

CAS Information Engineering

Leistungsnachweis für Modul DB & DWH

Gruppenmitglieder:

Andreas Fischer
Bernd Novotny
Tobias Schieferdecker

Inhaltsverzeichnis:

Leistungsnachweis für Modul DB & DWH.....	1
1. Übersetzung des gegebenen ERMs in den uns bekannten Dialekt.....	2
2. Auslagern der relevanten Daten in csv-Dateien.....	5
3. Erstellung der staging-area.....	6
4. Reparieren der korrupten Daten.....	8
5. Konzipierung des Sternschemas für das DWH.....	10
6. Erstellung und Befüllen des DWH.....	11
7. Auswertungen auf Basis des DWH.....	30
8. Fazit und Schlusswort.....	32

1. Übersetzung des gegebenen ERM in den uns bekannten Dialekt

Es soll die gegebene Datenbank "Northwind" genommen und gemäss einem neu anzufertigenden Schema migriert werden. Das Schema soll im ERM-Dialekt von H.-W. Buff ausgeführt werden.

A. Ausgangsdaten

Zur Erzeugung der Datenbank liegen die beiden SQL-Skripte

- Northwind-Database.sql
- Northwind-Daten.sql

vor. Sie können in MySQL-Workbench geöffnet und ausgeführt werden.

Nach erfolgreichem Durchlauf liegt die Datenbank mit 20 Entitäten und einigen Testdaten eingelese vor.

Das Schema der ERP-Datenbank "Northwind" (siehe Bild 1) liegt in der "Bachman's crow-foot notation" vor. Sie kann z.B. erzeugt werden durch das MySQL-Tool Workbench.

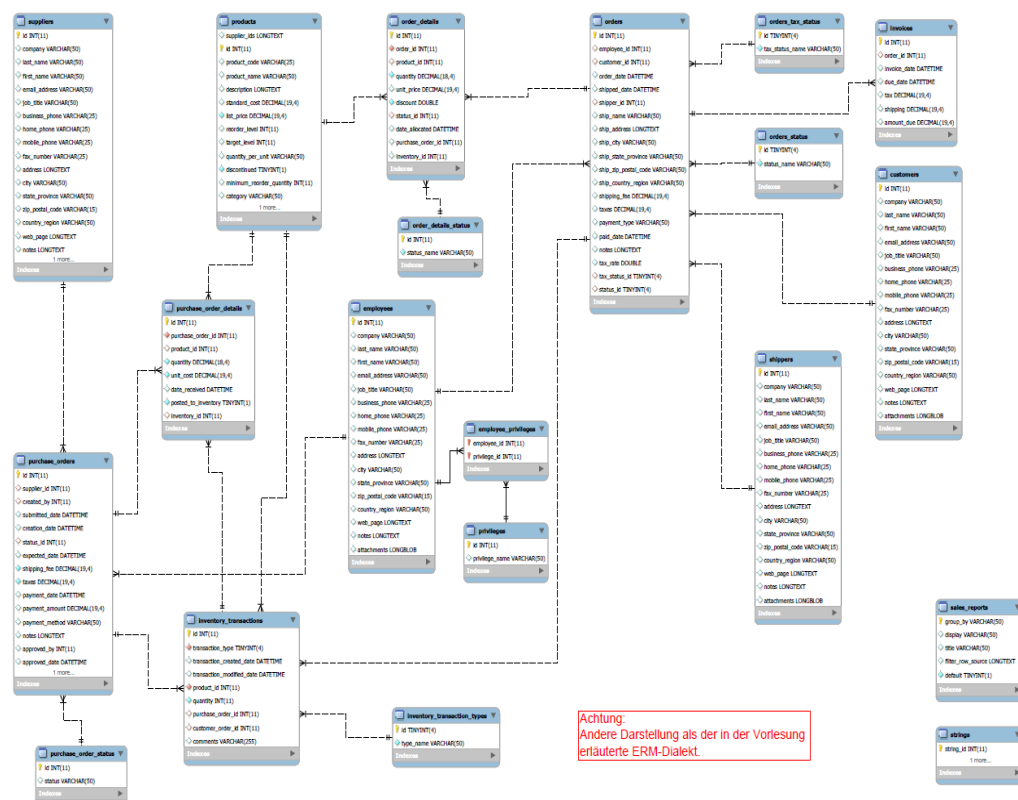


Bild 1: Schema der ursprünglichen ERP-Datenbank "Northwind".

Die Relationen sind hier durch Verbindungen mit speziellen Symbolen dargestellt, die die Kardinalitäten angeben. Bild 2 listet diese auf.

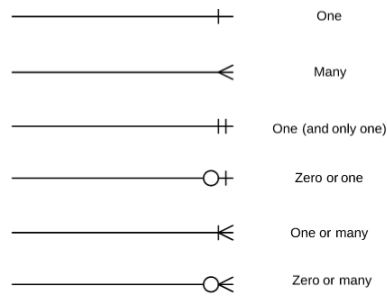


Bild 2: Bedeutung der verschiedenen Symbole in Bachmann's crow foot Notation

Um das neue ERM-Diagramm zu zeichnen wird das Tool Dia verwendet. Siehe dazu [https://de.wikipedia.org/wiki/Dia_\(Software\)](https://de.wikipedia.org/wiki/Dia_(Software)) oder <http://dia-installer.de/index.html.de>.

B. Entity-Relationship-Modell (ERM)

Bild 3 zeigt das neu erstellt ERM-Diagramm in Buff-Notation.

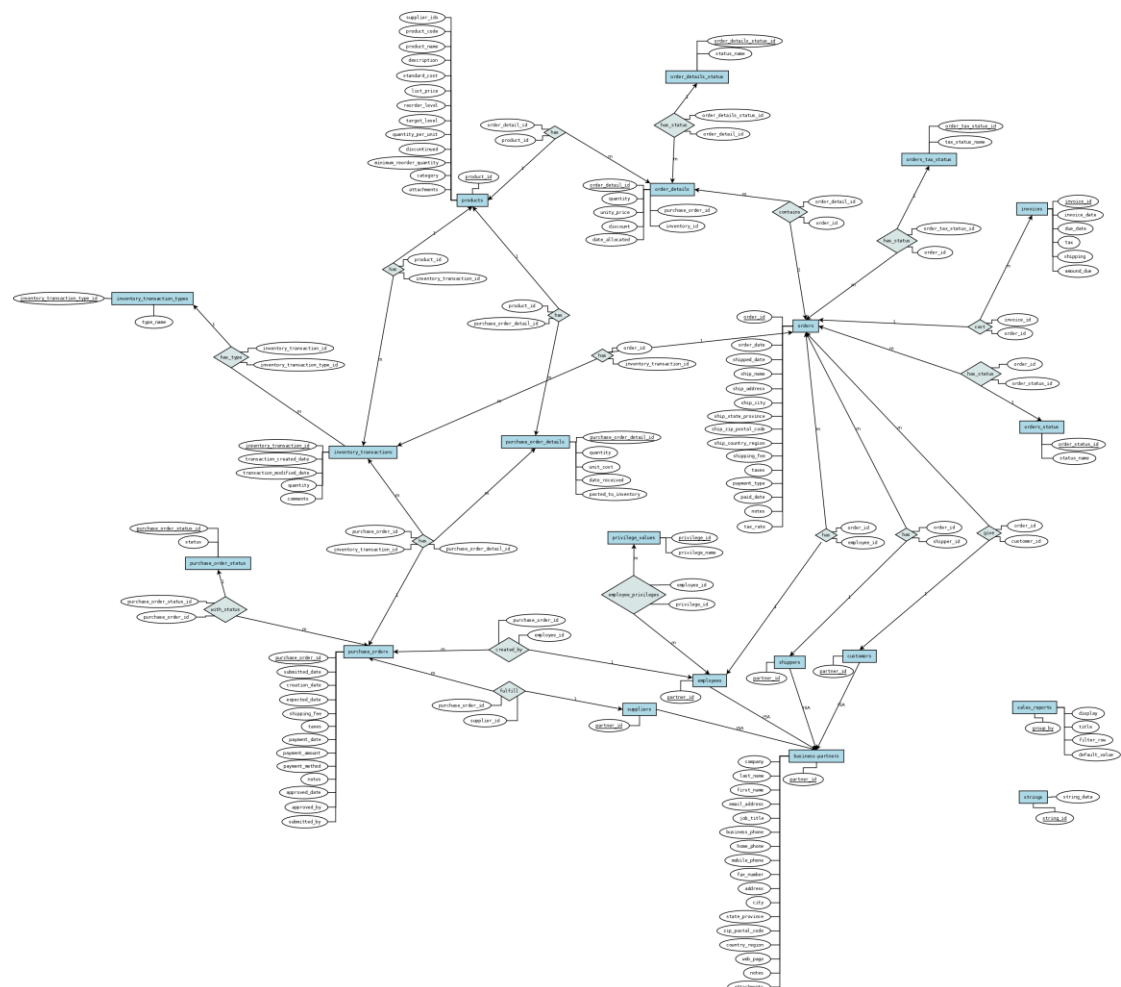


Bild 3: Übersetzung des Northwind-Schemas in den Dialekt von Buff.

Speziell wurde versucht ISA-Beziehungen einzubauen, da die Entitäten "suppliers", "employees", "shippers" und "customers" identische Attribute aufweisen. Dazu wurde neu die Entität "business-partners" eingefügt, vergleiche Bild 4.

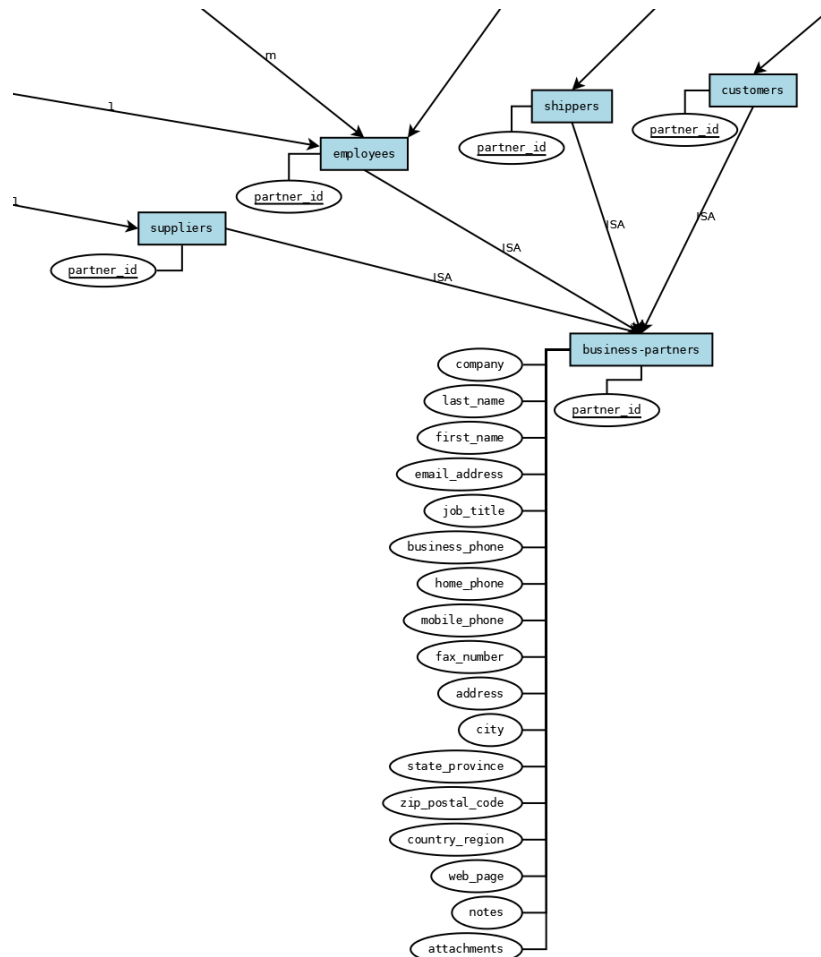


Bild 4: Die Entität "business-partners".

Beobachtungen:

- Das Diagramm ist grafisch anspruchsvoller und grösser als das ursprüngliche. Vor allem wenn viele Attribute vorhanden sind.
- Es sind mehr Elemente nötig, denen gute Namen gegeben werden müssen. Dies ist nicht immer einfach.
- Das gewählte Tool ist nicht ideal. Das Handling und die Darstellung könnten verbessert werden. Vor allem das Zeichnen der Relationen und deren Beschriftung sollte besser sein.
- Den Schlüsseln der Entitäten müssen eindeutige Bezeichnungen gegeben werden, da sie nicht nur in den Entitäten sondern auch in den Relationen verwendet werden.

C. SQL Skripte

Zur Umsetzung des ERM-Diagramms in eine Datenbank werden SQL-Skripte benötigt. Als Basis werden die ursprünglichen Skripte genommen und angepasst.

Zur Erzeugung der Datenbank muss als erstes immer das Skript **Northwind2-Database.sql** ausgeführt werden. Es löscht eine schon vorhandene Datenbank und baut dann eine neue wieder auf.

Danach können die Test-Daten eingelesen werden durch das Skript **Northwind2-Data.sql**.

Beobachtungen:

- Die ursprünglichen Skripte wurden scheinbar aus der Datenbank im MySQL-Workbench exportiert. Darin sind die Bezeichner alle "gequoted" mit "backticks". Dies wurde in den neuen Skripten nicht mehr beibehalten.
- Die Schlüssel wurden wie im Diagramm konsequent mit Entitätsnamen und angehängtem "_id" bezeichnet.
- Die Reihenfolge beim Einlesen der Daten ist wichtig. Daten auf die später verwiesen wird, müssen schon in der Datenbank sein. Die Constraints werden sonst nicht erfüllt.
- In den ursprünglichen Daten sind einzelne Records vorhanden, die auf Entitäten verweisen sollten aber mit NULL gefüllt. Diese konnten erst eingelesen werden, als die NULL-Werte durch vorhandene Fremdschlüssel ersetzt wurden.
- Da die Entitäten "suppliers", "employees", "shippers" und "customers" nur noch aus Schlüsseln bestehen und eine ISA-Beziehung zu der Entität "business_partners" haben, mussten die Schlüssel angepasst werden.

Die beiden Skripte laufen ohne Fehlermeldungen oder Warnungen durch und erzeugen bzw. füllen die Datenbank.

2. Auslagern der relevanten Daten in csv-Dateien

Nachdem die Northwind-Datenbank mit der uns zur Verfügung gestellten Datei "Northwind-Database.sql" mittels MySQL-Workbench erzeugt und im Folgenden im selben Tool mittels der Datei "Northwind-Data.sql" befüllt wurde, haben wir uns der Aufgabe gewidmet, die für unsere späteren Auswertungen relevanten Daten in csv-Dateien auszulagern.

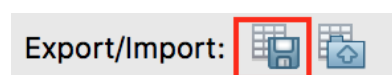


Bild 5 Der Knopf für den Export einer Datentabelle aus MySQL-Workbench ist rot umrandet.

Für diesen Schritt haben wir kein Skript verwendet. Wir haben vielmehr einfach die Funktionalität von MySQL-Workbench verwendet haben, auf Knopfdruck ein csv-File aus der gerade betrachteten Tabelle zu erzeugen (siehe Bild 5). Die Tabellen, die uns für spätere Auswertungen interessant schienen, waren:

- “customers”
- “products”
- “orders”
- “order_details”

Hierbei wurden an dieser Stelle keine Einschränkung der Daten vorgenommen, sondern das dem SQL-Befehl
“SELECT * FROM ‘TABLE’ ;” (mit ‘TABLE’ gleich einer der vier von uns
ausgesuchten Tabellen) entsprechenden Resultat in das csv-Format exportiert.
Die entsprechenden csv-Files sind dem zip-File
“skripte_und_daten_CAS_DB_DWH.zip” beigefügt, das im Email-Anhang
mitverschickt wurde.

3. Erstellung der staging-area

Zunächst ein paar Worte zu unserem Umgang mit der staging-area: Zur effizienteren Arbeitsaufteilung haben wir die Aufgaben der Einrichtung der staging-area und der Erzeugung des multidimensionalen Modells getrennt. Aus Gründen der Einfachheit und der Möglichkeit, schneller mit der Arbeit zu beginnen, haben wir anfangs die Daten für die verschiedenen Transformationen (Reparatur der korrupten Daten, Befüllen des multidimensionalen Modells, ...) aus csv-Dateien eingelesen und teils auch wieder als csv-Dateien rausgeschrieben. Die staging-area bestand somit letztlich aus dem Filesystem.

Aus zeitlichen Gründen haben wir darauf verzichtet, dies noch so umzustellen, dass die staging-area in Form einer Datenbank implementiert würde, d.h. dass das Einlesen – bis auf den einen Schritt zur Befüllen der staging-area Datenbank – immer aus Datenbanktabellen vorgenommen würde.

Wir wollen aber gerne demonstrieren, wie theoretisch die Integration der Daten über die Datenbank ausgesehen hätte. Dazu haben wir die staging-area Datenbank erstellt und mit den notwendigen Tabellen befüllt. Nur eben die Integration in den Daten-Flow ist nicht gegeben.

Es folgt eine kurze Erläuterung, wie unsere staging-area-Datenbank erzeugt und befüllt wird.

1. Das Skript zur Erzeugung der leeren Datenbank ist in MySQL-Workbench auszuführen und lautet “create_staging_area_schema.sql”. Bei der Ausführung des Skripts werden die verschiedenen Tabellen mitsamt ihren Variablen erzeugt, die Reihen bleiben dabei jedoch leer (siehe Bild 6). An dieser Stelle werden schon nicht mehr alle Variablen der ursprünglichen Tabellen übernommen, sondern nur noch jene für die späteren Abfragen notwendigen.

```

1 DROP SCHEMA IF EXISTS `northwind_staging_area` ;
2 CREATE SCHEMA IF NOT EXISTS `northwind_staging_area` DEFAULT CHARACTER SET latin1 ;
3 USE `northwind_staging_area` ;
4
5 -----
6 -- Table `northwind_staging_area`.`customers`
7 -----
8 CREATE TABLE IF NOT EXISTS `northwind_staging_area`.`customers` (
18 # INDEX `first_name` (`first_name` ASC),
19 # INDEX `last_name` (`last_name` ASC),
20 # INDEX `zip_postal_code` (`zip_postal_code` ASC),
21 # INDEX `state_province` (`state_province` ASC))
22 ENGINE = InnoDB
23 DEFAULT CHARACTER SET = UTF8MB4;
24
25 -----
26 -- Table `northwind_staging_area`.`orders`
27 -----
28 CREATE TABLE IF NOT EXISTS `northwind_staging_area`.`orders` (
39 ENGINE = InnoDB
40 DEFAULT CHARACTER SET = UTF8MB4;
41
42 -----
43 -- Table `northwind_staging_area`.`products`
44 -----
45 CREATE TABLE IF NOT EXISTS `northwind_staging_area`.`products` (
51 ENGINE = InnoDB
52 DEFAULT CHARACTER SET = UTF8MB4;
53
54 -----
55 -- Table `northwind_staging_area`.`order_details`
56 -----
57 CREATE TABLE IF NOT EXISTS `northwind_staging_area`.`order_details` (
75 ENGINE = InnoDB
76 DEFAULT CHARACTER SET = UTF8MB4;

```

Bild 6: Skript "create_staging_area_schema.sql". Für eine bessere Übersicht sind die Abschnitte für die Erzeugung der einzelnen Spalten zusammengeklappt.

2. Das Befüllen der Datenbank erfolgt mittels des Pentaho Data Integration (PDI) Spoon Skripts "add_data_to_staging_area.ktr" (siehe Bild 7). Jedes der csv-Files wird einzeln mit einem "CSV Input"-Schritt eingelesen und jeweils mit einem "Insert / Update"-Schritt in die Datenbank geschrieben. Es hat sich hierbei gezeigt, dass durch die Einschränkung durch die Verlinkung der Tabellen untereinander durch Schlüssel-Fremdschlüssel-Beziehungen, das gleichzeitige Befüllen der Tabellen eine Fehlermeldung provoziert. Um dies zu vermeiden, muss man zunächst die Verbindung zwischen "read orders" und "Insert orders into staging area" sowie diejenige zwischen "read order details" und "Insert order details into staging area" deaktivieren und die Spoon-Transformation laufen lassen. Nachdem dies erfolgreich geschehen ist, kann man die beiden deaktivierten Verbindungen wieder aktivieren und die Transformation erneut starten. Danach ist die staging-area Datenbank gefüllt.

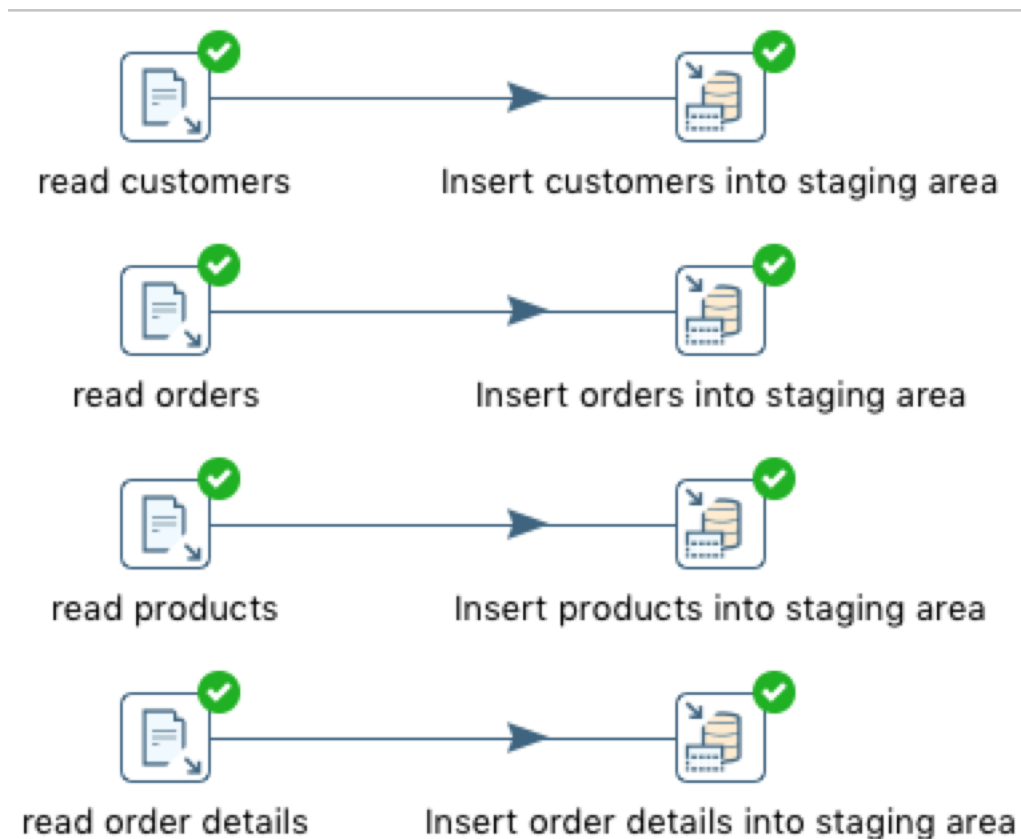


Bild 7: Transformation in Spoon zum Befüllen der staging-area-Datenbank nach erfolgreicher Ausführung.

4. Reparieren der korrupten Daten

Um – wie in der Aufgabenstellung angegeben – herauszufinden, inwiefern sich die Einträge der identischen Produkte unterscheiden, wurde ein kleines Skript namens "study_products_table.R" in der Programmiersprache R verwendet. Dieses ist auch Teil des Lieferumfangs unseres zip-Files. Zu seiner Ausführung ist die Installation des Pakets "dplyr" nötig. Das Skript tut prinzipiell folgendes:

1. Einlesen der csv-Datei "products.csv".
2. Es erzeugt einen logischen Vektor mit der Information, wann ein Produktname zum mindestens zweiten Mal auftritt (man sieht hier, dass nur die letzten 3 Stellen im Vektor "TRUE" sind)
3. Da ein Datensatz von allen doppelten Einträgen (inklusive des ersten Auftretens eines später wiederholten Eintrags) erwünscht ist, wird ein filternder "semi-join()" des ursprünglichen Datensatzes gegen den Datensatz mit den doppelt vorkommenden Produktnamen gemacht. Der Key ist der Produktname. Hierbei entsteht der Datensatz mit den doppelten Einträgen. Dieser hat sechs Zeilen.
4. Mit dem Befehl "View()" wird der Datensatz visuell dargestellt (siehe Bild 8).
5. Es zeigt sich, dass drei verschiedene Produkte jeweils doppelt vorkommen.
6. Um sicher zu gehen, dass auch wirklich alle Einträge bis auf "id" doppelt sind, wird für die drei Produkte mit dem Befehl "identical()" jeweils genau das überprüft – mit Erfolg. Das heißt also, dass diese drei Zeilen bis auf den

Eintrag "id" identisch sind, und die letzten drei Zeilen des ursprünglichen Datensatzes (bzw. allgemeiner, die doppelten) entfernt werden müssen.

supplier_ids	id	product_code	product_name	description	standard_cost	list_price	reorder_level	target
2;6	6	NWTJP-6	Northwind Traders Boysenberry Spread	NULL	18.75	25.0	25	
2;6	106	NWTJP-6	Northwind Traders Boysenberry Spread	NULL	18.75	25.0	25	
6	92	NWTCFV-92	Northwind Traders Green Beans	NULL	1.00	1.2	10	
6	192	NWTCFV-92	Northwind Traders Green Beans	NULL	1.00	1.2	10	
2;6	20	NWTJP-6	Northwind Traders Marmalade	NULL	60.75	81.0	10	
2;6	120	NWTJP-6	Northwind Traders Marmalade	NULL	60.75	81.0	10	

Bild 8: Ausschnitt (Reihen vollständig, Spalten unvollständig) aus Datensatz doppelter Einträge, extrahiert mittels "R" aus der Tabelle Produkte der Northwind-Datenbank.

Wie schon erwähnt, passiert das Reparieren der fehlerhaften Daten bei unserer Lösung ausserhalb der staging-area Datenbank. Die Daten werden von dem Spoon-Skript "staging_area-v2.ktr" aus den csv-Files "products.csv" und "order_details.csv" eingelesen und die reparierten – und zumindest im Falle der "order details"-Tabelle auch um weitere Variablen aus den anderen csv-Files ergänzten – Daten werden in die csv-Files "products_corrected.csv" und "order_details_corrected.csv" geschrieben. Das Hinzufügen von Spalten zur Tabelle der Bestelldetails erfolgt im Hinblick auf eine Vereinfachung der Überführung der Daten in das Sternschema.

Die Transformationen zur Behebung der Datenprobleme in der ursprünglichen Northwind-Datenbank laufen folgendermassen ab (siehe auch Bild 9):

1. Produkt-Tabelle
 - 1.1. Einlesen der csv-Datei "products.csv"
 - 1.2. Sortierung der Zeilen der Tabelle nach Produktnamen mit einem "Sort rows"-Schritt. Dies ist notwendig, da der folgende Schritt nicht funktioniert, falls doppelte Reihen nicht nacheinander kommen.
 - 1.3. Entfernen der doppelten Einträge mit einem "Unique rows"-Schritt. Hierbei werden alle Variablen bis auf die Variable "id" verglichen.
 - 1.4. Sortierung der Zeilen der Tabelle nach "id", so wie sie ursprünglich geordnet waren.
 - 1.5. Schreiben der Daten in das csv-File "products_corrected.csv"
2. Bestelldetails-Tabelle
 - 2.1. Einlesen der csv-Datei "order_details.csv".
 - 2.2. Hinzufügen der Variablen "customer_id", "employee_id" (diese Variable ist in der staging-area Datenbank nicht vorhanden. Sie wird aber auch für die Auswertungen nicht benötigt), "order_date", "payment_type" aus der Tabelle "orders" mittels eines Schrittes vom Typ "Stream Value Lookup".
 - 2.3. Hinzufügen der Variable "standard_cost" aus der Tabelle "products"
 - 2.4. Hinzufügen der Variablen "city", "state_province" und "country_region" aus der Tabelle "customer"
 - 2.5. Kreieren einer neuen Spalte "discount_corr", die den Betragswert der Variable "discount" enthält. Dies geschieht durch einen Schritt vom Typ "Calculator".

2.6. Mit einem Schritt "Select / Rename values" werden 15 Variablen ausgewählt und gleichzeitig die Variable "discount" durch "discount_corr" ersetzt.

2.7. Im letzten Schritt werden die Daten in das csv-File "order_details_corrected.csv" geschrieben.

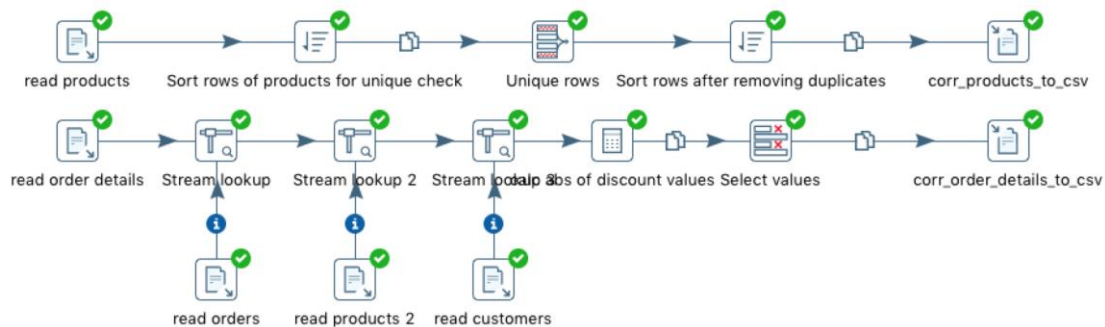


Bild 9: "staging_area-v2.ktr" nach erfolgreichem Durchlauf. Hier werden fehlerhafte Daten korrigiert und im Fall der Bestelldetails auch weitere Spalten hinzugefügt. Am Ende werden die Daten in die zwei im Text erwähnten csv-Files geschrieben.

Wie in Abschnitt 3 erwähnt, sollte das Einlesen von sowohl der "products"- als auch der "order_details"-Tabellen eigentlich aus der staging-area Datenbank durch einen "Table input"-Schritt geschehen. Das Schreiben der beiden Tabellen sollte eigentlich ebenso in die staging-area Datenbank geschehen, entweder mit einem "Dimension lookup / update"-Schritt oder mit einem "Table output"-Schritt.

5. Konzipierung des Sternschemas für das DWH

Für das multidimensionale Modell haben wir ein einfaches Stern-Schema gewählt. Die Faktentabelle im Zentrum basiert auf der Relation "order_details" aus dem Ausgangsschema. In "order_details" finden sich die Einzelpositionen aller Bestellungen, über die sich geeignete Auswertungen durchführen lassen. So sollen später zum Beispiel Umsätze pro Monat oder Gewinnzahlen für bestimmte Produkte ermittelt werden.

Bei den Dimensionen beschränken wir uns auf die Bereiche "Kunde", "Produkt", "Region", "Bestellung" und "Zeit". Zusätzlich wären Auswertungen über die Spediteure, die Lieferanten und die zuständigen Sachbearbeiter denkbar, was entsprechende weitere Dimensionen erfordern würde.

Bei der Dimension "Region" wäre eine Erweiterung auf das Schneeflocken-Schema denkbar gewesen, da es sich bei Ländern, Bundesstaaten und Städten um unterschiedliche Granularitäten räumlicher Angaben handelt. Allerdings erlaubt die geringe Datenmenge ohnehin keine fein granularen Auswertungen, so dass wir auf diese Unterteilung verzichtet haben.

Bild 10 zeigt das vollständige Schema.

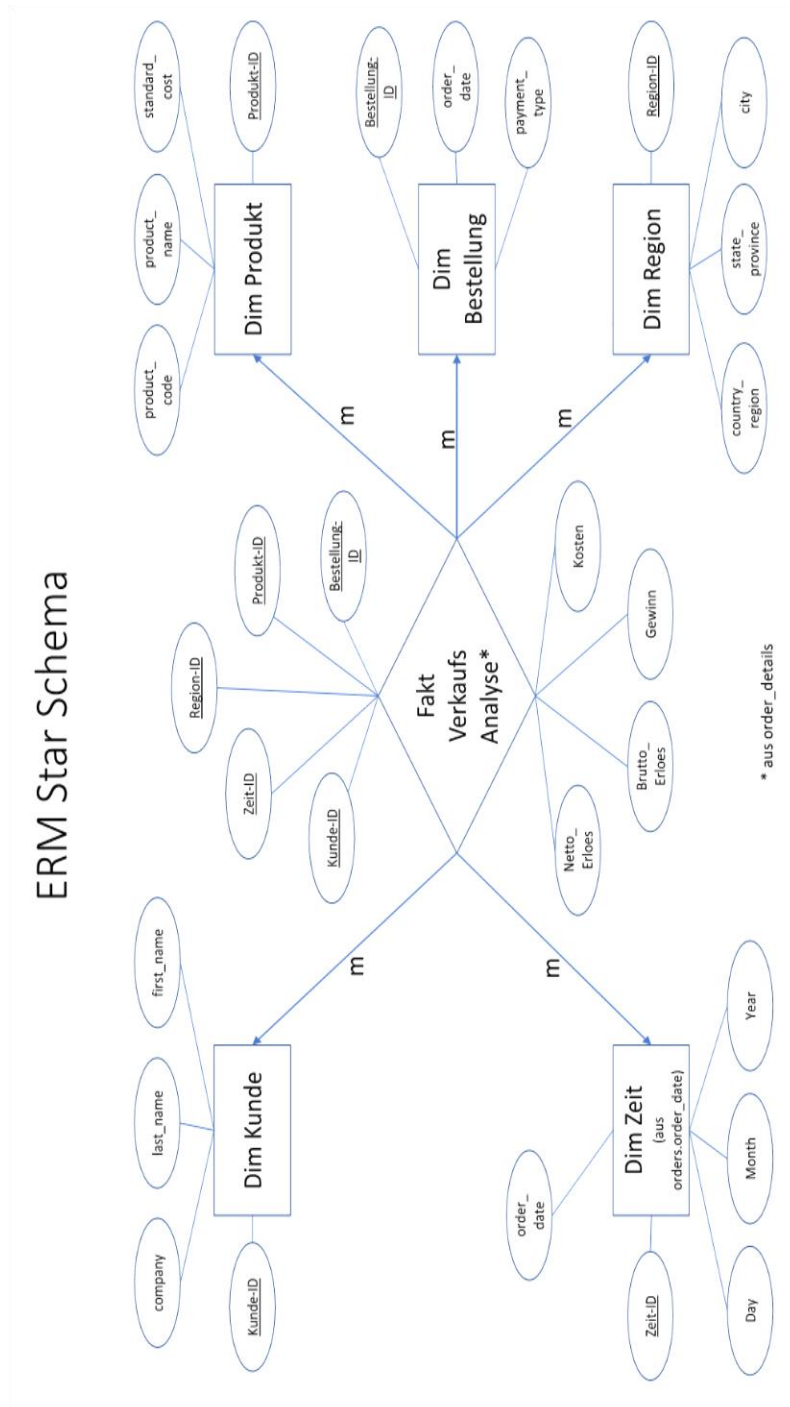


Bild 10: ERM-Diagramm unseres Sternschema-Modells.

6. Erstellung und Befüllen des DWH

Das Ziel-Schema wurde direkt durch SQL-Befehle in MySQL implementiert. Die notwendigen Erweiterungen, insbesondere für "Slowly Changing Dimensions" werden durch Pentaho vorgenommen. Hier die SQL-Befehle zum Erstellen des Data Warehouse:

```

DROP DATABASE IF EXISTS NORTHWIND_DWH;
CREATE DATABASE NORTHWIND_DWH;
USE NORTHWIND_DWH;

```

```

CREATE TABLE kunde_dim (
    KUNDE_ID_DIM      INT(11) NOT NULL AUTO_INCREMENT,
    CUSTOMER_ID       INT(11) NOT NULL,
    COMPANY            VARCHAR(50) NULL DEFAULT NULL,
    FIRST_NAME        VARCHAR(50) NULL DEFAULT NULL,
    LAST_NAME         VARCHAR(50) NULL DEFAULT NULL,
    CONSTRAINT PK_KUNDE_DIM PRIMARY KEY (KUNDE_ID_DIM)
);

```

```

CREATE TABLE bestellung_dim (
    BESTELLUNG_ID_DIM INT(11) NOT NULL AUTO_INCREMENT,
    BESTELLUNG_ID      INT(11) NOT NULL,
    PAYMENT_TYPE       VARCHAR(11) NULL DEFAULT NULL,
    ORDER_DATE         DATE,
    CONSTRAINT PK_BESTELLUNG_DIM PRIMARY KEY
                        (BESTELLUNG_ID_DIM)
);

```

```

CREATE TABLE produkt_dim (
    PRODUKT_ID_DIM    INT(11) NOT NULL AUTO_INCREMENT,
    PRODUKT_ID        INT(11) NOT NULL,
    PRODUKT_CODE      VARCHAR(25) NULL DEFAULT NULL,
    PRODUKT_NAME      VARCHAR(50) NULL DEFAULT NULL,
    STANDARD_COST     DECIMAL(19 , 4 ) NULL DEFAULT '0.0000',
    CONSTRAINT PK_PRODUKT_DIM PRIMARY KEY (PRODUKT_ID_DIM)
);

```

```

CREATE TABLE zeit_dim (
    ZEIT_ID_DIM       INT(11) NOT NULL AUTO_INCREMENT,
    ORDER_DATE        DATE,
    ORDER_DAY         INTEGER,
    ORDER_MONTH       INTEGER,
    ORDER_YEAR        INTEGER,
    CONSTRAINT PK_ZEIT_DIM PRIMARY KEY (ZEIT_ID_DIM)
);

```

```

CREATE TABLE region_dim (
    REGION_ID_DIM     INT(11) NOT NULL AUTO_INCREMENT,
    CITY              VARCHAR(25) NULL DEFAULT NULL,
    STATE_PROVINCE    VARCHAR(2) NULL DEFAULT NULL,
    COUNTRY_REGION    VARCHAR(15) NULL DEFAULT NULL,
    CONSTRAINT PK_REGION_DIM PRIMARY KEY (REGION_ID_DIM)
);

```

```

CREATE TABLE verkauf_fact (
    KUNDE_ID_DIM      INT(11) NOT NULL,

```

```

PRODUKT_ID_DIM      INT(11) NOT NULL,
ZEIT_ID_DIM          INT(11) NOT NULL,
REGION_ID_DIM        INT(11) NOT NULL,
BESTELLUNG_ID_DIM    INT(11) NOT NULL,
BRUTTO_ERLOES        INTEGER,
NETTO_ERLOES         INTEGER,
KOSTEN               INTEGER,
GEWINN               INTEGER
);

ALTER TABLE verkauf_fact
ADD CONSTRAINT verkauf_kunde_fk
FOREIGN KEY (KUNDE_ID_DIM) REFERENCES northwind_dwh.kunde_dim
(KUNDE_ID_DIM);

ALTER TABLE verkauf_fact
ADD CONSTRAINT verkauf_produkt_fk
FOREIGN KEY (PRODUKT_ID_DIM) REFERENCES
northwind_dwh.produkt_dim (PRODUKT_ID_DIM);

ALTER TABLE verkauf_fact
ADD CONSTRAINT verkauf_region_fk
FOREIGN KEY (REGION_ID_DIM) REFERENCES region_dim
(REGION_ID_DIM);

ALTER TABLE verkauf_fact
ADD CONSTRAINT verkauf_bezahlung_fk
FOREIGN KEY (BESTELLUNG_ID_DIM) REFERENCES bestellung_dim
(BESTELLUNG_ID_DIM);

```

Das Befüllen des Schemas mit den Daten erfolgt über zwei Pentaho Transformationen:

- 1) "star-schema.ktr": diese Transformation befüllt die Dimensionstabellen
- 2) "facts-table.ktr": durch diese Transformation wird die Fakten-Tabelle befüllt

Befüllen der Dimensionstabellen

Wir beginnen mit dem Befüllen der Produkte-Dimension. Diese besteht aus zweif Schritten (siehe Bild Bild 11):



Bild 11: Schritte zum Befüllen der Tabelle "produkt_dim" der Datenbank "NORTHWIND_DWH" aus dem Spoon-File "star-schema.ktr".

Der Schritt "Produkte korrigiert" liest die Informationen aus der Staging Area ein, d.h. das csv-File, das wir in den Schritten 2 bis 4 erstellt haben. Im

konkreten Fall das File “products-corrected.csv”. Dazu wird in Pentaho das Element “Input -> CSV file input” gewählt und mit einem Doppelklick geöffnet. Der Schritt wird umbenannt, der Pfad auf das CSV-File selektiert und als Delimiter das Komma gesetzt (siehe Bild 12).

Step name: **Produkte korrigiert**

Filename: C:\Users\Bernd\Documents\CAS Information Engineering\DB_DWH\Leistungsnachweis\products_corrected.csv

Delimiter: ,

Enclosure: "

NIO buffer size: 50000

Lazy conversion? ☐

Header row present? ☒

Add filename to result? ☐

The row number field name (optional):

Running in parallel? ☐

New line possible in fields? ☐

File encoding:

#	Name	Type	Format	Length	Precision	Currency	Decimal	Group	Trim type
1	supplier_ids	String		3		SFr.	.	*	both
2	id	Integer	#	15	0	SFr.	.	*	none
3	product_code	String		9		SFr.	.	*	both
4	product_name	String		40		SFr.	.	*	both
5	description	String		4		SFr.	.	*	both
6	standard_cost	Number	##	4	1	SFr.	.	*	none
7	list_price	Number	##	4	1	SFr.	.	*	none
8	reorder_level	Integer	#	15	0	SFr.	.	*	none
9	target_level	Integer	#	15	0	SFr.	.	*	none
10	quantity_per_unit	String		20		SFr.	.	*	both
11	discontinued	Integer	#	15	0	SFr.	.	*	none
12	minimum_reorder_quantity	String		4		SFr.	.	*	both
13	category	String		25		SFr.	.	*	both
14	attachments	Boolean				SFr.	.	*	none

Bild 12: Spoon-Schritt "CSV Input" für Tabelle "produkt_dim".

Nachdem mit “Get Fields” die Attribute aus dem CSV File identifiziert wurden, werden die Datentypen überprüft. Wo nötig werden Anpassungen vorgenommen. Vor allem bei String Feldern könnten in Zukunft längere Werte vorkommen, so dass sich unter Umständen eine Erhöhung der Feldlänge lohnt. Ein Blick auf das CSV File zeigt zudem, dass sich durch den Korrekturschritt im CSV File Leerzeichen eingeschlichen haben (siehe Bild 13).

products_corrected.csv - Editor

Datei Bearbeiten Format Ansicht ?

reorder_quantity	category	attachments	supplier_ids	id	product_code	product_name	description	standard_cost	list_price	reorder_level	target_level	quantity_per_unit	discontinued	minimum
4	,1,NWTB-1	,Northwind Traders	Chai	,NULL	13.5	18	10	40	10	boxes x 20 bags	,0,10	,Beverages	,	
10	,3,NWTCO-3	,Northwind Traders	Syrup	,NULL	7.5	10	25	100	12	- 550 ml bottles	,0,25	,Condiments	,	
10	,4,NWTCO-4	,Northwind Traders	Cajun Seasoning	,NULL	16.5	22	10	40	48	- 6 oz jars	,0,10	,Condiments	,	
10	,5,NWTCO-5	,Northwind Traders	Olive Oil	,NULL	16	21.4	10	40	36	boxes	,0,10	,Oil	,	
2;6,6	,NWTJP-6	,Northwind Traders	Boysenberry Spread	,NULL	18.8	25	25	100	12	- 8 oz jars	,0,25	,Jams, Preserves	,	
2	,7,NWTFN-7	,Northwind Traders	Dried Pears	,NULL	22.5	30	10	40	12	- 1 lb pkgs.	,0,10	,Dried Fruit & Nuts	,	
8	,8,NWTS-8	,Northwind Traders	Curry Sauce	,NULL	30	40	10	40	12	- 12 oz jars	,0,10	,Sauces	,	
2;6,14	,NWTDFN-14	,Northwind Traders	Walnuts	,NULL	17.4	23.2	10	40	40	- 100 g pkgs.	,0,10	,Dried Fruit & Nuts	,	
6	,17,NWTCFV-17	,Northwind Traders	Fruit Cocktail	,NULL	29.2	39	10	40	15.25	OZ	,0,10	,Canned Fruit & Vegetables	,	
1	,19,NWTBGM-19	,Northwind Traders	Chocolate Biscuits Mix	,NULL	6.9	9.2	5	20	10	boxes x 12 pieces	,0,5	,Baked Goods & Mixes	,	
2;6,20	,NWTJP-6	,Northwind Traders	Marmalade	,NULL	60.8	81	10	40	30	gift boxes	,0,10	,Jams, Preserves	,	
1	,21,NWTBGM-21	,Northwind Traders	Scones	,NULL	7.5	10	5	20	24	pkgs. x 4 pieces	,0,5	,Baked Goods & Mixes	,	
4	,34,NWTB-34	,Northwind Traders	Beer	,NULL	10.5	14	15	60	24	- 12 oz bottles	,0,15	,Beverages	,	
7	,40,NWTM-40	,Northwind Traders	Crab Meat	,NULL	13.8	18.4	30	120	24	- 4 oz tins	,0,30	,Canned Meat	,	
6	,41,NWTSO-41	,Northwind Traders	Clam Chowder	,NULL	7.2	9.7	10	40	12	- 12 oz cans	,0,10	,Soups	,	
3;4,43	,NWTB-43	,Northwind Traders	Coffee	,NULL	34.5	46	25	100	16	- 500 g tins	,0,25	,Beverages	,	
10	,48,NWTCA-48	,Northwind Traders	Chocolate	,NULL	9.6	12.8	25	100	10	pkgs	,0,25	,Candy	,	
2	,51,NWTFN-51	,Northwind Traders	Dried Apples	,NULL	39.8	53	10	40	50	- 300 g pkgs.	,0,10	,Dried Fruit & Nuts	,	
1	,52,NWTG-52	,Northwind Traders	Long Grain Rice	,NULL	5.2	7	25	100	16	- 2 kg boxes	,0,25	,Grains	,	
1	,56,NWTP-56	,Northwind Traders	Gnocchi	,NULL	28.5	38	30	120	24	- 250 g pkgs.	,0,30	,Pasta	,	
1	,57,NWTP-57	,Northwind Traders	Ravioli	,NULL	14.6	19.5	20	80	24	- 250 g pkgs.	,0,20	,Pasta	,	
8	,65,NWTS-65	,Northwind Traders	Hot Pepper Sauce	,NULL	15.8	21.1	10	40	32	- 8 oz bottles	,0,10	,Sauces	,	
8	,66,NWTS-66	,Northwind Traders	Tomato Sauce	,NULL	12.8	17	20	80	24	- 8 oz jars	,0,20	,Sauces	,	
5	,72,NWTD-72	,Northwind Traders	Mozzarella	,NULL	26.1	34.8	10	40	24	- 200 g pkgs.	,0,10	,Dairy products	,	
2;6,74	,NWTDFN-74	,Northwind Traders	Almonds	,NULL	7.5	10	5	20	5	kg pkg.	,0,5	,Dried Fruit & Nuts	,	
10	,77,NWTCO-77	,Northwind Traders	Mustard	,NULL	9.8	13	15	60	12	boxes	,0,15	,Condiments	,	
2	,80,NWTFN-80	,Northwind Traders	Dried Plums	,NULL	3.3	5	50	75	1	lb bag	,0,25	,Dried Fruit & Nuts	,	
3	,81,NWTB-81	,Northwind Traders	Green Tea	,NULL	2	3	100	125	20	bags per box	,0,25	,Beverages	,	
1	,82,NWTC-82	,Northwind Traders	Granola	,NULL	2	4	20	100	NULL		,0, NULL	,Cereal	,	

Bild 13: Durch den Korrekturschritt sind im csv-File Produkte-Tabelle Leerzeichen in einigen Spalten erzeugt worden.

Daher wird im Transformationsschritt beim Datentyp String unter "Trim type" die Option "both" gewählt, um die Leerzeichen zu entfernen. Die Leerzeichen scheinen zwar nur am Ende eingefügt zu sein, aber um auf sicher zu gehen werden Leerzeichen am Anfang und am Ende des Strings entfernt. Mit "Preview" werden die Daten nochmal überprüft und dann mit "OK" der Schritt geschlossen.

Im zweiten Transformationsschritt werden aus dem Eingabe-Stream die für die Dimensionstabelle relevanten Attribute extrahiert und dann in die Datenbank geschrieben. Dafür wählen wir das Transformationselement "Data Warehouse -> Dimension lookup/update" und verbinden es mit dem vorherigen Schritt. Ein Doppelklick öffnet folgendes Fenster:

Dimension Lookup / Update

Step name: **Produkte Dimension Update**

Update the dimension? ☒

Connection: DWH [Edit... New... Wizard...]

Target schema: northwind_dwh [Browse...]

Target table: produkt_dim [Browse...]

Commit size: 100

Enable the cache? ☐

Pre-load the cache? ☐

Cache size in rows (0 = cache all):

Keys **Fields**

Key fields (to look up row in dimension):

#	Dimension field	Field in stream
1	PRODUKT_ID	id

Technical key field: PRODUKT_ID_DIM [New name]

Creation of technical key:

☐ Use table maximum + 1

☐ Use sequence

☒ Use auto increment field

Version field: version

Stream Datefield:

Date range start field: date_from Min. year: 1900

Use an alternative start date? ☐ <Select Option>

Table date range end: date_to Max. year: 2199

OK Cancel Get Fields SQL

Help

Wir lassen Pentaho den Surrogat-Schlüssel "PRODUKT_ID_DIM" befüllen und verwenden den Auto Increment der Datenbank.

Um die für uns interessanten Felder der Dimension zu wählen und "Slowly Changing Dimensions" vom Typ 2 zu implementieren, wechseln wir ins Tab "Fields":

Dimension Lookup / Update

Step name:

Update the dimension? ☒

Connection:

Target schema:

Target table:

Commit size:

Enable the cache? ☐

Pre-load the cache? ☐

Cache size in rows (0 = cache all)

Keys **Fields**

Lookup/Update fields

#	Dimension field	Stream field to compare with	Type of dimension update
1	PRODUKT_CODE	product_code	Insert
2	PRODUKT_NAME	product_name	Insert
3	STANDARD_COST	standard_cost	Insert

Technical key field: New name:

Creation of technical key

☐ Use table maximum + 1

☐ Use sequence

☒ Use auto increment field

Version field:

Stream Datefield:

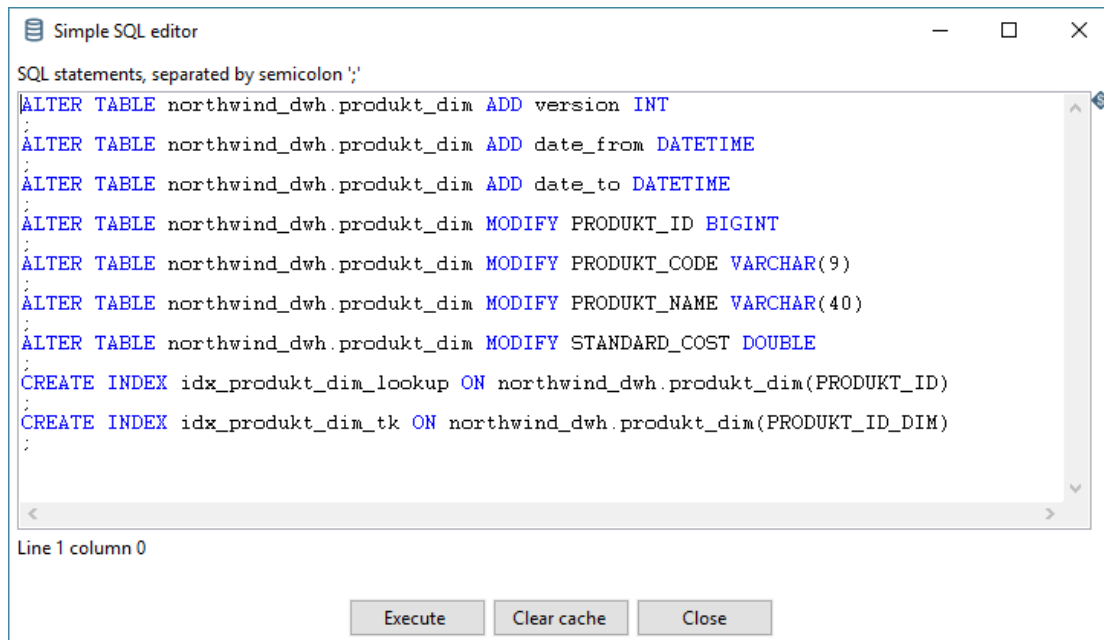
Date range start field: Min. year:

Use an alternative start date? ☐

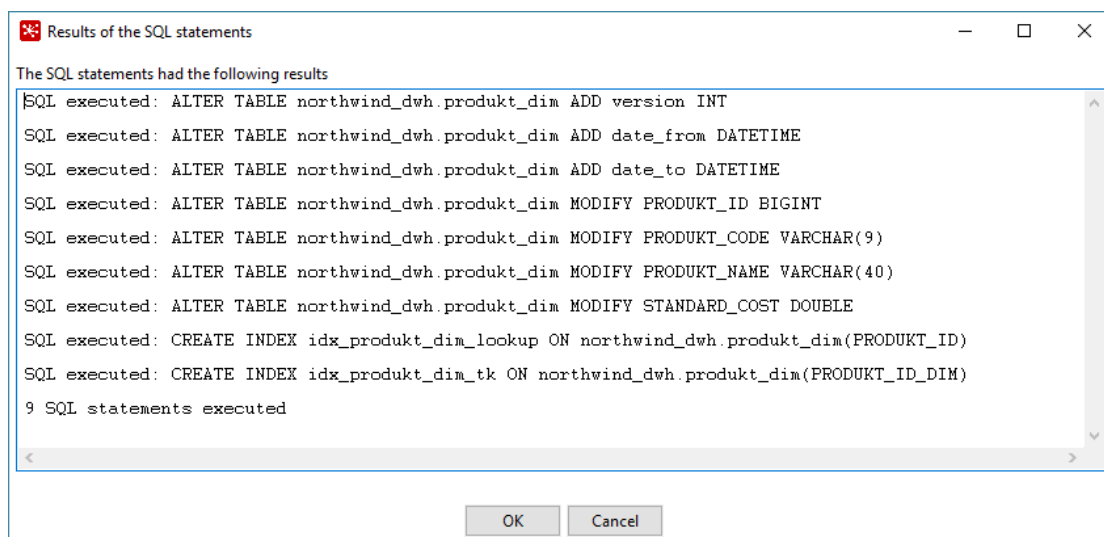
Table date range end: Max. year:

Wie gut zu erkennen ist, beschränken wir uns auf drei Felder aus dem Eingabe-Stream (product_code, product_name und standard_cost) und mappen sie auf die Felder unseres Datenbankschemas. Für die Typ 2 SCD wird als "Type of dimension update" der Wert "Insert" gewählt.

Bevor die Transformation ausgeführt werden kann, muss dass generierte SQL ausgeführt werden. Sonst fehlen die notwendigen Schemaänderungen in der Datenbank. Dazu wird auf den Button "SQL" geklickt:



Ein Klick auf “Execute” führt die notwendigen Änderungen in der Datenbank durch. Pentaho bestätigt uns die erfolgreiche Ausführung:



Danach können die Fenster durch “OK” bzw. “Close” wieder geschlossen werden.

In analoger Weise wird für die Dimensionen “Kunde” und “Bestellung” vorgegangen.

Bei “Region” ist zu beachten, dass dafür in der Ausgangs-DB keine eigene Relation vorliegt. Wir beziehen diese Daten aus der Relation “customers” über die Felder “city”, “state_province” und “country_region”.

CSV Input

Step name:

Filename:

Delimiter:

Enclosure:

NIO buffer size:

Lazy conversion? ☐

Header row present? ☒

Add filename to result ☐

The row number field name (optional):

Running in parallel? ☐

New line possible in fields? ☐

File encoding:

#	Name	Type	Format	Length	Precision	Currency	Decimal	Group	Trim type
1	id	Integer	#	15	0	SFr.	.	'	none
2	city	String		25		SFr.	.	'	both
3	state_province	String		2		SFr.	.	'	both
4	country_region	String		15		SFr.	.	'	both

Wir dürfen für die Identifikation der Zeilen nicht den Schlüssel "id" dieser Relation heranziehen, da wir sonst doppelte Einträge in der Dimensionstabelle erhalten würden. Stattdessen wählen wir die Kombination der drei Attribute als Key für die Identifikation von Datensätzen in der customers-Relation.

Dimension Lookup / Update

Step name:

Update the dimension? ☒

Connection:

Target schema:

Target table:

Commit size:

Enable the cache? ☒

Pre-load the cache? ☐

Cache size in rows (0 = cache all):

Keys Fields

Key fields (to look up row in dimension):

#	Dimension field	Field in stream
1	CITY	city
2	STATE_PROVINCE	state_province
3	COUNTRY_REGION	country_region

Technical key field: New name:

Creation of technical key

☐ Use table maximum + 1

☐ Use sequence

☒ Use auto increment field

Version field:

Stream Datefield:

Date range start field: Min. year:

Use an alternative start date? ☐

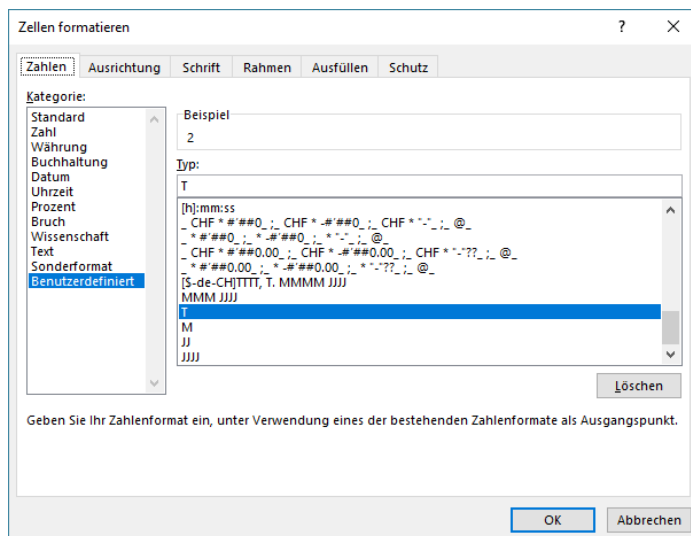
Table date range end: Max. year:

Zusätzliche Felder sind nicht zu wählen, da wir in der Dimensionstabelle nur an diesen Werten interessiert sind.

Für die Zeit-Dimension schauen wir zuerst in den Ausgangsdaten, welcher Zeitraum relevant ist. Offensichtlich handelt es sich um das Jahr 2006. Die Dimensionstabelle soll also mit den Tagen dieses Jahres gefüllt werden. Der einfachste Ansatz wäre gewesen, eine solche Tabelle mit ein paar Mausklicks in Excel zu erzeugen, als CSV zu speichern und dieses dann in gewohnter Weise einzulesen und in das DWH zu schreiben.

	A	B	C	D	E
1	Datum	Tag	Monat	Jahr	
2	01.01.2006	1	1	2006	
3	02.01.2006	2	1	2006	
4	03.01.2006	3	1	2006	
5	04.01.2006	4	1	2006	
6	05.01.2006	5	1	2006	
7	06.01.2006	6	1	2006	
8	07.01.2006	7	1	2006	
9	08.01.2006	8	1	2006	
10	09.01.2006	9	1	2006	
11	10.01.2006	10	1	2006	
12	11.01.2006	11	1	2006	
13	12.01.2006	12	1	2006	

Die Spalte B lässt sich leicht durch eine benutzerdefinierte Formatierung füllen. Gleiches gilt dann für die Spalten C und D.



Statt des Ansatzes mit Excel haben wir einen Ansatz mit Pentaho Transformationsschritten gewählt:



Im ersten Schritt werden mit dem Element "Generate Rows" 365 Zeilen erzeugt, für jeden Tag des Jahres eine, beginnend am 1 Januar 2006:

Generate Rows

Step name:

Limit:

Never stop generating rows: ☐

Interval in ms (delay):

Current row time field name:

Previous row time field name:

Fields:

#	Name	Type	Format	Length	Precision	Currency	Decimal	Group	Value	Set empty string?
1	StartDate	Date	dd-MM-yyyy						01-01-2006	N

Nach diesem Schritt steht zunächst in jeder Zeile der 1. Januar 2006. Im zweiten Schritt wird ein Zähler generiert ("Add Sequence"), der für jedes Element, das er aus dem Eingangsstream erhält jeweils um 1 hochzählt. Wir erhalten also eine Sequenz von 0 bis 364

Get Value From Sequence

Step name:

Name of value:

Use a database to generate the sequence

Use DB to get sequence? ☐

Connection:

Schema name:

Sequence name:

Use a transformation counter to generate the sequence

Use counter to calculate sequence? ☒

Counter name (optional):

Start at value:

Increment by:

Maximum value:

Im dritten Schritt wird dieser Zähler auf die Datumssequenz angewendet ("Calculator" Schritt). Wichtig ist es, bei "Conversion Mask" das gleiche Format wie im ersten Schritt zu definieren.

Calculator

Step name:

Fields:

#	New field	Calculation	Field A	Field B	Field C	Value type	Length	Precision	Remove	Conversion mask	Decimal symbol	Grouping symbol	Currency
1	Sales_Date	Date A + B Days	StartDate	increment_date		Date			N	dd-MM-yyyy			

Nun wird der Datenstrom auf das Feld reduziert, um das es eigentlich geht: das Datumsfeld. Das geschieht mit Hilfe des "Select Values" Elements.

The screenshot shows the 'Select / Rename values' dialog box. The 'Step name' field contains 'Select values'. The 'Fields' table has the following data:

#	Fieldname	Rename to	Length	Precision
1	Sales_Date			

The 'Include unspecified fields, ordered by' checkbox is unchecked. The 'Get fields to select' and 'Edit Mapping' buttons are on the right. The 'OK', 'Cancel', and 'Help' buttons are at the bottom.

Für die Sequenz an Datumsfeldern (ein Element pro Tag im Jahr) können nun die anderen Felder der Zeit-Dimensionstabelle berechnet werden mithilfe eines "Calculator" Schritts:

Calculator

Step name: Calculate Additional Fields

Fields:

#	New field	Calculation	Field A	Field B	Field C	Value type	Length	Precision	Remove	Conversion mask	Decimal symbol	Grouping symbol	Currency
1	Day	Day of month of date A	Sales_Date			Integer			N				
2	Month	Month of date A	Sales_Date			Integer			N				
3	Year	Year of date A	Sales_Date			Integer			N				

< >

Help OK Cancel

Als abschliessender Schritt wird die Dimensionstabelle wie zuvor gefüllt:

Dimension Lookup / Update

Step name:

Update the dimension? ☒

Connection:

Target schema:

Target table:

Commit size:

Enable the cache? ☒

Pre-load the cache? ☐

Cache size in rows (0 = cache all):

Keys Fields

Lookup/Update fields

#	Dimension field	Stream field to compare with	Type of dimension update
1	ORDER_DAY	Day	Update
2	ORDER_MONTH	Month	Update
3	ORDER_YEAR	Year	Update

Technical key field: New name:

Creation of technical key

☐ Use table maximum + 1

☐ Use sequence

☒ Use auto increment field

Version field:

Stream Datefield:

Date range start field: Min. year:

Use an alternative start date? ☐

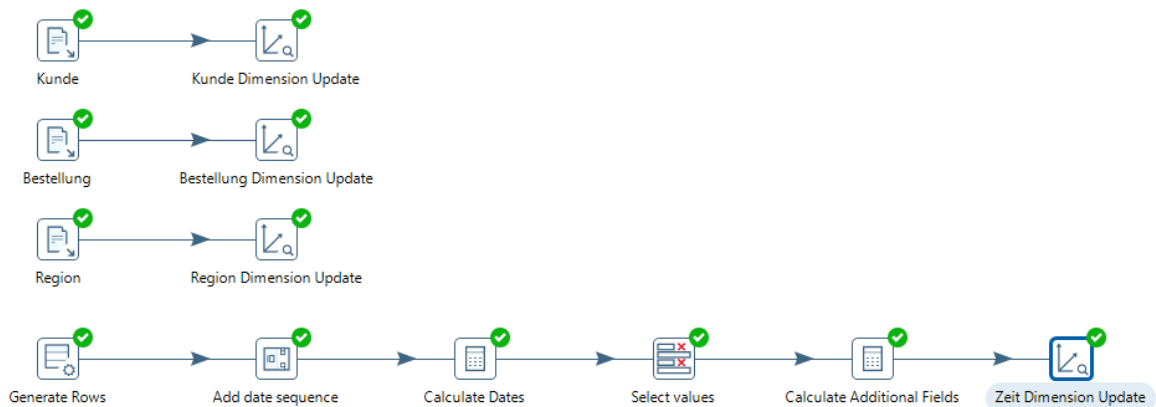
Table date range end: Max. year:

Keys Fields

Key fields (to look up row in dimension):

#	Dimension field	Field in stream
1	ORDER_DATE	Sales_Date

Nun kann die komplette Transformation ausgeführt werden:



Ein exemplarischer Blick in die Datenbank auf die Dimensionstabellen am Beispiel der Zeit-Dimension:

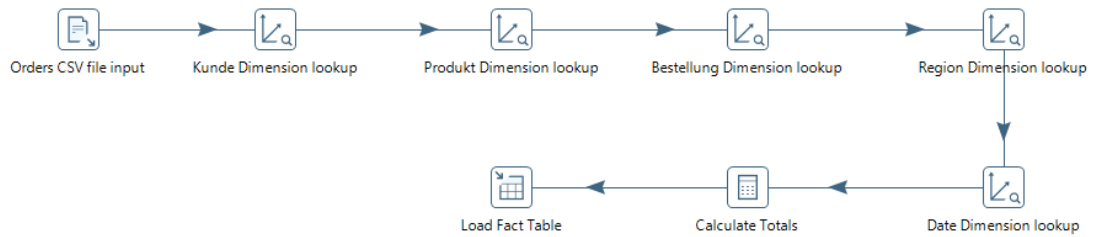
1 • `SELECT * FROM northwind_dwh.zeit_dim;`

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content: `⌘`

	ZEIT_ID_DIM	ORDER_DATE	ORDER_DAY	ORDER_MONTH	ORDER_YEAR	version	date_from	date_to
	16	2006-01-15	15	1	2006	1	1900-01-01 00:00:00	2200-01-01 00:00:00
	17	2006-01-16	16	1	2006	1	1900-01-01 00:00:00	2200-01-01 00:00:00
	18	2006-01-17	17	1	2006	1	1900-01-01 00:00:00	2200-01-01 00:00:00
	19	2006-01-18	18	1	2006	1	1900-01-01 00:00:00	2200-01-01 00:00:00
	20	2006-01-19	19	1	2006	1	1900-01-01 00:00:00	2200-01-01 00:00:00
	21	2006-01-20	20	1	2006	1	1900-01-01 00:00:00	2200-01-01 00:00:00
	22	2006-01-21	21	1	2006	1	1900-01-01 00:00:00	2200-01-01 00:00:00
	23	2006-01-22	22	1	2006	1	1900-01-01 00:00:00	2200-01-01 00:00:00
	24	2006-01-23	23	1	2006	1	1900-01-01 00:00:00	2200-01-01 00:00:00
	25	2006-01-24	24	1	2006	1	1900-01-01 00:00:00	2200-01-01 00:00:00
	26	2006-01-25	25	1	2006	1	1900-01-01 00:00:00	2200-01-01 00:00:00
	27	2006-01-26	26	1	2006	1	1900-01-01 00:00:00	2200-01-01 00:00:00
	28	2006-01-27	27	1	2006	1	1900-01-01 00:00:00	2200-01-01 00:00:00
	29	2006-01-28	28	1	2006	1	1900-01-01 00:00:00	2200-01-01 00:00:00
	30	2006-01-29	29	1	2006	1	1900-01-01 00:00:00	2200-01-01 00:00:00
	31	2006-01-30	30	1	2006	1	1900-01-01 00:00:00	2200-01-01 00:00:00
	32	2006-01-31	31	1	2006	1	1900-01-01 00:00:00	2200-01-01 00:00:00
	33	2006-02-01	1	2	2006	1	1900-01-01 00:00:00	2200-01-01 00:00:00
	34	2006-02-02	2	2	2006	1	1900-01-01 00:00:00	2200-01-01 00:00:00
	35	2006-02-03	3	2	2006	1	1900-01-01 00:00:00	2200-01-01 00:00:00
	36	2006-02-04	4	2	2006	1	1900-01-01 00:00:00	2200-01-01 00:00:00
	37	2006-02-05	5	2	2006	1	1900-01-01 00:00:00	2200-01-01 00:00:00
	38	2006-02-06	6	2	2006	1	1900-01-01 00:00:00	2200-01-01 00:00:00
	39	2006-02-07	7	2	2006	1	1900-01-01 00:00:00	2200-01-01 00:00:00

Erzeugen der Fakten

Wie eingangs erwähnt basiert die Faktentabelle auf “order_details” aus der Ausgangs-DB. Diese Relation wird im ersten Schritt eingelesen.



Auch hier schauen wir wieder, dass allfällige Leerzeichen am Anfang und Ende von String entfernt werden, damit das Matching sauber funktioniert.

CSV Input

Step name: Orders CSV file input

Filename: C:\Users\Bernd\Documents\CAS Information Engineering\DB_DWH\Leistungsnachweis\order_details_corrected.c Browse...

Delimiter: , Insert TAB

Enclosure: "

NIO buffer size: 50000

Lazy conversion? ☒

Header row present? ☒

Add filename to result ☐

The row number field name (optional)

Running in parallel? ☐

New line possible in fields? ☐

File encoding

#	Name	Type	Format	Length	Precision	Currency	Decimal	Group	Trim type
7	date_allocated	String		4		SFr.	.	'	none
8	discount	Integer	#	15	0	SFr.	.	'	none
9	employee_id	Integer	#	15	0	SFr.	.	'	none
10	customer_id	Integer	#	15	0	SFr.	.	'	none
11	order_date	Date	yyyy-MM-dd HH:mm:ss			SFr.	.	'	none
12	payment_type	String		11		SFr.	.	'	both
13	city	String		14		SFr.	.	'	both
14	country_region	String		3		SFr.	.	'	both
15	state_province	String		2		SFr.	.	'	both

Help OK Get Fields Preview Cancel

Danach werden im “Dimension lookup/update” Schritten für jeden Datensatz die Surrogatschlüssel aus den Dimensionstabellen gelesen.

Dimension Lookup / Update

Step name: Kunde Dimension lookup

Update the dimension? ☐

Connection: DWH Edit... New... Wizard...

Target schema: Browse...

Target table: kunde_dim Browse...

Commit size: 100

Enable the cache? ☐

Pre-load the cache? ☐

Cache size in rows (0 = cache all):

Keys **Fields**

Key fields (to look up row in dimension):

#	Dimension field	Field in stream
1	CUSTOMER_ID	customer_id

Technical key field: KUNDE_ID_DIM New name

Creation of technical key

☒ Use table maximum + 1

☐ Use sequence

☐ Use auto increment field

Version field: version

Stream Datefield:

Date range start field: date_from Min. year 1900

Use an alternative start date? ☐ <Select Option>

Table date range end: date_to Max. year 2199

OK Cancel Get Fields SQL

? Help

Für die abschliessende Kalkulation der für die Faktentabelle definierten Werte (Netto_Erloes, Brutto_Erloes, Gewinn und Kosten) wird aus den Produkten noch der Wert für die Standard-Kosten in den Datenstream eingefügt.

Dimension Lookup / Update

Step name: Produkt Dimension lookup

Update the dimension? ☐

Connection: DWH Edit... New... Wizard...

Target schema: northwind_dwh Browse...

Target table: produkt_dim Browse...

Commit size: 100

Enable the cache? ☐

Pre-load the cache? ☐

Cache size in rows (0 = cache all):

Keys **Fields**

Lookup/Update fields

#	Dimension field	New name of output field	Type of return field
1	STANDARD_COST	standard_cost	String

Technical key field: PRODUKT_ID_DIM New name

Creation of technical key

☐ Use table maximum + 1

☐ Use sequence

☒ Use auto increment field

Version field: version

Stream Datefield:

Date range start field: date_from Min. year 1900

Use an alternative start date? ☐ <Select Option>

Table date range end: date_to Max. year 2199

OK Cancel Get Fields SQL

Help

Diese Information wird dann im “Calculator” Step (vorletzter Schritt der Transformation) in den Berechnungen genutzt:

Calculator

Step name: Calculate Totals

Fields:

#	New field	Calculation	Field A	Field B	Field C	Value type	Length	Precision	Remove	Conversion mask
1	brutto_erloes	A * B	quantity	unit_price		Integer			N	
2	netto_erloes	A - (A * B / 100)	brutto_erloes	discount		Integer			N	
3	kosten	A * B	standard_cost	quantity		Integer			N	
4	gewinn	A - B	netto_erloes	kosten		Integer			N	

Help OK Cancel

Im letzten Schritte werden dann alle Datensätze der Faktentabelle einschl. kalkulierter Attribute in das Data-Warehouse geschrieben:

Step name: Load Fact Table

Connection: DWH

Target schema: northwind_dwh

Target table: verkauf_fact

Commit size: 1000

Truncate table: ☐

Ignore insert errors: ☐

Specify database fields: ☒

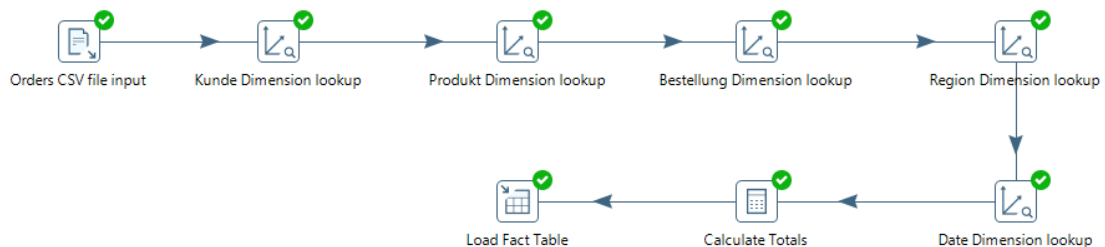
Main options / Database fields

Fields to insert:

#	Table field	Stream field
1	KUNDE_ID_DIM	KUNDE_ID_DIM
2	PRODUKT_ID_DIM	PRODUKT_ID_DIM
3	BESTELLUNG_ID_DIM	BESTELLUNG_ID_DIM
4	REGION_ID_DIM	REGION_ID_DIM
5	ZEIT_ID_DIM	ZEIT_ID_DIM
6	BRUTTO_ERLOES	brutto_erloes
7	NETTO_ERLOES	netto_erloes
8	KOSTEN	kosten
9	GEWINN	gewinn

Buttons: Get fields, Enter field mapping, Help, OK, Cancel, SQL

Nach dem alle Schritte definiert sind, lassen wir die Transformation laufen.



Ein Kontrollblick in die Datenbank bestätigt, dass die Faktentabelle geschrieben wurde:

1 • SELECT * FROM northwind_dwh.verkauf_fact;

	KUNDE_ID_DIM	PRODUKT_ID_DIM	ZEIT_ID_DIM	REGION_ID_DIM	BESTELLUNG_ID_DIM	BRUTTO_ERLOES	NETTO_ERLOES	KOSTEN	GEWINN
▶	28	14	16	13	2	1400	1400	1050	350
	28	28	16	13	2	120	120	90	30
	5	7	21	5	3	300	252	225	27
	5	19	21	5	3	530	530	398	132
	5	28	21	5	3	40	40	30	10
	13	2	23	13	4	270	270	203	67
	13	17	23	13	4	920	810	690	120
	9	11	31	9	5	270	227	207	20
	5	11	38	5	6	180	180	138	42
	30	18	42	15	7	130	130	96	34
	4	16	55	4	8	2000	2000	1440	560
	7	8	66	7	9	680	680	510	170
	29	17	70	14	10	13800	13800	10350	3450
	9	18	82	9	11	1300	1300	960	340
	11	29	84	11	12	600	600	400	200
	8	17	84	8	13	13800	13800	10350	3450
	11	6	84	11	14	250	250	188	62
	11	4	84	11	14	220	220	165	55
	11	11	84	11	14	90	90	69	21
	12	28	84	12	15	80	64	60	4
	12	29	84	12	15	150	150	100	50
	2	2	84	2	16	450	450	338	112
	2	17	84	2	16	1150	1150	863	287
	2	29	84	2	16	75	75	50	25

7. Auswertungen auf Basis des DWH

Wir beginnen mit einer sehr einfachen Auswertung, bei der wir den Gesamtgewinn berechnen, den wir mit unserem Kunden "Company A" erzielt haben:

1 • SELECT SUM(f.GEWINN)
2 FROM northwind_dwh.verkauf_fact f
3 JOIN northwind_dwh.kunde_dim k
4 ON f.KUNDE_ID_DIM = k.KUNDE_ID_DIM
5 WHERE k.COMPANY = "Company A";

	SUM(f.GEWINN)
▶	592

Als nächstes wollen wir uns anschauen, wieviel Gewinn wir mit diesem Kunden mit unserem Produkt "Northwind Traders Coffee" erzielt haben:

7	
8	SELECT SUM(f.GEWINN)
9	FROM northwind_dwh.verkauf_fact f
10	JOIN northwind_dwh.kunde_dim k
11	ON f.KUNDE_ID_DIM = k.KUNDE_ID_DIM
12	JOIN northwind_dwh.produkt_dim p
13	ON f.PRODUKT_ID_DIM = p.PRODUKT_ID_DIM
14	WHERE k.COMPANY = "Company A" AND
15	p.PRODUKT_NAME = "Northwind Traders Coffee";
16	

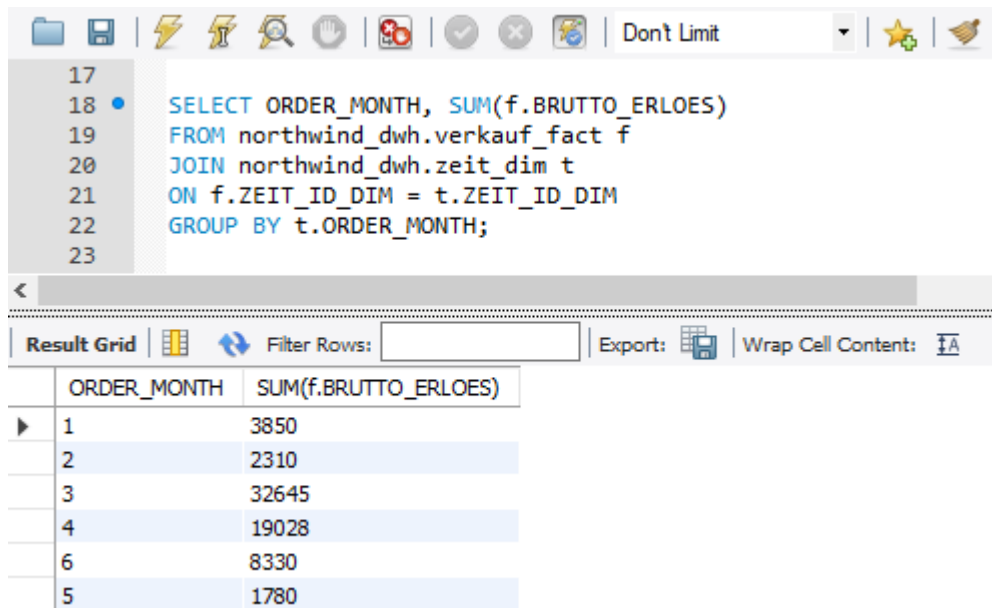
Result Grid	Filter Rows:	Export:	Wrap Cell Content:
SUM(f.GEWINN)			
287			

Beim Produkt "Northwind Traders Chocolate" interessiert uns nun vor allem, wie sich der Verkauf bei unseren Kunden in Oregon im Lauf des Jahres entwickelt hat.

25	SELECT ORDER_MONTH, r.STATE_PROVINCE, f.BRUTTO_ERLOES
26	FROM northwind_dwh.verkauf_fact f
27	JOIN northwind_dwh.region_dim r
28	ON f.REGION_ID_DIM = r.REGION_ID_DIM
29	JOIN northwind_dwh.produkt_dim p
30	ON f.PRODUKT_ID_DIM = p.PRODUKT_ID_DIM
31	JOIN northwind_dwh.zeit_dim t
32	ON f.ZEIT_ID_DIM = t.ZEIT_ID_DIM
33	WHERE p.PRODUKT_NAME = "Northwind Traders Chocolate" AND r.STATE_PROVINCE = "OR";
34	

Result Grid	Filter Rows:	Export:	Wrap Cell Content:
ORDER_MONTH	STATE_PROVINCE	BRUTTO_ERLOES	
3	OR	1300	
6	OR	520	

Und last but not least wollen wir uns einen Überblick über die monatlichen Brutto-Erlöse verschaffen, d.h. die Erlöse vor Abzug aller Rabatte:



```

17
18 • SELECT ORDER_MONTH, SUM(f.BRUTTO_ERLOES)
19 FROM northwind_dwh.verkauf_fact f
20 JOIN northwind_dwh.zeit_dim t
21 ON f.ZEIT_ID_DIM = t.ZEIT_ID_DIM
22 GROUP BY t.ORDER_MONTH;
23

```

	ORDER_MONTH	SUM(f.BRUTTO_ERLOES)
▶	1	3850
	2	2310
	3	32645
	4	19028
	6	8330
	5	1780

8. Fazit und Schlusswort

Die Arbeit an diesem durchaus umfangreichen Objekt war sicher lehrreich auf verschiedenen Ebenen, zu denen die folgenden zählen:

- Besseres Kennenlernen der für uns neuen Tools “PDI” und “MySQL-Workbench”
- Bewusst werden, wie ERM-Schemas aufgebaut sind und Umgang mit unterschiedlichen ERM-Dialekten
- Umgang mit SQL-Syntax
- Effektive Zusammenarbeit in einer Gruppe (da lernt man nie aus)
- Arbeiten mit Versionskontrolle durch “git”.

Die Arbeit mit “git” hat gut funktioniert und den Austausch von Ideen und Dateien sicher erleichtert und es ermöglicht, den Überblick über die Änderungen im Filesystem zu behalten. Sicher wäre die Arbeit an diesem doch überschaubaren Projekt auch ohne git möglich gewesen, aber es hat – gefühlt – selbst auf diesem Niveau schon Vorteile.

Eine Sache, deren wir uns vorher nicht bewusst waren, ist, dass aufgrund der Definition von Fremdschlüssel-Constraints die Reihenfolge der Befüllung (und auch der Löschung) von Tabellen eine Rolle spielt.

Dadurch, dass das Projekt wirklich viele Aspekte des Moduls “DB & DWH” berührt hat, war es für uns sehr nützlich bei der Nachbereitung des Lehrmaterials.