zum Laden von Daten aus Dateien (zum Beispiel CSV, Excel...) sei auf Michael Lundholms Dokument „*Loading data into R*“ [6] verwiesen.

Grafiken speichern

Plots können natürlich nicht nur in einem Fenster ausgegeben werden, sondern auch in eine Datei geschrieben werden. Diese Grafik-Dateien können im Anschluss in die eigenen Auswertungen integriert werden.

```
> fit <- lm(some ~ model)
> png(filename="your/file/location/name.png")
> plot(fit)
> dev.off()
```

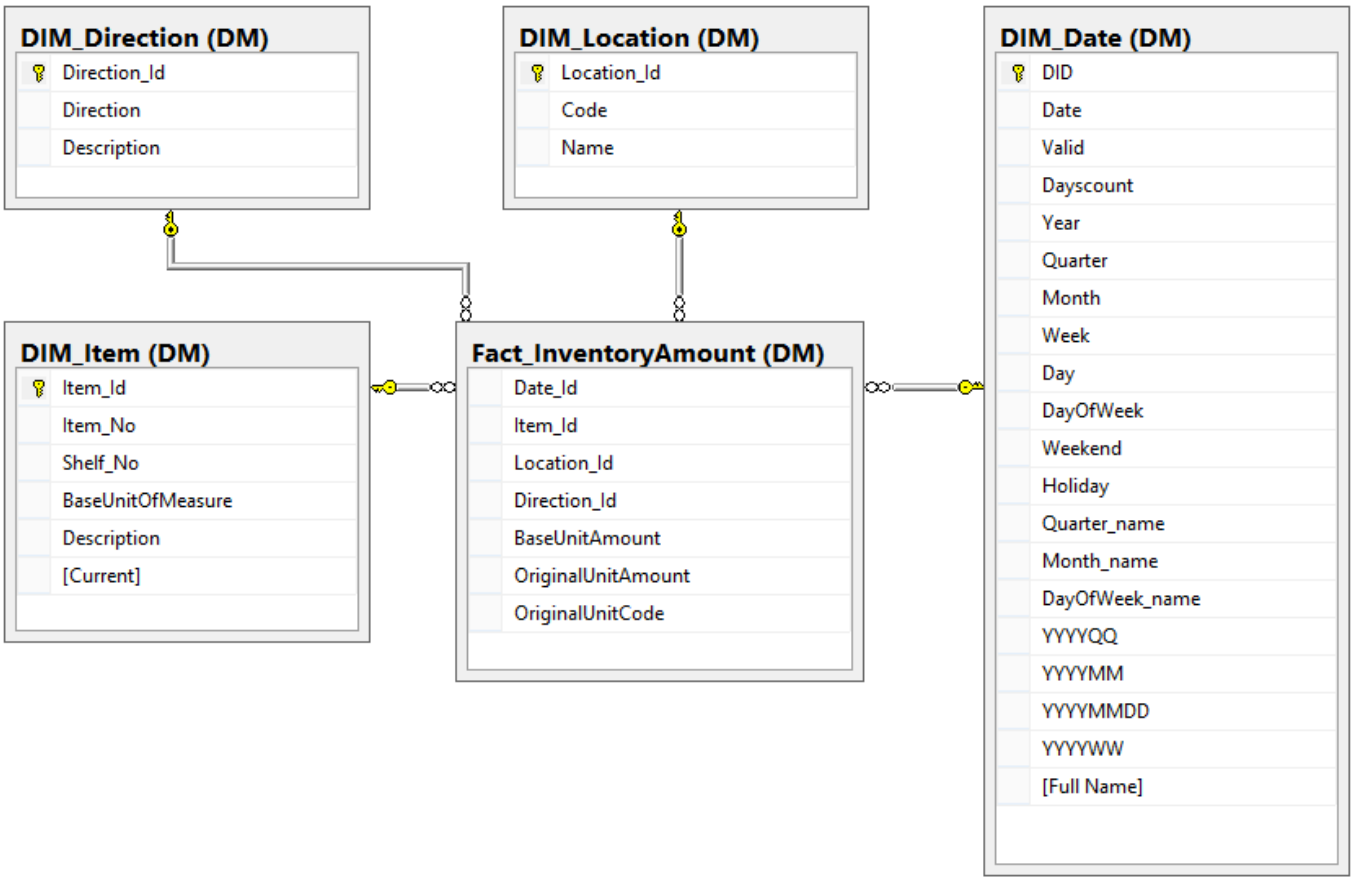
Mittels `EvaluateNoReturn` können die Befehle an R übergeben werden. Das Tool erstellt die Grafik-Dateien und im Anschluss daran kann die Delphi Anwendung diese laden und weiterverarbeiten.

Anhang A - Logical Data Map

Logical Data Map

Target					Source				Transform
Table Name	Column Name	Data Type	Table Type	SCD Type	Database Name	Table Name	Column Name	Data Type	
Dim_Item	Item_Id	int	Dimension	n/a					Surrogate Key
Dim_Item	Item_No	nvarchar(20)	Dimension	bk	Demo Database NAV (6-0)	CRONUS AG$Item	No_	varchar(20)	Trim([No_])
Dim_Item	Shelf_No	nvarchar(10)	Dimension	2	Demo Database NAV (6-0)	CRONUS AG$Item	[Shelf No]	varchar(10)	Trim([Shelf No_])
Dim_Item	Description	nvarchar(30)	Dimension	1	Demo Database NAV (6-0)	CRONUS AG$Item	Decription	varchar(30)	Trim(description)
Dim_Item	BaseUnitOfMeasure	nvarchar(10)	Dimension	2	Demo Database NAV (6-0)	CRONUS AG$Item	[Base Unit of Measure]	varchar(10)	Trim([Base Unit of Measure])
Dim_Location	Location_Id	int	Dimension	n/a					Surrogate Key
Dim_Location	Code	nvarchar(10)	Dimension	bk	Demo Database NAV (6-0)	CRONUS AG$Location	Code	varchar(10)	Trim(Code)
Dim_Location	Name	nvarchar(30)	Dimension	1	Demo Database NAV (6-0)	CRONUS AG$Location	Name	varchar(30)	Trim(Name)
Dim_Direction	Direction_Id	int	Dimension	n/a					Surrogate Key
Dim_Direction	Direction	nvarchar(3)	Dimension	bk	Flat File: direction.csv		Direction	string	Natural Key for Direction
Dim_Direction	Description	nvarchar(50)	Dimension	fixed	Flat File: direction.csv		Description	string	
Dim_Date	DID	int	Dimension	n/a					Surrogate Key
Dim_Date	Date	date	Dimension	bk	Flat File: date.csv		Date	string	Natural Key for Date
Dim_Date	Valid	nvarchar(10)	Dimension	fixed	Flat File: date.csv		Valid	string	
Dim_Date	Dayscount	int	Dimension	fixed	Flat File: date.csv		Dayscount	string	
Dim_Date	Year	int	Dimension	fixed	Flat File: date.csv		Year	string	
Dim_Date	Quarter	int	Dimension	fixed	Flat File: date.csv		Quarter	string	
Dim_Date	Month	int	Dimension	fixed	Flat File: date.csv		Month	string	
Dim_Date	Week	int	Dimension	fixed	Flat File: date.csv		Week	string	
Dim_Date	Day	int	Dimension	fixed	Flat File: date.csv		Day	string	
Dim_Date	DayOfWeek	int	Dimension	fixed	Flat File: date.csv		DayOfWeek	string	
Dim_Date	Weekend	nvarchar(10)	Dimension	fixed	Flat File: date.csv		Weekend	string	
Dim_Date	Holiday	nvarchar(10)	Dimension	fixed	Flat File: date.csv		Holiday	string	
Dim_Date	Quarter_name	nvarchar(10)	Dimension	fixed	Flat File: date.csv		Quarter_name	string	
Dim_Date	Month_name	nvarchar(10)	Dimension	fixed	Flat File: date.csv		Month_name	string	
Dim_Date	DayOfWeek_name	nvarchar(10)	Dimension	fixed	Flat File: date.csv		DayOfWeek_name	string	
Dim_Date	YYYYQQ	nvarchar(10)	Dimension	fixed	Flat File: date.csv		YYYYQQ	string	
Dim_Date	YYYYMM	nvarchar(10)	Dimension	fixed	Flat File: date.csv		YYYYMM	string	
Dim_Date	YYYYMMDD	nvarchar(10)	Dimension	fixed	Flat File: date.csv		YYYYMMDD	string	
Dim_Date	YYYYWW	nvarchar(10)	Dimension	fixed	Flat File: date.csv		YYYYWW	string	
Dim_Date	[Full Name]	nvarchar(30)	Dimension	fixed	Flat File: date.csv		[Full Name]	string	
Fact_InventoryAmount	Date_Id	int	Fact		DWH_BB_4	Dim_Date	DID	int	select DID from Dim_Date where (Date = [CRONUS AG$Item Ledger Entry].[Posting Date])
Fact_InventoryAmount	Item_Id	int	Fact		DWH_BB_4	Dim_Item	Item_Id	int	select Item_Id from Dim_Item where (Item_No = [CRONUS AG $Item].[No_])
Fact_InventoryAmount	Location_Id	int	Fact		DWH_BB_4	Dim_Location	Location_Id	int	select Location_Id from Dim_Location where (Code = [CRONUS AG $Location].[Code])
Fact_InventoryAmount	Direction_Id	int	Fact		DWH_BB_4	Dim_Direction	Direction_Id	int	select Direction_Id from Dim_Direction where Direction = IIF(Fact_InventoryAmount.BaseUnitAmount < 0 „OUT“ : „IN“)
Fact_InventoryAmount	BaseUnitAmount	decimal(38,20)	Fact		Demo Database NAV (6-0)	[CRONUS AG$Item Ledger Entry]	Quantity	decimal(38,20)	select Quantity from [CRONUS AG$Item Ledger Entry]
Fact_InventoryAmount	OriginalUnitAmount	decimal(38,20)	Fact		Demo Database NAV (6-0)	[CRONUS AG$Item Ledger Entry]	[Qty_per unit of measure]	decimal(38,20)	select [Qty_per unit of measure] from [CRONUS AG$Item Ledger Entry]
Fact_InventoryAmount	OriginalUnitCode	nvarchar(10)	Fact		Demo Database NAV (6-0)	[CRONUS AG$Item Ledger Entry]	[Unit of measure code]	varchar(10)	select [Unit of measure code] from [CRONUS AG$Item Ledger Entry]

Anhang B - Datenmodell des Beispiels



Literatur

[1] http://www.r-project.org

[2] RAndFriendsSetup2152V3.2-9-1

[3] http://www.statconn.com/products.html

[4] http://mahendramahfood.blogspot.co.at/2007/12/how-to-access-r-engine-with-borland.html

[5] R - R-Einführung: Einführung durch angewandte Statistik; Pearson Studium; 1 Auflage (9. Februar 2011), ISBN 978-3868940602

[6] Loading data into R; Michael Lundholm; Version 1.1; http://people.su.se/~lundh/reproduce/loading_data.pdf

[7] Dr. Dieter Menne; Menne Biomed Consulting; http://www.menne-biomed.de/download/download.html

[8] R Development Core Team, 2011] R Development Core Team (2011). R: A Language and Environment for Statistical Computing. R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-07-0

[9] The Data Warehouse Toolkit; Second Edition; Ralph Kimball;

[10] Datawarehousing; David Meyer;

[11] BI-Praktikum IBM – WS 2008/09; Christian Kurze

[12] http://community.qlikview.com/blogs/qlikviewdesignblog/2013/03/25/dimensions-and-measures

[13] http://www.symcorp.com/downloads/ADAPT_white_paper.pdf

[14] Microsoft SQL Server Integration Services: Erstellen von Datenintegrations- und Datentransformationslösungen auf Unternehmensebene; Bernd Jungbluth; 1. Auflage (25. Februar 2011); ISBN 3-866456-54-9

[15] http://kettle.pentaho.com