

Institut für Parallele und Verteilte Systeme (IPVS)

Abteilung Anwendersoftware

Universität Stuttgart
Universitätsstr. 38
D-70569 Stuttgart

Diplomarbeit Nr. 2288

OLAP-Analyse von Propagationsprozessen

Anastasios Kozas

Studiengang:	Informatik
Prüfer:	Prof. Bernhard Mitschang
Betreuer:	Rodrigo Monteiro, Uwe Heinkel
Begonnen am:	24.11.2004
Beendet am:	24.05.2005
CR-Klassifikation:	H.2.1, H.2.7, H.2.8

Inhaltsverzeichnis

1	Einführung	1
1.1	Zielsetzung.....	2
1.2	Aufbau der Arbeit.....	2
2	Grundlagen	4
2.1	SIES Architektur.....	4
2.1.1	Propagationsmanager.....	6
2.1.2	Logmanager.....	8
2.2	Data Warehouse.....	12
2.2.1	Definition.....	12
2.2.2	Data Warehouse Architektur.....	13
2.2.3	Dimensionsmodellierung.....	16
2.3	Business Intelligence Tools.....	18
2.3.1	OLAP.....	18
2.3.1.1	Definition.....	18
2.3.1.2	OLAP Architekturen.....	19
2.3.1.3	OLAP Funktionen.....	20
2.3.2	Data Mining.....	21
2.3.3	Unterschiede zwischen OLAP und Data Mining.....	22
3	Anforderungen	23
3.1	Analyse der Logstruktur.....	23
3.2	Anforderungsanalyse.....	26
4	Design des Data Warehouses	28
4.1	Architektur.....	28
4.2	Konzeptueller Entwurf.....	29
4.3	Logischer Entwurf.....	37
4.4	Physischer Entwurf.....	39

5	Datenbereitstellung aus dem Log (ETL)	41
5.1	ETL-Prozess.....	41
5.1.1	Extraktion	42
5.1.2	Transformation	44
5.1.3	Laden.....	45
5.2	Historisierung der Dimensionen.....	46
6	OLAP Anfragen	47
6.1	Vordefinierte Anfragen	47
6.2	Ad hoc Anfragen	50
7	Implementationsaspekte	52
7.1	Aufbau des Data Warehouses	52
7.1.1	Grundlagen zu DB2 Data Warehouse Center	52
7.1.2	Umsetzung von ETL-Prozessen	55
7.1.3	Umsetzung der Konsistenz	64
7.2	Implementierung der Anfragen	65
7.2.1	Grundlagen zu Microstrategy 6.0	65
7.2.2	Bereitstellen der Daten.....	69
7.2.3	Umsetzung der Anfragen.....	73
8	Zusammenfassung und Ausblick	80

Anhang

A	Literaturverzeichnis	81
B	Abbildungsverzeichnis	83
C	Tabellenverzeichnis	86
D	Stored Procedures	87
E	Schemata	89

1 Einführung

Ein wichtiger Erfolgsfaktor für Unternehmen ist heutzutage das schnelle Reagieren auf die sich immer schneller ändernden Marktverhältnisse. Um jedoch schnell reagieren zu können sind Informationen eine wichtige Voraussetzung. Da es sich aber bei den operativen Datenbeständen eines Unternehmens um rohe Daten handelt, die nicht ohne weiteres genutzt werden können, ist es notwendig diese Daten in für die Analyse geeignete Informationen umzuwandeln. Wichtig in diesem Zusammenhang ist, dass Informationen möglichst einfach und verständlich dargestellt werden, denn die Hauptanwender bzw. Entscheidungsträger sind meistens Manager und somit nicht zwangsweise IT-Fachleute.

Durch die Verwendung eines Data Warehouses ist es beispielsweise möglich Informationen auf eine einfache und verständliche Art darzustellen. Innerhalb des Data Warehouses werden die operativen Daten integriert und vor allem konsistent bereitgestellt, was für die weitere Verwendung besonders wichtig ist. Konsistent bedeutet unter anderem, dass Informationen einheitlich gespeichert werden. In den operativen Datenbeständen kann zum Beispiel zu einem Angestellten in einem Geschäftsbereich nur der Name gespeichert werden und in einem anderen Geschäftsbereich Name und Vorname. Im Data Warehouse wird zum Beispiel nur der Name gespeichert. Die Informationen in einem Data Warehouse werden verwendet, um Wissen zu extrahieren, das wiederum genutzt werden kann, um Entscheidungen treffen zu. Um Wissen aus einem Data Warehouse zu extrahieren sind Business Intelligence Tools wie zum Beispiel OLAP erforderlich. Erst der Einsatz von Business Intelligence Tools in Zusammenhang mit einem Data Warehouse ermöglicht es Wissen zu extrahieren, das genutzt werden kann, um auf ändernde Marktverhältnisse reagieren und vor allem agieren zu können.

Data Warehouse und OLAP können auch innerhalb des *Stuttgarter Informations- und Explorationssysteme (SIES)* eingesetzt werden. Das Hauptziel von SIES ist es Datenänderungen zwischen verbundenen Informationssystemen zu verwalten. Der Prozess der Weiterleitung von Datenänderungen zwischen Informationssystemen kann mit Hilfe eines Data Warehouses und OLAP analysiert werden, um beispielsweise anschließend Optimierungen durchzuführen. Durch das Analysieren des Prozesses erhält man ein besseres Verständnis für die Beziehung zwischen den unterschiedlichen Informationssystemen, was wiederum zu einem besseren Verständnis der Geschäftsprozesse innerhalb eines Unternehmens führt. Ein weiterer Punkt warum das Analysieren der SIES Daten interessant sein könnte ist das Überwachen von Fehlern.

1.1 Zielsetzung

Ziel dieser Arbeit ist es durch OLAP Anfragen Informationen über das Propagieren von Datenänderungen zwischen Informationssystemen zu gewinnen. Die OLAP Anfragen basieren auf Daten, die während des Austausches von Datenänderungen zwischen Informationssystemen entstehen und im Log gespeichert werden. Bevor jedoch derartige OLAP Anfragen formuliert werden können, ist eine Analyse der Logstruktur durchzuführen. Erst danach ist es möglich interessante Fragestellungen zu formulieren, die durch OLAP Anfragen beantwortet werden können. Als Grundlage für die OLAP Anfragen wurde zunächst ein Data Warehouse aufgebaut, in dem die Daten aus dem Log integriert und strukturiert bereitgestellt werden. Um jedoch die Daten aus dem Log ins Data Warehouse zu laden sind ETL-Prozesse erforderlich. Zum Schluss werden die implementierten OLAP Anfragen auf dem Data Warehouse ausgeführt. Dadurch werden Informationen gewonnen, die genutzt werden können um zum Beispiel Optimierungspotenziale zu ermitteln.

1.2 Aufbau der Arbeit

In *Kapitel 2* werden zunächst einige Begriffe erläutert, die für das Verständnis dieser Arbeit erforderlich sind. Hierzu zählen SIES, Data Warehouse, OLAP und Data Mining.

Kapitel 3 befasst sich mit den Anforderungen, welche die Grundlage für das weitere Vorgehen sind.

Kapitel 4 beschreibt die Vorgehensweise für den Aufbau des Data Warehouses, wobei hier zunächst die Architektur erläutert wird und anschließend die einzelnen Entwurfsphasen (konzeptuell, logisch und physisch) .

In *Kapitel 5* geht es um die Datenbereitstellung aus dem Log (ETL), d.h. wie werden die Daten aus dem Log ins Data Warehouse geladen. Hierbei geht es lediglich um die Erläuterung des Konzeptes und nicht Umsetzung mit einem speziellen Werkzeug.

Kapitel 6 beschreibt verschiedene OLAP Anfragen zur Analyse eines Propagationsprozesses, wobei hier zwischen vordefinierten und ad hoc Anfragen unterschieden wird.

In *Kapitel 7* wird einerseits der Aufbau des Data Warehouses und andererseits die Implementierung der aus Kapitel 6 definierten OLAP Anfragen erläutert. Für den Aufbau des Data Warehouses wurde das Werkzeug von IBM „DB2 Data Warehouse Center“ und für die Implementierung der Anfragen „Microstrategy 6.0“ verwendet. In beiden Abschnitten erfolgt zunächst eine Einführung in die jeweiligen Werkzeuge.

Kapitel 8 fasst die Ergebnisse der Arbeit zusammen und gibt Anregungen für weitere Arbeiten, die sich mit den Themen Data Warehouse und OLAP beschäftigen.

2 Grundlagen

Dieses Kapitel erläutert zunächst die SIES Architektur und hier speziell die beiden Komponenten Propagationsmanager und Logmanager. Außerdem wird erläutert was ein Data Warehouse, das die Grundlage für Business Intelligence Tools darstellt, ist. Zum Schluss erfolgt die Definition der beiden Business Intelligence Tools OLAP und Data Mining, wobei im Anschluss auch auf die Unterschiede der beiden Techniken eingegangen wird.

2.1 SIES Architektur

Durch die sich immer schneller ändernden Marktverhältnisse ist es wichtig, dass sich Unternehmen mit der Frage *wie kann sich das Unternehmen in Zukunft an die sich immer schneller ändernden Marktverhältnisse anpassen* beschäftigen. Und mit dieser Frage beschäftigt sich der Sonderforschungsbereich 467 (Wandlungsfähige Unternehmensstrukturen für die variantenreichen Serienfertigung). Ziel des Teilprojektes A5 - Modellierung von und Exploration in komplexen Unternehmensinformationen – des Sonderforschungsbereichs 467 ist die Integration von Informationssystemen. Hierfür ist eine Infrastruktur erforderlich, die mit Hilfe von Änderungspropagationen die Informationssysteme (IS) eines wandlungsfähigen Unternehmens integriert. Datenänderungen werden erkannt und an die von dieser Änderung abhängigen IS weitergeleitet. In Abbildung 1 ist diese Infrastruktur, auch *Stuttgarter Informations- und Explorationssystem (SIES)* genannt, abgebildet. Sie wurde entworfen, um heterogene Informationssysteme zu einem funktionalen Gesamtsystem zu verbinden.

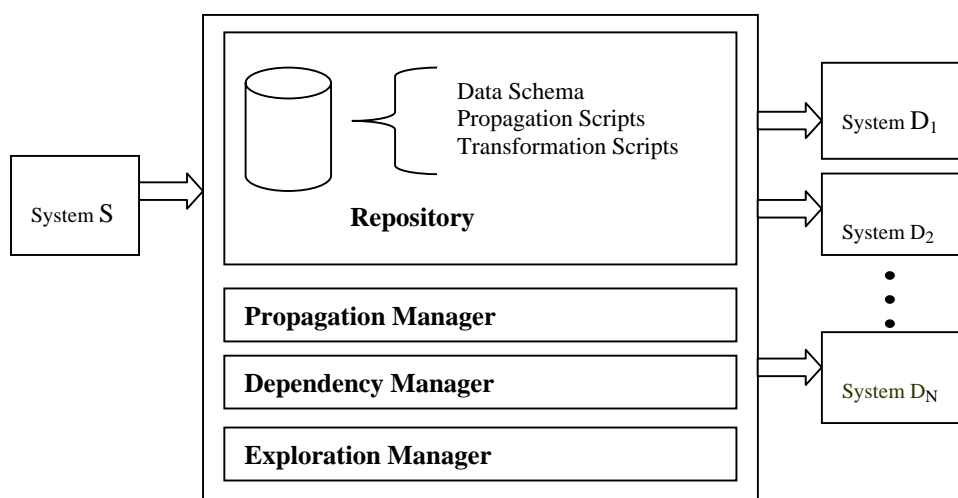


Abbildung 1 SIES Architektur [CHR02]

SIES besteht aus vier Komponenten, nämlich Dependency Manager, Repository, Propagationsmanager (PM) und Exploration Manager. Der Dependency Manager ist zuständig für das Definieren der Abhängigkeiten zwischen IS. Eine derartige Abhängigkeit wird als Dependency bezeichnet. Anders ausgedrückt, durch den Dependency Manager wird der Datenfluss zwischen IS definiert. Im Repository werden sämtliche Informationen, die für den Austausch von Datenänderungen zwischen IS benötigt werden, gespeichert. Hierzu zählen Propagationsskripte (PS), Transformationskripte und Datenschemata. Die Aufgabe des PM ist das Transformieren von Datenobjekten des Quellsystems in das Format des Zielsystems und das anschließende Propagieren der Datenänderungen an die entsprechenden IS. Eine weitere Komponente von SIES ist der Exploration Manager, wobei dafür noch keine Implementierung vorhanden war. In Abbildung 2 ist die Architektur der Exploration Umgebung abgebildet. Innerhalb dieser Architektur ist der Logmanager (LM) eine wichtige Komponente. Der LM speichert Informationen zu speziellen Ereignissen, die während einer Propagation von Datenänderungen zwischen IS auftreten.

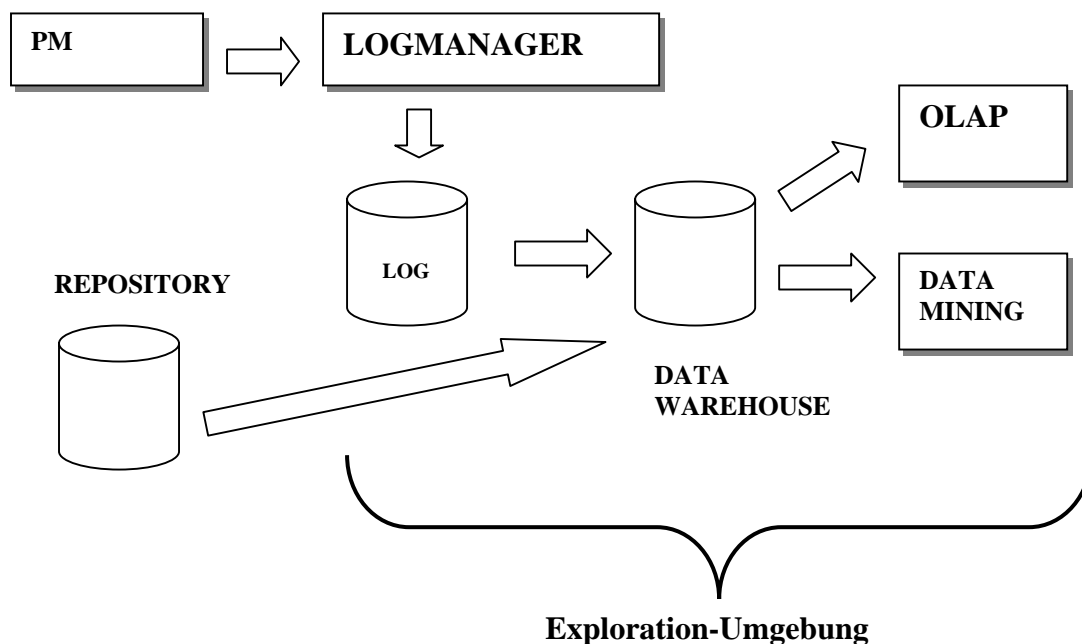


Abbildung 2 Architektur der Exploration-Umgebung

In den nächsten beiden Abschnitten werden die beiden Komponenten PM und LM näher erläutert. Grundlagen zu Data Warehouse, OLAP und Data Mining werden in den Abschnitten 2.2 und 2.3 erläutert.

2.1.1 Propagationsmanager

Der Propagationsmanager (PM) ist verantwortlich für das Propagieren von Datenänderungen (*Insert*, *Update* und *Delete*) zwischen Informationssystemen (IS). Er ist eine Laufzeit Komponente und greift auf ein Repository, welches die notwendigen Informationen zum Propagieren der Datenänderungen enthält, zu. Auf Grund der Tatsache, dass IS nicht unbedingt dasselbe Datenformat verwenden, ist eine Transformation von Datenänderungen nicht zu vermeiden. Als Transformation wird eine Operation verstanden, die empfangene Daten in das Format des jeweiligen Zielsystems umwandelt. Für die Transformationen wird eine XML-basierte Sprache (XSLT) verwendet [CRM01], [CHM02].

Da in dieser Arbeit Daten verwendet werden, die während eines Propagationsprozesses entstehen, wird im Folgenden erläutert was ein Propagationsprozess ist.

Die Ausführung eines Propagationsskriptes (PS) entspricht einem Propagationsprozess. Für jede Dependency gibt es ein PS, welches als Prozess ausgeführt wird. Ein PS wird in XPDL (**X**ML **P**ropagation **D**efinition **L**anguage), die verbesserte Variante von XRL+, implementiert und durch die Propagation Engine ausgeführt. In Abbildung 3 ist ein Beispiel für ein PS zu sehen.

```
1 <xpdl:propagation_script xmlns:xpdl="http://www.sfb467.uni-stuttgart.de/sies2/xpdl"
2   created_by="Anastasios Kozas" creation_date="30.12.2005" version="0.8"
3   depType="simple" debug="true">
4
5   <xpdl:input_declaration>
6     <xpdl:start_input system="A" schema="position" chg_type="update" xml_out="a"/>
7   </xpdl:input_declaration>
8   <xpdl:sequence>
9     <xpdl:transform xml_in="a" xml_out="res" script="atores" reduceTo="after"/>
10    <xpdl:propagate xml_in="res" system="B" schema="Bresource"/>
11  </xpdl:sequence>
12 </xpdl:propagation_script>
```

Abbildung 3 Beispiel für ein Propagationsskript (XPDL)

In Zeile 1 bis 3 erfolgt die Definition des PS. Hierzu gehören zum Beispiel Angaben zum Erstellungsdatum, Ersteller, etc. Jedes PS enthält zum Empfangen von Daten eine Input Deklaration (*Input_Declaration*, Zeile 4). In Zeile 5 ist ein Start Input (*Start_Input*) definiert. Ein PS besitzt genau ein Start Input, das sobald Daten vom jeweiligen System (Hier: „A“) empfangen werden, den Propagationsprozess startet. Optional können in einem PS auch Change Inputs (*Change_Input*, N:M Abhängigkeit) definiert werden und zwar wenn der PM auf Daten von mehreren Systemen angewiesen ist. Das bedeutet, dass ein *Timeout* festgelegt werden muss. Innerhalb

der Sequenz (Zeile 7 bis 10) erfolgt das Transformieren und anschließende Propagieren der Datenänderung an das Zielsystem (Hier: „B“). Für eine exakte Definition der jeweiligen Strukturen innerhalb eines PS sei auf die Dokumentation [HEI04] verwiesen.

Wie bereits erwähnt wurde, entspricht die Ausführung eines PS einem Propagationsprozess. Bevor jedoch ein PS ausgeführt werden kann, sind einige Schritte notwendig. In Abbildung 4 ist der Ablauf vom Empfang bis zur Propagation der Datenänderung abgebildet. In diesem Beispiel wird eine Abhängigkeit zwischen zwei IS abgebildet ist. Das bedeutet aber nicht, dass Abhängigkeiten lediglich zwischen zwei IS bestehen. Es können auch mehrere Zielsysteme von den Datenänderungen innerhalb des Quellsystems abhängen. In diesem Beispiel würde es bedeuten, dass neben dem System B weitere Systeme abgebildet werden müssten.

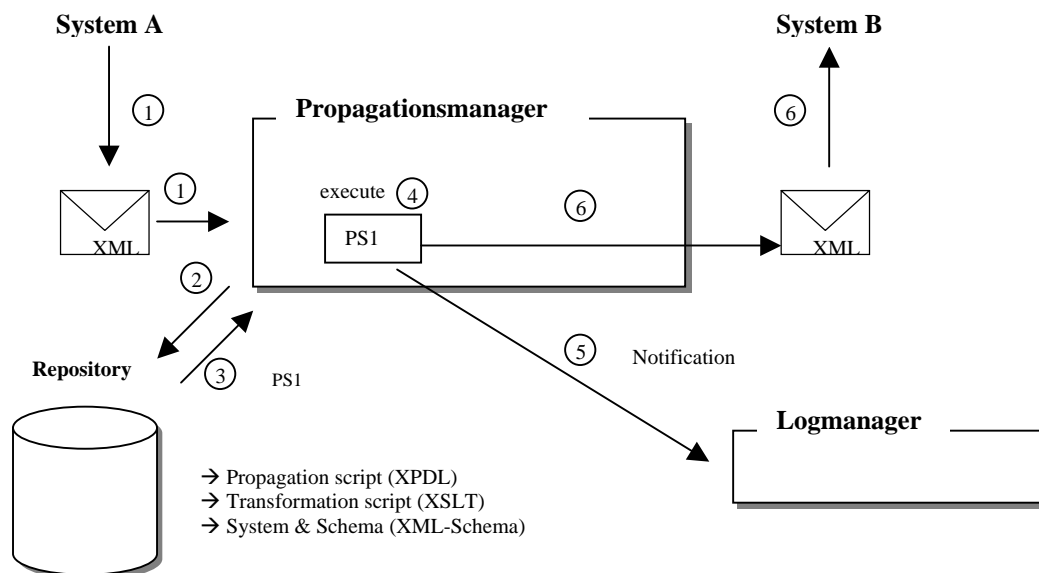


Abbildung 4 Ablauf vom Empfang bis zur Propagation einer Datenänderung

Der PM muss zunächst Datenänderungen innerhalb eines IS erkennen (1), wobei hierfür Adaptere verwendet werden. Es wird zwischen aktiven und passiven Adaptere unterschieden [RCH02]. Aktive Adaptere prüfen eigenständig, ob Datenänderungen auftreten und benachrichtigen den PM. Im Gegensatz zu den aktiven Adaptere müssen passive Adaptere durch die entsprechenden Datenquellen über Datenänderungen benachrichtigt werden. Beim Erhalt einer Nachricht prüft der PM, ob im Repository ein passendes PS gespeichert ist, d.h. ob es IS gibt, die von der Datenänderung betroffen sind (2). Wenn es IS gibt, die davon betroffen sind, erhält der PM das für die spezielle Datenänderung erforderliche PS (3) (hier: PS1), welches er als Prozess anschließend ausführt (4). Während der Ausführung eines Propagationsprozesses können verschiedene Ereignisse auftreten, wie zum Beispiel ein Ereignis

nis, das anzeigt, dass ein Propagationsprozess gestartet wurde (Start Ereignis). Sämtliche Ereignisse einschließlich Erläuterungen sind im nächsten Abschnitt aufgeführt. Nur wenn eines der vordefinierten Ereignisse auftritt wird eine Benachrichtigung an den LM gesendet (5). Dieser speichert anschließend zu den jeweiligen Ereignissen diverse Informationen bzw. Details. Bei einem Start Ereignis wird zum Beispiel als Detail der Name des Quellsystems (*Source_System*) gespeichert. Der letzte Schritt innerhalb eines Propagationsprozesses ist das Propagieren der Datenänderungen an die jeweiligen IS (6), wobei die Datenänderungen in das jeweilige Format des Zielsystems transformiert werden. Anhand von diesem Ablauf wird auch das Zusammenwirken des PM und LM, welcher im nächsten Abschnitt näher erläutert wird, ersichtlich.

2.1.2 Logmanager

Die Aufgabe des Logmanagers (LM) ist das Speichern von Informationen zu den jeweiligen Ereignissen. Wie im vorherigen Abschnitt bereits erwähnt wurde, werden während eines Propagationsprozesses beim Eintreten von bestimmten Ereignissen Benachrichtigungen an den LM gesendet. Angenommen es tritt beispielsweise während eines Propagationsprozesses ein Start Ereignis auf. Beim Eintreten wird eine Benachrichtigung an den Logmanager gesendet, der anschließend Informationen zu diesem Ereignis speichert. Die Informationen, die zum Start Ereignis gespeichert werden sind Name des Quellsystems (*Source_System*, zum Beispiel „Sales System“) und das verwendete Schema des Quellsystems (*Source_System_Schema*, zum Beispiel „Sales“).

Bevor die Benachrichtigungen erläutert werden, muss zunächst die Logstruktur, die in Abbildung 5 abgebildet ist, beschrieben werden.

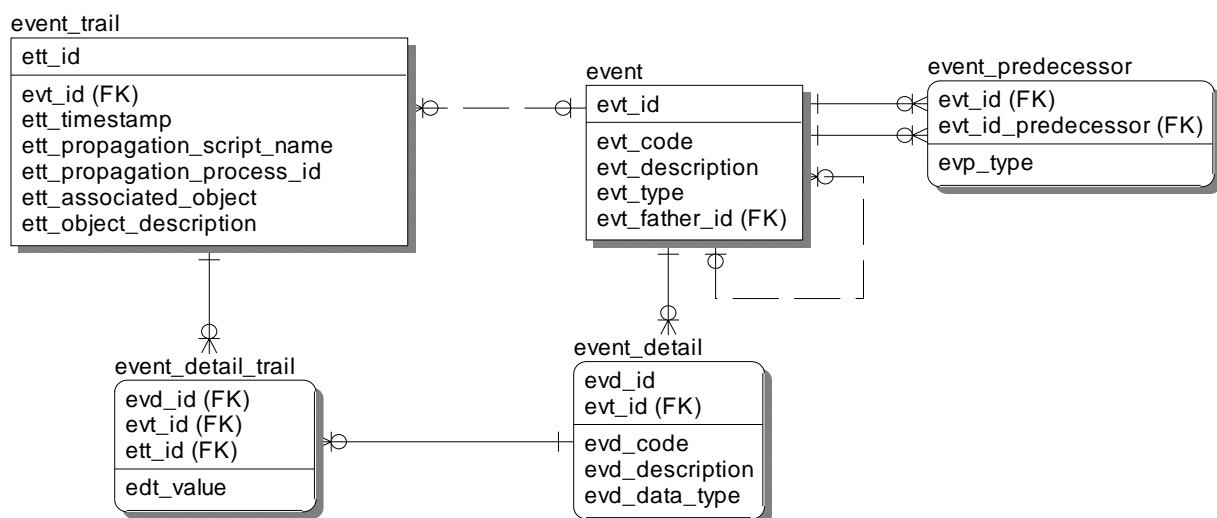


Abbildung 5 Logstruktur

Die Tabellen *Event*, *Event_Detail* und *Event_Predecessor* enthalten die Metadaten zu den Ereignissen, die durch den LM verwaltet werden. Bevor Benachrichtigungen vom PM empfangen und gespeichert werden können, müssen die Metadaten zu den jeweiligen Ereignissen spezifiziert werden. Die Tabelle *Event* enthält sämtliche Informationen zu den Ereignissen, wobei *evt_code* ein eindeutiger Name des Ereignisses ist. Zusätzlich besitzt *Event* noch eine 1:N Beziehung mit sich selbst bzw. mit *father_id*. Anders ausgedrückt, ein Ereignis kann mehrere Unter-Ereignisse besitzen. Ein Beispiel hierfür könnte sein, dass ein Failure Ereignis aufgeteilt ist in mehrere Failure Ereignisse. Die zu den Ereignissen gehörenden Informationen bzw. Details werden in der Tabelle *Event_Detail* gespeichert. *Event_Predecessor* enthält den unmittelbaren Vorgänger eines Ereignisses. Beispielsweise wird in *Event_Predecessor* die ID des Vorgängers (Sleep Ereignis) von Wake_Up abgespeichert, d.h. Wake_Up hat Sleep immer als Vorgänger. In Abbildung 6 ist ein Beispiel abgebildet, das einen Ausschnitt davon zeigt, welche Informationen bzw. Metadaten zum Start Ereignis gespeichert werden.

Event

Evt_id	Evt_code	Evt_father_id (FK)
<u>1</u>	Start	—

Event_Detail

Evd_id	Evt_id (FK)	Evd_code
1	<u>1</u>	Source_System
2	<u>1</u>	Source_System_Schema

Abbildung 6 Beispiel Metadaten zum Start Ereignis

Die eigentlichen Daten zu einem Propagationsprozess werden in *Event_Trail* und *Event_Detail_Trail* gespeichert. In *Event_Trail* werden Informationen wie zum Beispiel Zeitstempel des jeweiligen Ereignisses innerhalb des Propagationsprozesses oder Name des Propagationsskriptes gespeichert. Um jedoch derartige Informationen speichern zu können müssen die Metadaten zu den jeweiligen Ereignissen zuvor spezifiziert worden sein. Da ein PS mehrfach ausgeführt werden kann und der LM sämtliche Ausführungen speichern muss, ist eine eindeutige ID für jede Ausführung erforderlich. Hierzu wird die *ett_propagation_process_id* verwendet.

In *Event_Detail_Trail* werden die entsprechenden Werte zu den jeweiligen Details gespeichert. Zum Detail *Source_System* wird beispielsweise der Name des verwendeten Quellsystems (Bsp. „Sales System“) abgelegt. Auch hier gilt, dass die Metadaten zu den jeweiligen Details der Ereignisse in der Tabelle *Event_Detail* zuvor spezifi-

ziert wurden. Ein Beispiel für die Ausführung eines PS mit entsprechenden Ereignissen wird in Kapitel 3.1 näher erläutert.

Um Informationen über Ereignisse innerhalb eines Propagationsprozesses speichern zu können, muss der PM Benachrichtigungen, die im PM Code eingebettet sind, an den LM senden. Abbildung 7 zeigt ein Beispiel für eine einfache Benachrichtigung.

```
...
{
Zeile 1 LogManagerClient client = new LogManagerClient();
Zeile 2 LogEventNotification notification;

Zeile 3 notification=newLogEventNotificationTrackEvent (START_EVENT,
Zeile 4 propagationScriptName, processID);

Zeile 5 client.send(notification);
}
```

Abbildung 7 Einfache Benachrichtigung (Java)

In den Zeilen 1 und 2 erfolgt die Deklaration, um Benachrichtigungen (Notifications) senden zu können. Die Benachrichtigung mit den Informationen Name des Ereignisses (*Event_Name*), Propagationskriptname (*PS_Name*) und Propagationsprozess ID (*processID*) wird in Zeile 3 und 4 erzeugt. Anschließend wird in Zeile 5 die Benachrichtigung an den LM gesendet. Das nächste Beispiel in Abbildung 8 enthält neben *Event_Name*, *PS_Name* und *processID* zusätzliche Informationen zu den Ereignissen (Zeile 7, 8 und 9) wie zum Beispiel zu *Source_System* „Sales System“ und zu *Source_System_Schema* „Product“. In Zeile 1 bis 3 erfolgt die Deklaration, um Benachrichtigungen senden zu können. Zeile 4 ist analog zu Zeile 3 der einfachen Benachrichtigung und in Zeile 10 wird die Benachrichtigung an den LM gesendet.

```
...
{
Zeile 1 LogManagerClient client = new LogManagerClient();
Zeile 2 LogEventNotification notification;
Zeile 3 LogEventNotificationDetailContainer details;
Zeile 4 notification = new LogEventNotificationTrackEvent (START_EVENT,
Zeile 5 propagationScriptName, processID);

Zeile 6 details = notification.getDetailContainer();
Zeile 7 details.addNotificationDetail("SOURCE_SYSTEM", "Sales System");
Zeile 8 details.addNotificationDetail("SOURCE_SCHEMA", "Product");
Zeile 9 notification.setNotificationDetails(details);

Zeile 10 client.send(notification);
}
```

Abbildung 8 Umfangreiche Benachrichtigung (Java)

Zusammenfassend werden im Folgenden sämtliche Ereignisse (Tabelle 1) und die jeweiligen Details (Tabelle 2), die im Log abgelegt werden, aufgeführt. Es gibt zwei Typen von Ereignissen und zwar die allgemeinen (*Process Manager Events*) und die spezifischen (*Propagation Process Events*) Ereignisse. Zu den allgemeinen Ereignissen gehört beispielsweise die Initialisierung des Prozessmanagers. Spezifische Ereignisse hingegen sind Ereignisse, die während der Ausführung eines Propagationsprozesses auftreten, wie zum Beispiel ein Start Ereignis.

Ereignis Typ	Ereignis	Beschreibung
Process Manager Events	Initialization	Prozessmanager wurde erfolgreich initialisiert
	Shutdown	Prozessmanager wurde erfolgreich terminiert
Propagation Process Events	Start	Propagationsprozess wurde gestartet
	Sleep	Es fehlen erforderliche Daten und der Prozess wird sozusagen still gelegt
	Wake Up	Entweder alle erforderlichen Daten sind vorhanden oder Timeout ist abgelaufen
	Transform	Umwandlung von Datenänderungen an das Format der jeweiligen Zielsysteme
	Propagate	Datenänderungen werden an Zielsysteme weitergeleitet
	Condition	Eine Bedingung innerhalb des PS wird ausgewertet
	Filter	Ein Filter innerhalb des PS wird ausgewertet
	Termination Successful	Propagationsprozess wurde erfolgreich terminiert
	Forced Termination	Propagationskriptterminierung durch ein Propagationskriptbefehl
	Partial Rollback Termination	Der Prozess wurde mehr als eine definierte Anzahl wiedergestartet oder es treten bestimmte Fehler auf
	Restart	Propagationsprozess wurde wiedergestartet
Failure	Ein Fehler tritt auf	

Tabelle 1 Vordefinierte Ereignisse

Ereignis	Ereignis Detail	Beschreibung
Start	Source_System Source_System_Schema	Name des Quellsystems Schema
Sleep	Timeout	Timeout Wert
Wake_Up	Timed out	Timeout abgelaufen
Condition	Condition Clause	Condition Ausdruck
Filter	Filter Clause	Filter Ausdruck
Transform	Trans_Script	Name des Transformationskriptes
Propagate	Destination_System Destination_System_Schema	Name des Zielsystems Schema

Tabelle 2 Informationen zu den jeweiligen Ereignissen

2.2 Data Warehouse

In diesem Abschnitt wird zunächst definiert was ein Data Warehouse (DW), welches die Grundlage für Business Intelligence Tools darstellt, ist. Anschließend werden die einzelnen Komponenten des DWs erläutert. Im letzten Abschnitt wird die Dimensionsmodellierung beschrieben.

2.2.1 Definition

Vereinfacht ausgedrückt ist ein DW eine Ansammlung von Informationen, die zur Analyse genutzt werden. Eine weitaus genauere Definition liefert Inmon. „A Data Warehouse is a subject-oriented, integrated, time-variant, and non-volatile collection of Data in support of management’s Decision support process.“ [INM02]

Dabei bedeuten die einzelnen Teile dieser Definition:

- Subject-oriented: Daten werden nach betriebswirtschaftlichen Kriterien organisiert bzw. an die Belange der Manager ausgerichtet.
- Integrated: DW integriert verschiedene Datenquellen des Unternehmens, d.h. ein DW beinhaltet Daten aus unterschiedlichen operativen Datenbeständen bzw. auch externen Datenquellen.
- Time-variant: DW enthält Beschreibungen zu verschiedenen Zeitpunkten, d.h. Daten repräsentieren meist definierte Zeiträume.
- Non-volatile: Daten werden dauerhaft gespeichert und können nicht durch spätere Analysen verändert werden.
- Decision support process: DW stellt analytisch und strategisch Informationen für Entscheider zur Verfügung und unterstützt so den Entscheidungsprozess.

Ein DW muss strikt von den operativen Datenbeständen getrennt werden. Dafür gibt es eine Reihe von Gründen. Einer davon ist, dass in einem DW üblicherweise ein bestimmter Zeitpunkt in der Vergangenheit beschrieben wird und die operativen Da-

tenbestände meist tages-aktuell sind. Weitere Unterschiede sind in der untenstehenden Tabelle aufgeführt:

	Operationale Datenbanken	Data Warehouse
Entstehung	Jeweils für eine Applikation oder aus einer bestimmten Perspektive heraus	Mehrere Perspektiven gleichzeitig
Anforderungen	Bekannt	Vage
Bedeutung	Alltägliche Geschäftsabläufe	Entscheidungen des Managements, die sich auf Profitabilität auswirken
Datenzugriff	Ein Aufruf liefert wenige Zeilen zurück	Große Datenmengen werden zugegriffen, um das Ergebnis zu ermitteln
Tuning	Abgestimmt für häufige Zugriffe auf kleine Datenmengen	Abgestimmt für eher seltene Zugriffe auf große Datenmengen
Datenvolumen	Datenbestand wird für operationales Geschäft gebraucht	Großer Datenbestand wird für statistische Analysen, Vorhersagen, ad hoc Reports gebraucht
Datenaufbewahrung	Solange es das Tagesgeschäft Erfordert	Langfristig, um Reporting über Zeiträume oder Vergleiche zu Ermöglichen
Verfügbarkeit	Hohe Verfügbarkeit erforderlich	Nicht ganz so hoch wie in Produktionsumgebungen, abhängig davon, ob weltweiter Zugriff
Entwurfsziel	Hohe Performance	Flexibilität

Tabelle 3 Gegenüberstellung operationale Datenbanken und Data Warehouse

In der Literatur wird in Zusammenhang mit einem DW häufig der Begriff Data Mart erwähnt. Ein Data Mart enthält Daten, die für einen bestimmten Benutzerkreis gedacht sind, wie zum Beispiel für den Vertrieb. Der Unterschied zu einem DW ist der, dass ein DW eine Sammlung von Daten über das Unternehmen ist und ein Data Mart ein Datenausschnitt davon ist. Eine Gegenüberstellung von DW und Data Mart ist in [INM02] aufgeführt.

Oft verwenden mehrere Data Marts teilweise dieselben Dimensionen. Es wäre nicht sinnvoll für jedes einzelne Data Mart eine eigene Dimension anzulegen. Vielmehr sollten sie die Dimensionen gemeinsam nutzen. Hierfür kann die DW Bus Architektur verwendet werden. Data Marts, die über gemeinsame Dimensionen zusammenhängen, können über die DW Bus Architektur miteinander verbunden werden [KIM98].

2.2.2 Data Warehouse Architektur

In Abbildung 9 ist die Architektur eines DWs abgebildet. Ein DW wird nicht einmal befüllt, sondern es handelt sich hierbei um einen iterativen Prozess. Ein wichtiger Begriff in diesem Zusammenhang ist das Monitoring. Unter Monitoring wird das Überwachen der Operational Source Systems auf Datenänderungen, die für das DW relevant sind, verstanden. Hierfür gibt es unterschiedliche Techniken wie zum Bei-

spiel Trigger-basiert, Ereignis-basiert, Zeitstempel-basiert oder auch Snapshot-basiert [BAU01].

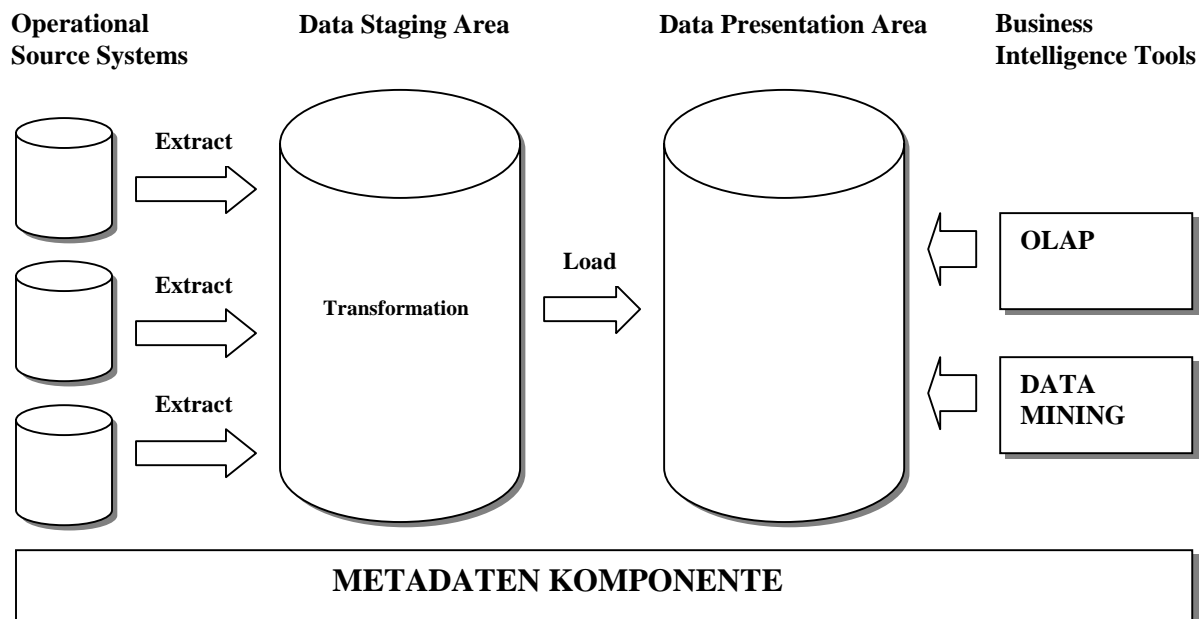


Abbildung 9 Data Warehouse Architektur

Zu den Komponenten der DW Architektur gehören:

- *Operational Source Systems (Quellsysteme)*
Hier befinden sich die operationalen Datenbestände eines Unternehmens, d.h. sie befinden sich außerhalb des DWs. Innerhalb der operationalen Datenbestände sind die Daten meist tages-aktuell, d.h. Daten werden innerhalb der operationalen Datenbestände überschrieben und folglich gibt es keine Historie [KIM02].
- *Data Staging Area*
Data Staging Area ist sowohl Speicherbereich als auch eine Menge von Prozessen. Er befindet sich zwischen den operational Source Systems und dem Data Presentation Area.

Ralph Kimball hat einen sehr treffenden bildlichen Vergleich angestellt. Er vergleicht ein Data Staging Area mit der Küche eines Restaurants. Die Zutaten (Daten) für ein Gericht werden zum Beispiel aus verschiedenen Märkten (Operational Source Systems) zusammengetragen. Innerhalb der Küche (Data Staging Area) werden diese anschließend zubereitet und dem Restaurantbesucher an den Tisch (Data Presentation Area) gebracht [KIM02].

Innerhalb des Data Staging Areas ist der Zugriff durch Benutzer nicht erlaubt, da die Daten noch transformiert bzw. konsistent zusammengetragen werden müssen. Es würde auch keinen Sinn machen auf Daten innerhalb des Data

Staging Areas zuzugreifen, da sie inkonsistent sind. Anders gesagt, ein Restaurantbesucher würde auch nicht auf rohe Zutaten in der Küche zugreifen, sondern auf das fertige Menü warten. Es gibt eine Reihe von Schritten, die notwendig sind, um die operativen Daten in für die Analyse geeignete Informationen umzuwandeln.

Der erste Schritt ist die Extraktion der Daten aus den operativen Datenbeständen in den Data Staging Area. Extraktion bedeutet das Lesen und Kopieren der erforderlichen Daten in den Data Staging Area. Sobald die Daten in den Data Staging Area extrahiert wurden erfolgt eine Transformation. Zur Transformation gehört die Bereinigung der extrahierten operativen Daten (zum Beispiel das Eliminieren von Duplikaten, Korrigieren von Rechtschreibfehlern, usw.), Vereinheitlichung, Zuweisung von Surrogate Keys (um unabhängig von Datenänderungen innerhalb der Operational Source Systems zu sein), Anreicherung oder auch wie Datenänderungen innerhalb der Quellsysteme gehandhabt werden sollen. Vereinheitlichen bedeutet, dass Synonyme und Homonyme entfernt werden. Synonyme, d.h. zwei verschiedene Wörter mit derselben Bedeutung, werden aufgelöst und es wird lediglich einer der Begriffe verwendet. Wenn ein Wort für verschiedene Bedeutungen verwendet wird, d.h. ein Homonym ist, wird dieses Wort aufgelöst, so dass es nur eine eindeutige Bedeutung dafür gibt. Unter Anreicherung wird das Bilden und Speichern von betriebswirtschaftlichen Kenngrößen wie zum Beispiel Umsatz verstanden. Im Anschluss an die Transformation hat man eine Ansammlung von integrierten Daten, die bereinigt, standardisiert und zusammengefasst sind. Der letzte Schritt ist das Laden dieser integrierten Daten in den Data Presentation Area [KIM02]. Das Extrahieren, Transformieren und Laden wird als ETL-Prozess bezeichnet.

- *Data Presentation Area*
Das ist der Bereich in dem Informationen organisiert und tatsächlich dauerhaft abgelegt werden. Hier werden die Informationen für die Analyse zur Verfügung gestellt [KIM02].
- *Business Intelligence Tools*
Mit Hilfe von Business Intelligence Tools wird auf die Daten im Data Presentation Area zugegriffen. Hierzu zählen zum Beispiel OLAP und Data Mining (Vgl. Abschnitte 2.3.1 und 2.3.2).
- *Metadaten Komponente*
Metadaten sind Daten über Daten. Zu den Metadaten zählen sämtliche Informationen in der DW Umgebung, die nicht zu den eigentlichen Daten gehören. Hierzu zählen Informationen zu den logischen Datenstrukturen, Informationen über die Dateien, Indizes etc. Metadaten sind für ein DW von großer Bedeutung, da sie zur Steuerung der Prozesse benötigt werden. Metadaten wer-

den sowohl für den ETL-Prozess als auch für das eigentliche DW, d.h. bei der Analyse, Datenzugriffe, etc., verwendet. In einem Metadaten Repository werden die Metadaten gespeichert und durch den Metadaten-Manager verwaltet. Metadaten können mit den Gelben Seiten verglichen werden, da beide Informationen zu bestimmten Bereichen enthalten [KIM02]. Man unterscheidet zwischen technischen und fachlichen Metadaten. Zu den technischen zählen sämtliche Metadaten, die für Administratoren und Anwendungsentwickler wichtig sind, wie zum Beispiel Zugangsrechte und Datenbankkataloge. Als fachliche Metadaten werden sämtliche Informationen bezeichnet, die dem Benutzer helfen sollen die Daten innerhalb des DWs zu verstehen, wie zum Beispiel Informationen zu vordefinierten Anfragen. Metadaten können auch aus Datensicht klassifiziert werden. Hierbei wird zwischen operationalen Metadaten (Speicherplatzinformationen, usw.), Extraktion und Transformation Metadaten (Transformations- und Bereinigungsregeln, usw.) und End-Benutzer Metadaten (Definitionen der Tabellen innerhalb des Data Presentation Areas, usw.) unterschieden [KIM02].

2.2.3 Dimensionsmodellierung

Da normalerweise die üblichen Mitarbeiter, Geschäftsanalytiker und Manager, d.h. keine IT-Fachleute, die eigentlichen Benutzer eines DWs sind, ist die Ergonomie eines DW sehr wichtig. Ein DW sollte leicht verständlich und vor allem benutzerfreundlich sein. Daher ist das dimensionale Datenmodell besonders gut geeignet. Im Folgenden wird das Konzept der Dimensionsmodellierung erläutert:

- **Fakten:** Fakten entsprechen den Informationen, die für die Analyse besonders interessant sind, wie zum Beispiel Umsatz oder Verkaufsmengen. Fakten sind meist numerische Attribute. Es wird zwischen additiven, semi-additiven und non-additiven Fakten unterschieden [KIM02]. Additive, d.h. sämtliche Aggregationsfunktionen sind anwendbar, werden in einem DW angestrebt. Semi-additive bedeutet, dass nicht sämtliche der Aggregationsfunktionen angewendet werden können. Typisches Beispiel hierfür sind Fakten, die einen Zustand beschreiben, wie zum Beispiel Lagerbestand. Es würde hierbei keinen Sinn machen die Aggregationsfunktion SUM zu verwenden, da jedes Tupel in der Faktentabelle den Zustand, d.h. den aktuellen Lagerbestand, beschreibt. Non-additive Fakten sollten vermieden werden, da entsprechende Anfragen nicht leicht zu formulieren sind. Doch manchmal kann es sinnvoll sein sie trotzdem zu verwenden, was sich im Laufe dieser Arbeit zeigen wird.
- **Dimensionen:** Dimensionen enthalten die beschreibenden Attribute zu den Fakten. Die Attribute innerhalb der Dimension beschreiben Fakten. Ein Beispiel hierfür wäre, Umsatz nach Kunde, Produkt und Zeit, wobei Kunde, Produkt und Zeit die Dimensionen sind [KIM02]. Bei späteren Analysen werden die Attribute einer Dimension verwendet, um Einschränkungen zu definie-

ren, d.h. um die Ergebnisse der Analyse einzuschränken, zum Beispiel „Gib nur die Verkaufszahlen für den Monat Januar“.

In Abbildung 10 ist ein Beispiel für ein dimensionales Datenmodell abgebildet. Innerhalb der Dimensionsmodellierung gibt es unter anderem das Star und Snowflake Schema. In einem Star Schema gibt es eine Faktentabelle und jeweils eine Dimensionstabelle pro Dimension. Innerhalb der Dimensionen gibt es Redundanzen, d.h. es findet keine Normalisierung statt, was aber gleichzeitig zu erheblichen Performancesteigerungen führt. Eine Faktentabelle besteht aus einer Reihe von Fakten und den jeweiligen Primärschlüssel der Dimensionen.

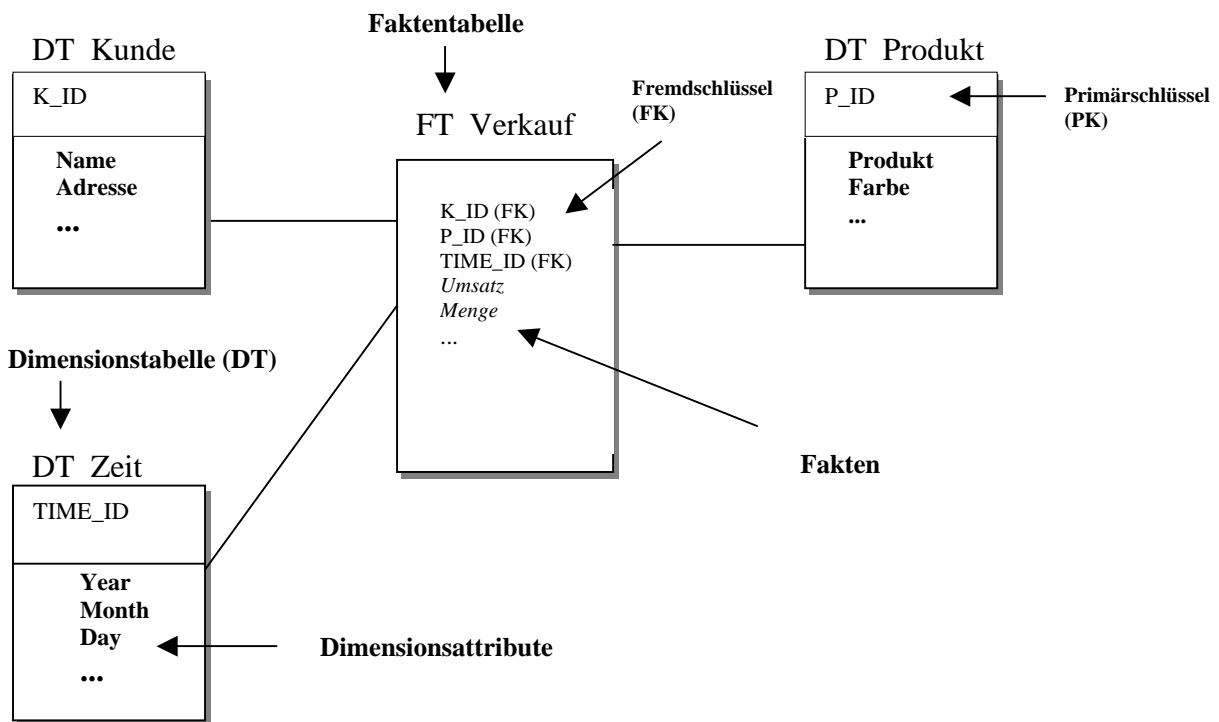


Abbildung 10 Dimensionsmodell – Star Schema Beispiel

Ein Snowflake Schema dagegen besteht aus einer Faktentabelle und mehreren Dimensionenstabellen für jede Dimension, wobei die Dimensionstabellen teilweise normalisiert sind. Beim Snowflake Schema werden Redundanzen eliminiert indem redundante Attribute in weiteren Dimensionen ausgelagert werden. Wenn zum Beispiel die Dimension Produkt ein Attribut „Produktgruppe“ enthält, so würde das innerhalb des Snowflake Schemas bedeuten, dass dieses Attribut in eine weitere Dimensionstabelle Produktgruppe ausgelagert wird. Für weitere Informationen zur Normalisierung sei auf [MIT01] verwiesen.

Es ist nicht zwingend, dass es innerhalb beider Schemata nur eine Faktentabelle gibt. Bei der Galaxy oder auch Constellation genannt, gibt es mehrere Faktentabellen, die aber über gemeinsame Dimensionstabellen zusammenhängen müssen.

2.3 Business Intelligence Tools

In Abschnitt 2.3 werden die Business Intelligence Tools, die im vorherigen Abschnitt kurz erläutert wurden, näher beschrieben. Zunächst wird OLAP erläutert, wobei hier zuerst eine Definition erfolgt und anschließend die unterschiedlichen OLAP Architekturen und OLAP Funktionen beschrieben werden. Im Anschluss erfolgt die Definition von Data Mining, ein weiteres Business Intelligence Tool. Der letzte Teil dieses Abschnitts befasst sich mit den Unterschieden zwischen OLAP und Data Mining.

2.3.1 OLAP

In diesem Abschnitt wird das Business Intelligence Tool OLAP, das auf ein DW zugreift, näher erläutert. Zunächst wird der Begriff OLAP definiert und anschließend werden die unterschiedlichen OLAP Architekturen aufgezeigt. Zum Schluss werden noch die unterschiedlichen OLAP Funktionen beschrieben.

2.3.1.1 Definition

Wie bereits in vorangegangenen Abschnitten erwähnt wurde, greifen verschiedene Analyse Anwendungen auf ein DW zu. Im Folgenden wird eine derartige Analyse Anwendung erläutert. OLAP steht für Online Analytical Processing und bedeutet, dass der Endanwender auf das DW zugreift, um Erkenntnisse zu gewinnen, die ihn in seinen Entscheidungen unterstützen sollen. Es gibt aber noch eine weitere Art der Online-Verarbeitung, nämlich OLTP (Online Transaction Processing). Während OLTP lesend und schreibend auf operative Datenbestände zugreift, greift OLAP nur lesend auf ein DW zu. In der untenstehenden Tabelle 4 sind weitere Unterschiede zwischen OLTP und OLAP aufgeführt.

	OLTP (transaktional)	OLAP (analytisch)
Fokus	Lesen, Schreiben, Modifizieren, Löschen	Lesen, periodisches Hinzufügen
Transaktionsdauer und -typ	kurze Lese-/Schreibtransaktionen	lange Lesetransaktionen
Anfragestruktur	einfach strukturiert	Komplex
Datenvolumen einer Anfrage	wenige Datensätze	viele Datensätze
Datenvolumen eines Systems	Gigabyte (GB) - Terabyte (TB)	Terabyte (TB) - Petabyte (PB)
Datenalter	aktuell - mehrere Monate	aktuell – viele Jahre
Datenmodell	Anfrageflexibles Datenmodell	Analysebezogenes Datenmodell
Verfügbarkeit	sehr hoch	Hoch
Anzahl Benutzer	Hoch	Gering

Tabelle 4 OLTP vs. OLAP

Für OLAP Analysen werden die Daten in einem multidimensionalen Datenwürfel, auch Cube genannt, präsentiert. Der große Vorteil ist, dass Daten aus verschiedenen Perspektiven betrachtet werden können und vor allem ist diese Art der Datenpräsentation einfach zu verstehen.

In Abbildung 11 ist ein Beispiel für einen multidimensionalen Datenwürfel abgebildet. Die Dimensionen entsprechen den Achsen, d.h. sie spannen den Datenwürfel auf. Jede Zelle des Würfels enthält genau einen Wert, der zum Beispiel den Umsatz (Faktum) angibt. Ein derartiger Würfel kann mit Hilfe von OLAP Funktionen (Vgl. 2.3.1.3) analysiert werden.

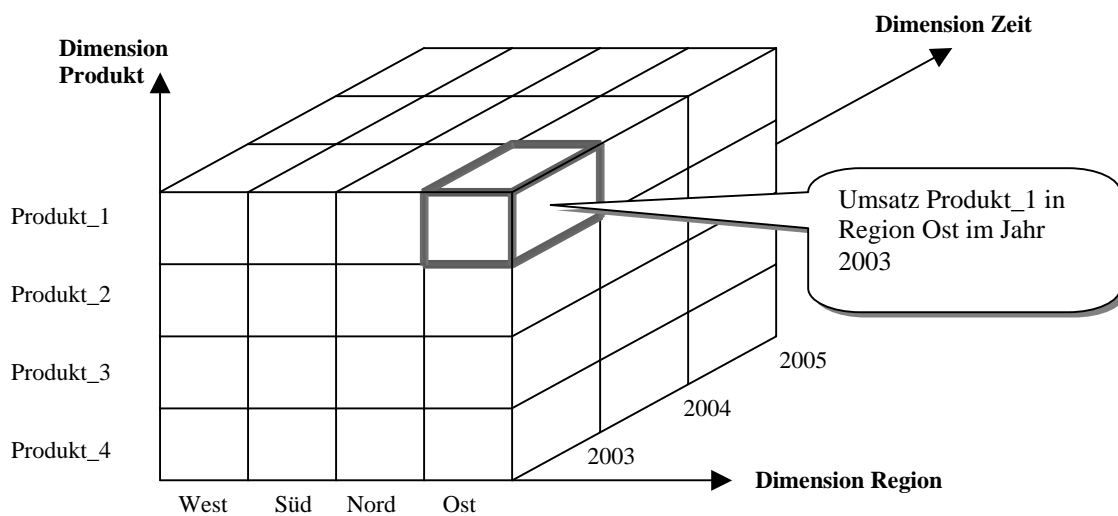


Abbildung 11 Beispiel Datenwürfel

2.3.1.2 OLAP Architekturen

Aus logischer und physischer Sicht unterscheidet man, je nach Art des DWs, zwischen ROLAP (Relationales OLAP), MOLAP (Multidimensionales OLAP) und HOLAP (Hybrid OLAP). Im folgenden werden diese Architekturen erläutert:

- ROLAP: ROLAP verwendet eine gewöhnliche relationale Datenbank. Die relationalen Tabellen bilden die Dimensionen in ein Star Schema ab. Wenn die Dimensionen normalisiert werden, d.h. Entfernung der Redundanzen durch Auslagerung der redundanten Attribute in weitere Dimensionstabellen, erhält man ein Snowflake Schema [KUR99].
- MOLAP: MOLAP verwendet (meist proprietäre) multidimensionale Datenbanken, d.h. Daten werden nicht nur für den Benutzer multidimensional dargestellt, sondern auch so gespeichert. Bei MOLAP wird bereits beim Vorbereiten

ten und Speichern der Rohdaten ein Teil der Analyse vorweggenommen, um Analysezeit zu sparen [KUR99].

- **HOLAP:** Bei HOLAP wird zur Speicherung von Detaildaten eine relationale Datenbank verwendet und für aggregierte Daten eine multidimensionale Datenbank [KUR99].

Des Weiteren gibt es noch FOLAP (Flat-File OLAP) und DOLAP (Desktop OLAP) [KUR99]. Doch diese sind nicht so weit verbreitet wie die oben erwähnten und werden daher nicht näher erläutert. Die Vor- und Nachteile der verschiedenen OLAP Architekturen sind in der untenstehenden Tabelle 5 aufgelistet.

Architektur	Vorteile	Nachteile
ROLAP	<ul style="list-style-type: none"> • Robuste Datenbanktechnologie • Standard-Abfragesprachen (SQL) • Einfacher Datenimport via SQL • Sicherheitsmechanismen auf relationaler Ebene • Große Datenmengen möglich • Am häufigsten für DW verwendet 	<ul style="list-style-type: none"> • Standard-SQL/92 für OLAP nicht ausreichend
MOLAP	<ul style="list-style-type: none"> • Sehr gute Antwortzeiten bei kleinen Datenmengen (<1 Gbyte) • Intuitive Abfragesprache • Einfaches Hinzufügen von Dimensionen und Aggregationsstufen 	<ul style="list-style-type: none"> • Problematik dünn besetzter Würfel • Proprietäre DBMS und bisher keine Standard Abfragesprache • Schlechte Antwortzeiten bei größeren Datenvolumina • Zur Integration externer Quellen Transformation zu SQL nötig • Sinkende Effizienz bei Zugriff auf persistentes Speichermedium (miss hit)
HOLAP	<ul style="list-style-type: none"> • Vereinigt Vorteile von ROLAP und MOLAP • MDDB-System greift nicht mehr direkt auf operative Systeme zu, sondern auf die Detaildaten im relationen Teil des DW • Nur geringe Anzahl persistenter Aggregationen nötig 	<ul style="list-style-type: none"> • Höherer Implementations- und Wartungsaufwand • Keine einheitliche Abfragesprache

Tabelle 5 Vor- und Nachteile der verschiedenen OLAP Architekturen [KUR99]

2.3.1.3 OLAP Funktionen

Es gibt verschiedene OLAP Funktionen mit denen die Abfragen durchgeführt werden können. Die Datenanalyse mit einem OLAP Werkzeug ist ein dynamischer Prozess bei dem der Anwender mit Hilfe von multidimensionalen Operatoren durch die multidimensionalen Datenstrukturen navigieren kann [BAU01].

Zu den Standard Funktionen zählen:

- *Drill down:* Detaillierungsgrad wird erhöht, d.h. das Ergebnis wird in einer feineren Form zurückgegeben, wie zum Beispiel Umsatz pro Monat → drill down → Umsatz pro Tag

- *Roll up*: Roll up ist die inverse Operation zu drill down, d.h. das Ergebnis wird auf eine höhere Stufe in der Hierarchie aggregiert. Zum Beispiel Umsatz pro Monat → roll up → Umsatz pro Quartal
- *Pivot*: Die Berichtsdaten können mit Hilfe der Pivot Funktion aus unterschiedlichen Perspektiven betrachtet werden. Mit anderen Worten, drehen des Datenwürfels durch Vertauschen der Dimensionen.
- *Dice*: Mit Hilfe von Dice kann ein Teil aus dem bestehenden Datenwürfel sozusagen „ausgeschnitten“ werden. Das Ergebnis entspricht wieder einem Würfel.
- *Slice*: Slice bedeutet „in Scheiben schneiden“. Anders gesagt, bei Slice erfolgt eine Projektion auf einen Unterraum des betrachteten Datenwürfels durch Weglassen einer Dimension.
- *Drill Across*: Mit Hilfe dieser Funktion können Daten aus mehreren Faktentabellen in einen OLAP Report dargestellt werden. Voraussetzung hierfür ist, dass die Faktentabellen eindeutig über gemeinsame Dimensionstabellen zusammenhängen.

2.3.2 Data Mining

Ein weiteres Business Intelligence Tool, das auf ein DW zugreifen kann, ist Data Mining. Data Mining wird in diesem Abschnitt näher erläutert.

Als Business Intelligence Tool wurde für diese Arbeit ausschließlich OLAP verwendet und nicht Data Mining. Da Data Mining aber innerhalb der Exploration Architektur in Zukunft verwendet werden soll, wird im Folgenden der Begriff Data Mining erläutert.

Grob gesagt geht es bei Data Mining um das Durchsuchen von Unternehmensdaten, um unbekannte Zusammenhänge herauszufinden. Bei Data Mining werden Analysen durch das Werkzeug durchgeführt, um Zusammenhänge in großen Datenmengen aufzudecken, und nicht durch den Benutzer. Hierfür gibt es verschiedene Verfahren wie zum Beispiel die Assoziationsanalyse. Bei der Assoziationsanalyse wird nach statistischen Mustern gesucht, die Aussagen wie bei folgendem bekannten **Beispiel** ermöglichen:

Die Assoziationsanalyse hat ergeben, dass zum Beispiel 50% der Leute, die Bier kaufen, auch Chips kaufen. Der Anwender muss anschließend derartige Aussagen interpretieren und mögliche Maßnahmen einleiten. Mögliche Maßnahme zu obiger Aussage könnte sein, dass Chips und Bier im selben Regal abgelegt werden. Weitere Verfahren sind zum Beispiel Clusterbildung, Klassifikation und Regression [BAU01].

Ein DW ist ideal für Data Mining, da ein DW den großen Vorteil besitzt, dass die Informationen konsistent sind, was sehr wichtig für Data Mining ist, um sinnvolle Zusammenhänge zu ermitteln. Ein weiterer Vorteil ist, dass ein DW in der Regel sehr große Datenmengen beinhaltet und somit für den Benutzer unüberschaubar wird. Somit ist der Einsatz von derartigen Anwendungen unverzichtbar um Zusammenhänge zu entdecken. Auch wenn großen Datenmengen ideal für den Einsatz von Data Mining sind, bedeuten sie gleichzeitig einen Nachteil, da Data Mining Anwendungen oft zuviel Rechenzeit benötigen um interessante Ergebnisse bzw. Zusammenhänge zu entdecken. Daher wird Data Mining eher auf Data Marts angewendet, weil Data Marts bekanntermaßen geringere Datenmengen besitzen und somit schneller analysiert werden können [MAR98].

2.3.3 Unterschiede zwischen OLAP und Data Mining

Es gibt eine Reihe von Unterschieden zwischen OLAP und Data Mining, die im Folgenden erläutert werden:

- OLAP verwendet konkrete Fragestellungen, d.h. der Anwender muss zuvor Fragestellungen formulieren. Bei Data Mining hingegen muss der Anwender nur die Analysemethode auswählen. Der Analysevorgang läuft anschließend ohne weitere Eingriffe durch den Benutzer ab.
- OLAP Anwender müssen keine Experten sein, um Analysen durchzuführen. Während bei Data Mining Experten bei der Analyse benötigt werden, um die Ergebnisse des Analysevorgangs interpretieren zu können.
- OLAP verwendet multidimensionale Strukturen um Analysen durchführen zu können. Bei Data Mining sind multidimensionale Strukturen nicht zwingend erforderlich.
- OLAP Analysen liefern konkrete Zahlen wie zum Beispiel Umsatz oder Verkaufszahlen. Data Mining Analysen hingegen liefern irgendwelche Aussagen, die anschließend interpretiert werden müssen.

3 Anforderungen

Das Ziel dieser Arbeit ist die Analyse des Prozesses der Weiterleitung von Datenänderungen zwischen Informationssystemen, um beispielsweise anschließend Optimierungen durchführen zu können. Hierfür wird eine Umgebung benötigt, welche eine derartige Analyse ermöglicht.

Dieses Kapitel befasst sich mit der Ermittlung der Anforderungen für die Exploration Manager Umgebung, welche eine Analyse des Prozesses ermöglicht. Hierbei wird zunächst die Logstruktur anhand eines Beispiels analysiert, um anschließend die eigentlichen Anforderungen mit Hilfe von ersten möglichen Anfragen für das DW zu ermitteln. Mit Hilfe der ermittelten Anfragen können die Fakten festgelegt werden.

3.1 Analyse der Logstruktur

Bevor mit der eigentlichen Anforderungsanalyse begonnen werden kann muss zunächst die Logstruktur analysiert werden. Hierbei geht es speziell darum zu verstehen, welche Daten sich im Log befinden bzw. welche Daten während eines Propagationsprozesses vom LM im Log gespeichert werden. Wie bereits erläutert wurde besteht der Log aus den 5 Tabellen *Event_Trail*, *Event*, *Event_Detail*, *Event_Detail_Trail* und *Event_Predecessor*. Im Folgenden wird ein Beispiel erläutert, um klar zu machen, welche Informationen während eines Propagationsprozesses im Log gespeichert werden können.

Beispiel: Angenommen das Propagationsskript PS_1 definiert die Abhängigkeit zwischen Quellsystem „Sales System“ und Zielsystem „FLP System“. Sobald eine Datenänderung im Quellsystem auftritt führt der PM PS_1, welches er aus dem Repository erhält, als Prozess aus. Innerhalb des Prozesses treten die Ereignisse Start, Transform, Propagate und Successful_Termination (Succ_Term) auf. Beim Eintreten dieser Ereignisse wird eine Benachrichtigung an den LM gesendet, welcher anschließend zu den jeweiligen Ereignisse diverse Informationen speichert. In Abbildung 12 ist der Ablauf innerhalb des PS_1 abgebildet.

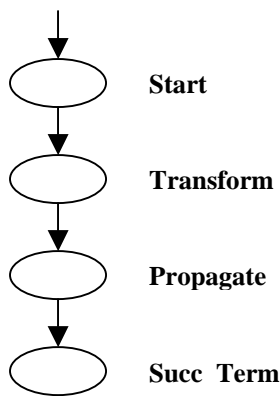


Abbildung 12 Beispiel für Sequenz von Ereignissen während der Ausführung eines Propagationsprozesses

In der untenstehenden Tabelle sind die einzelnen Werte der Details bzw. der Informationen zu den einzelnen Ereignissen, aufgelistet:

Ereignis	Detail	Wert
Start	<ul style="list-style-type: none"> • Source_System • Source_System_Schema 	<ul style="list-style-type: none"> • Sales System • Sales
Transform	<ul style="list-style-type: none"> • Trans_Script 	<ul style="list-style-type: none"> • Sales.xslt
Propagate	<ul style="list-style-type: none"> • Destination_System • Destination_System_Schema 	<ul style="list-style-type: none"> • FLP System • FLP
Succ_Term	-	-

Tabelle 6 Details und Werte zu den Ereignissen des Beispiels

In Abbildung 13 sind die einzelnen Tabellen zu sehen, wobei die jeweiligen Schlüssel der Tabellen nicht abgebildet sind.

Beim Eintreten einer der oben genannten Ereignisse wird eine Benachrichtigung an den LM gesendet. Voraussetzung für das Empfangen und Speichern von Informationen ist, dass die Metadaten, d.h. Informationen zu den Ereignissen, in den Tabellen *Event* und *Event_Detail* zuvor spezifiziert wurden. Beim Empfang der Benachrichtigung speichert der LM, die in Tabelle 6 aufgeführten Werte zu den jeweiligen Ereignissen. In der Tabelle *Event_Detail_Trail* werden also die Werte „Sales System“, „Sales“, „Sales.xslt“, „FLP System“ und „FLP“ abgelegt. In *Event_Trail* wird zu jedem Ereignis ein Tupel gespeichert, welches den Zeitstempel, die Propagationsprozess ID (Hier: P_P_ID_1) und Propagationsskriptnamen (Hier: PS_1) enthält. In diesem Beispiel gibt es insgesamt vier Tupel in *Event_Trail*. Beispielsweise wird hier zum Start Ereignis der Zeitstempel (25.03.2005 12:00:00), der Propagationsskriptname (PS_1) und die Propagationsprozess ID (P_P_ID_1) gespeichert.

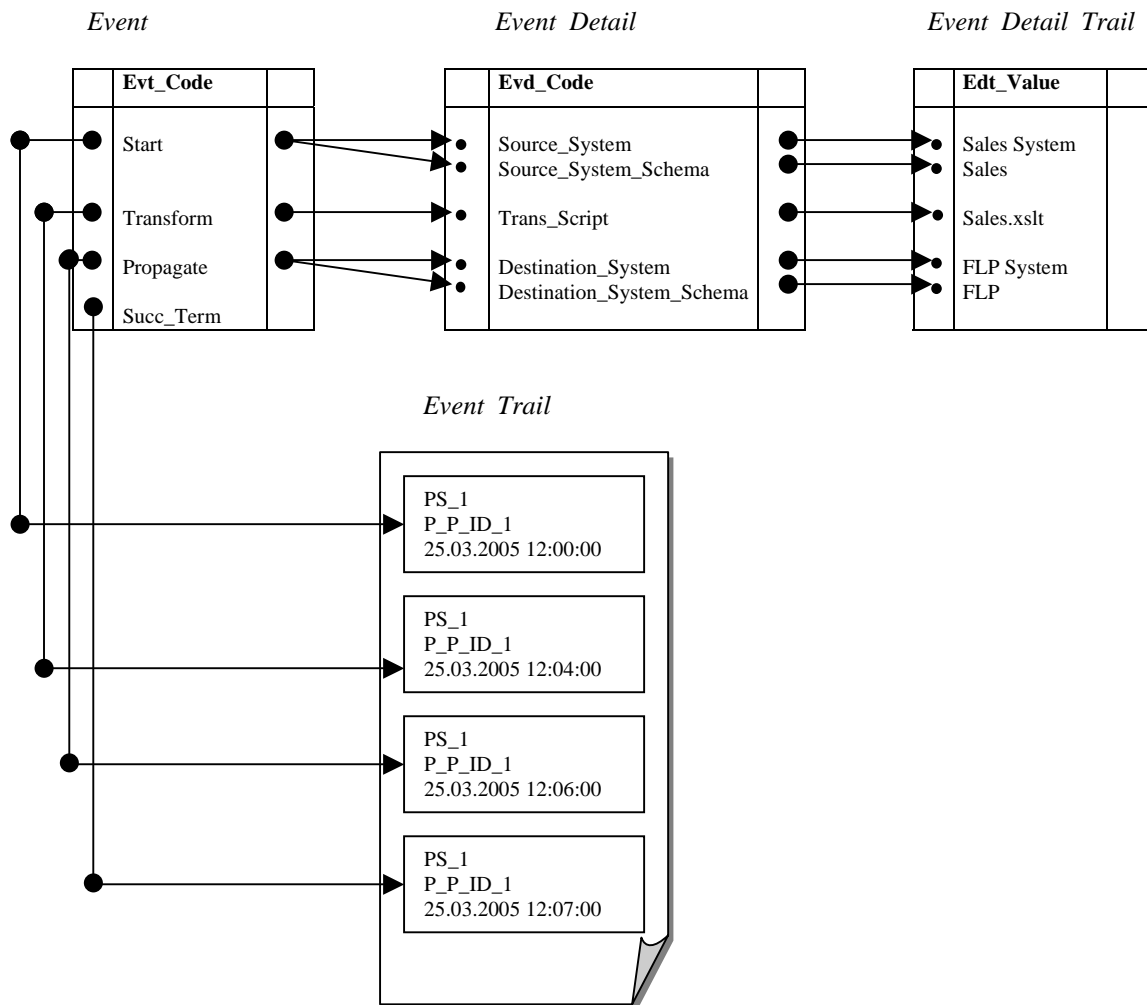


Abbildung 13 Tabellen des Logs mit den entsprechenden Informationen

Aus obigem Beispiel wird ersichtlich, welche Informationen zu einem Propagationsprozess gespeichert werden können. Hieraus lassen sich erste Hinweise für interessante Fragestellungen, die innerhalb der Anforderungsanalyse formuliert werden müssen, gewinnen. Für weitere Hinweise müssen die aus Abschnitt 2.1.2 definierten Ereignissen bezüglich ihrer Bedeutung und der jeweiligen Details betrachtet werden, wobei nur die *Propagation Process Events* betrachtet werden. Hierbei ist es sinnvoll die Ereignisse in zwei Gruppen zu unterteilen. Die eine Gruppe enthält sämtliche Ereignisse, die den inaktiven und aktiven Teil der Ausführung eines Propagationsprozesses trennen. Hierzu zählen die Ereignisse Start, Sleep, Wake Up, Termination Successful, Forced Termination, Partial Rollback Termination, Restart und Failure. Wenn die Ausführung eines Propagationsprozesses vor Eintritt eines solchen Ereignisses aktiv war, ist sie nach Eintritt inaktiv. Umgekehrt gilt, wenn die Ausführung eines Propagationsprozesses vor Eintritt eines solchen Ereignisses inaktiv war, ist sie nach Eintritt aktiv. Beispielsweise ist die Ausführung eines Propagationsprozesses vor Eintritt eines Sleep Ereignisses aktiv und nach Eintritt inaktiv. Für die andere Gruppe, die aus den Ereignissen Transform, Propagate, Condition und Filter besteht, gilt eine derartige Trennung nicht. Die Ausführung eines Propagationsprozesses ist vor

und nach Eintritt eines solchen Ereignisses aktiv. Zum Beispiel ist die Ausführung eines Propagationsprozesses vor und nach Eintritt eines Transform Ereignisses aktiv.

3.2 Anforderungsanalyse

Aufbauend auf der Log Analyse und der aus Abschnitt 2.1.2 definierten Ereignisse war es möglich erste Anfragen zu formulieren. Diese Anfragen stellten die Grundlage für die Festlegung der Fakten dar.

Im Folgenden wird zunächst beschrieben wie erste allgemeine Anfragen ermittelt werden konnten. Erste Hinweise konnten aus dem im vorherigem Abschnitt beschriebenen Beispiel gewonnen werden. Um weitere Anfragen formulieren zu können mussten die aus Abschnitt 2.1.2 definierten Ereignisse betrachtet werden.

Aus dem in 3.1 beschriebenen Beispiel ergibt sich eine interessante Fragestellung bezüglich der Dauer eines Propagationsprozesses, da zu jedem Ereignis innerhalb der Tabelle *Event_Trail* der Zeitstempel gespeichert wird. Was ebenfalls interessant sein könnte bezüglich der Dauer eines Propagationsprozesses ist die Ermittlung der realen Dauer, da während eines Sleep Ereignis der Propagationsprozess sozusagen stillgelegt ist. Eine weitere Anfrage ergibt sich aus der Bedeutung des Start Ereignisses, welches den Beginn eines Propagationsprozesses kennzeichnet. In diesem Zusammenhang kann beispielsweise die Anzahl der Propagationsprozesse ermittelt werden.

Zusammenfassend sind in Tabelle 7 erste allgemein definierte Anfragen einschließlich der zueinander in Beziehung stehenden Tabellen aus dem Log aufgelistet.

Anfragen	Benötigte Tabellen (Log)
1) Anzahl der Propagationsprozesse	Event, Event_Trail
2) (Reale) Dauer eines Propagationprozesses	Event, Event_Trail
3) Welche Details hat ein Propagationsprozess?	Event, Event_Trail, Event_Detail, Event_Detail_Trail
4) Wie oft wurde ein Propagationsprozess wiedergestartet?	Event, Event_Trail
5) Welche Systeme wurden wie oft miteinander verbunden?	Event_Detail, Event_Detail_Trail, Event_Trail

Tabelle 7 Beispiele für erste mögliche Anfragen

Erst durch die Formulierung der obigen Anfragen konnten die Fakten festgelegt werden und zwar indem die unterschiedlichen Ebenen der einzelnen Anfragen identifiziert wurden. Im Folgenden werden die unterschiedlichen Ebenen zu den Anfragen ermittelt:

1) *Anzahl der Propagationsprozesse*: Zur Ermittlung der Anzahl der Propagationsprozesse werden die Informationen lediglich auf Propagationsprozessebene benötigt.

2) *(Reale) Dauer eines Propagationsprozesses*:

Hierfür wird im Gegensatz zu 1) eine Ebene tiefer benötigt, d.h. es müssen die einzelnen Ereignisse betrachtet werden.

3) *Details eines Propagationprozesses*:

Für diese Anfrage ist eine Betrachtung auf Ereignis – Detail Ebene, d.h. noch eine Ebene tiefer als 2), erforderlich.

4) *Wie oft wurde ein Propagationsprozess wiedergestartet*: Betrachtung derselben Ebene wie 2), da hierfür das Ereignis Restart betrachtet werden muss.

5) *Welche Systeme wurden wie oft miteinander verbunden*: Betrachtung derselben Ebene wie 2). Es spielt keine Rolle, dass auf die Tabelle *Event_Detail_Trail* zugegriffen wird, denn die Granularität ist ausschlaggebend. Da ein Propagation Ereignis nur ein Zielsystem haben kann ist die Anfrage auf Ereignis Ebene.

Hieraus wird ersichtlich, dass es insgesamt drei verschiedene Ebenen, die in Abbildung 14 abgebildet sind, gibt. Die unterschiedlichen Ebenen beschreiben den jeweiligen Detaillierungsgrad, das bedeutet wie detailliert werden die Informationen für die entsprechenden Anfragen benötigt.

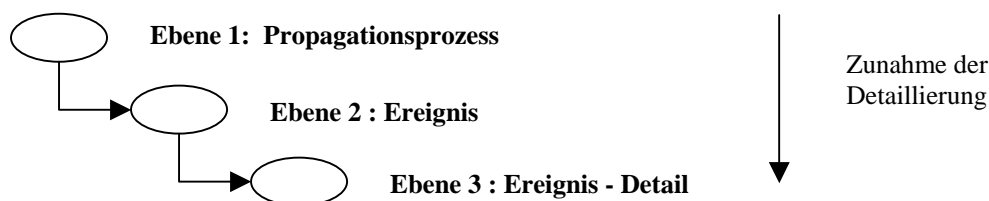


Abbildung 14 Ebenen der Anfragen

Zusammenfassend sind in Tabelle 8 die Fakten einschließlich der zugehörigen Anfragen und Ebenen aufgelistet. Das Festlegen der Fakten stellt die Grundlage für den konzeptuellen Entwurf, der im nächsten Kapitel beschrieben wird, dar.

Anfrage	Detailebene	Fakten
1) Anzahl der Propagationsprozesse	1	Number_of_Process
2) (Reale) Dauer eines Propagationprozesses	2	Duration
3) Welche Details hat ein Propagationprozess?	3	Value
4) Wie oft wurde ein Propagationsprozess wiedergestartet?	2	Number_of_Restart
5) Welche Systeme wurden wie oft miteinander verbunden?	2	Connection

Tabelle 8 Definierte Fakten für die jeweiligen Anfragen

4 Design des Data Warehouses

Die ermittelten Anforderungen aus Kapitel 3 bilden die Grundlage für den Entwurf des DWs. In Kapitel 4 geht es um den Entwurf des DWs, wobei zunächst die Architektur erläutert wird und anschließend die einzelnen Entwurfsphasen. Begonnen wird mit dem konzeptuellen Entwurf, der die Grundlage für den logischen Entwurf darstellt. Der letzte Abschnitt befasst sich mit dem physischen Entwurf. Anzumerken sei, dass in Kapitel 4 lediglich der Entwurf behandelt wird und nicht die Umsetzung mit einem speziellen Werkzeug. Die eigentliche Umsetzung wird in Kapitel 7 erläutert.

4.1 Architektur

In Abbildung 15 ist die gesamte Architektur einschließlich der verschiedenen Bereiche abgebildet.

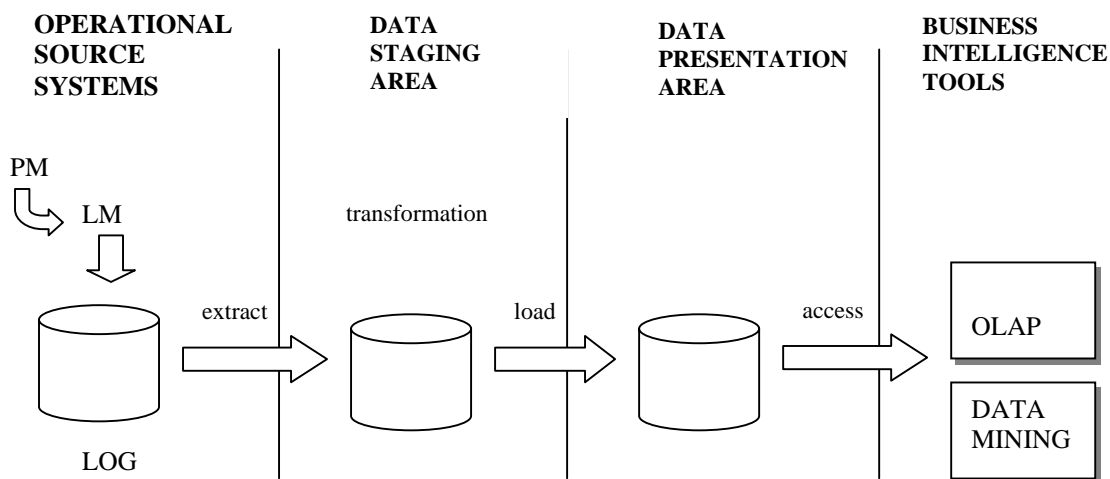


Abbildung 15 Exploration Manager Architektur

Aus obiger Abbildung wird ersichtlich, dass der Log zu den Operational Source Systems gehört und sich damit außerhalb des DWs befindet. Da die Daten im LOG nicht ohne weiteres genutzt werden können, um Analysen durchzuführen, werden sie in einem DW organisiert und dauerhaft abgelegt. Mit anderen Worten, sie werden strukturiert und so organisiert, dass sie auch für Laien verständlich sind, um anschließend Analysen durchzuführen. Dazu müssen aber zunächst die erforderlichen Daten aus dem Log in den Data Staging Area extrahiert, d.h. gelesen und kopiert, werden. Innerhalb des Data Staging Areas werden diverse Transformationen der Daten durchgeführt, wie zum Beispiel das Definieren von einheitlichen Namen. Im Anschluss an die Transformation erfolgt das Laden der aufbereiteten Informationen in den Data Presentation Area, wo die Informationen integriert und tatsächlich dauerhaft abgelegt werden.

Erst wenn die Informationen sich im Data Presentation Area befinden, ist ein Zugriff durch Benutzer möglich, um zum Beispiel OLAP oder Data Mining Analysen durchzuführen.

4.2 Konzeptueller Entwurf

Beim Konzeptuellen Entwurf wird ein Schema für das Zielinformationssystem erstellt und damit die Grobstruktur der Daten, die das IS verwaltet. Dabei soll das Schema unabhängig vom Datenbanksystem (relational oder multidimensional) sein, das für das DW verwendet werden soll. Es gibt eine Reihe von Modellen für den konzeptuellen Entwurf wie zum Beispiel COCOM (Common Conceptual warehouse Model) [SCH03], StarER [TBC99] oder Dimensional Fact Model [GR98] [GR99] [GMR98]. Für diese Arbeit wurde das Dimensional Fact Model (DFM) von Rizzi und Golfarelli verwendet.

Das DFM besagt, dass beim Konzeptuellen Entwurf für jede Faktentabelle Attributsbäume, Dimensionen, Fakten und Hierarchien definiert werden müssen. Des Weiteren ist das Definieren der Granularität ein sehr wichtiger Punkt. Unter Granularität wird der Detaillierungsgrad der Informationen verstanden. Es ist deshalb so wichtig, weil die falsche Wahl des Detaillierungsgrades dazu führen kann, dass Analysen in Zukunft nicht durchgeführt werden können. Angenommen es wird als Granularität Umsatz pro Monat gewählt, da momentan nur Analysen auf Monatebene von Interesse sind. Im Laufe der Zeit aber auch Analysen auf Tagesebene durchgeführt werden sollen. Dies wäre dann aber nicht möglich, da als Granularität Umsatz pro Monat definiert wurde. Aus diesem Grund wird in der Regel der feinste Detaillierungsgrad gewählt, um so auch flexibel zu sein, d.h. um zukünftige Analyse Anforderungen zu gewährleisten.

Innerhalb des DFM wird der Realitätsausschnitt als Dimensionsschema bezeichnet, der aus einer Menge von Faktenschemata, dessen Grundelemente Fakten, Dimensionen und Hierarchien sind, besteht. Eine genauere Definition ist in den Papern [GR98] und [GR99] aufgeführt.

Im vorherigen Kapitel wurde festgestellt, dass es drei verschiedene Ebenen für die jeweiligen Fakten gibt. Bei der Definition der Faktentabellen spielt die Granularität eine wichtige Rolle. Jedoch müssen auch andere Gesichtspunkte bei der Definition der Faktentabellen betrachtet werden. Zunächst wird ein Überblick über die verschiedenen Faktentabellen gezeigt und anschließend die einzelnen Faktenschemata zu den Faktentabellen erläutert.

Aus Designgründen wurde entschieden, dass sowohl die Anzahl (*Number_of_Prozess*) als auch die Dauer (*Duration*) eines Propagationsprozesses mit einer Faktentabelle bestimmt werden sollte. Der Grund dafür ist, dass die Anzahl der Propagationsprozesse durch einfache Aggregation ermittelt werden kann. Genaueres hierzu wird im Faktenschema der jeweiligen Faktentabelle beschrieben. Des Weiteren wurden zwei weitere Faktentabellen definiert, wobei eine genutzt wird, um die Details eines Propagationsprozesses (*Value*) zu analysieren und die andere um die Verbindungen (*Connection*) zwischen Quell- und Zielsystemen zu analysieren. Der Grund wieso für *Connection* eine separate Faktentabelle definiert wurde, obwohl es sich auf derselben Ebene wie zum Beispiel *Duration* befindet, ist der, dass hierzu zusätzliche Informationen benötigt werden. Diese zusätzlichen Informationen sind jedoch für die Fakten *Duration*, *Number_of_Process* und *Number_of_Restart* nicht erforderlich. Beispielsweise beschreiben Quell- und Zielsystem das Faktum *Connection*, was aber wiederum für *Number_of_Process* nicht erforderlich ist. Genaueres wird in den jeweiligen Faktenschemata beschrieben. In Tabelle 9 sind die Faktentabellen einschließlich der Fakten für die sie genutzt werden können aufgelistet.

Fakten	Faktentabelle
Number_of_Process	Fact_Table_1
Duration	Fact_Table_1
Number_of_Restart	Fact_Table_1
Connection	Fact_Table_2
Value	Fact_Table_3

Tabelle 9 Faktentabellen einschließlich der Anfragen

Im Folgenden sind die Faktenschemata zu den Faktentabellen erläutert und im Anschluss daran die Überlappungen zwischen den einzelnen Faktenschemata:

- *Fact_Table_1*
Semantik der Faktentabelle *Fact_Table_1*: Ein Tupel in der *Fact_Table_1* entspricht dem Intervall zwischen zwei nachfolgenden Ereignissen, d.h. wenn zum Beispiel während eines Propagationsprozesses fünf Ereignisse eintreten, werden in der *Fact_Table_1* vier Tupel dazu gespeichert (Vergleiche Abbildung 16).

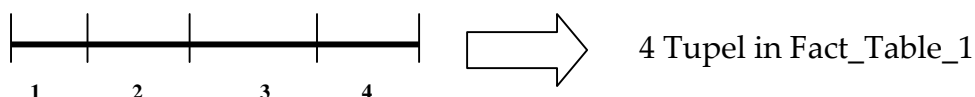


Abbildung 16 Intervall zwischen zwei nachfolgenden Ereignissen

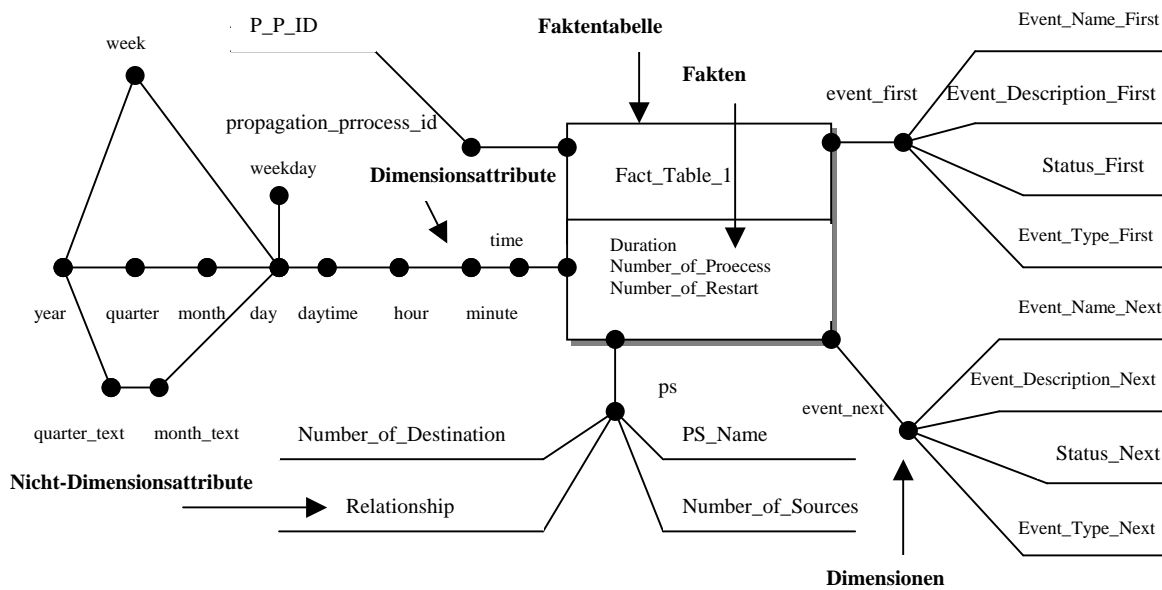


Abbildung 17 Faktenschema zu Fact_Table_1

In Abbildung 17 ist das Faktenschema zu Fact_Table_1 abgebildet, aus dem ersichtlich wird, dass fünf Dimensionen (*Propagation_Process_ID*, *PS*, *Time*, *Event_First* und *Event_Next*) definiert wurden. Diese beschreiben die Fakten *Number_of_Process*, *Duration* und *Number_of_Restart*. Beispielsweise ergibt sich das Faktum *Duration* aus der Anfrage „(Reale) Dauer eines Propagationsprozesses“. Anzumerken sei, dass auf Grund dessen, dass in *Fact_Table_1* ein Intervall zwischen zwei nachfolgenden Ereignissen gespeichert wird, zwei Dimensionen notwendig sind. Die Dimension *Event_First* wird für das erste Ereignis und die Dimension *Event_Next* für das nachfolgende Ereignisse eines Intervalls verwendet. In Tabelle 10 sind alle Fakten der *Fact_Table_1* aufgelistet.

Fakten	Faktenart	Semantik
Duration	Additive	Dauer eines Ereignisses
Number_of_Process	Semi-additive	Zur Ermittlung der Anzahl der Propagationsprozesse. Faktum kann den Wert 1 oder 0 haben. Beim Start Ereignis 1, sonst 0
Number_of_Restart	Semi-additive	Zur Ermittlung der Anzahl der wiedergestarteten Propagationsprozesse, was für spätere Analyse interessant sein könnte. Analog zu Number_of_Process kann dieses Faktum den Wert 1 oder 0 haben, mit dem Unterschied, dass die 1 nur beim Restart Ereignis gesetzt wird, sonst 0

Tabelle 10 Semantik und Art der Fakten

Es wird zwischen Dimensionsattributen und Nicht-Dimensionsattributen unterschieden. Der Unterschied besteht darin, dass Dimensionsattribute zur Aggregation genutzt werden können, Nicht-Dimensionsattribute dagegen nicht. Dimensionsattribute werden verwendet um Hierarchien zu definieren und Nicht-Dimensionsattribute sind zur Beschreibung. Wie aus Abbildung 17 ersichtlich wird, gibt es nur in der *Time* Dimension Hierarchien. Innerhalb der *Event_First*, *Event_Next*, *PS* und *Propagation_Process_ID* Dimensionen sind Aggregationen nicht möglich, das bedeutet sie beinhalten ausschließlich Nicht-Dimensionsattribute.

Die Semantik der verschiedenen Attribute der *Event_First*, *Event_Next*, *Time*, *PS* und *Propagation_Process_ID* Dimensionen sind in Tabelle 11 definiert. Anzumerken sei, dass *Event_First* und *Event_Next* dieselbe Struktur besitzen.

Attribut	Dimension	Semantik
Number_of_Sources	PS	Anzahl der Quellsysteme, die das Propagationsskript besitzt
Number_of_Destination	PS	Anzahl der Zielsystem, die das Propagationsskript besitzt
Relationship	PS	Beziehung zwischen Quell- und Zielsystem. Entweder 1:1, 1:N oder N:M
PS_Name	PS	Name des Propagationsskriptes
Event_Name_First/Event_Name_Next	Event_First/Event_Next	Name des ersten und nachfolgenden Ereignisses
Event_Description_First/ Event_Description_Next	Event_First/Event_Next	Beschreibung des Ereignisses
Event_Type_First/ Event_Type_Next	Event_First/Event_Next	Typ des Ereignisses. Entweder Process Manager Events oder Propagation Process Events
Status_First/ Status_Next	Event_First/Event_Next	Definiert den Status eines Ereignisses, d.h. ob Propagationsprozess aktiv oder inaktiv ist
Minute	Time	Feinste Granularität, d.h. Einträge in Faktentabelle auf Minutenebene
Hour	Time	Stunde
Daytime	Time	Tageszeit (morgens, mittags, nachmittag, abends und nachts)
Day	Time	Tag, zum Beispiel 24
Weekday	Time	Wochentag, zum Beispiel Montag ; Anmerkung: Weekday befindet sich auf derselben Ebene wie Day
Month	Time	Monat, zum Beispiel 2
Month_Text	Time	Monat, zum Beispiel Februar
Quarter	Time	Quartal, zum Beispiel 2
Quarter_Text	Time	Quartal, zum Beispiel Quartal 2
Week	Time	Woche, zum Beispiel 3
Year	Time	Jahr, zum Beispiel 2005
Time_F	Time	Gesamte Zeit, zum Beispiel 13:00:00
Date_F	Time	Gesamtes Datum, zum Beispiel 23.03.2005
P_P_ID	Propagation_Process_ID	Eine ID entspricht der Ausführung eines Propagationsskriptes

Tabelle 11 Semantik der Attribute

Innerhalb der *Time* Dimension gibt es drei Hierarchien, d.h. dass Aggregationen möglich sind. Die unterste Aggregationsebene ist in allen drei Hierarchien „minute“. Indem sowohl *month* als auch *month_text* definiert wurden, ist es möglich bei der späteren Analyse als Einschränkung Monatsnamen oder Monat als Zahl zu wählen. Für *Quarter* und *Quarter_Text* gilt dasselbe.

- *Fact_Table_2*

Semantik der *Fact_Table_2*: Ein Tupel entspricht einer Verbindung zwischen Quell- und Zielsystem, d.h. bei jeder Ausführung eines Propagationsprozesses wird ein Tupel in *Fact_Table_2* abgelegt.

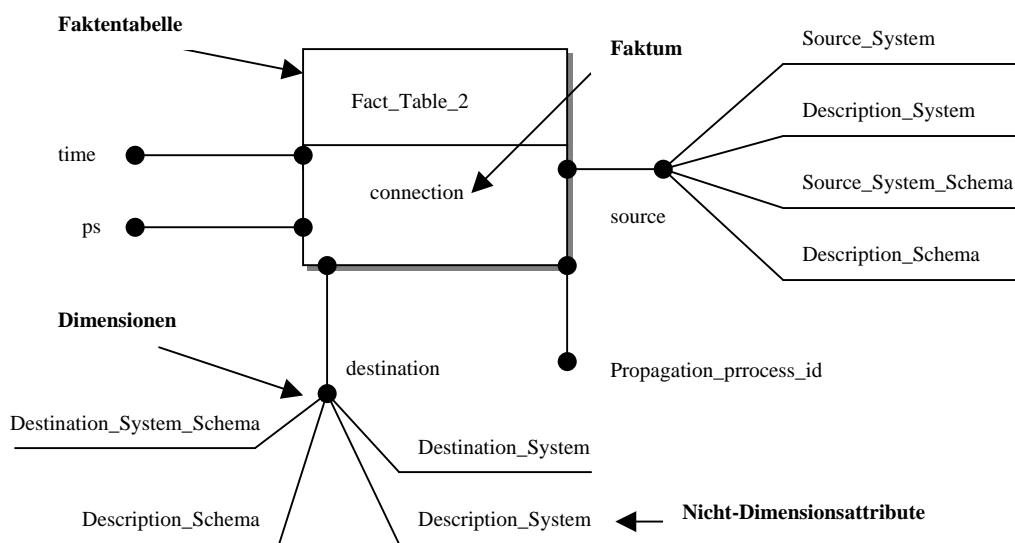


Abbildung 18 Faktenschema zu *Fact_Table_2*

Wie aus Abbildung 18 ersichtlich wird, werden für *Fact_Table_2* fünf Dimensionen verwendet. Hierzu zählen die *Time*, *PS*, *Source*, *Destination* und *Propagation_Process_ID* Dimensionen. Die *Time*, *PS* und *Propagation_Process_ID* Dimensionen wurden bereits im Faktenschema zu *Fact_Table_1* definiert.

Bei *Fact_Table_2* handelt es sich um eine Factless Fact Table, d.h. sie beinhaltet kein echtes Faktum, sondern lediglich ein Dummy-Fakt, um spätere Analysen zu erleichtern. Um zum Beispiel die Anzahl der Verbindungen zwischen IS zu bekommen würde lediglich ein $SUM(Connection)$ genügen. Als Dummy-Fakt wird *Connection* verwendet, das immer den Wert 1 besitzt, wenn ein Tupel in *Fact_Table_2* abgelegt wird.

Sowohl in der *Source* Dimension als auch in der *Destination* Dimension gibt es keine Dimensionsattribute, d.h. es gibt keine Hierarchien und damit sind Aggregationen innerhalb dieser Dimensionen nicht möglich. Beide Dimensionen enthalten ausschließlich Nicht-Dimensionsattribute. In Tabelle 12 sind die einzelnen Attribute einschließlich Semantik aufgeführt.

Attribut	Dimension	Semantik
Source_System	Source	Name des Quellsystem
Description_System	Source	Beschreibung des Quellsystems
Source_System_Schema	Source	Name des Schemas des Quellsystems
Description_Schema	Source	Beschreibung des Schemas
Destination_System	Destination	Name des Zielsystems
Description_System	Destination	Beschreibung des Zielsystems
Destination_System_Schema	Destination	Name des Schemas des Zielsystems
Description_Schema	Destination	Beschreibung des Schemas

Tabelle 12 Semantik der Attribute

- *Fact_Table_3*

Semantik der *Fact_Table_3*: Ein Tupel entspricht einem Detail bzw. den Wert eines Details eines Ereignisses innerhalb eines Propagationsprozesses, d.h. *Fact_Table_3* beinhaltet sämtliche Details wie zum Beispiel Name des Transformationskriptes, welcher innerhalb eines Propagationsprozess verwendet wurde.

Value ist im Allgemeinen non-additive, jedoch kann ein einzelner Wert in der Spalte *Value* auch semi-additive bzw. additive sein. In der *Detail* Dimension gibt es ein Attribut *Status*, das definiert, um welche Art von Faktum es sich handelt. Das Detail Transformationskript ist non-additive, aber das Detail Timeout könnte semi-additive sein.

Der Grund wieso *Fact_Table_3* definiert wurde ist der, dass einerseits dadurch sämtliche Informationen zu einem Propagationsprozess festgehalten werden und andererseits um flexibel zu sein. Flexibel in diesem Zusammenhang bedeutet, dass durch die *Fact_Table_3* sichergestellt ist, dass sämtliche neue bzw. unvorhersehbare Informationen ins DW ohne Probleme hinzugefügt werden können. Ein Beispiel hierfür wäre, wenn im Laufe der Zeit entschieden wird den Benutzer, der die Datenänderung vorgenommen hat, auch zu speichern. Das Hinzufügen des Benutzernamens ist ohne weiteres möglich. In die *Detail* Dimension wird lediglich ein Detail „Benutzername“ hinzugefügt und *Fact_Table_3* enthält den Wert, d.h. den Benutzernamen. Ein Nachteil der *Fact_Table_3* ist jedoch, dass sie keinen großen Spielraum für OLAP Anfragen lässt. Das Faktenschema zu *Fact_Table_3* ist in Abbildung 19 abgebildet.

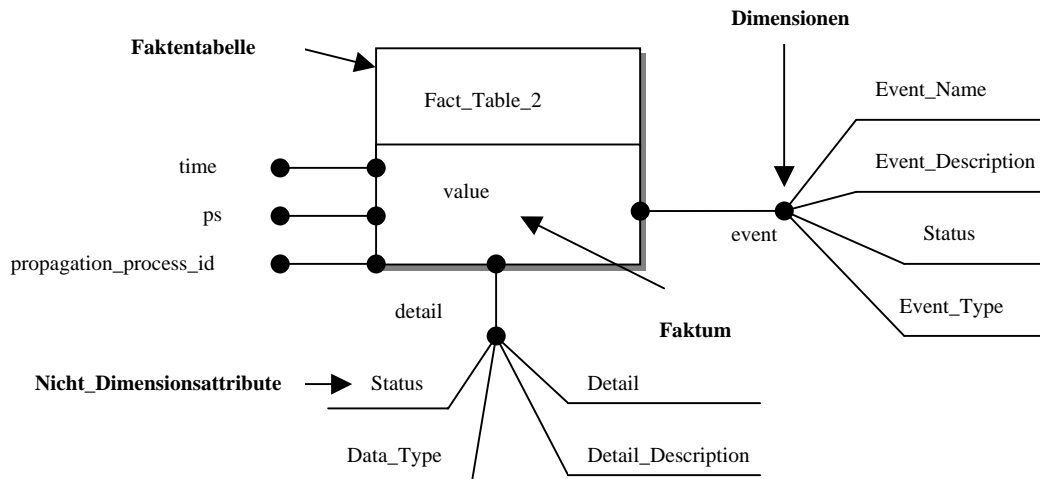


Abbildung 19 Faktenschema zu Fact_Table_3

Für *Fact_Table_3* waren genau wie für *Fact_Table_2* fünf Dimensionen erforderlich. Es werden die *PS*, *Time*, *Propagation_Process_ID*, *Detail* und *Event* Dimensionen verwendet. In der *Detail* Dimension sind keine Aggregationen möglich, d.h. sie enthält ausschließlich Nicht-Dimensionsattribute. In Tabelle 13 sind die Attribute der *Detail* und *Event* Dimensionen einschließlich Semantik aufgelistet.

Attribut	Dimension	Semantik
Detail	Detail	Name des Details, zum Beispiel Name des Transformationsskriptes
Data_Type	Detail	Datentyp des Details
Status	Detail	Status bezieht sich auf die Faktenart, d.h. ob es additive, semi-additive oder non-additive ist. Im Moment enthält Fact_Table_3 nur non-additive Fakten
Detail_Description	Detail	Beschreibung des Details
Event_Name	Event	Name des Ereignisses
Event_Description	Event	Beschreibung des Ereignisses
Status	Event	Definiert den Status eines Ereignisses, d.h. ob Propagationsprozess aktiv oder inaktiv ist
Event_Type	Event	Typ des Ereignisses. Entweder Process Manager Events oder Propagation Process Events

Tabelle 13 Semantik der Attribute

- Überlappungen der Faktenschemata
Sämtliche Faktentabellen wurden in separaten Faktenschemata definiert. Jedoch ist es oft der Fall, dass der Benutzer Informationen aus mehreren Faktentabellen kombinieren möchte. Dabei handelt es sich um sogenannte Cross-Over Operationen. Doch damit derartige Operationen möglich sind müssen die verschiedenen Faktentabellen über gemeinsame Dimensionen zusammenhängen.

Eine mögliche Anfrage könnte zum Beispiel lauten: Anzahl der Propagationsprozesse bei dem Quellsystem (Source_System) „Sales System“ ist. Hierzu

werden *Fact_Table_1* und *Fact_Table_2* benötigt. Auf diese Weise lassen sich weitere Anfragen formulieren, die auf mehrere Faktentabellen zugreifen.

Beispiele:

- *Fact_Table_1* AND *Fact_Table_3*: Anzahl der Propagationsprozesse, die das Transformationskript „Sales.xslt“ verwendet haben.
- *Fact_Table_2* AND *Fact_Table_3*: Welche Systeme wurden miteinander verbunden und haben als Transformationskript „Sales.xslt“ verwendet?
- *Fact_Table_1* AND *Fact_Table_2* AND *Fact_Table_3*: Anzahl der Propagationsprozesse für Source_System „Sales System“, die als Transformationskript „Sales.xslt“ verwenden.

Wie bereits aus obigen Faktenschemata ersichtlich wird, verwenden die drei Faktentabellen teilweise dieselben Dimensionen. Die *Time* Dimension wird zum Beispiel von allen Faktentabellen genutzt. Für die Überlappung von *Fact_Table_1* und *Fact_Table_3* wird die *Event* Dimension verwendet, da die Dimensionen *Event*, *Event_First* und *Event_Next* dieselbe Struktur haben. Sie unterscheiden sich lediglich darin, dass sie für unterschiedliche Zwecke eingesetzt werden. *Event* Dimension beschreibt ein Ereignis für das in *Fact_Table_3* Details gespeichert werden. *Event_First* wird für das erste Ereignis und *Event_Next* für das nachfolgende Ereignis eines Intervalls verwendet. In Abbildung 20 werden die möglichen Überlappungen aufgeführt:

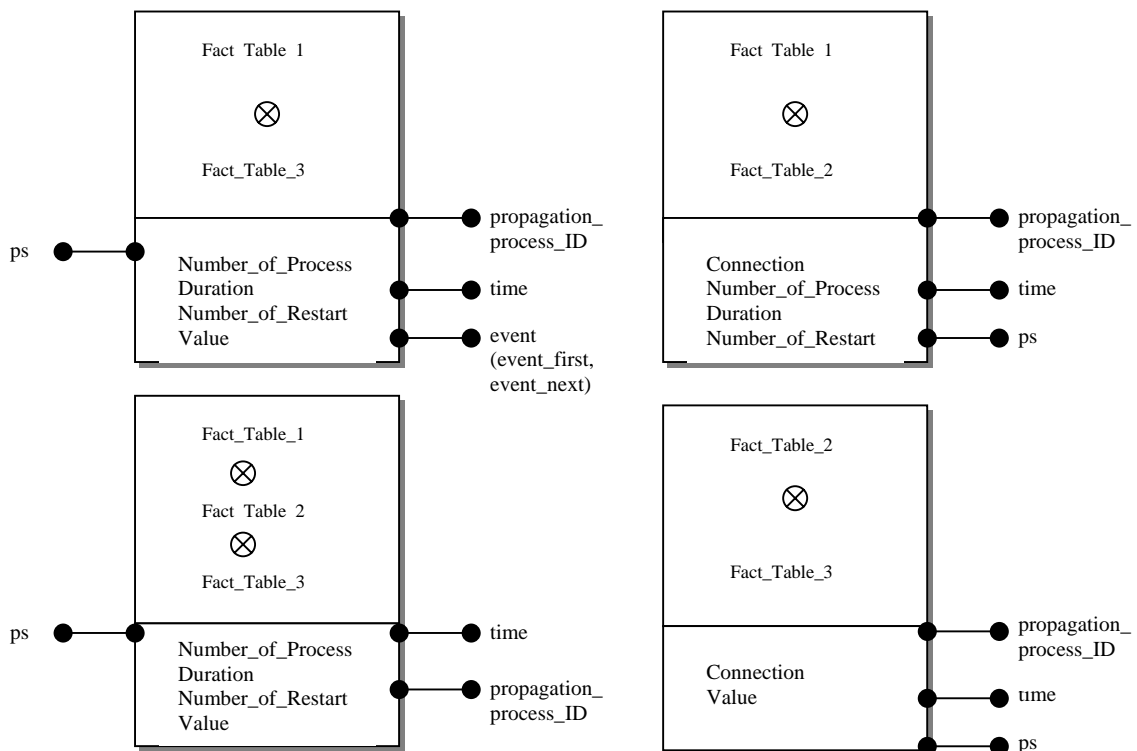


Abbildung 20 Überlappungen der Faktenschemata

4.3 Logischer Entwurf

Die einzelnen Faktenschemata aus dem konzeptuellen Entwurf bilden die Grundlage für den logischen Entwurf.

Ziel des logischen Entwurfs ist das Abbilden der Dimensionsschemata aus dem konzeptuellen Entwurf auf ein Schema, das das gegebene Ziel-Datenbanksystem berücksichtigt, d.h. ob es sich um ein relationales oder multidimensionales Datenbanksystem handelt. Beim Ziel-Datenbanksystem handelte es sich um ein relationales Datenbanksystem, d.h. beim logischen Entwurf erfolgt die Abbildung des konzeptuellen Schemas auf Tabellen. Als logisches Schema wurde das Star Schema verwendet. Abbildung 21 zeigt das Star Schema für das DW. Für *Event*, *Event_First* und *Event_Next* wurde eine Dimensionstabelle verwendet, da alle drei dieselbe Struktur aufweisen. Aus der Abbildung 21 wird ersichtlich, dass es zwischen *Event* und *Fact_Table_1* zwei Beziehungen gibt. Die eine ist für das erste Ereignis und die andere für das nachfolgende Ereignis eines Intervalls.

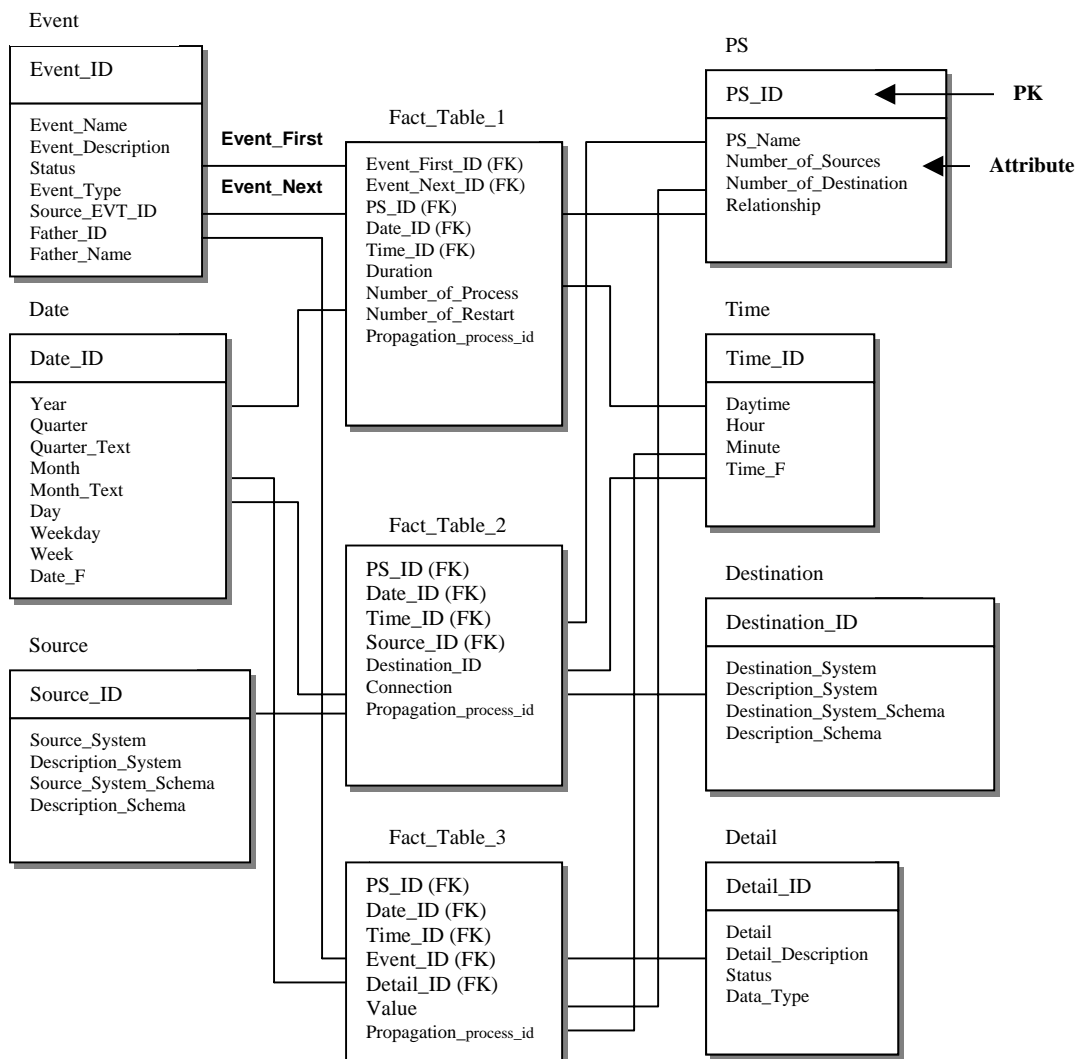


Abbildung 21 Star Schema (Galaxy)

In der Regel besteht ein Star Schema aus einer Faktentabelle und einer Dimensionstabelle pro Dimension. Da aber alle drei Faktentabellen über gemeinsame Dimensionen zusammenhängen wird von einer Galaxy oder auch Constellation gesprochen. Derartige Konstellationen sind bei Cross-Over Operationen von Interesse. In Abbildung 21 ist eine zusätzliche Date Dimension definiert. Dazu wurde die *Time* Dimension aus dem konzeptuellen Entwurf unterteilt in *Time* und *Date* Dimension. *Time* Dimension enthält die Attribute *daytime*, *minute*, *hour* und die Gesamtzeit *time_f*. Sie gilt als statisch, da sie nur einmal gefüllt wird. Insgesamt beinhaltet sie 1440 Tupel ($60 * 24$). Der Grund für diese Aufteilung war der, dass dadurch eine Menge Speicherplatz gespart wird und nicht zu jedem Datum Minute, Stunde und Tageszeit erneut gespeichert werden müssen. Die *Propagation_Process_ID* Dimension wurde nicht als Dimension ins Star Schema aufgenommen. Es handelt sich hierbei um eine degenerierte Dimension. Die *Propagation_Process_ID* enthält lediglich die *P_P_ID* und keine weiteren Attribute. Daher wird die *P_P_ID* in die Faktentabelle als weiteres Attribut abgespeichert. Diese Möglichkeit wird genutzt, falls ein Attribut sehr viele Ausprägungen besitzt, wie zum Beispiel Auftragsnummer, und dieses in Kombination mit anderen Attributen zu einem stärkeren Anwachsen der Dimensionstabelle gegenüber der Faktentabelle führt und somit die Anfragen-Performance verschlechtern würde.

Als Primärschlüssel (PK) für die Dimensionen wird ein Surrogate Key, d.h. künstlich erzeugter Schlüssel, der unabhängig vom Primärschlüssel der Operational Source Systems ist, verwendet. Die Faktentabelle besteht aus diesen Surrogate Keys der Dimensionen und den definierten Fakten. Die Dimensions- und Faktentabellen mit den Attributen und Schlüsseln sehen folgendermaßen aus, wobei die Primärschlüssel der Dimensionen unterstrichen sind und in den Faktentabellen als Fremdschlüssel gelten:

Dimensionstabellen:

```
Event_Dimension(Event_ID, Event_Name, Event_Description, Status,
Source_EVT_ID, Father_ID, Father_Name)
PS_Dimension (PS_ID, PS_Name, Number_of_Sources, Number_of_Destination,
Relationship)
Time_Dimension (Time_ID, Daytime, Hour, Minute, Time_F)
Date_Dimension (Date_ID, Year, Quarter, Quarter_Text, Month, Month_Text,
Day, Week, Weekday, Date_F)
Source_Dimension (Source_ID, Source_System, Description_System,
Source_System_Schema, Description_Schema)
Destination_Dimension (Destination_ID, Destination_System, Description_System,
Destination_System_Schema, Description_Schema)
Detail_Dimension (Detail_ID, Detail, Data_Type, Detail_Description, Status)
```

Faktentabellen:

```
Fact_Table_1 (PS_ID, Time_ID, Date_ID, Event_First_ID, Event_Next_ID,
Duration, Number_of_Process, Number_of_Restart, Propagation_Process_ID)
Fact_Table_2 (PS_ID, Time_ID, Date_ID, Source_ID, Destination_ID,
Connection, Propagation_Process_ID)
Fact_Table_3 (PS_ID, Event_ID, Time_ID, Date_ID, Detail_ID, Value, Propagation_Process_ID)
```

Des Weiteren war es notwendig zwei Views zu definieren, da sonst für OLAP Anfragen die Join Operation zwischen *Event* Dimension und *Fact_Table_1* nicht durchgeführt werden kann. Der Grund ist, dass *Fact_Table_1* die Attribute *Event_ID_First* und *Event_ID_Next* beinhaltet und *Event* Dimension nur *Event_ID*. Da Joins über Namensgleichheit durchgeführt werden, ist es notwendig zwei Views von der *Event* Dimension zu definieren. Eine View beinhaltet als PK *Event_ID_First* und die andere *Event_ID_Next*. Erst dadurch ist ein Join zwischen *Fact_Table_1* und der *Event* Dimension bzw. der *Event_View_First* und *Event_View_Next*. In Abbildung 22 ist dieser Zusammenhang abgebildet.

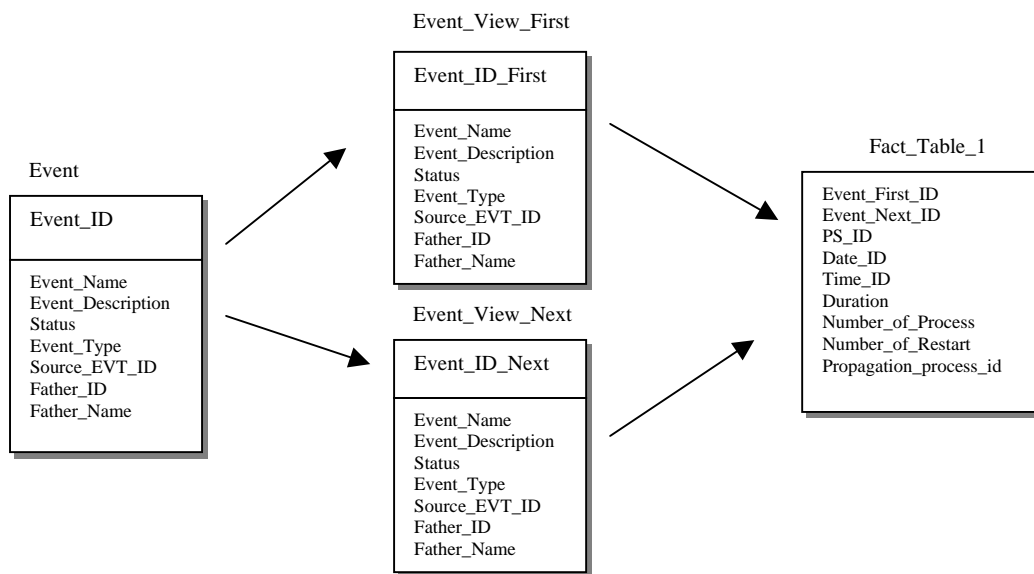


Abbildung 22 Views der Event Dimension

4.4 Physischer Entwurf

Aus dem Star Schema wird in einem weiteren Schritt die physische Implementierung in ein relationales DBMS durchgeführt. Als Beispiel ist die Implementierung der *PS* und *Source* Dimensionen in Abbildung 23 abgebildet. Der vollständige SQL Code zu sämtlichen Tabellen ist im Anhang abgedruckt.

Des Weiteren geht es beim physischen Entwurf um Optimierung der Antwortzeiten und Verfügbarkeit. Im DW werden Indizes aus Performancegründen für PK's und FK's verwendet. Eine geeignete Indexart für eine DW Umgebung ist das Bitmap Indexing, welche das Ergebnis vorausgeführter Joins zwischen Fakten- und Dimensionstabellen darstellt. Für weitere Informationen sei auf [CHY98] verwiesen. Zur Optimierung wird beispielsweise auch die Partitionierung verwendet. Bei der Partitionierung werden physisch eigenständige Partitionen aus den Datenbeständen erstellt. Für weitere Informationen sei auf [KIM98] verwiesen.

```

CREATE TABLE PS_Dimension (
    PS_Surrogate_ID          integer not null,
    PS_Name                  varchar(30),
    Number_of_Sources        integer not null,
    Number_of_Destination    integer not null,
    Relationship              varchar(20) not null );

CREATE TABLE Source_Dimension (
    Source_Surrogate_ID      integer not null,
    Source_System             varchar(30) not null,
    Description_System        varchar(200) not null,
    Source_System_Schema     varchar(30) not null,
    Description_Schema       varchar(200) not null);

```

Abbildung 23 DDL Befehle zu den Dimensionen PS und Source

Ein weiterer Optimierungsansatz ist die Extent- bzw. Speicherbereichszuweisung. Extentzuweisung ist eine Methode zur Zuweisung von neuem Speicherplatz für ein Datenbankobjekt wie zum Beispiel einer Tabelle. Unter Extent wird eine Speichereinheit verstanden, deren Speichergröße bestimmt wie viel Speicherbereich bei der Erzeugung eines Datenbankobjektes reserviert wird. Beispielsweise ist der Standardwert für ein Extent in Oracle 4KB, d.h. wenn ein Datenbankobjekt erzeugt wird, wird diesem Objekt 4KB zugewiesen. Durch die Extentzuweisung werden die Zugriffe innerhalb der Datenbankobjekte auf Grund der geringen Fragmentierung des Speichers beschleunigt.

5 Datenbereitstellung aus dem Log (ETL)

In Kapitel 5 geht es um den ETL-Prozess, d.h. wie bekomme ich die Daten aus dem Log in das DW. Zu Beginn wird der ETL-Prozess erläutert und anschließend werden die unterschiedlichen Schritte des ETL-Prozesses, d.h. Extraktion, Transformation und Laden, beschrieben. Im Abschnitt zur Extraktion wird die Vorgehensweise beschrieben wie sichergestellt werden kann, dass sich nach jedem ETL-Prozess dieselben Daten in den Dimensions- und Faktentabellen befinden.

Ein weiterer wichtiger Punkt ist die Historisierung von Dimensionen. Hierbei geht es darum wie Änderungen innerhalb von Dimensionen gehandhabt werden. Auch hier ist anzumerken, dass in diesem Kapitel lediglich die Konzepte erläutert werden und nicht die Umsetzung mit einem speziellen Werkzeug.

5.1 ETL-Prozess

Es wurde ein ETL-Prozess definiert, wobei sich dieser wiederum in mehrere Sub-ETL-Prozesse unterteilt. Jeder dieser Sub-ETL-Prozesse beschreibt das Extrahieren, Transformieren und Laden von Daten aus dem Log ins DW. In Abbildung 24 sind mehrere Sub-ETL-Prozesse einschließlich der entsprechenden Abhängigkeiten zwischen den einzelnen Tabellen abgebildet. Die Sub-ETL-Prozesse entsprechen den Kreisen im Data Staging Area.

Zu den einzelnen Schritten, wobei diese Vorgehensweise für sämtliche Dimensions- und Faktentabellen gilt, gehören:

1. Laden der erforderlichen Daten aus den Quelltabellen *Ps_Use_Syschema* und *Event_Trail* in PS Dimension. Es werden nur Daten geladen, die sich noch nicht im DW befinden.
2. Laden der erforderlichen Daten aus der Quelltable *Event* in die *Event* Dimension.
3. Füllen der *Time* Dimension, wobei sie lediglich einmal mit Hilfe eines Skriptes gefüllt werden muss.
4. Füllen der *Date* Dimension analog zu *Time* Dimension mit dem Unterschied, dass das Skript bei jedem neuen ETL-Prozess ausgeführt wird, um die aktuellen Daten zu speichern.
5. Ermitteln der neuen Tupel für *Fact_Table_1* durch Zugriff auf die Quelltable *Event_Trail* und die zuvor gefüllten Dimensionen.

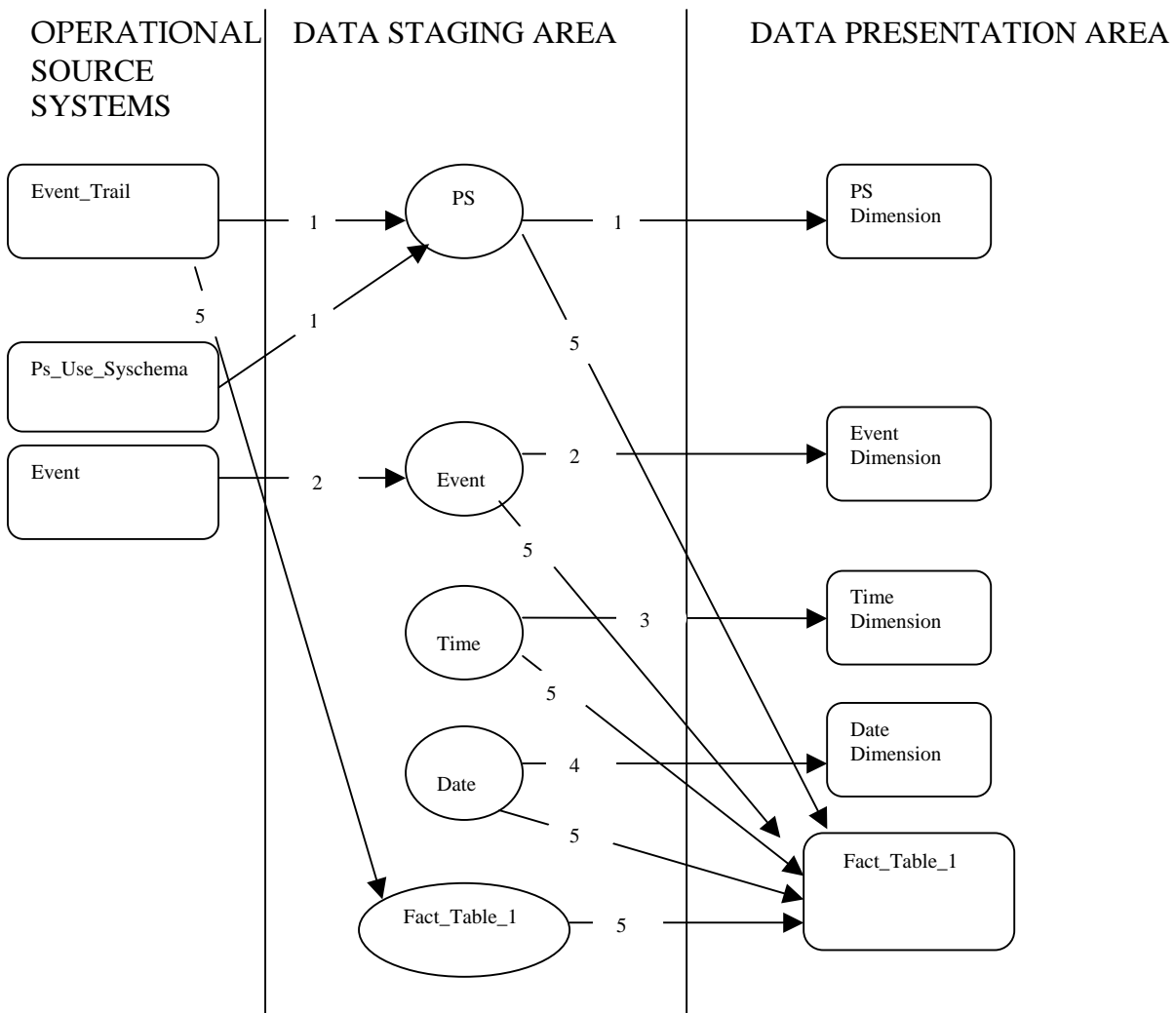


Abbildung 24 Beispiele für Abhängigkeiten und Sequenz einer ETL-Prozess Ausführung

Es macht keinen Unterschied welche der Faktentabellen zuerst gefüllt wird. Jedoch müssen zunächst immer die Dimensionen gefüllt werden, da sonst die Gefahr besteht, dass zu Tupeln in den Quelltabellen keine passende Einträge in den Dimensionen, d.h. keine Primärschlüssel, vorhanden sind. Wenn sämtliche Dimensionen und Faktentabellen gefüllt wurden, d.h. erfolgreiche Ausführung der Sub-ETL-Prozesse, ist ein ETL-Prozess beendet. Jeder weitere ETL-Prozess läuft analog zur beschriebenen Vorgehensweise ab.

5.1.1 Extraktion

Das Extrahieren der Daten aus den operativen Datenbeständen ist der erste Schritt eines ETL-Prozesses. Unter Extraktion versteht man das Lesen und Kopieren von operativen Daten in den Data Staging Area. Hierbei ist es besonders wichtig die Konsistenz zu wahren.

Beim Extrahieren der Daten aus den Quelltabellen *Event* und *Event_Detail* werden bei jedem ETL-Prozess sämtliche Daten in den Data Staging Area extrahiert. Das liegt daran, dass das Datenvolumen in diesen Tabellen immer sehr gering sein wird. Für die Tabellen *Event_Trail* und *Event_Detail_Trail* gilt das nicht, denn diese Tabellen weisen ein großes Datenvolumen auf. Bei jedem ETL-Prozess werden nur die neuen Daten aus *Event_Trail* und *Event_Detail_Trail* in den Data Staging Area extrahiert.

Innerhalb des ETL-Prozesses ist es besonders wichtig, dass nach jedem erfolgreichen Durchlauf eines ETL-Prozesses dieselben Daten in sämtlichen Dimensions- und Faktentabellen gespeichert wurden. Hierfür ist es erforderlich, dass bei der Extraktion der neuen Daten aus den Quelltabellen eine Prüfung durchgeführt wird. Diese Prüfung muss sicherstellen, dass für alle Dimensions- und Faktentabellen dieselben Daten betrachtet werden. Sowohl für die Dimensionen *Source*, *PS* und *Destination* als auch für sämtliche Faktentabellen ist eine derartige Prüfung durchzuführen. Für die Dimensionen *Event* und *Detail* ist eine Prüfung nicht erforderlich, da wie bereits erwähnt wurde, bei jedem ETL-Prozess sämtliche Daten aus den jeweiligen Quelltabellen in den Data Staging Area extrahiert werden.

Im Folgenden wird die Vorgehensweise zur Konsistenzsicherung in den Dimensions- und Faktentabellen erläutert:

Bevor die Daten aus den Quelltabellen in den Data Staging Area extrahiert werden, ist eine Prüfung notwendig, so dass lediglich die neuen Daten betrachtet werden. Um diese Prüfung durchführen zu können, ist es erforderlich nach jedem Sub-ETL-Prozess die ID des zuletzt gespeicherten Tupels zu sichern. Dadurch wird sichergestellt, dass nach erfolgreichem Sub-ETL-Prozess die bereits gespeicherten Informationen nicht erneut in den Data Presentation Area extrahiert werden. Dies ist besonders dann hilfreich und spart vor allem viel Zeit, wenn nach erfolgreicher Ausführung eines Sub-ETL-Prozesses ein Fehler innerhalb des Data Staging Areas auftritt und dadurch alle Sub-ETL-Prozesse erneut ausgeführt werden müssen. Des Weiteren ist es sehr wichtig, insbesondere für die so genannten Cross-Over Operationen zwischen Faktentabellen, dass nach erfolgreichem Ablauf eines ETL-Prozesses dieselben Tupel aus den Quelltabellen betrachtet und gespeichert wurden. Um dies sicherzustellen ist es notwendig zu Beginn die für den aktuellen ETL-Prozess letzte ID zu sichern, welche als Grenze für sämtliche Sub-ETL-Prozesse gilt. Das bedeutet, dass bei Ausführung eines Sub-ETL-Prozesses nur die Tupel betrachtet werden, die sich im Intervall „größer vorheriger letzter ID und kleiner-gleich aktueller letzter ID“ befinden. In Abbildung 25 ist ein mögliches Szenario abgebildet, welches obige Vorgehensweise verdeutlicht.

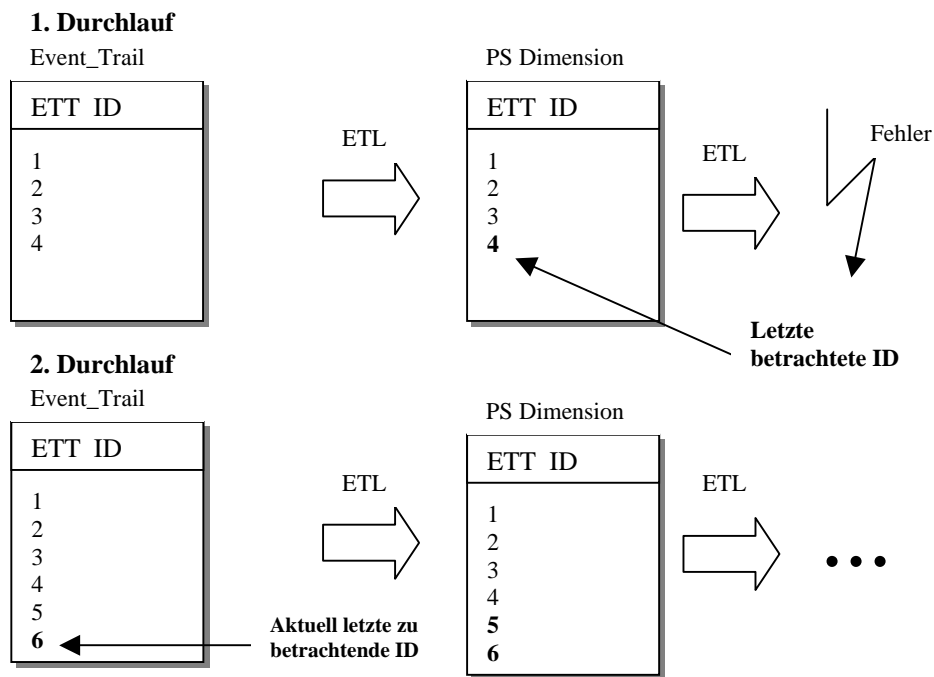


Abbildung 25 Mögliches Szenario eines ETL-Prozesses

Angenommen der Sub-ETL-Prozess für die *PS Dimension* wurde erfolgreich ausgeführt, d.h. nach dem **1. Durchlauf** wurden die neuen Daten in der *PS Dimension* (Data Presentation Area) erfolgreich gespeichert. Anschließend erfolgt ein Fehler innerhalb des Data Staging Areas und sämtliche Sub-ETL-Prozesse müssen im nächsten Durchlauf (**2. Durchlauf**) erneut ausgeführt werden. In diesem Fall ist es möglich, dass sich in der *PS Dimension* mehr Daten befinden bzw. betrachtet wurden als in den anderen Dimensionen. Wichtig ist jedoch, dass nach einem erfolgreichen Ablauf sämtlicher Sub-ETL-Prozesse dieselben Daten aus den Quelltabellen betrachtet und gespeichert wurden. Wie aus obiger Abbildung ersichtlich wird, sind im **2. Durchlauf** weitere Tupel in der Quelltable hinzugekommen. Durch die oben beschriebene Vorgehensweise zur Konsistenzsicherung würden bei erneutem Ausführen des Sub-ETL-Prozesses für die *PS Dimension* lediglich die neu hinzugekommenen Tupel (5 und 6) betrachtet werden. Das betrachtete Intervall wäre in diesem Fall $4 < ETT_ID \leq 6$. Der **2. Durchlauf** verläuft ohne Fehler, d.h. nach Ausführung sämtlicher Sub-ETL-Prozesse befinden sich dieselben Daten in den Dimensions- und Faktentabellen.

5.1.2 Transformation

Innerhalb des Data Staging Areas sind eine Reihe von Transformationen erforderlich. Eine wichtige Transformation innerhalb des Data Staging Areas ist das Erzeugen eines Surrogate Keys. Für die Dimensionen wird, wie bereits erwähnt wurde, nicht der Primärschlüssel aus den Quelltabellen verwendet, sondern ein künstlich erzeugter Schlüssel, der keine weitere Bedeutung hat. Die Frage *Warum nicht die Primärschlüssel*

der Quelltabellen verwendet werden sollten lässt sich leicht beantworten. Angenommen die Primärschlüssel in den Quelltabellen werden wiederverwendet. Dies ist dann möglich, wenn ein Unternehmen sich zum Beispiel entscheidet jedes Jahr den Primärschlüssel zu ändern und den alten Primärschlüssel für andere Tupel wieder zu verwenden. Innerhalb des DWs würde es zu Unstimmigkeiten kommen und dadurch zu erheblichen Fehlern. In Tabelle 14 sind einige Transformationen aufgeführt:

Quelltabellen	Transformation	Data Presentation Area
Event	Ermitteln des Attributes Status aus Event_Name. Wenn Event_Name Sleep oder Failure ist, dann ist Status INACTIVE, sonst ACTIVE. INACTIVE bedeutet, dass der Propagationsprozess sozusagen still gelegt ist.	Event Dimension
Event	Ermitteln des Attributes Father_Name aus der Father_ID und Event_ID. Da Father_ID einer Event_ID eines Ereignisses entspricht, muss lediglich der Event_Name herausgesucht werden, bei dem Father_ID gleich Event_ID ist	Event Dimension
Event_Trail, Ps_Use_Syschema	Beide Tabellen werden benötigt, um Number_of_Sources, Number_of_Destination und Relationship zu ermitteln. Relationship ergibt sich aus Number_of_Sources und Number_of_Destination. Wenn zum Beispiel Number_of_Sources 1 und Number_of_Destination 3 sind, dann besteht eine 1:3 Relationship.	PS Dimension
Event_Trail, Event Dimension, PS Dimension, Date Dimension, Time Dimension	Ermittlung des Faktums Duration durch Berechnung der Differenz zwischen zwei aufeinander folgenden Ereignissen. Die Fakten Number_of_Process und Number_of_Restart ergeben sich aus den Ereignissen Start und Restart	Fact_Table_1

Tabelle 14 Beispiele für Transformationen

5.1.3 Laden

Beim Laden werden die extrahierten und transformierten Daten aus dem Data Staging Area in den Data Presentation Area geladen. Erst wenn der Ladevorgang beendet wurde ist ein Zugriff durch Benutzer zur Analyse der Informationen möglich. Wichtig ist, dass erst nach Ablauf eines gesamten ETL-Prozesses auf die Informationen zugegriffen werden kann, da sonst beispielsweise bei Cross-Over Operationen, welche auf mehrere Faktentabellen zugreifen, Konsistenzproblemen auftreten können.

5.2 Historisierung der Dimensionen

Bei der Historisierung geht es darum wie Änderungen in den Dimensionen gehandhabt werden sollen. Mit anderen Worten was passiert mit den Attributwerten in den Dimensionen, wenn Änderungen in den operationalen Datenbeständen auftreten.

Angenommen der Name eines Ereignisses wird in der Quelltablelle *Event* geändert, so wird beim nächsten ETL-Prozess geprüft, ob sich dieses Ereignis in der *Event* Dimension bereits befindet und falls nicht, wird es hinzugefügt. Innerhalb des ETL-Prozesses wird lediglich geprüft, ob sich die *Event_ID* in der *Event* Dimension befindet. Für diese Arbeit war diese Prüfung ausreichend, da wie bereits gesagt, nicht von Änderungen innerhalb der Quelltabellen ausgegangen wird. Diese Prüfung könnte erweitert werden, indem zum Beispiel weitere Prüfungen durchgeführt werden. Es könnte zum Beispiel überprüft werden, ob der *Event_Type* eines Ereignisses geändert wurde. Derartige Erweiterungen gelten auch für die restlichen Dimensionen.

Im Folgenden werden drei verschiedene Historisierungsmethoden erläutert:

Die wohl einfachste Möglichkeit der Historisierung kann durch die Überschreibung des alten Wertes in der Dimension erfolgen. Die wesentlichen Vorteile dieser Vorgehensweise sind Einfachheit und Schnelligkeit. Jedoch ist das Problem, dass sämtliche Historie, was zu einem wichtigen Ziel des DWs gehört, dabei verloren geht. Diese Art der Historisierung wird lediglich verwendet, wenn es sich bei Attributsänderung um Korrekturen handelt oder wenn der alte Wert nicht mehr benötigt wird. Eine zweite Möglichkeit ist, dass bei einer Attributsänderung einfach eine neue Zeile hinzugefügt wird, um somit Historisierung zu ermöglichen. Für jedes neue Tupel in der Dimension wird eine neue Zeile eingefügt und ein Surrogate Key erzeugt. Jeder dieser Surrogate Keys identifiziert ein Tupel in der Dimension eindeutig für einen bestimmten Zeitraum. Anhand dieser Methode wird die Wichtigkeit der Surrogate Keys besonders deutlich. Bei der dritten Historisierungsmethode wird in der Dimension eine weitere Spalte hinzugefügt, in welcher der alte Attributwert gespeichert wird. Es gäbe demnach zwei Spalten, wobei in der einen der aktuelle und in der anderen der alte Wert gespeichert wird. Im Falle einer Änderung wird der Attributwert durch den neuen Wert ersetzt und in die Spalte „Altes Attribut“ geschrieben.

6 OLAP Anfragen

In diesem Kapitel werden sämtliche definierten Anfragen erläutert, wobei eine Unterscheidung zwischen vordefinierten Anfragen und ad hoc Anfragen erfolgt. Die vordefinierten Anfragen müssen lediglich vom Benutzer ausgeführt werden. Im zweiten Teil des Kapitels werden Hinweise gegeben, die für die Definition von ad hoc Anfragen durch den Benutzer beachtet werden sollten.

6.1 Vordefinierte Anfragen

In diesem Abschnitt werden sämtliche Anfragen für die jeweiligen Faktentabellen aufgeführt und im Anschluss Beispiele für Cross-Over Operationen. Als Grundlage wurden die allgemein formulierten Anfragen aus Abschnitt 3.2 genommen und weiter ausgebaut.

- *Fact_Table_1*

Nummer	Anfrage	Semantik	Verwendete Fakten
1	Wie lang ist die durchschnittliche Dauer eines Propagationsprozesses in einem bestimmten Zeitraum?	Durchschnittliche Dauer eines Propagationsprozesses. Als Zeitraum kann zum Beispiel Monat, Jahr, etc. angegeben werden	Duration
2	Wie lang ist die durchschnittliche reale Dauer eines Propagationsprozesses in einem bestimmten Zeitraum?	Analog zu 1, mit dem Unterschied, dass hier nur Ereignisse betrachtet werden, bei denen der Propagationsprozess aktiv ist, d.h. alle außer den Sleep und Failure Ereignissen werden betrachtet	Duration
3	Wie hoch ist der prozentuale inaktive Anteil eines Propagationsprozesses zum Gesamtprozess für einem bestimmten Zeitraum?	Berechnung des prozentualen Anteils eines inaktiven Intervalls; zuerst wird inaktive Anteil berechnet, dann Gesamtdauer eines Propagationsprozesses und zum Schluss $(\text{Inaktive Anteil} * 100 / \text{Gesamtdauer})$; Betrachtung erfolgt pro Propagationsprozess, d.h. für jede Propagation_Process_ID	Duration
4	Wie lang ist die durchschnittliche Dauer eines inaktiven Intervalls in einem bestimmten Zeitraum?	Betrachtung der durchschnittlichen Dauer eines Inaktiven Anteils, d.h. Einschränkung ist Event = „Failure“ oder „Sleep“; mögliche Betrachtung pro Propagationsskript oder pro Propagation_Process_ID	Duration
5	Welcher Intervall innerhalb eines Propagationsprozesses hat am längsten benötigt in einem bestimmten Zeitraum?	Ermittlung des Intervalls, der zum Beispiel im Monat = „Januar“ pro Propagation_Process_ID die längste Zeit benötigt hat	Duration

6	Welche Propagationsprozesse haben mehr Zeit benötigt als ein bestimmter Wert? zum Beispiel > 100000 Mikrosekunden	Hier wird für die Metrik eine Einschränkung definiert, nämlich nur Propagationsprozesse auflisten, die > 100000. Interessante Anfrage, falls zuvor ein Durchschnittswert für die Dauer von Propagationsprozessen ermittelt wurde und man anschließend wissen will, welche der Propagationsprozesse diesen Wert überschreiten, um so eventuelle Optimierungen vornehmen zu können	Duration
7	Gib alle Propagationsprozesse, bei denen der Inaktive Intervall nicht größer ist als ein bestimmter Wert	Wenn zuvor ein Grenzwert für die Dauer eines Inaktiven Anteils ermittelt wurde, können mit Hilfe dieser Anfrage nur die Propagationsprozesse ausgegeben werden, die den Definierten Grenzwert nicht überschreiten	Duration
8	Welcher Propagationsprozess hat insgesamt am längsten Zeit benötigt in einem bestimmten Zeitraum?	Hier erhält man den Propagationsprozess, der zum Beispiel im Monat Januar am längsten Zeit benötigt hat. Dadurch erhält man eine Obere Grenze. Analog dazu könnte eine Anfrage definieren werden, die die Untergrenze mit Hilfe der MIN Funktion ermittelt	Duration
9	Wie viele und welche Propagationsprozesse wurden erfolgreich terminiert in einem bestimmten Zeitraum?	Ermitteln sämtlicher Propagationsprozesse, die in einem bestimmten Zeitraum erfolgreich terminiert wurden. Hierzu wird die Summe über das Faktum Number_of_Process gebildet, aber mit der Einschränkung Event= „Successful_Termination“	Number_of_Process
10	Wie viele und welche Propagationsprozesse wurden nicht erfolgreich terminiert?	Analog zu 8, mit dem Unterschied, dass Einschränkung Event = „Forced_Termination“ ist.	Number_of_Process
11	Wie viele Propagationsprozesse wurden in einem bestimmten Zeitraum ausgeführt?	Berechnung sämtlicher Propagationsprozesse zum Beispiel für Monat = „Januar“	Number_of_Process
12	Welche Propagationsprozesse wurden in einem bestimmten Zeitraum nach Restart erfolgreich terminiert? (prozentualer Anteil)	Ermittlung sämtlicher Propagationsprozesse, die nach Ereignis „Restart“ erfolgreich terminiert wurden (Successful_Termination). Hierzu wird zunächst die Summe aller Propagationsprozesse mit Event = „Restart“ ermittelt. Anschließend Summe über alle mit Event = „Restart“ und Event = „Successful_Termination“. Zum Schluss noch eine Division	Number_of_Process
13	Welche Propagationsprozesse wurden in einem bestimmten Zeitraum mehr als ein bestimmter Wert, zum Beispiel drei mal, wiedergestartet?	Einschränkung der Metrik durch Festlegung eines Wertes, zum Beispiel nur Propagationsprozesse die weniger als drei mal wiedergestartet wurden	Number_of_Process

Tabelle 15 Vordefinierte Anfragen für Fact_Table_1

- *Fact_Table_2*

Innerhalb dieser Faktentabelle kann das Faktum *Connection* nur verwendet werden, um die Summe zu bilden. Die Aggregationsfunktion AVG würde hier keinen Sinn machen, da das Ergebnis jedes mal 1 wäre. Demnach ist zum Beispiel die Anfrage „Durchschnittliche Anzahl der Verbindungen“ nicht leicht zu implementieren bzw. nicht durch direkte Anwendung von AVG möglich. In Tabelle 16 sind mögliche Anfragen aufgelistet.

Nummer	Anfrage	Semantik	Verwendete Fakten
1	Welche Informationssysteme wurden in einem bestimmten Zeitraum wie oft miteinander verbunden?	Ermittlung sämtlicher Quell- und Zielsysteme, die innerhalb eines bestimmten Zeitraums, zum Beispiel im Monat Januar miteinander verbunden wurden	Connection
2	Welche Informationssysteme wurden in einem bestimmten Zeitraum mehr als ein bestimmter Wert miteinander verbunden?	Sämtliche Informationssysteme werden aufgelistet, die öfter als bestimmter Wert, zum Beispiel 5 mal in zum Beispiel Monat Januar, verbunden wurden	Connection
3	Welche Informationssysteme wurden in einem bestimmten Zeitraum am häufigsten miteinander verbunden?	Hier werden die Informationssysteme ermittelt, die zum Beispiel im Monat Januar am meisten miteinander verbunden wurden. Gibt Auskunft, wo Datenänderungen am häufigsten vorkommen	Connection

Tabelle 16 Vordefinierte Anfragen für *Fact_Table_2*

- *Fact_Table_3*

Wie bereits im vorherigen Kapitel erwähnt wurde, lassen sich auf *Fact_Table_3* Anfragen nicht einfach formulieren bzw. es gibt keinen großen Spielraum für Anfragen. Innerhalb der *Fact_Table_3* kann lediglich die Aggregationsfunktion COUNT bzw. COUNT (DISTINCT) verwendet werden.

Nummer	Anfrage	Semantik	Verwendete Fakten
1	Wie oft wurde ein bestimmtes Detail in einem bestimmten Zeitraum verwendet bzw. tritt in einem Propagationsprozess ein? (Erweiterung: Welche Details wurden zum Beispiel mehr als vier mal in einem bestimmten Zeitraum verwendet?)	Lediglich derartige Anfragen lassen sich formulieren. Hierbei wird ermittelt, wie oft bestimmte Details wie zum Beispiel TIMEOUT in einem bestimmten Zeitraum verwendet wurde. Es erfolgt ein Zählen der Values. Als Einschränkung wird ein Detail aus der Detail Dimension verwendet	Value

Tabelle 17 Vordefinierte Anfragen für *Fact_Table_3*

Anzumerken sei, dass Anfragen wie zum Beispiel „Wie oft wurde ein Transformationskript verwendet?“ schwierig zu implementieren sind. Da das verwendete Faktum non-additive ist, lassen sich nur wenige Anfragen für die Faktentabelle *Fact_Table_3* formulieren.

- Cross-Over Operationen

Cross-Over Operationen werden verwendet, wenn der Benutzer sich für Fakten aus mehreren Faktentabellen interessiert. In der untenstehenden Tabelle 18 sind zwei Beispiele aufgelistet.

Nummer	Anfrage	Semantik	Verwendete Fakten
1	Was ist die durchschnittliche Dauer eines Propagationsprozesses bei dem Source_System = „Sales System“ ist?	Hier erfolgt eine Cross-Over Operation zwischen Fact_Table_1 und Fact_Table_2. Hier wird zwar nur das Faktum Duration benötigt, jedoch ist ein Join zwischen den beiden Faktentabellen erforderlich. Dies geschieht über die Propagation_Process_ID. Des Weiteren gibt es noch eine Einschränkung, nämlich Source_System = „Sales System“ ; auch hier lassen sich natürlich Zeiträume angeben	Duration
2	Was ist die durchschnittliche Dauer eines Propagationsprozesses bei dem Source_System „Sales System“ und Destination_System „FLP“ ist?	Analog zu 1, lediglich mit einer weiteren Einschränkung: Destination_System = „FLP“	Duration

Tabelle 18 Beispiele für Cross-Over Operationen

6.2 Ad hoc Anfragen

Im vorherigen Abschnitt wurden mögliche Anfragen zu den jeweiligen Faktentabellen aufgelistet. Dabei handelte es sich um vordefinierte Anfragen, die der Benutzer lediglich ausführen muss. Natürlich ist es auch möglich, dass der Benutzer ad hoc Anfragen definiert. Dabei muss er darauf achten welche Anfragen auf welchen Faktentabellen möglich sind. Um das zu verstehen, muss der Benutzer vor allem wissen, um welche Art von Faktum es sich handelt. In Tabelle 19 sind sämtliche Fakten der Faktentabellen einschließlich der Art und der Aggregationsfunktionen, die für die jeweiligen Fakten sinnvoll sind, aufgeführt.

Faktentabelle	Fakten	Faktenart	Aggregationsfunktionen
Fact_Table_1	Duration	Additive	SUM, AVG, MIN UND MAX
Fact_Table_1	Number_of_Process	Semi-Additive	SUM
Fact_Table_1	Number_of_Restart	Semi-Additive	SUM
Fact_Table_2	Connection	Semi-Additive	SUM
Fact_Table_3	Value	Non-Additive	COUNT, COUNT (DISTINCT)

Tabelle 19 Sämtliche Fakten einschließlich möglicher Aggregationsfunktionen

Duration ist ein perfektes Faktum, d.h. sämtliche Aggregationsfunktionen sind möglich. Wohingegen *Number_of_Process*, *Number_of_Restart* und *Connection* semi-additive sind und somit nicht sämtliche Aggregationen möglich sind. Anfragen, die den Durchschnitt (oder auch `Min` und `Max`) ermitteln wollen, sind demnach durch die direkte Anwendung dieser Aggregationsfunktionen auf das Faktum nicht möglich. Außerdem würden sie auch keinen Sinn machen, da das Resultat der Anfrage immer falsch wäre. Angenommen jemand interessiert sich für die durchschnittliche Anzahl der Verbindungen zwischen Quell- und Zielsystemen, d.h. $AVG(Connection)$. Da aber *Connection* immer 1 ist, würde der Durchschnitt ebenfalls immer 1 sein und somit würde die Anfrage kein sinnvolles Resultat liefern. Dasselbe gilt für die Fakten *Number_of_Process* und *Number_of_Restart*. Das Faktum *Value* ist non-additive und somit lassen sich Anfragen hierauf nicht leicht formulieren. Lediglich `COUNT` bzw. `COUNT(DISTINCT)` können ausgeführt werden. Wie bereits im vorherigen Kapitel erwähnt wurde, wurde *Fact_Table_3* zusätzlich definiert, um sämtliche Informationen aus den Quellsystemen im DW zu sichern.

Aus obiger Beschreibung wird deutlich, dass nicht alle Anfragen durch die direkte Anwendung der Aggregationsfunktionen auf Fakten möglich sind. Beispielsweise lässt sich die durchschnittliche Anzahl der Propagationsprozesse pro Monat durch $AVG(Number_of_Process)$ nicht ermitteln. Das bedeutet aber nicht, dass derartige Anfragen nicht formuliert werden können. Allerdings ist die Formulierung nicht leicht. Um beispielsweise die Anzahl der Propagationsprozesse pro Monat ermitteln zu können, muss zunächst $SUM(Number_of_Process)$ für jeden einzelnen Monat berechnet werden und anschließend kann davon der Durchschnitt ermittelt werden. Eine weitere Anfrage, die sich zum Beispiel nicht durch die Aggregationsfunktion `AVG` direkt ermitteln lässt, bezieht sich auf das Faktum *Value*. Beispielsweise ist es nicht möglich die durchschnittliche Anzahl der Ausführungen eines Transformationskriptes pro Monat durch $AVG(Value)$ zu ermitteln. Hierzu müsste der Benutzer zunächst sämtliche Tupel der *Fact_Table_3* für jeden Monat zählen, die das Transformationskript verwenden. Erst dann kann der Durchschnitt aus den ermittelten Werten berechnet werden.

7 Implementationsaspekte

In den vorherigen Kapiteln wurden sämtliche Konzepte erläutert, wobei dabei die praktische Umsetzung mit Hilfe von speziellen Werkzeugen nicht betrachtet wurde. Die praktischen Umsetzungen werden in diesem Kapitel beschrieben, d.h. hier wird erläutert wie das DW aufgebaut wurde, wie der ETL-Prozess bzw. die Sub-ETL-Prozesse umgesetzt wurden und was nötig war, um die Anfragen zu implementieren.

Da das DW die Grundlage für die OLAP Anfragen darstellt, wird zunächst der Aufbau des DWs mit Hilfe des Werkzeugs DB2 Data Warehouse Center von IBM beschrieben. Im Anschluss erfolgt die Implementierung der Anfragen mit Microstrategy 6.0. In beiden Unterkapiteln erfolgt zunächst eine Einführung in die jeweiligen Werkzeuge.

7.1 Aufbau des Data Warehouses

Für den Aufbau des DW mussten zunächst zwei Datenbanken zusätzlich zur Datenbank für die Quelltabellen definiert werden. Diese Datenbanken stellen die verschiedenen Bereiche eines DW Systems dar. Die Datenbank PMLOGDB, welche den Log darstellt und in der sämtliche Quelltabellen gespeichert sind, gilt als Operational Source System. Die Datenbank STAGE_DB entspricht dem Data Staging Area und beinhaltet sämtliche Tabellen, Stored Procedures, Einschränkungen (Constraints), Sequenzen, usw., die für die entsprechenden Sub-ETL-Prozesse benötigt werden. Um OLAP Analysen durchführen zu können müssen sämtliche Daten aus den Operational Source Systems über den Data Staging Area in den Data Presentation Area geladen werden. DWLOGDB ist die Datenbank, die den Data Presentation Area darstellt und in der sämtliche Informationen in den Dimensions- und Faktentabellen dauerhaft abgelegt sind, um Analysen durchführen zu können.

Bevor die Umsetzung mit dem Werkzeug DB2 Data Warehouse Center beschrieben wird, erfolgt im nächsten Unterkapitel eine Einführung in dieses Werkzeug.

7.1.1 Grundlagen zu DB2 Data Warehouse Center

Um das Data Warehouse zu erstellen und anschließend zu befüllen (ETL-Prozess) wurde das Werkzeug DB2 Data Warehouse Center von IBM verwendet.

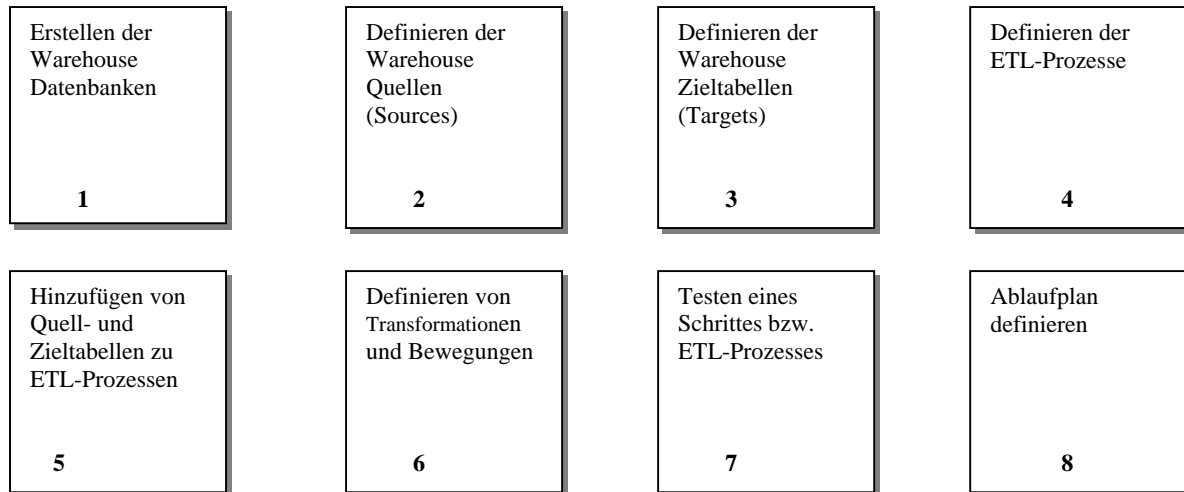


Abbildung 26 Ablauf eines DW Aufbaus mit Data Warehouse Center

In obiger Abbildung ist der Ablauf dargestellt, um ein DW zu definieren und anschließend zu befüllen. Die einzelnen Schritte aus obiger Abbildung werden im Folgenden erläutert [DWC]:

Schritt 1 Erstellen der Warehouse Datenbanken: Hierbei werden, wie bereits oben erläutert wurde, zunächst die benötigten Datenbanken definiert, d.h. für Operational Source Systems, Data Staging Area und Data Presentation Area.

Schritt 2 Definieren der Warehouse Quellen: In diesem Schritt werden die Tabellen definiert, die operative Daten enthalten.

Schritt 3 Definieren der Warehouse Zieltabellen: Hier geht es um das Definieren der benötigten Tabellen, sowohl für Data Staging Area als auch für Data Presentation Area. Hierzu müssen analog zur Definition der Quelltabellen zunächst die Zieltabellen innerhalb der Steuerzentrale definiert werden und anschließend ins Werkzeug importiert werden. Bei beiden Schritten (1 und 2) ist es jedoch erforderlich im Werkzeug zunächst die Quell- und Zieltabellen zu definieren, um anschließend, die in der Steuerzentrale definierten Tabellen zu importieren.

Schritt 4 Definieren von ETL-Prozessen: Ein ETL-Prozess besteht aus einer Reihe von Schritten zum Extrahieren, Transformieren und Laden von Daten. Zuvor muss jedoch ein Themenbereich definiert werden. Themenbereiche werden verwendet, um Prozesse zu organisieren. Es kann zum Beispiel ein Marketing Themenbereich definiert und innerhalb diesen Themenbereiches ETL-Prozesse hinzugefügt werden, die zum Marketingbereich gehören.

Schritt 5 Hinzufügen von Quell- und Zieltabellen zu einem Prozess:

Die Prozesse werden, wie bereits erwähnt wurde, verwendet um Daten von einer bzw. mehreren Quelltable(n) in eine oder mehrere Zieltabelle(n) zu laden. Dazu

muss innerhalb des ETL-Prozesses Quell- und Zieltabellen eingefügt werden. Sobald sie eingebunden wurden können sie zum Beispiel über SQL-Schritte verbunden werden.

Schritt 6 Definieren von Datentransformationen und Ladevorgängen(Bewegungen):

Hierfür gibt es die unterschiedliche Schritte wie zum Beispiel SQL-Schritte. Zur Transformation gehört zum Beispiel das Auswählen von Daten aus mehreren Quelltabellen und das anschließende Laden der verbundenen Daten in Zieltabellen. Ein weiteres Beispiel für eine Transformation ist das Erzeugen der Surrogate Keys (Primärschlüssel der Dimensionen).

Schritt 7 Testen eines Schrittes bzw. ETL-Prozesses:

Schritte können im Anschluss der Definition getestet werden. Hierzu muss aber der Modus des Schrittes auf Test-Modus gesetzt werden. Insgesamt gibt es drei Modi, nämlich Entwicklung, Test und Produktion. Schritte können lediglich im Entwicklungsmodus erstellt und geändert werden. Sobald ein Schritt in den Produktionsmodus hochgestuft wurde, kann er nicht geändert werden. Das Wechseln des Modus ist in Abbildung 27 zu sehen.

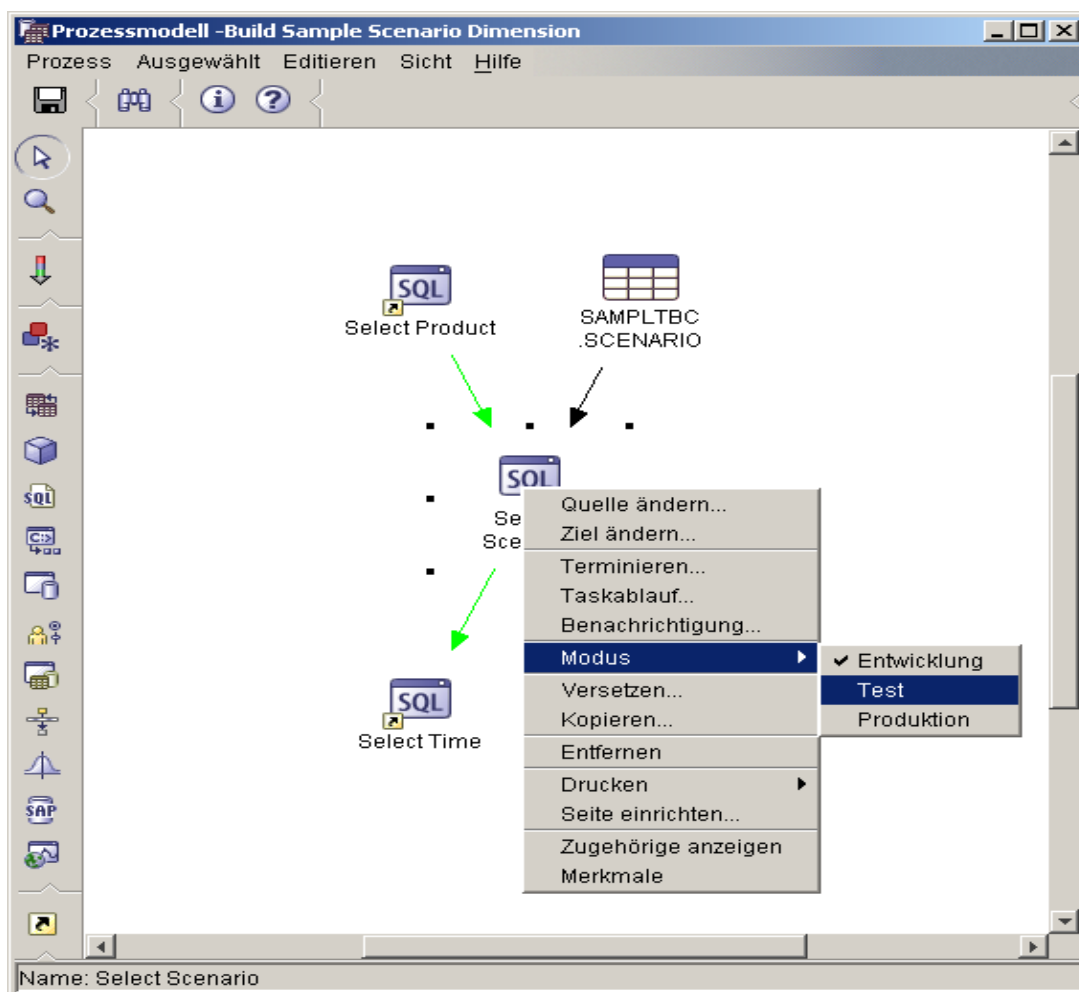


Abbildung 27 Wechseln des ETL-Prozess Modus

Schritt 8 Definition eines Ablaufplans für ETL-Prozesse:

Hierbei kann spezifiziert werden in welchen Zeitabschnitten Prozesse durchgeführt werden sollen. Zusätzlich ist es möglich eine Reihenfolge der Schritte zu definieren, in der die einzelnen Schritte ausgeführt werden sollen [DWC]. In Abbildung 28 ist ein Beispiel für einen Prozess abgebildet.

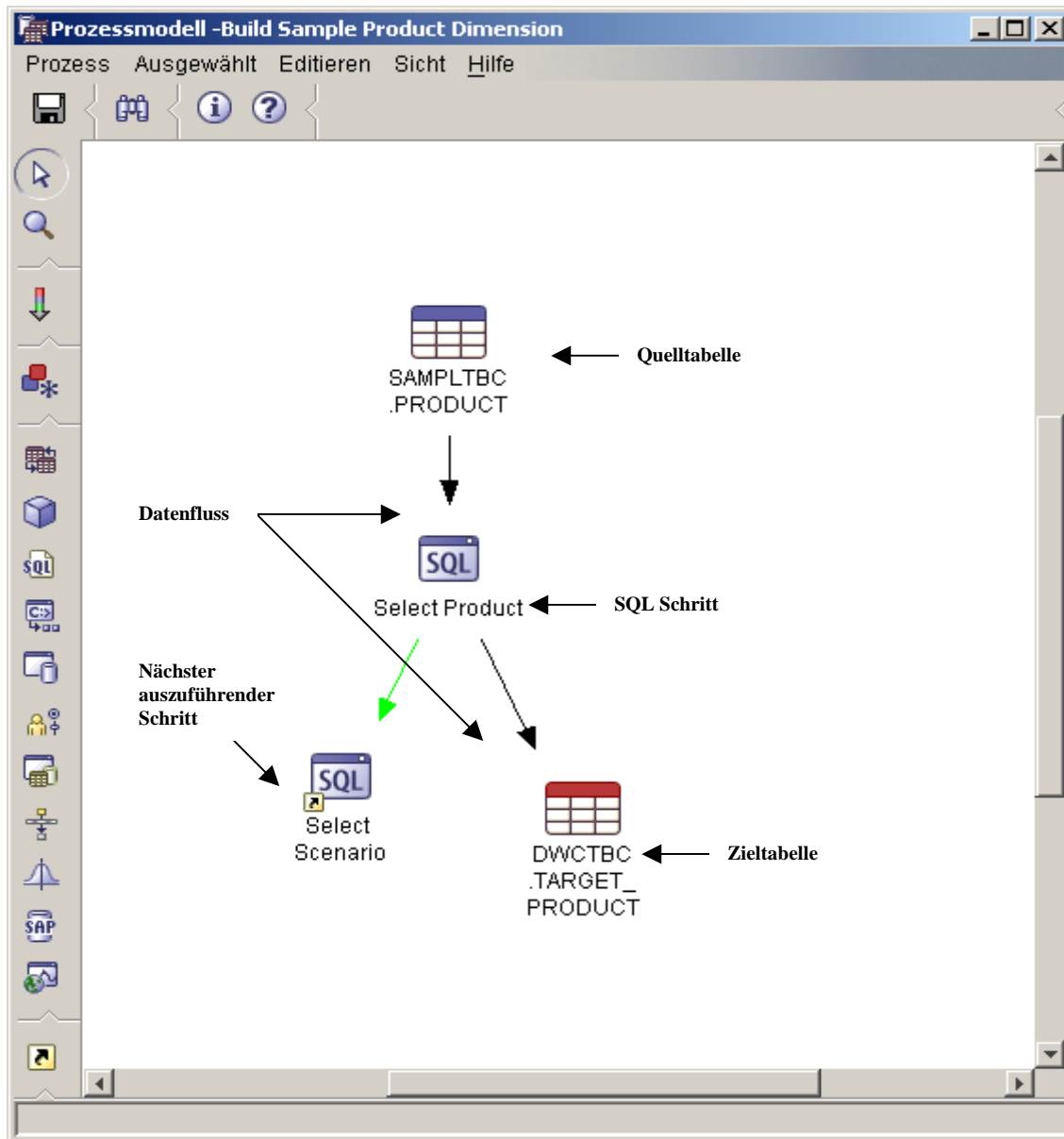


Abbildung 28 Abbildung eines ETL-Prozesses

7.1.2 Umsetzung von ETL-Prozessen

Wie bereits in vorherigen Kapiteln erläutert wurde, besteht der ETL-Prozess aus mehreren Sub-ETL-Prozessen. Im Folgenden wird die Umsetzung zwei dieser Sub-

ETL-Prozesse mit dem Werkzeug DB2 Data Warehouse Center erläutert, um zu verdeutlichen wie ein ETL-Prozess mit diesem Werkzeug implementiert wird. Dabei werden die Sub-ETL-Prozesse für die *Event* Dimension und für die *Fact_Table_1* nach obiger Vorgehensweise erläutert.

- *Event* Dimension

Zunächst werden die Quell- und Zieltabellen definiert. In Tabelle 20 sind die jeweils benötigten Tabellen aufgelistet, wobei anzumerken ist, dass eine Zieltabelle auch als Quelltable für einen anderen Schritt fungieren kann.

Reihenfolge	Quelltabellen	Zieltabellen
1	Event (PMLOGDB)	Source_Event_ID (STAGE_DB)
2	Source_Event_ID (STAGE_DB)	Surrogate_Event_ID (STAGE_DB)
3	Source_Event_ID (STAGE_DB), Surrogate_Event_ID (STAGE_DB)	Event Dimension (STAGE_DB)
4	Event Dimension (STAGE_DB)	Event Dimension (DWLOGDB)

Tabelle 20 Quell- und Zieltabellen des Sub-ETL-Prozesses für Event Dimension

Im Anschluss wurde der Themenbereich Logfile_Analyse definiert, in dem sämtliche Sub-ETL-Prozesse organisiert werden. Es wurden die drei Prozesse *Create_Fact_Table_1*, *Create_Fact_Table_2* und *Create_Fact_Table_3* definiert. Der Sub-ETL-Prozess für die *Event* Dimension wurde innerhalb des Prozesses *Create_Fact_Table_1* definiert, welcher aus folgenden vier Schritten besteht:

1. **Create_Event_Surrogate_Key_Part_1**

In diesem Schritt werden sämtliche Ereignisse aus der Quelltable *Event* in die Tabelle *Source_Event_ID* extrahiert. Bei diesem Schritt handelt es sich um eine einfache SQL-Anweisung bei der die erforderlichen Daten selektiert und anschließend in Zieltabelle eingefügt werden.

SQL-Anweisung:

```

SELECT
    PMLOG.EVENT.EVT_CODE AS EVT_CODE,
    PMLOG.EVENT.EVT_ID AS EVT_ID,
    PMLOG.EVENT.EVT_DESCRIPTION AS EVT_DESCRIPTION,
    PMLOG.EVENT.EVT_TYPE AS EVT_TYPE,
    PMLOG.EVENT.EVT_FATHER_ID AS EVT_FATHER_ID
FROM
    PMLOG.EVENT

```

Abbildung 29 Selektieren der Daten aus der Quelltable Event

Um die selektierten Daten in die Zieltabelle einzufügen muss innerhalb des Schrittes die Spaltenzuordnung definiert werden. Dazu gibt es die Optionen Ersetzen und Hinzufügen. Beim Ersetzen wird jedes mal der gesamte Inhalt in der jeweiligen Tabelle ersetzt und beim Hinzufügen werden die neuen Tupel an die jeweilige Tabelle angehängt. Innerhalb des Schrittes **Create_Event_Surrogate_Key_Part** wurde die

Option Ersetzen gewählt. Das bedeutet, dass die Daten in der jeweiligen Zieltabelle im Data Staging Area durch die neuen selektierten Daten ersetzt werden.

2. Create_Event_Surrogate_Key_Part_2

Innerhalb dieses Schrittes werden sämtliche Ereignisse geladen, die noch nicht in der *Event* Dimension enthalten sind. Des Weiteren erfolgt eine Erzeugung der Surrogate Keys (Primärschlüssel der Dimensionen). Doch bevor Surrogate Keys erzeugt werden können muss eine Sequenz innerhalb des Data Staging Areas definiert werden.

```
CREATE SEQUENCE seq_surrogate_event_id AS INTEGER START WITH 1  
INCREMENT BY 1;
```

Jedes Tupel bekommt eine Zahl als Surrogate Key aus dieser Sequenz zugewiesen.

Eine andere Möglichkeit wäre gewesen, bei Definition der Tabelle mit Hilfe der DB2 Steuerzentrale eine Spalte als Identitätsspalte zu definieren. Dadurch würde jedem neuen Tupel ein eindeutiger Integerwert zugewiesen.

SQL-Anweisung zu diesem Schritt:

```
SELECT  
    KOZASAS.SOURCE_EVENT_ID.SOURCE_ID AS SOURCE_ID,  
    nextval for kozasas.seq_surrogate_event_id AS surrogate_id  
FROM  
    KOZASAS.SOURCE_EVENT_ID  
WHERE  
    (  
        (KOZASAS.SOURCE_EVENT_ID.SOURCE_ID NOT IN (SELECT  
            B.SOURCE_ID FROM KOZASAS.SURROGATE_EVENT_ID B))  
    )
```

Abbildung 30 Selektieren der neuen Daten und Schlüsselzuweisung

Wenn ein Ereignis bzw. die *Event_ID* noch nicht in der Zieltabelle vorhanden ist, wird die *Event_ID* in die Zieltabelle eingefügt und erhält einen eindeutigen Surrogate Key. Die Zuweisung erfolgt über den Befehl **nextval for kozasas.seq_surrogate_event_id**. Das bedeutet, dass die *Event_ID* den nächsten Wert der definierten Sequenz zugewiesen bekommt. Auch hier erfolgt das Einfügen über das Definieren der Spaltenzuordnungen mit der Option Ersetzen.

Innerhalb der WHERE-Klausel wird geprüft, ob die *Event_ID* schon vorhanden ist. Da die Quelltablelle *Event* und damit auch die Zieltabelle

Surrogate_Event_ID nicht sehr groß sind, ist diese Art der Prüfung auch nicht ineffizient.

3. Create_Event_Target

Bei diesem Schritt werden die Daten aus *Surrogate_Event_ID* und *Source_Event_ID* in die Zieltabelle *Event Dimension* (STAGE_DB) geladen.

SQL-Anweisung:

```
SELECT
  KOZASAS.SURROGATE_EVENT_ID.SURROGATE_ID AS SURROGATE_ID,
  KOZASAS.SURROGATE_EVENT_ID.SOURCE_ID AS SOURCE_ID,
  A.EVENT_NAME AS EVENT_NAME,
  A.EVENT_DESCRIPTION AS EVENT_DESCRIPTION,
  A.EVENT_TYPE AS EVENT_TYPE,
  CASE WHEN A.EVENT_NAME = 'SLEEP' THEN 'INACTIV'
        WHEN A.EVENT_NAME = 'FAILURE' THEN 'INACTIV'
        ELSE 'ACTIVE' END AS STATUS,
  A.FATHER_ID AS FATHER_ID,
  (SELECT B.EVENT_NAME FROM KOZASAS.SOURCE_EVENT_ID B
   WHERE A.FATHER_ID= B.SOURCE_ID) AS FATHER_NAME
FROM
  KOZASAS.SURROGATE_EVENT_ID, KOZASAS.SOURCE_EVENT_ID A
WHERE
  (
    (
      KOZASAS.SURROGATE_EVENT_ID.SOURCE_ID = A.SOURCE_ID
    )
  )
```

Abbildung 31 Laden der neuen Daten in die vorläufige Event Dimension (STAGE_DB)

Innerhalb dieses Schrittes kommt es zusätzlich zu Transformationen. Die Attribute *Status* und *Father_Name* werden aus den vorhandenen Daten ermittelt. Der Wert des *Status* ist abhängig von den Ereignissen und erfolgt über die CASE-Anweisung. In der CASE-Anweisung wird den Ereignissen *Failure* und *Sleep* der Wert *INACTIVE* und allen anderen Ereignissen der Wert *ACTIVE* zugewiesen. Um *Father_Name* zu ermitteln ist es notwendig eine weitere Sub-Anfrage zu definieren, die im Falle einer *Father_ID* den *Event_Name* ermittelt, bei dem *Event_ID* = *Father_ID* ist.

In der WHERE-Klausel ist die Join-Bedingung zwischen den beiden Tabellen *Surrogate_Event_ID* und *Source_Event_ID* definiert. Es werden lediglich die Ereignisse in die Zieltabelle *Event Dimension* (STAGE_DB) geladen, welche die Bedingung erfüllen. Das Einfügen der Werte in die Zieltabelle erfolgt analog zu den vorherigen Schritten über die Spaltenzuordnungen.

4. Create_Event_Dimension

Im letzten Schritt werden lediglich die Daten aus dem Data Staging Area in den Data Presentation Area geladen, um sie für Benutzer zugänglich zu machen. Hierbei werden die Daten aus der *Event Dimension* (STAGE_DB) in die *Event Dimension* (DWLOGDB) geladen.

SQL-Anweisung:

```

SELECT
    KOZASAS.EVENT_DIMENSION.EVENT_ID AS EVENT_ID,
    KOZASAS.EVENT_DIMENSION.SOURCE_EVT_ID AS SOURCE_EVT_ID,
    KOZASAS.EVENT_DIMENSION.EVENT_NAME AS EVENT_NAME,
    KOZASAS.EVENT_DIMENSION.EVENT_DESCRIPTION AS EVENT_DESCRIPTION,
    KOZASAS.EVENT_DIMENSION.EVENT_TYPE AS EVENT_TYPE,
    KOZASAS.EVENT_DIMENSION.STATUS AS STATUS,
    KOZASAS.EVENT_DIMENSION.FATHER_ID AS FATHER_ID,
    KOZASAS.EVENT_DIMENSION.FATHER_NAME AS FATHER_NAME
FROM KOZASAS.EVENT_DIMENSION

```

Abbildung 32 Laden der Daten in den Data Presentation Area (DWLOGDB)

Sämtliche Informationen aus dem Data Staging Area werden in den Data Presentation Area geladen, indem zunächst alle Informationen aus der *Event* Dimension (STAGE_DB) selektiert und anschließend über die Spaltenzuordnung in die *Event* Dimension (DWLOGDB) eingefügt werden. Hierfür wird die Option Hinzufügen verwendet, d.h. die neuen Informationen werden an die jeweilige Tabelle angehängt. Erst wenn sich die Informationen im Data Presentation Area befinden, sind sie dauerhaft abgelegt.

- *Fact_Table_1*
Analog zu *Event* Dimension müssen auch hier zunächst die Quell- und Zieltabellen definiert werden, die in Tabelle 21 aufgelistet sind.

Reihenfolge	Quelltabellen	Zieltabellen
1	Event (PMLOGDB), Event_Trail (PMLOGDB)	Fact_Table_1_Part_1 (STAGE_DB)
2	Fact_Table_1_Part_1 (STAGE_DB)	Fact_Table_1_Part_2 (STAGE_DB)
3	Time Dimension (STAGE_DB), Date Dimension (STAGE_DB), PS Dimension (STAGE_DB), Event Dimension (STAGE_DB), Fact_Table_1_Part_2 (STAGE_DB)	Fact_Table_1_Part_3 (STAGE_DB)
4	Time Dimension (STAGE_DB), Date Dimension (STAGE_DB), PS Dimension (STAGE_DB), Event Dimension (STAGE_DB), Fact_Table_1_Part_2 (STAGE_DB)	Fact_Table_Part_4 (STAGE_DB)
5	Fact_Table_1_Part_2 (STAGE_DB), Fact_Table_1_Part_3 (STAGE_DB), Fact_Table_1_Part_4 (STAGE_DB)	Fact_Table_1 (STAGE_DB)
6	Fact_Table_1 (STAGE_DB)	Fact_Table_1 (DWLOGDB)

Tabelle 21 Quell- und Zieltabellen des Sub-ETL-Prozesses für Fact_Table_1

Dieser Sub-ETL-Prozess gehört ebenfalls zum Themenbereich Logfile_Analyse. Sämtliche Schritte sind unter dem Prozess *Create_Fact_Table_1* abgelegt.

Zu diesen Schritten gehören:

1. CREATE_FACT_TABLE_1_PART_1

Innerhalb des ersten Schrittes werden sämtliche neuen Informationen aus den Quelltabellen *Event* und *Event_Trail* in die entsprechende Ziel-tabelle *Fact_Table_1_Part_1* extrahiert. Neue Informationen bedeutet, dass nur die Informationen in den Data Staging Area extrahiert werden, die zwischen *Last_ETT_ID* und *Last_Current_ETT_ID* liegen.

Last_ETT_ID ist die letzte *ETT_ID* aus vorherigem ETL-Prozess und *Last_Current_ETT_ID* die letzte *ETT_ID* des aktuellen ETL-Prozesses. Dies wird in den beiden unteren Abschnitten der SQL-Anweisung geprüft. Anzumerken sei, dass *COALESCE* verwendet wird, weil beim aller ersten Durchlauf dieses Schrittes die jeweiligen Tabellen keinen Werte enthalten. *COALESCE* bedeutet hier, dass wenn ein Wert für *Last_ETT_ID* nicht vorhanden ist, der Wert 0 genommen wird, d.h. beim ersten Durchlauf werden sämtliche *ETT_ID*'s betrachtet, die größer 0 sind. Angenommen die Tabelle *Event_Trail* enthält keine Werte. Dann aber wäre auch die Tabelle, welche die letzte *ETT_ID* für den aktuellen ETL-Prozess enthält, leer. Auf diese Weise würde die Bedingung beim aller ersten Durchlauf von diesem Sub-ETL-Prozess für *Fact_Table_1* die Bedingung $0 < ETT_ID \leq 0$ entstehen und somit nichts in den Data Staging Area extrahiert.

SQL-Anweisung:

```
SELECT
    PMLOG.EVENT.EVT_CODE AS EVT_CODE,
    PMLOG.EVENT.EVT_ID AS EVT_ID,
    PMLOG.EVENT_TRAIL.ETT_ID AS ETT_ID,
    PMLOG.EVENT_TRAIL.ETT_TIMESTAMP AS ETT_TIMESTAMP,
    PMLOG.EVENT_TRAIL.ETT_PROPAGATION_SCRIPT_NAME AS
        ETT_PROPAGATION_SCRIPT_NAME,
    PMLOG.EVENT_TRAIL.ETT_PROPAGATION_PROCESS_ID AS
        ETT_PROPAGATION_PROCESS_ID
FROM
    PMLOG.EVENT, PMLOG.EVENT_TRAIL
WHERE
    (
        (
            PMLOG.EVENT.EVT_ID = PMLOG.EVENT_TRAIL.ETT_EVT_ID
        )
    )
AND
    ((PMLOG.EVENT_TRAIL.ETT_ID > (SELECT COALESCE (MAX
(KOZASAS.LAST_ETT_ID_FACT_TABLE_1.LAST_ETT_ID), 0 )
FROM KOZASAS.LAST_ETT_ID_FACT_TABLE_1))
AND
    ((PMLOG.EVENT_TRAIL.ETT_ID <= (SELECT COALESCE ( MAX
(KOZASAS.LAST_ETT_ID_CURRENT_LOAD.LAST_CURRENT_ETT_ID), 0)
FROM KOZASAS.LAST_ETT_ID_CURRENT_LOAD)))
    )
```

Abbildung 33 Selektieren der neuen Daten aus Event und Event_Trail

Um die selektierten Daten in die Zieltabelle einzufügen muss innerhalb dieses Schrittes die Spaltenzuordnung definiert werden. Als Option wurde Ersetzen gewählt, d.h. in der jeweiligen Zieltabelle wird der gesamte Inhalt durch die selektierten neuen Daten ersetzt.

2. **CREATE_FACT_TABLE_1_PART_2 (stored procedure)**

Im zweiten Schritt wird das Faktum *Duration* berechnet, wobei hierzu eine Stored Procedure verwendet wurde. Um die Differenz zwischen zwei Tupel bzw. zwei Ereignissen zu berechnen ist es leichter eine Stored Procedure zu verwenden. Zur Implementierung von Stored Procedure wird die Sprache PL/SQL (Procedure Language) verwendet. Der Code für die Stored Procedure ist zusammen mit den Stored Procedures für die *Time* und *Date* Dimensionen im Anhang abgelegt. Innerhalb der Prozedur wird zunächst das erste Tupel und die Anzahl der Tupel in der Tabelle *Fact_Table_1_Part_1* ermittelt. Diese Werte werden verwendet, um das Intervall festzulegen, d.h. die Prozedur läuft solange bis sämtliche Tupel aus *Fact_Table_1_Part_1*, welche lediglich die neuen Tupel beinhaltet, in die *Fact_Table_1_Part_2* eingefügt worden sind.

3. **CREATE_FACT_TABLE_1_PART_3**

In diesem Schritt werden sämtliche Primärschlüssel aus den Dimensionstabellen in die Zieltabelle *Fact_Table_1_Part_3* gespeichert, die zu den jeweiligen Tupel aus *Fact_Table_1_Part_2* gehören. Zur Ermittlung der Primärschlüssel für die *Time* und *Date* Dimensionen war es notwendig aus dem Zeitstempel jeweils die Minute und Stunde herauszulösen. Hierzu gibt es die Funktionen `Minute` und `Hour`.

SQL-Anweisung:

```

SELECT DISTINCT
  KOZASAS.EVENT_DIMENSION.EVENT_ID AS EVENT_ID,
  KOZASAS.PS_DIMENSION.PS_SURROGATE_ID AS PS_SURROGATE_ID,
  KOZASAS.TIME_DIMENSION.TIME_ID AS TIME_ID,
  KOZASAS.DIMENSION_DATE.DATE_ID AS DATE_ID,
  KOZASAS.FACT_TABLE_1_PART_2.DURATION AS DURATION,
  KOZASAS.FACT_TABLE_1_PART_2.PROPAGATION_PROCESS_ID AS
    PROPAGATION_PROCESS_ID,
  KOZASAS.FACT_TABLE_1_PART_2.EVENT_ID_FIRST AS
    EVENT_ID_FIRST,
  KOZASAS.FACT_TABLE_1_PART_2.EVENT_ID_NEXT AS EVENT_ID_NEXT
FROM
  KOZASAS.TIME_DIMENSION,
  KOZASAS.FACT_TABLE_1_PART_2,
  KOZASAS.PS_DIMENSION,
  KOZASAS.EVENT_DIMENSION,
  KOZASAS.DIMENSION_DATE
WHERE
  (
    (
      KOZASAS.PS_DIMENSION.PS_NAME =
      KOZASAS.FACT_TABLE_1_PART_2.PS_NAME
      AND
      KOZASAS.FACT_TABLE_1_PART_2.DATE_F =
      KOZASAS.DIMENSION_DATE.DATE_D
      AND
      KOZASAS.FACT_TABLE_1_PART_2.EVENT_NAME_FIRST =
      KOZASAS.EVENT_DIMENSION.EVENT_NAME
    )
  )
AND
  (( MINUTE( KOZASAS.FACT_TABLE_1_PART_2.TIME_F ) =
    KOZASAS.TIME_DIMENSION.MINUTE_T )
  )
AND
  ( HOUR( KOZASAS.FACT_TABLE_1_PART_2.TIME_F ) =
    KOZASAS.TIME_DIMENSION.HOUR_T ) )
)

```

Abbildung 34 Auswahl der jeweiligen Primärschlüssel

Das Einfügen der Werte in die jeweilige Zieltabelle erfolgt analog zu den vorherigen Schritten über die Spaltenzuordnungen.

4. CREATE_FACT_TABLE_1_PART_4

Dieser Schritt ist analog zu vorherigem, aber mit dem Unterschied, dass hier der Name für das Attribut *Event_Name_Next* ermittelt wird. Die SQL-Anweisung ist analog zu vorherigem Schritt. Unterschied besteht jedoch darin, dass in der WHERE-Klausel die Join Bedingung lautet:

```

KOZASAS.FACT_TABLE_1_PART_2.EVENT_NAME_NEXT =
KOZASAS.EVENT_DIMENSION.EVENT_NAME

```

5. CREATE_FACT_TABLE_1_STAGING_AREA

In Schritt 5 geht es darum, dass sämtliche Informationen aus vorherigen Schritten in eine gemeinsame Zieltabelle, nämlich *Fact_Table_1* (STAGE_DB), gespeichert werden. Hier finden die Transformationen für die Attribute *Number_of_Process* und *Number_of_Restart* statt. Diese

Transformationen, die in Abschnitt 5.1.2 erläutert wurden, werden innerhalb der CASE-Anweisungen durchgeführt. Sämtliche Join Bedingungen sind innerhalb der WHERE-Klausel aufgeführt.

SQL-Anweisung:

```

SELECT DISTINCT
  KOZASAS.FACT_TABLE_1_PART_3.EVENT_ID_FIRST AS EVENT_ID_FIRST,
  KOZASAS.FACT_TABLE_1_PART_4.EVENT_ID_NEXT AS EVENT_ID_NEXT,
  KOZASAS.FACT_TABLE_1_PART_3.PS_ID AS PS_ID,
  KOZASAS.FACT_TABLE_1_PART_3.TIME_ID AS TIME_ID,
  KOZASAS.FACT_TABLE_1_PART_3.DATE_ID AS DATE_ID,
  KOZASAS.FACT_TABLE_1_PART_3.DURATION AS DURATION,
  CASE WHEN KOZASAS.FACT_TABLE_1_PART_2.EVENT_NAME_FIRST =
'START'          THEN 1
      WHEN KOZASAS.FACT_TABLE_1_PART_2.EVENT_NAME_FIRST =
'FAILURE'        THEN 1 ELSE 0 END AS NUMBER_PROCESS,
  CASE WHEN KOZASAS.FACT_TABLE_1_PART_2.EVENT_NAME_FIRST =
'RESTART'       THEN 1 ELSE 0 END AS NUMBER_RESTART,
  KOZASAS.FACT_TABLE_1_PART_3.PROPAGATION_PROCESS_ID AS
  PROPAGATION_PROCESS_ID
FROM
  KOZASAS.FACT_TABLE_1_PART_4,
  KOZASAS.FACT_TABLE_1_PART_2,
  KOZASAS.FACT_TABLE_1_PART_3
WHERE
  (
    (
      KOZASAS.FACT_TABLE_1_PART_4.ETT_ID_1 =
      KOZASAS.FACT_TABLE_1_PART_3.ETT_ID_1
      AND
      KOZASAS.FACT_TABLE_1_PART_3.ETT_ID_2 =
      KOZASAS.FACT_TABLE_1_PART_4.ETT_ID_2
      AND
      KOZASAS.FACT_TABLE_1_PART_4.PS_ID =
      KOZASAS.FACT_TABLE_1_PART_3.PS_ID
      AND
      KOZASAS.FACT_TABLE_1_PART_4.TIME_ID =
      KOZASAS.FACT_TABLE_1_PART_3.TIME_ID
      AND
      KOZASAS.FACT_TABLE_1_PART_3.DATE_ID =
      KOZASAS.FACT_TABLE_1_PART_4.DATE_ID
      AND
      KOZASAS.FACT_TABLE_1_PART_4.DURATION =
      KOZASAS.FACT_TABLE_1_PART_3.DURATION
      AND
      KOZASAS.FACT_TABLE_1_PART_3.PROPAGATION_PROCESS_ID =
      KOZASAS.FACT_TABLE_1_PART_4.PROPAGATION_PROCESS_ID
      AND
      KOZASAS.FACT_TABLE_1_PART_2.EVENT_ID_FIRST =
      KOZASAS.FACT_TABLE_1_PART_4.ETT_ID_1
      AND
      KOZASAS.FACT_TABLE_1_PART_4.ETT_ID_2 =
      KOZASAS.FACT_TABLE_1_PART_2.EVENT_ID_NEXT
      AND
      KOZASAS.FACT_TABLE_1_PART_2.EVENT_ID_FIRST =
      KOZASAS.FACT_TABLE_1_PART_3.ETT_ID_1
      AND
      KOZASAS.FACT_TABLE_1_PART_2.EVENT_ID_NEXT =
      KOZASAS.FACT_TABLE_1_PART_3.ETT_ID_2
    )
  )
)

```

Abbildung 35 Zusammenführung sämtlicher Informationen

Das Einfügen der Werte in die Zieltabelle erfolgt analog zu den vorherigen Schritten über die Spaltenzuordnungen mit der Option Ersetzen.

6. CREATE_FACT_TABLE_1_DWLOGDB

Im letzten Schritt werden alle Informationen aus der *Fact_Table_1* (STAGE_DB) in *Fact_Table_1* (DWLOGDB) geladen. Bei der Selektionsanweisung erfolgt durch die ORDER BY Klausel eine Sortierung der Informationen, welche anschließend durch die Spaltenzuordnungen in den Data Presentation Area geladen werden.

SQL-Anweisung:

```
SELECT DISTINCT
  KOZASAS.FACT_TABLE_1.EVENT_ID_FIRST AS EVENT_ID_FIRST,
  KOZASAS.FACT_TABLE_1.PS_ID AS PS_ID,
  KOZASAS.FACT_TABLE_1.EVENT_ID_NEXT AS EVENT_ID_NEXT,
  KOZASAS.FACT_TABLE_1.DATE_ID AS DATE_ID,
  KOZASAS.FACT_TABLE_1.TIME_ID AS TIME_ID,
  KOZASAS.FACT_TABLE_1.DURATION AS DURATION,
  KOZASAS.FACT_TABLE_1.NUMBER_OF_PROCESS AS NUMBER_OF_PROCESS,
  KOZASAS.FACT_TABLE_1.NUMBER_OF_RESTART AS NUMBER_OF_RESTART,
  KOZASAS.FACT_TABLE_1.PROPAGATION_PROCESS_ID AS
    PROPAGATION_PROCESS_ID
FROM
  KOZASAS.FACT_TABLE_1
ORDER BY
  PS_ID,
  EVENT_ID_FIRST,
  EVENT_ID_NEXT,
  TIME_ID,
  DATE_ID,
  DURATION,
  NUMBER_OF_PROCESS,
  NUMBER_OF_RESTART,
  PROPAGATION_PROCESS_ID
```

Abbildung 36 Selektion der neuen Informationen für *Fact_Table_1* (DWLOGDB)

Durch die Spaltenzuordnung werden die selektierten Informationen anschließend in die *Fact_Table_1* (DWLOGDB) geladen. Als Option wurde hier Hinzufügen verwendet.

7.1.3 Umsetzung der Konsistenz

In diesem Abschnitt wird erläutert wie die in Abschnitt 5.1.1 beschriebene Vorgehensweise zur Konsistenzsicherung mit Hilfe des Werkzeuges DB2 Data Warehouse Center umgesetzt wurde. Um sicherzustellen, dass sich nach jedem erfolgreichen Durchlauf eines ETL-Prozesses dieselben Informationen im DW befinden, wurden sechs zusätzliche Tabellen, die in Tabelle 22 aufgelistet sind, definiert.

Tabelle	Semantik	Genutzt von
Last_ETT_ID_Current_Load	Enthält die letzte ETT_ID des aktuellen ETL-Prozesses	PS, Source, Destination Dimension und Fact_Table_1, Fact_Table_2 und Fact_Table_3
Last_PS_ETT_ID_Staging_Area	Enthält die letzte ETT_ID aus vorherigem ETL-Prozess für PS Dimension	PS Dimension
Last_Source_ETT_ID_Staging_Area	Enthält die letzte ETT_ID aus vorherigem ETL-Prozess für Source Dimension	Source Dimension
Last_Dest_ETT_ID_Staging_Area	Enthält die letzte ETT_ID aus vorherigem ETL-Prozess für Destination Dimension	Destination Dimension
Last_ETT_ID_Fact_Table_1	Enthält die letzte ETT_ID aus vorherigem ETL-Prozess für Fact_Table_1	Fact_Table_1
Last_ETT_ID_Fact_Table_2	Enthält die letzte ETT_ID aus vorherigem ETL-Prozess für Fact_Table_2	Fact_Table_2
Last_ETT_ID_Fact_Table_3	Enthält die letzte ETT_ID aus vorherigem ETL-Prozess für Fact_Table_3	Fact_Table_3

Tabelle 22 Definierte Tabellen zur Konsistenzsicherung

Für die Dimensionen *Detail* und *Event* wurden keine derartigen Tabellen benötigt, da, wie bereits in Abschnitt 5.1.1 erwähnt wurde, beim Extrahieren immer die gesamten Daten aus den Quelltabellen *Event* und *Event_Detail* in den Data Staging Area extrahiert werden.

Bei jedem Durchlauf eines Sub-ETL-Prozesses erfolgt jeweils eine Prüfung der Art $Last_ETT_ID < ETT_ID \leq Last_Current_ETT_ID$, d.h. es werden lediglich noch nicht in den Dimensions- und Faktentabellen befindlichen Informationen in den Data Staging Area extrahiert.

7.2 Implementierung der Anfragen

Die in Kapitel 6 definierten Anfragen werden mit Hilfe von Microstrategy 6.0 implementiert. Doch zuvor werden die Grundlagen dieses Werkzeugs erläutert. Im Anschluss erfolgt der Aufbau der OLAP Architektur, welche die Grundlage für die eigentliche Implementierung der Anfragen darstellt.

7.2.1 Grundlagen zu Microstrategy 6.0

Zur Implementierung der Anfragen wurde Microstrategy 6.0 verwendet. Microstrategy verwendet eine 3-Tier-Architektur und zwar mit den Komponenten Intelligence Server, der als Herzstück von Microstrategy gilt, Agent und Architect. Die 3-Tier-Architektur ist grob in Abbildung 37 abgebildet.

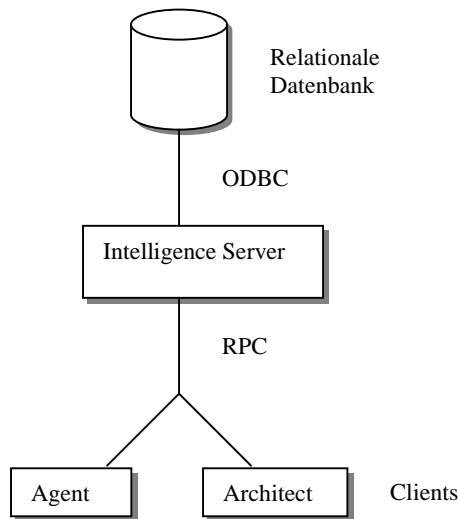


Abbildung 37 3-Tier-Architektur von Microstrategy 6.0 [MSI99]

Es wäre auch möglich zunächst den 2-Tier Modus zu verwenden, d.h. ohne Intelligence Server und nach erfolgreichem Erstellen eines Projektes den Intelligence Server hinzuzufügen. Als Kommunikationsmechanismen werden RPC und ODBC verwendet, wobei ODBC zur Kommunikation mit der Datenbank und RPC zur Kommunikation mit den Clients verwendet werden. Im Folgenden werden die drei Komponenten (Intelligence Server, Agent und Architect) erläutert:

- **Intelligence Server:** Der Intelligence Server, ein erweiterter OLAP-Server, fungiert als Middleware, welcher Anfragen von Clients empfängt und Anfragen an die Datenbank sendet [MSI99].
- **Architect:** Der Architect wird verwendet um ein multidimensionales Modell zu entwerfen und zwar für die Daten, die im DW gespeichert sind. Er erstellt ein Mapping zwischen End-Benutzer Objekten und dem physischen Schema des DWs durch Definieren und Pflegen von Metadaten. Diese Metadaten ermöglichen das dynamische Verbinden von Decision Support Systemen (zum Beispiel Agent) mit dem DW. Der Architect besteht insgesamt aus fünf Schritten:

Schritt 1 Connection: Der erste Schritt beinhaltet das Definieren des Projektnamens, System Administrator, Metadaten und DW Umgebung. Anzumerken sei, dass als Projekt der Aufbau eines multidimensionalen Modells bezeichnet wird.

Schritt 2 Warehouse: Hierbei werden sämtliche benötigten Tabellen aus dem DW festgelegt, d.h. die Tabellen die für den Aufbau des multidimensionalen Modells die notwendigen Informationen beinhalten.

Schritt 3 Fakten: In diesem Schritt werden sämtliche Fakten ausgewählt, die für das multidimensionale Datenmodell benötigt werden bzw. die Grundlage für den Agent zur Definition von Metriken sind.

Schritt 4 Attribute: Dieser Schritt ist der Aufwendigste von allen, denn es müssen sämtliche Dimensionen definiert werden und zwar mit den entsprechenden Attributen und Attribut-Hierarchien innerhalb dieser Dimensionen.

Schritt 5 Komponenten: In Schritt 4 werden die Attribute für das multidimensionale Modell definiert, während in Schritt 5 die Präsentations Struktur für das multidimensionale Modell im Agent.

Bevor der Architect jedoch genutzt werden kann, um Projekte zu definieren, müssen zunächst die Metadaten Tabellen installiert werden. Hierfür gibt es je nach verwendeter Datenbanksystem Skripte, die nach Ausführung die erforderlichen Tabellen anlegen.

Das definierte multidimensionale Datenmodell setzt sozusagen auf dem DW auf. Innerhalb des Architekten können je nach Anforderungen der End-Benutzer lediglich die relevanten Informationen aus dem DW geladen werden, d.h. eine Definition sämtlicher Dimensionen aus dem DW ist nicht zwingend erforderlich. Bei der Definition des multidimensionalen Datenmodells richtet man sich stets an die Anforderungen des Benutzers. Ein anderer Aspekt ist, dass für unterschiedliche Gruppen unterschiedliche Datenmodelle definiert werden können. Was für die eine Gruppe interessant bzw. erlaubt ist, kann für die andere Gruppe uninteressant bzw. nicht erlaubt sein [MSC99].

- **Agent:** Mit Hilfe des Agents, der relationalen OLAP-Schnittstelle, ist ein Zugriff durch den End-Benutzer auf unternehmensweite Daten des DWs bzw. multidimensionalen Datenmodells möglich. Der Agent erlaubt unter anderem das Erstellen von Reports, enthält eine ROLAP Engine, welche die Reports in SQL-Anfragen übersetzt, und erweiterte Features wie Agenten usw. Agenten, nicht zu verwechseln mit dem Agent, werden verwendet, um mehrere zusammenhängende Reports nacheinander auszuführen. Dadurch müssen die Reports nicht jedes mal einzeln ausgeführt werden. Beim Definieren von Reports gibt es einige Aspekte die beachtet werden müssen. Einer davon ist, dass darauf geachtet werden muss, dass nicht zu viele Metriken verwendet werden, da dies ein Zeichen für unklare Reports ist. Des Weiteren sollte der Name der Metriken klar und verständlich sein, da es sonst beim Benutzer zu Verständnisprobleme kommen könnte. Außerdem sollte die Anzahl der Attribut-elemente im Report nicht zu groß sein. Dies ist meistens ein Zeichen dafür, dass der Benutzer nicht aufgefordert wird Einschränkungen festzulegen. Zum Inhalt eines Reports gehören:

→ *Template*: Innerhalb eines Templates wird einerseits definiert, welche Informationen aus dem DW geladen werden sollen und andererseits die Art der Darstellung des Reportergebnisses. Das Template liefert der Microstrategy Engine die Informationen für die SQL Klauseln `SELECT` und `GROUP BY`. Vorgehensweise sieht so aus, dass zunächst ein Ordner für die Templates definiert werden muss, in dem anschließend die Templates abgelegt werden. Es ist nicht notwendig die Templates separat zu definieren, denn innerhalb der Reportdefinition erfolgt eine Auswahl der Objekte für das Template, wobei das Werkzeug automatisch einen Template Ordner anlegt, in dem er die Templates der Reports ablegt.

→ *Filter*: Filter werden verwendet, um die Anzahl der Attributelemente im Report einzuschränken. Ein Filter besteht aus einem oder mehreren Attributelementen (zum Beispiel Monat = „Januar“), Attributeinschränkungen (zum Beispiel Ergebnis `BETWEEN 10000 AND 20000`), Metrikeinschränkungen (zum Beispiel Umsatz > 10000), weiteren Filter, etc. In SQL entspricht der Filter der `WHERE` Klausel. Analog zur Vorgehensweise für Templates muss auch hier zunächst ein Filter Ordner angelegt werden, in dem die Filter anschließend abgelegt werden. Innerhalb der Filterdefinition gibt es drei Filteroperatoren, die in Abbildung 38 abgebildet sind.

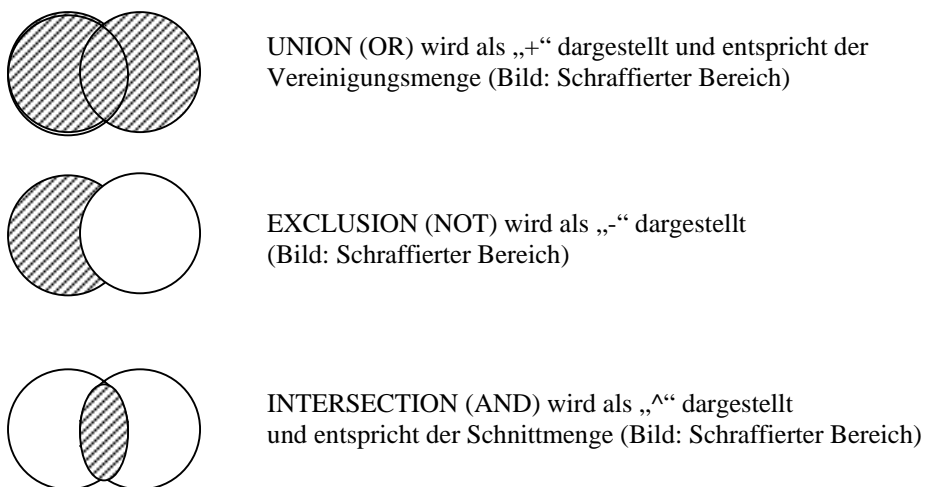


Abbildung 38 Filteroperatoren

Innerhalb der Filter Definition ist `AutoPrompt` ein sehr nützliches Feature. Mit Hilfe dieses Features ist es möglich, dass beim Ausführen eines Reports der Benutzer aufgefordert wird Filterkriterien auszuwählen. Hierbei können entweder einzelne Attributelemente ausgewählt werden oder es wird das Attribut gewählt, d.h. der Benutzer erhält beim Ausführen des Reports sämtliche möglichen Attributelemente als Liste, aus der ein oder mehrere Attributelemente auswählen kann.

→ *Metriken*: Metriken sind benutzer-definierte analytische Berechnungen auf Fakten des DWs und werden innerhalb von Templates verwendet. Sie können ebenfalls in Filter verwendet werden, um Metriken einzuschränken wie zum Beispiel Umsatz < 10000. Die Metrik Definition entspricht in SQL den Aggregationsfunktionen COUNT(DISTINCT), MIN, MAX, SUM, AVG und COUNT. Mögliche Metriken bestehen entweder aus aggregierten Fakten oder aus mehreren Metriken, aber nicht aus beiden [MSA99]

7.2.2 Bereitstellen der Daten

Die Grundlage für den Microstrategy Agent ist die OLAP Architektur, die zunächst mit Hilfe des Microstrategy Architect definiert werden muss. Erst wenn die Architektur definiert wurde ist es möglich Anfragen zu implementieren.

Hierbei ist aber anzumerken, dass das Werkzeug Microstrategy in der Regel von einem Snowflake Schema ausgeht. Darauf aufbauend werden für Dimensionen Lookup Tabellen, etc. definiert. Die Schwierigkeit lag jedoch darin, dass ein Star Schema verwendet wurde und es nicht in jeder Dimension Hierarchien gab. Lediglich die Dimension *Date* und *Time* beinhalten Hierarchien. Um diese Schwierigkeit zu vermeiden könnte für die OLAP Anfragen ein anderes Werkzeug verwendet werden wie zum Beispiel SAP Business Information Warehouse 3.5 (SAP BW 3.5). Dieses Werkzeug kann sowohl für den Aufbau des DWs als auch zur Implementierung der OLAP Anfragen verwendet werden. Der Vorteil von SAP BW 3.5 ist, dass ein Star Schema verwendet wird und kein Snowflake Schema. Für weitere Informationen sei auf [SAP] verwiesen.

Um ein Projekt anzulegen ist es jedoch erforderlich, dass bei der Definition der Attribute in jeder Dimension Hierarchien bzw. Parent-Child Beziehungen gibt. Das Werkzeug setzt voraus, dass Attribute mindestens einen gemeinsamen Nachkommen besitzen. Innerhalb der Dimensionen *Time* und *Date* war es kein Problem die Hierarchien zu definieren. Jedoch in den Dimensionen *PS*, *Source*, *Destination*, *Event* und *Detail* gibt es keine Hierarchien. Daher war es nötig bei der Definition der Hierarchien den Surrogate Key der Dimensionen mit in die Definition hinzuzufügen und anschließend sämtliche Attribute auf den Surrogate Key abzubilden.

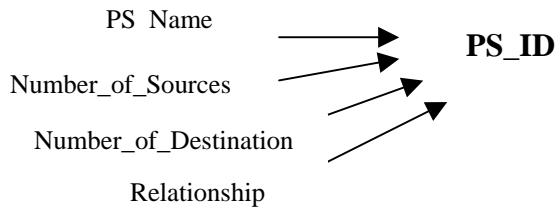


Abbildung 39 Mapping der Attribute auf PS_ID (Surrogate Key)

Abbildung 39 zeigt das Mapping zwischen den Attributen und der *PS_ID* (Surrogate Key) der *PS* Dimension. Analog wird das für die *Event*, *Destination*, *Source* und *Detail* Dimensionen gemacht. Zwar wurde der Surrogate Key als Attribut definiert, allerdings nicht innerhalb der Komponenten, d.h. es ist innerhalb des Agents nicht sichtbar. Folgende Schritte sind für den Aufbau der Architektur durchzuführen:

1.Schritt:

Zunächst musste eine Verbindung (ODBC-Verbindung) zum Datenbanksystem hergestellt werden. Innerhalb dieser Datenbank müssen jedoch zuvor die Metadaten Tabellen installiert werden.

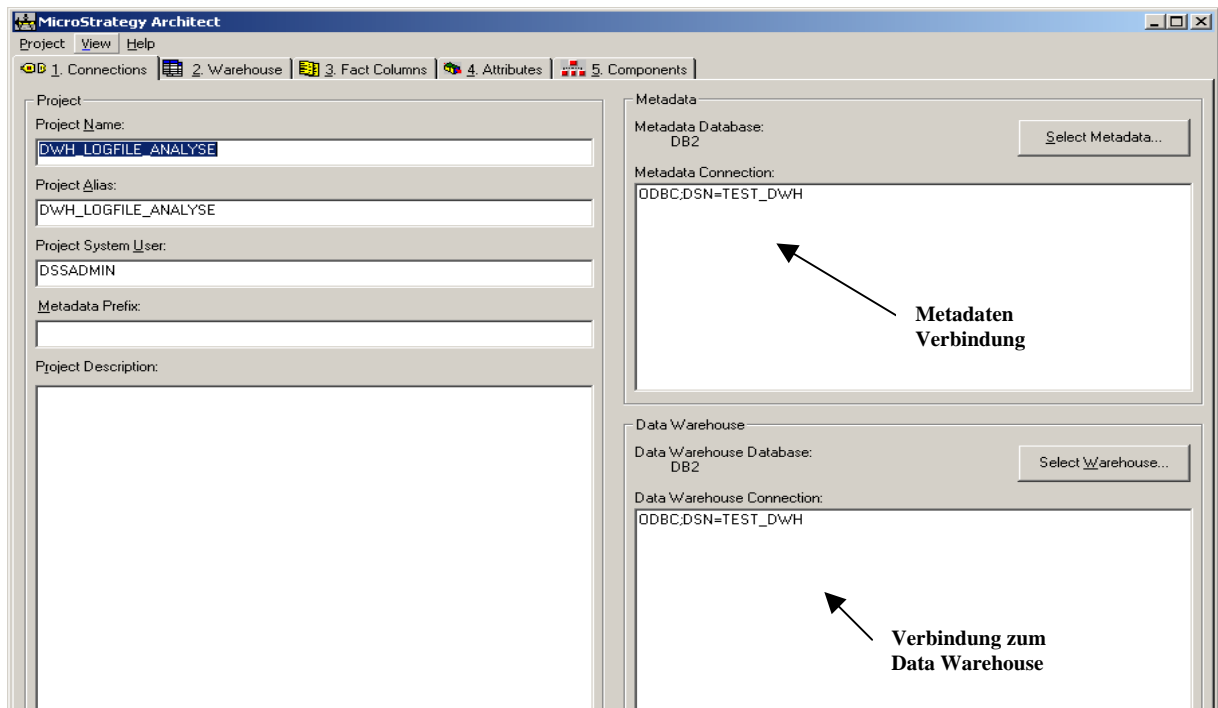


Abbildung 40 Microstrategy (MS) Architect – Projektdefinition

2.Schritt:

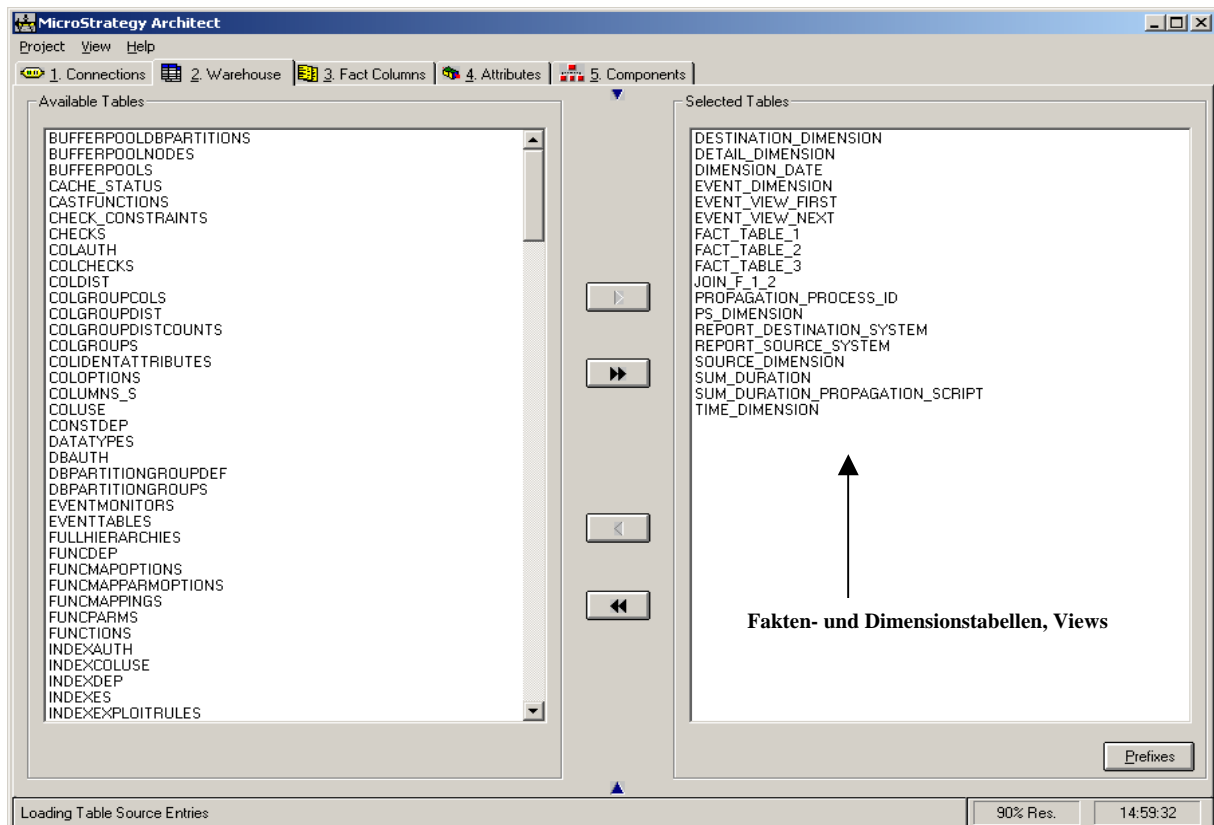


Abbildung 41 MS Architect – Auswahl Views, Dimensions- und Faktentabellen

Nachdem die Verbindung zum DW hergestellt wurde erfolgt die Auswahl der erforderlichen Faktentabellen, Dimensionstabellen und Views.

3.Schritt:

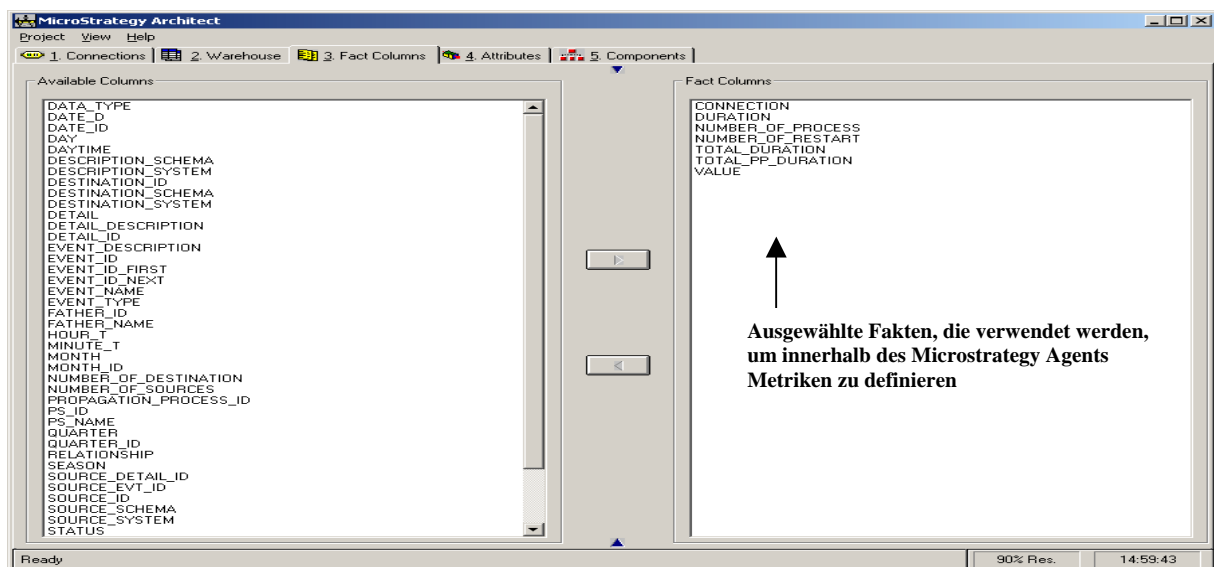


Abbildung 42 MS Architect – Auswahl der Fakten

4.Schritt :

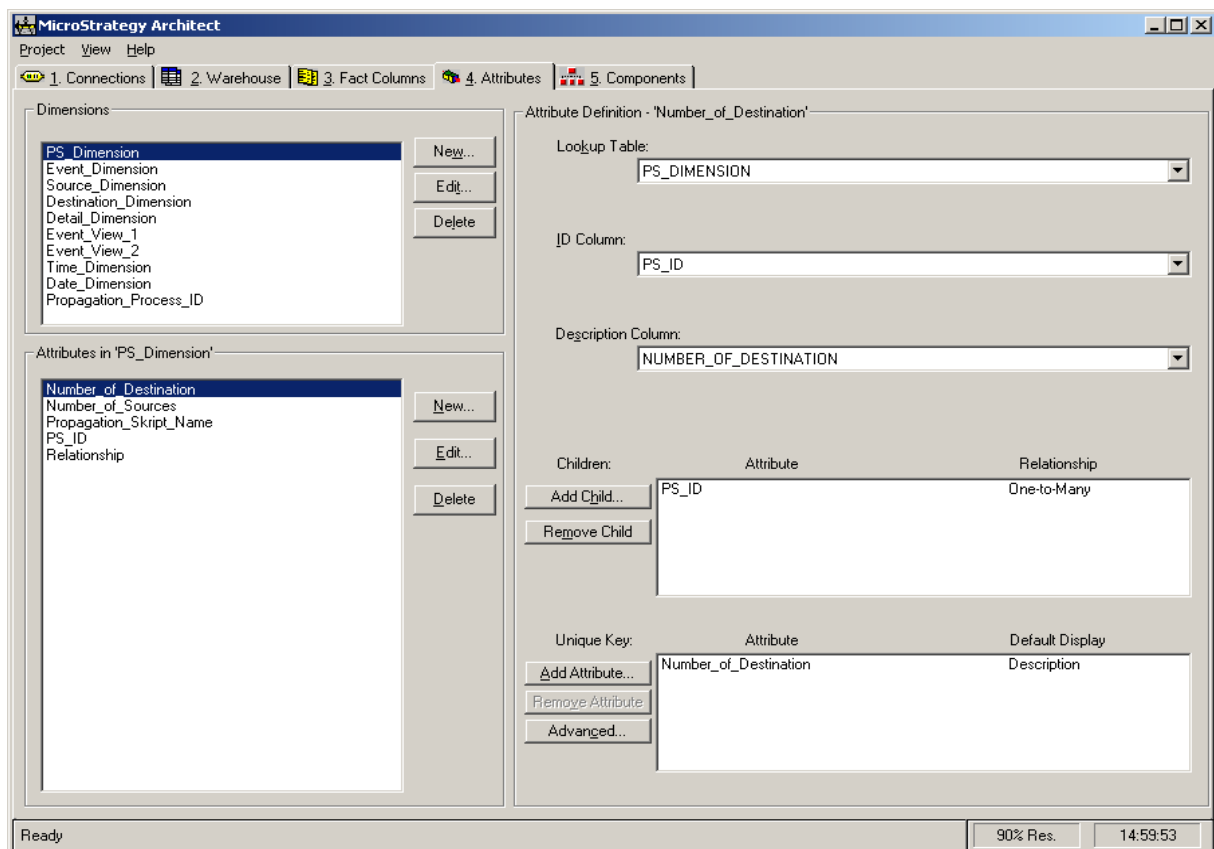


Abbildung 43 MS Architect – Definition der Attribute

Als nächstes müssen die Attribute der einzelnen Dimensionen einschließlich der jeweiligen Hierarchien definiert werden. Zu jedem Dimensionsattribut des Projektes sind die jeweiligen Lookup Tabellen, Identifikations- und Beschreibungsattribute anzugeben.

Wie bereits erwähnt wurde, gab es nicht für jedes Attribut eine eigene Lookup Tabelle, wie das beim Snowflake Schema der Fall ist. Vielmehr gab es für jedes Attribut eine einzige Lookup Tabelle, wobei die Lookup Tabellen den jeweiligen Dimensionstabellen entsprachen. Im Anschluss konnten die Hierarchien durch Angabe von Parent-Child Beziehungen definiert werden, wobei es lediglich innerhalb der *Time* und *Date* Dimensionen Hierarchien gibt. Für die Dimension *Time* zum Beispiel gibt es die Hierarchie *Daytime* → *Hour* → *Minute* .

5.Schritt

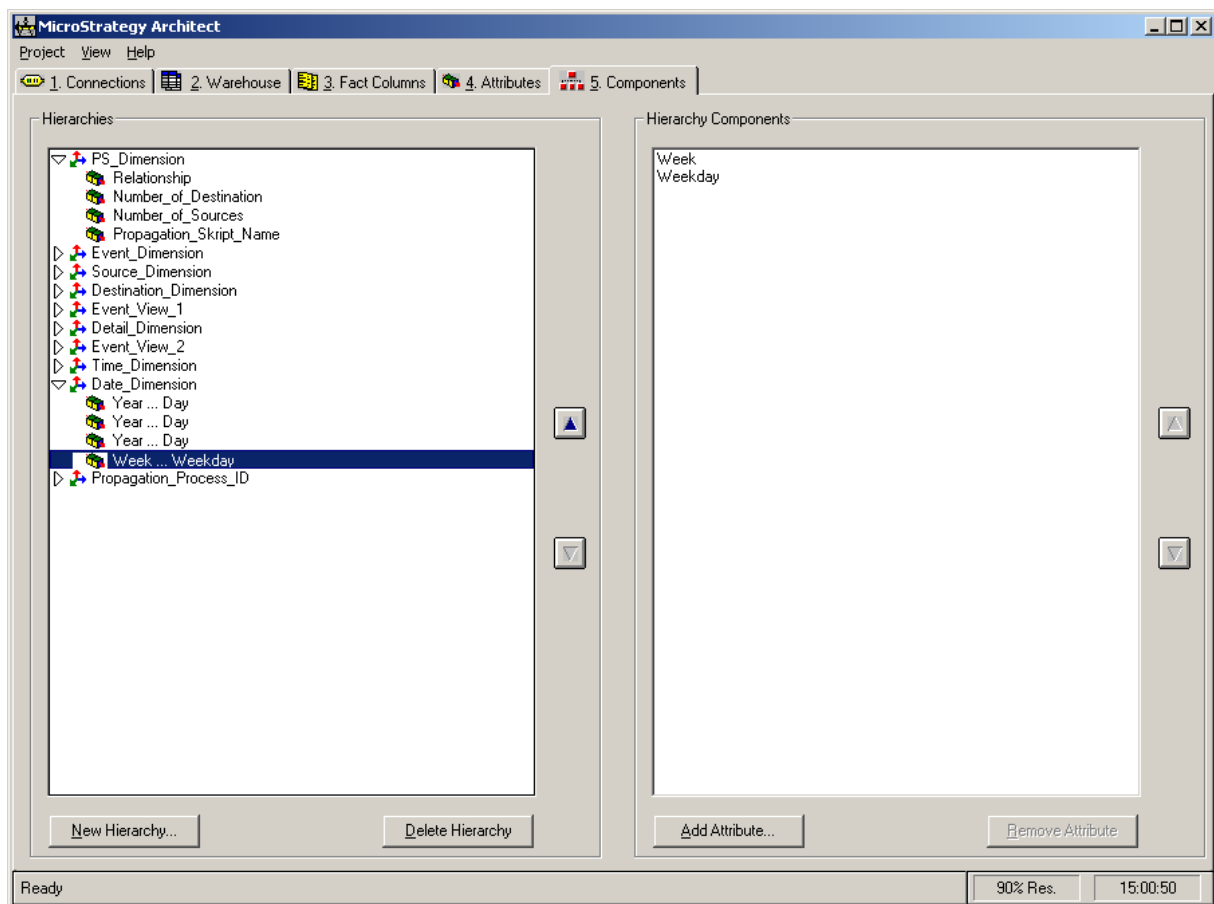


Abbildung 44 MS Architect – Bereitstellen der Komponenten für den Agent

Zum Schluss müssen noch die Attribute und Hierarchien, die im Microstrategy Agent zur Verarbeitung zur Verfügung stehen sollten, bestimmt werden. Erst dann ist es möglich den Microstrategy Agent zu verwenden, um Anfragen zu implementieren.

7.2.3 Umsetzung der Anfragen

Um die definierten Anfragen zu implementieren wurde der Microstragey Agent verwendet. Er stellt eine grafische Oberfläche zur Verfügung, die zur Modellierung und Ausführung von Anfragen genutzt werden kann.

Im Folgenden wird die Vorgehensweise zur Implementierung zweier Anfragen erläutert:

1. Anfrage : Wie lang ist die reale Dauer eines Propagationsprozesses in einem bestimmten Zeitraum?

→ Definition der Metriken

Bevor ein Report definiert werden kann, ist eine Identifikation der erforderlichen Metriken notwendig.

Um die reale Dauer eines Propagationsprozesses zu ermitteln waren drei Metriken erforderlich.

- Metrik *Summe_Duration* zur Ermittlung der Gesamtdauer eines Propagationsprozesses.

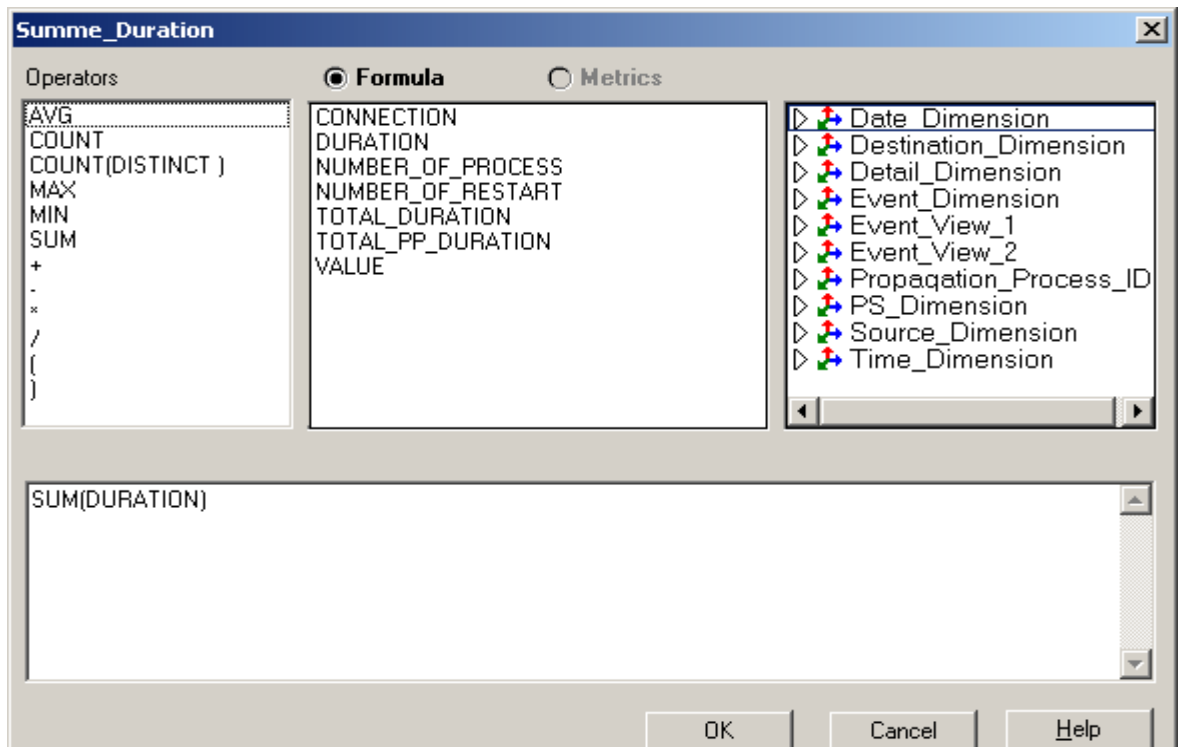


Abbildung 45 MS Agent – Definition *Summe_Duration*

- Die Metrik *Inaktive_Dauer_Propagation_Prozess* berechnet den inaktiven Anteil innerhalb eines Propagationsprozesses. Hierfür ist jedoch ein Filter notwendig, der im nächsten Schritt definiert werden muss.

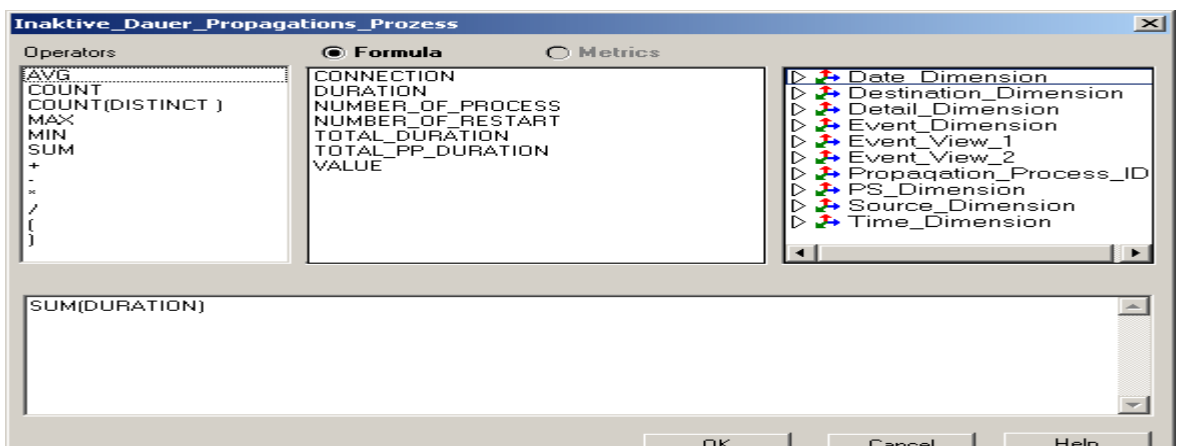


Abbildung 46 MS Agent – Definition *Inaktive_Dauer_Propagation_Prozess*

Innerhalb der Metrik muss eine Einschränkung definiert werden, d.h. es muss eine Filterbedingung hinzugefügt werden, da für den inaktiven Anteil lediglich die Dauer eines Sleep Ereignisses von Bedeutung ist.

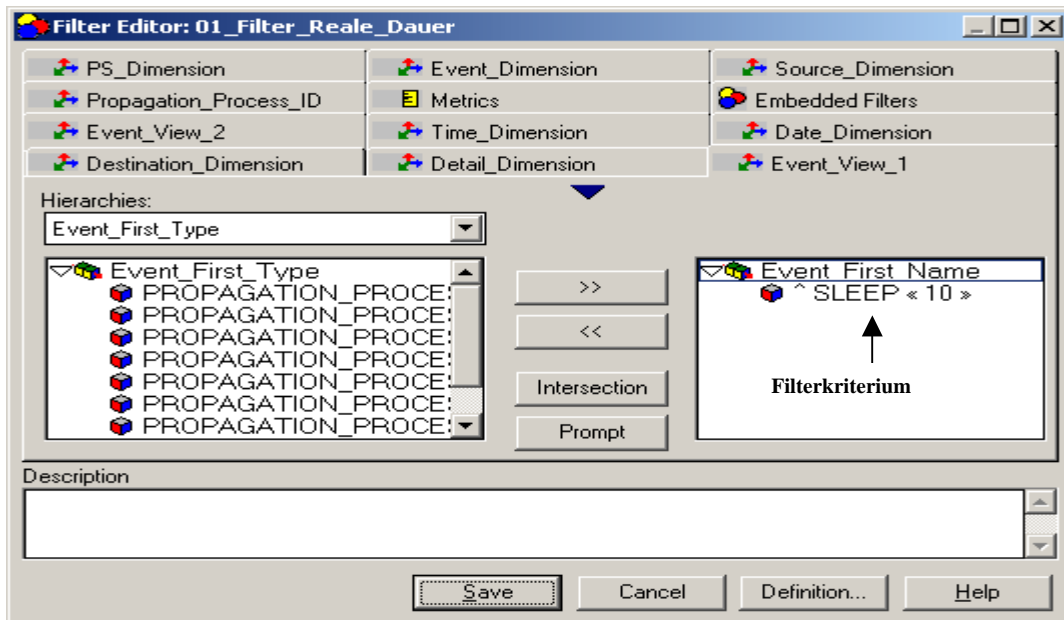


Abbildung 47 MS Agent – Definition des Filters

- Bei der Metrik *Reale_Dauer_Propagation_Process* handelt es sich um eine zusammengesetzte Metrik, welche die reale Dauer eines Propagationsprozesses ermittelt. Dazu werden die Werte, die aus den oberen beiden Metriken errechnet wurden, subtrahiert.

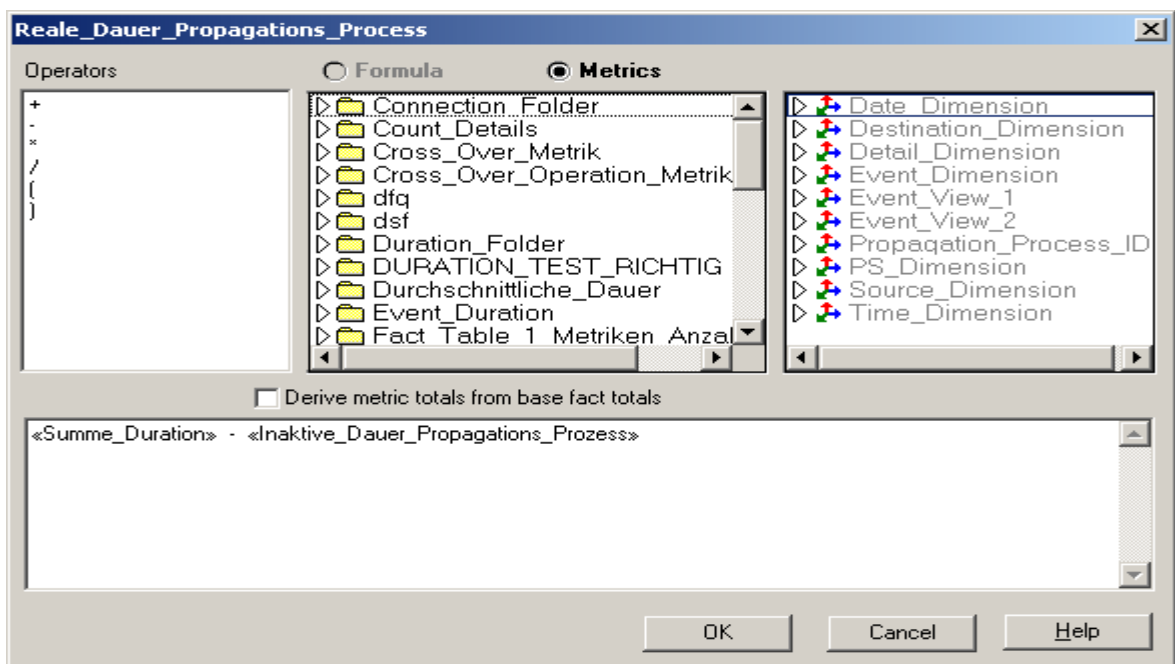


Abbildung 48 MS Agent – Definition Reale_Dauer_Propagation_Process

→ Definition der Templates

Hierbei geht es darum die Informationen, die im Report dargestellt werden sollen, festzulegen. Ein Template bestimmt, welche der definierten Metriken und welche Attribute aus dem DW im Report dargestellt werden. Des Weiteren wird durch ein Template festgelegt wie die Informationen auf dem Bildschirm dargestellt werden.

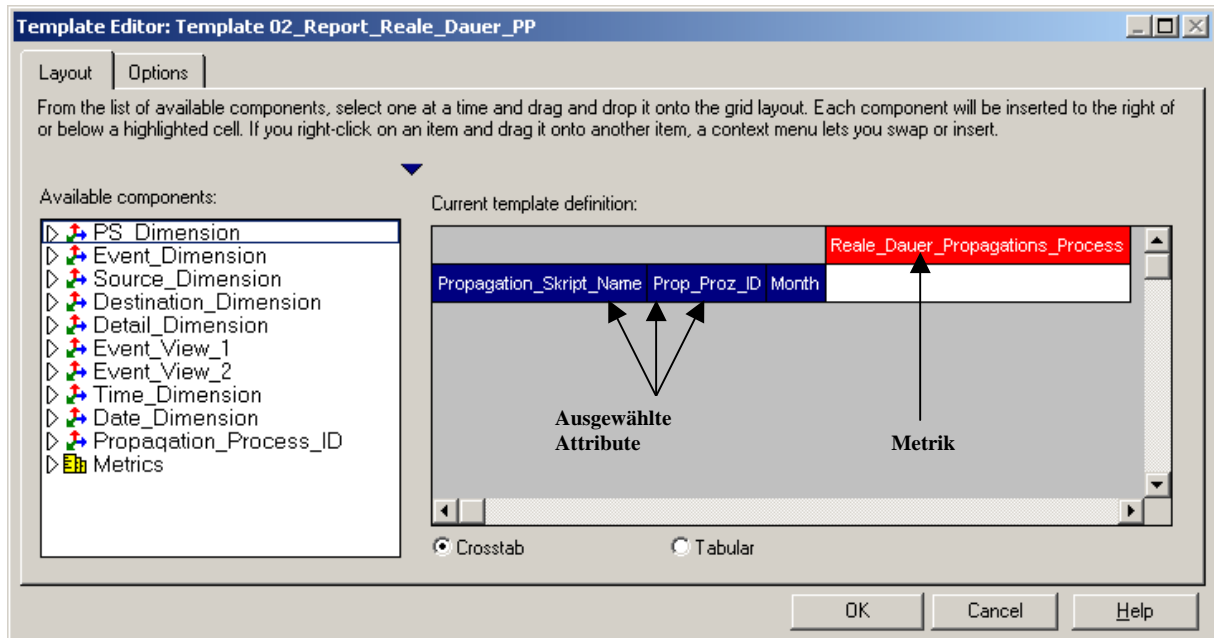


Abbildung 49 MS Agent – Definition des Templates für die 1. Anfrage

Das Template für die Anfrage ist in Abbildung 49 abgebildet. Das Attribut *Month* repräsentiert die *Date* Dimension, *Propagation_Skript_Name* die Dimension *PS* und *Prop_Proz_ID* die *Propagation_Process_ID* Dimension. Als Metrik für diese Anfrage wurde *Reale_Dauer_Propagation_Process*, welche sich auf das Faktum *duration* bezieht, gewählt.

Auf Grund der obigen Repräsentanten der jeweiligen Dimensionen und durch das Faktum erfolgt ein Join zwischen den Fakten- und Dimensionstabellen. Da die Attribute *Propagation_Process_Name*, *Prop_Proz_ID* und *Month* im Report ausgewählt wurden, verwendet die ROLAP Engine diese auch um die Joins zwischen den Dimensions- und Faktentabellen durchzuführen.

→ Ausführen des Reports

Durch die Kombination des definierten Templates mit obigem Filter wird ein ausführbarer Report erzeugt. Das Resultat des definierten Reports ist in Abbildung 50 abgebildet.

Propagation Skript Name	Prop Proz ID	Month	Reale Dauer Propagations Process
delmia_prod	P_P_ID_1	3	297.000
	P_P_ID_4	3	124.999
	P_P_ID_8	3	95.432
delmia_res	P_P_ID_2	3	141.000
	P_P_ID_5	3	78.000
fp_op	P_P_ID_3	3	78.000
	P_P_ID_6	3	157.001

Abbildung 50 MS Agent – Resultat der 1. Anfrage

2. Anfrage: Was ist die durchschnittliche Dauer eines Propagationsprozesses bei dem Source_System = „Sales System“ ist?

→ Definition der Metrik *Durschnittliche_Dauer_Source_System*

Für die 2. Anfrage ist eine Metrik erforderlich, welche die durchschnittliche Dauer eines Propagationsprozesses ermittelt. Hierfür ist die Aggregationsfunktion AVG zu verwenden.

Durschnittliche_Dauer_Source_System

Operators: Formula Metrics

Operators list: AVG, COUNT, COUNT(DISTINCT), MAX, MIN, SUM, +, -, *, /, (,)

Formula list: CONNECTION, DURATION, NUMBER_OF_PROCESS, NUMBER_OF_RESTART, TOTAL_DURATION, TOTAL_PP_DURATION, VALUE

Dimensions list: Date Dimension, Destination Dimension, Detail Dimension, Event Dimension, Event_View_1, Event_View_2, Propagation_Process_ID, PS_Dimension, Source_Dimension, Time_Dimension

Formula input: AVG(TOTAL_DURATION)

Buttons: OK, Cancel, Help

Abbildung 51 MS Agent – Definition Durschnittliche_Dauer_Source_System

→ Definition des Templates

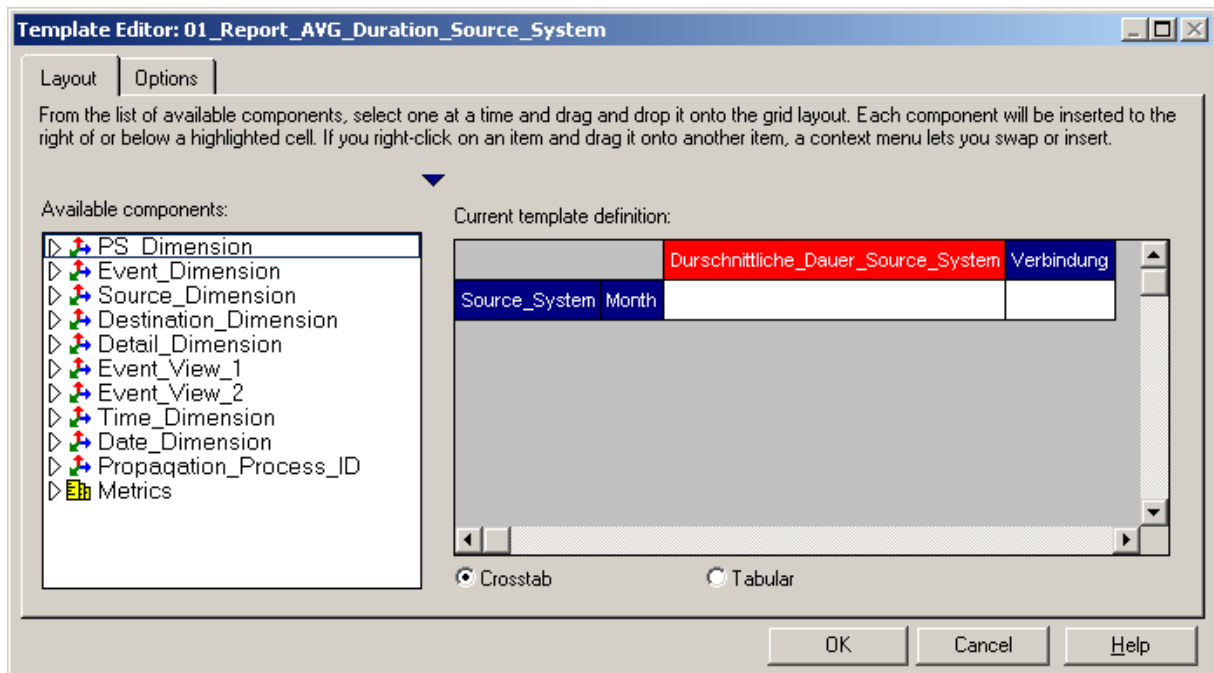


Abbildung 52 MS Agent – Definition des Templates für die 2. Anfrage

Source_System repräsentiert die *Source* Dimension und *Month* die *Date* Dimension. Zusätzlich wurde im Template *Verbindung*, welches dem Faktum der *Fact_Table_2* entspricht, definiert. Es ist notwendig um die Verbindung zwischen *Fact_Table_1* und *Fact_Table_2* herzustellen. Die ROLAP Engine verwendet automatisch *Source_System* und *Month* als Join-Attribute.

→ Definition des Filters

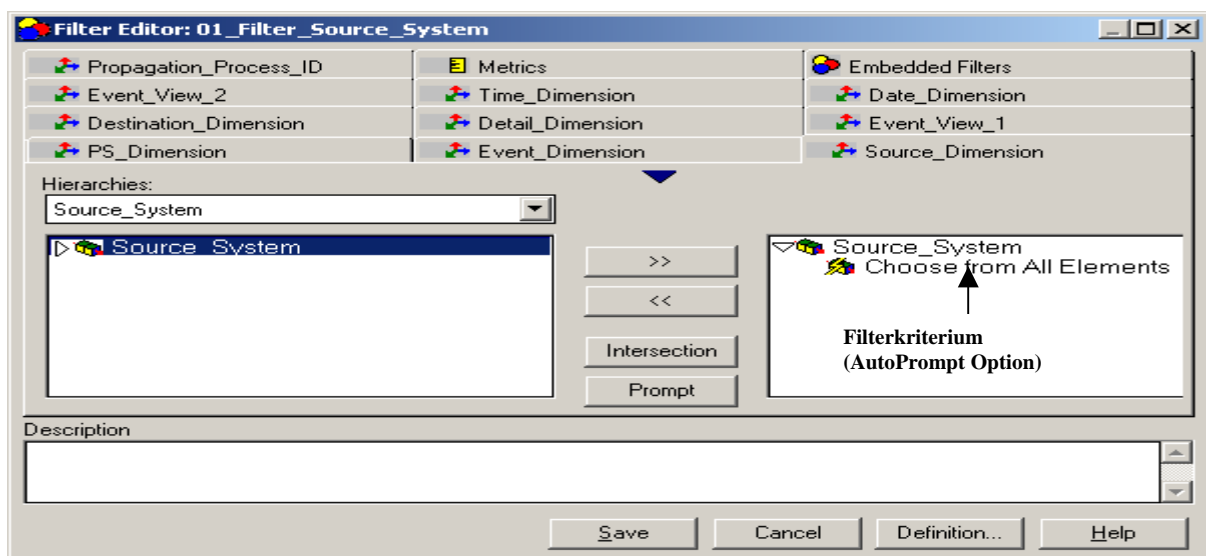
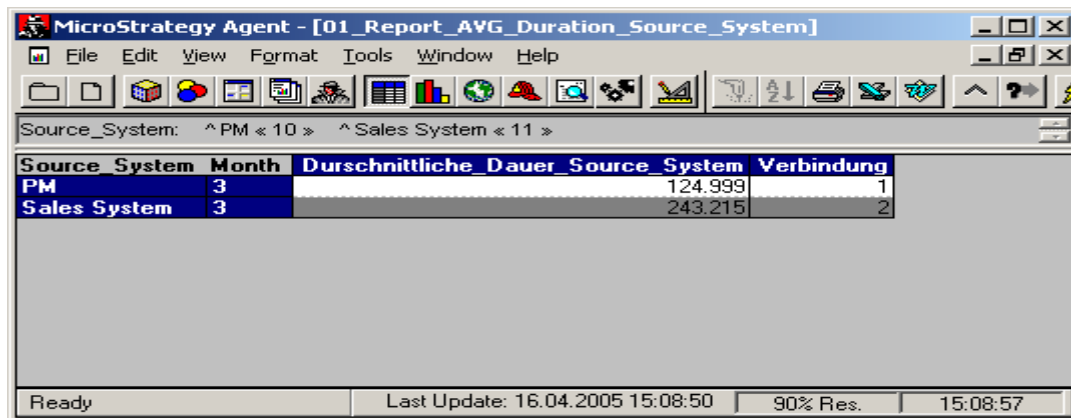


Abbildung 53 MS Agent – Definition des Filters

Es können Einschränkungen auf den zusammengestellten Attributen und Metriken im Template definiert werden. Um solche Einschränkungen machen zu können, ist es erforderlich ein Filter zu definieren. Für die 2.Anfrage wurde ein Filter definiert, um *Source_Systeme* auswählen zu können. Vor der Ausführung des Reports kann der Benutzer festlegen, welche *Source_Systeme* berücksichtigt werden sollen.

→ Ausführen des Reports

Durch die Kombination des definierten Templates mit obigem Filter wird ein ausführbarer Report erzeugt. Das Resultat des definierten Reports ist in Abbildung 54 abgebildet.



The screenshot shows the MicroStrategy Agent interface. The title bar reads "MicroStrategy Agent - [01_Report_AVG_Duration_Source_System]". The menu bar includes File, Edit, View, Format, Tools, Window, and Help. The toolbar contains various icons for report manipulation. Below the toolbar, there are two filter controls: "Source_System: ^ PM < 10 »" and "Sales System < 11 »". The main area displays a table with the following data:

Source System	Month	Durchschnittliche Dauer	Source System	Verbindung
PM	3		124.993	1
Sales System	3		243.215	2

At the bottom of the window, the status bar shows "Ready", "Last Update: 16.04.2005 15:08:50", "90% Res.", and "15:08:57".

Abbildung 54 MS Agent – Resultat der 2. Anfrage

8 Zusammenfassung und Ausblick

Das Hauptziel von SIES ist es Datenänderungen zwischen verbundenen Informationssystemen zu verwalten. Ziel dieser Arbeit war es den Prozess der Weiterleitung von Datenänderungen zwischen verschiedenen Informationssystemen zu analysieren, um beispielsweise Optimierungen durchführen zu können. Hierfür war es erforderlich eine Umgebung zu implementieren, die das Analysieren eines solchen Prozesses ermöglicht.

In dieser Arbeit wurde die Exploration Manager Umgebung erstellt. Dazu wurde einerseits ein DW modelliert und aufgebaut und andererseits OLAP Anfragen, welche auf dem DW ausgeführt werden, implementiert. Für die praktische Umsetzung des DWs wurde das Werkzeug DB2 Data Warehouse Center und für die OLAP Anfragen Microstrategy 6.0 verwendet. Die Daten, die mit Hilfe von ETL-Prozessen in das DW geladen werden, entstehen während der Propagation von Datenänderungen zwischen Informationssystemen.

Der konzeptuelle Entwurf stellte in dieser Arbeit den größten Aufwand dar. Meines Erachtens ist dieser Mehraufwand gerechtfertigt, da falsche Entscheidungen innerhalb dieser Phase für das weitere Vorgehen bis hin zur Nutzung des DWs gravierende Auswirkungen haben könnte.

Mit Hilfe dieser Arbeit wurde die Grundlage für das Analysieren von Propagationsprozessen geschaffen. In Zukunft kann das entwickelte DW dazu genutzt werden Informationen zu Propagationsprozessen zu speichern, welche anschließend mit Hilfe der vordefinierten Anfragen analysiert werden können. Das Formulieren weiterer Anfragen ist notwendig, wenn zusätzlich zu den bereits definierten Informationen zu einem Propagationsprozess weitere Informationen gespeichert werden sollen. Beispielsweise könnte irgendwann entschieden werden, dass der Benutzer, der die Datenänderung vorgenommen hat, auch gespeichert werden soll. Um eine Analyse bezüglich Benutzer durchführen zu können ist es notwendig eine neue Anfrage zu definieren, da derartige Analysen mit den aus Abschnitt 6.1 definierten Anfragen nicht möglich ist.

Literaturverzeichnis

- [BAU01] A. Bauer, H. Günzel
Data Warehouse Systeme
Dpunkt.verlag, 2001
- [CHM02] C. Constantinescu, U. Heinkel, H. Meinecke
A Data Change Propagation System for Enterprise Application Integration
In: Smari, Waleed W. (ed.); Melab, Nordine (ed.); Chen, Shu-Ching (ed.): The 2nd International Conference on Information Systems and Engineering (ISE 2002)
- [CHR02] C. Constantinescu, U. Heinkel, R. Rantza, B. Mitchang
A SYSTEM FOR DATA CHANGE PROPAGATION IN HETEROGENEOUS INFORMATION SYSTEMS
In: Proceedings of the International Conference on Enterprise Information Systems (ICEIS), Volume I, Ciudad Real, Spain, April 2002
- [CHY98] C. Y. Chan, Y. E. Ioannidis
Bitmap Index Design and Evaluation
Computer Sciences Dept., University of Wisconsin-Madiso, 1998
- [CRM01] C. Constantinescu, U. Heinkel, R. Rantza, B. Mitchang
SIES: An Approach for a Federated Information System in Manufacturing
In: Proceedings of the International Symposium on Information Systems and Engineering (ISE); Las Vegas, Nevada, USA, June 2001
- [DWC]
[GMR98] M. Golfarelli, D. Maio, S. Rizzi
Conceptual Design of Data Warehouses from E/R Schemes
Copyright 1998 IEEE, Published in the Proceedings of the Hawaii International Conference On System Sciences, January 6-9, 1998, Kona, Hawaii
- [GR98] M. Golfarelli, S. Rizzi
A Methodological Framework for Data Warehouse Design
Proceedings ACM First International Workshop on Data Warehousing and OLAP (DOLAP 98), Nov. 7, 1998, Washington, D.C., USA
- [GR99] M. Golfarelli, S. Rizzi
Designing the Data Warehouse: Key Steps and Crucial Issues
Journal of Computer Science and Information Management, Vol. 2, N. 3, 1999

- [HEI04] U. Heinkel
XPDL Dokumentation, 2004
- [INM02] W. H. Inmon
Building the Data Warehouse
Wiley Computer Publishing, John Wiley & Sons, Inc. u.a. New York, Third Edition, 2002
- [KIM98] R. Kimball, L. Reeves, M. Ross, W. Thornthwaite
THE DATA WAREHOUSE LIFECYCLE TOOLKIT: Expert Methods for Designing, Developing and Deploying Data Warehouses
John Wiley & Sons: New York u.a. 1998
- [KIM02] R. Kimball, M. Ross
The Data Warehouse Toolkit: The Complete Guide to Dimensional Modelling
Wiley Computer Publishing, John Wiley & Sons, Inc. u.a. New York, Second Edition, 2002
- [KUR99] A. Kurz
Data Warehousing Enabling Technologies
Verlag MITP, 1999
- [MAR98] W. Martin
Data warehousing, Data mining OLAP
Verlag International Thomson Publ., 1998
- [MIT01] Prof. Dr.-Ing. habil. B. Mitschang
Vorlesung Grundlagen der Datenbanksysteme und Informationssystemen, 2001/2002
- [MSA99] Advanced Topics **Microstrategy 6.0 Agent**
Application Development and Sophisticated Analysis, 1999
- [MSC99] User Guide, **Microstrategy 6.0 Architect**
Rapid Application Development, 1999
- [MSI99] User Guide, **Microstrategy 6.0 Intelligence Server**
Foundation for Intelligent E-Business, 1999
- [RCH02] R. Rantza, C. Constantinescu, U. Heinkel, H. Meinecke
CHAMPAGNE: DATA CHANGE PROPAGATION FOR HETEROGENEOUS INFORMATION SYSTEMS
In: Proceedings of the International Conference on Very Large Databases (VLDB); Demonstration Paper; Hong Kong, August 20-23, 2002
- [SAP] <http://www.sap.de>
- [SCH03] H. Schwarz
Konzeptueller und logischer Data-Warehouse-Entwurf: Datenmodelle und Schematypen für Data Mining und OLAP
Informatik Forsch. Entw. , 2003
- [TBC99] N. Tryfona, F. Busborg, J. G. Borch Christiansen
starER: A Conceptual Model for Data Warehouse Design

Abbildungsverzeichnis

Abbildung 1 SIES Architektur [CHR02]	4
Abbildung 2 Architektur der Exploration-Umgebung.....	5
Abbildung 3 Beispiel für ein Propagationsskript (XPDL).....	6
Abbildung 4 Ablauf vom Empfang bis zur Propagation einer Datenänderung	7
Abbildung 5 Logstruktur	8
Abbildung 6 Beispiel Metadaten zum Start Ereignis	9
Abbildung 7 Einfache Benachrichtigung (Java).....	10
Abbildung 8 Umfangreiche Benachrichtigung (Java).....	10
Abbildung 9 Data Warehouse Architektur	14
Abbildung 10 Dimensionsmodell – Star Schema Beispiel.....	17
Abbildung 11 Beispiel Datenwürfel	19
Abbildung 12 Beispiel für Sequenz von Ereignissen während der Ausführung eines Propagationprozesses	24
Abbildung 13 Tabellen des Logs mit den entsprechenden Informationen	25
Abbildung 14 Ebenen der Anfragen.....	27
Abbildung 15 Exploration Manager Architektur	28
Abbildung 16 Intervall zwischen zwei nachfolgenden Ereignissen.....	30
Abbildung 17 Faktenschema zu Fact_Table_1.....	31
Abbildung 18 Faktenschema zu Fact_Table_2.....	33
Abbildung 19 Faktenschema zu Fact_Table_3.....	35
Abbildung 20 Überlappungen der Faktenschemata	36
Abbildung 21 Star Schema (Galaxy).....	37
Abbildung 22 Views der Event Dimension	39
Abbildung 23 DDL Befehle zu den Dimensionen PS und Source.....	40
Abbildung 24 Beispiele für Abhängigkeiten und Sequenz einer ETL-Prozess Ausführung	42

Abbildung 25 Mögliches Szenario eines ETL-Prozesses	44
Abbildung 26 Ablauf eines DW Aufbaus mit Data Warehouse Center.....	53
Abbildung 27 Wechseln des ETL-Prozess Modus.....	54
Abbildung 28 Abbildung eines ETL-Prozesses.....	55
Abbildung 29 Selektieren der Daten aus der Quelltable Event.....	56
Abbildung 30 Selektieren der neuen Daten und Schlüsselzuweisung.....	57
Abbildung 31 Laden der neuen Daten in die vorläufige Event Dimension (STAGE_DB).....	58
Abbildung 32 Laden der Daten in den Data Presentation Area (DWLOGDB).....	59
Abbildung 33 Selektieren der neuen Daten aus Event und Event_Trail	60
Abbildung 34 Auswahl der jeweiligen Primärschlüssel	62
Abbildung 35 Zusammenführung sämtlicher Informationen.....	63
Abbildung 36 Selektion der neuen Informationen für Fact_Table_1 (DWLOGDB)....	64
Abbildung 37 3-Tier-Architektur von Microstrategy 6.0 [MSI99].....	66
Abbildung 38 Filteroperatoren.....	68
Abbildung 39 Mapping der Attribute auf PS_ID (Surrogate Key)	70
Abbildung 40 Microstrategy (MS) Architect – Projektdefinition.....	70
Abbildung 41 MS Architect – Auswahl Views, Dimensions- und Faktentabellen.....	71
Abbildung 42 MS Architect – Auswahl der Fakten	71
Abbildung 43 MS Architect – Definition der Attribute	72
Abbildung 44 MS Architect – Bereitstellen der Komponenten für den Agent	73
Abbildung 45 MS Agent – Definition Summe_Duration	74
Abbildung 46 MS Agent – Definition Inaktive_Dauer_Propagation_Process	74
Abbildung 47 MS Agent – Definition des Filters.....	75
Abbildung 48 MS Agent – Definition Reale_Dauer_Propagation_Process	75
Abbildung 49 MS Agent – Definition des Templates für die 1. Anfrage	76
Abbildung 50 MS Agent – Resultat der 1. Anfrage.....	77
Abbildung 51 MS Agent – Definition Durchschnittliche_Dauer_Source_System.....	77
Abbildung 52 MS Agent – Definition des Templates für die 2. Anfrage	78

Abbildung 53 MS Agent – Definition des Filters.....	78
Abbildung 54 MS Agent – Resultat der 2. Anfrage.....	79

Tabellenverzeichnis

Tabelle 1 Vordefinierte Ereignisse	11
Tabelle 2 Informationen zu den jeweiligen Ereignissen	12
Tabelle 3 Gegenüberstellung operationale Datenbanken und Data Warehouse	13
Tabelle 4 OLTP vs. OLAP	18
Tabelle 5 Vor- und Nachteile der verschiedenen OLAP Architekturen [KUR99]	20
Tabelle 6 Details und Werte zu den Ereignissen des Beispiels	24
Tabelle 7 Beispiele für erste mögliche Anfragen	26
Tabelle 8 Definierte Fakten für die jeweiligen Anfragen	27
Tabelle 9 Faktentabellen einschließlich der Anfragen	30
Tabelle 10 Semantik und Art der Fakten	31
Tabelle 11 Semantik der Attribute	32
Tabelle 12 Semantik der Attribute	34
Tabelle 13 Semantik der Attribute	35
Tabelle 14 Beispiele für Transformationen	45
Tabelle 15 Vordefinierte Anfragen für Fact_Table_1	48
Tabelle 16 Vordefinierte Anfragen für Fact_Table_2	49
Tabelle 17 Vordefinierte Anfragen für Fact_Table_3	49
Tabelle 18 Beispiele für Cross-Over Operationen	50
Tabelle 19 Sämtliche Fakten einschließlich möglicher Aggregationsfunktionen	50
Tabelle 20 Quell- und Zieltabellen des Sub-ETL-Prozesses für Event Dimension	56
Tabelle 21 Quell- und Zieltabellen des Sub-ETL-Prozesses für Fact_Table_1	59
Tabelle 22 Definierte Tabellen zur Konsistenzsicherung	65

Stored Procedures

1. Füllen der Dimension Time

```
create procedure KOZASAS.dimension_time ()
begin
declare Whole_t time default '00:00:00';
declare par_h integer default 0;
declare par_m integer default 0;
declare par_hour integer;
declare par_minute integer;
while par_h < 24 do
  set par_hour = hour (Whole_t);
  while par_m < 60 do
    set par_minute = minute (Whole_t) ;
    insert into KOZASAS.time_dimension (TIME_ID, HOUR_T, MINUTE_T , DAYTIME, TIME_D)
      VALUES (NEXTVAL FOR KOZASAS.seq_time_id, par_hour, par_minute,
CASE WHEN 0 < par_h AND par_h < 7 THEN 'MIDNIGHT'
  WHEN 6 < par_h AND par_h <12 THEN 'MORNING'
        WHEN 11 < par_h AND par_h < 16 THEN 'NOON'
        WHEN 15 < par_h AND par_h < 19 THEN 'AFTERNOON'
        WHEN 18 < par_h AND par_h < 24 THEN 'NIGHT'
        ELSE 'MIDNIGHT'
      END, Whole_t);
    set par_m = par_m + 1 ;
    set Whole_t = (Whole_t + 1 minute) ;
  end while;
set par_m = 0;
set par_h = par_h + 1;
end while;
end;
call KOZASAS.dimension_time ();
```

2. Füllen der Dimension Date

```
create procedure KOZASAS.date_dimension ()
begin
declare par_1 date;
declare par_2 date;

SET par_1 = (SELECT COALESCE(MAX (KOZASAS.DATE_DIMENSION_1.DATE_D), '31.12.2004') from KOZASAS.DATE_DIMENSION_1);
SET par_2 = (SELECT (CURRENT DATE) FROM SYSIBM.SYSDUMMY1);

IF par_1 < par_2 then
set par_1 = par_1 + 1 DAY;
While Par_1 < Par_2 or par_1 = par_2 Do
  INSERT into kozasas.DATE_DIMENSION_1 ( DAY , WEEKDAY, WEEKDAY_ID , WEEK, MONTH,
  MONTH_ID, QUARTER, QUARTER_ID , YEAR, SEASON ,DATE_D)
  VALUES ( day(par_1), dayname(par_1), dayofweek(par_1), week (par_1), monthname(par_1) ,
  month(par_1),
CASE WHEN month(par_1)> 0 AND month(par_1) < 4 THEN 'QUARTER_1'
  WHEN month(par_1)> 3 AND month(par_1)< 7 THEN 'QUARTER_2'
  WHEN month(par_1)> 6 AND month(par_1) < 10 THEN 'QUARTER_3'
  WHEN month(par_1)> 10 AND month(par_1) < 13 THEN 'QUARTER_4'
  END, CASE WHEN month(par_1)> 0 AND month(par_1)< 4 THEN 1
  WHEN month(par_1)> 3 AND month(par_1) < 7 THEN 2
  WHEN month(par_1)> 6 AND month(par_1) < 10 THEN 3
  WHEN month(par_1)> 10 AND month(par_1) < 13 THEN 4
  END, year(par_1),
CASE WHEN month(par_1)> 0 AND month(par_1)< 4 THEN 'WINTER'
  WHEN month(par_1)> 3 AND month(par_1) < 7 THEN 'SPRING'
  WHEN month(par_1)> 6 AND month(par_1) < 10 THEN 'SUMMER'
  WHEN month(par_1)> 10 AND month(par_1) < 13 THEN 'AUTUMN'
  END , par_1);
SET par_1 = par_1 + 1 DAY;
END WHILE;
end if;
END;
```

3. Berechnung von Duration

```
create procedure KOZASAS.duration_fact_1 ()

begin
declare par_1 integer ; -- entspricht erster Ett_id
declare par_2 integer ; -- entspricht Anzahl neuer Tupel
declare par_3 integer default 1 ; -- entspricht Laufvariabel
declare par_4 varchar (30) ; -- entspricht erstem Event_Name
declare par_5 timestamp ; -- entspricht Zeitstempel des ersten Events
declare par_6 integer; -- entspricht zweiter Ett_id
declare par_7 varchar (30) ; --entspricht zweitem Event_Namen
declare par_8 timestamp ; -- entspricht Zeitstempel des zweiten Events
declare par_9 varchar (100) ; -- entspricht P_P_ID des ersten Events
declare par_10 varchar (100) ; -- entspricht P_P_ID des zweiten Events
declare par_11 varchar (30) ; -- entspricht PS_Namen des ersten Events
declare par_12 varchar (30) ; -- entspricht PS_Namen des zweiten Events
declare par_13 integer ; -- entspricht der Dauer zwischen zwei Events
declare par_time time ; -- entspricht der Zeit, aber nur eines Start Events
declare par_date date ; -- entspricht dem Datum, aber nur eines Start Events

set par_1 = ( select min (kozasas.fact_table_1_part_1.ett_id) from kozasas.fact_table_1_part_1 );
set par_2 = ( select (count (kozasas.fact_table_1_part_1.ett_id) + 1) from kozasas.fact_table_1_part_1 );

while par_3 < par_2 do
    set par_4 = (select kozasas.fact_table_1_part_1.event_name from kozasas.fact_table_1_part_1
    where kozasas.fact_table_1_part_1.ett_id = par_1 );
    set par_5 = (select kozasas.fact_table_1_part_1.ett_timestamp from kozasas.fact_table_1_part_1
    where kozasas.fact_table_1_part_1.ett_id = par_1);
    set par_time = ( select time(kozasas.fact_table_1_part_1.ett_timestamp) from ko-
    zasas.fact_table_1_part_1
    where kozasas.fact_table_1_part_1.ett_id = par_1 ) ;
    set par_date = ( select date(kozasas.fact_table_1_part_1.ett_timestamp) from ko-
    zasas.fact_table_1_part_1 where kozasas.fact_table_1_part_1.ett_id = par_1 );
    set par_6 = ( par_1 + 1 ) ;
    set par_7 = (select kozasas.fact_table_1_part_1.event_name from kozasas.fact_table_1_part_1
    where kozasas.fact_table_1_part_1.ett_id = par_6 );
    set par_8 = ( select kozasas.fact_table_1_part_1.ett_timestamp from kozasas.fact_table_1_part_1
    where kozasas.fact_table_1_part_1.ett_id = par_6 );
    set par_9 = ( select kozasas.fact_table_1_part_1.propagation_process_id from ko-
    zasas.fact_table_1_part_1 where kozasas.fact_table_1_part_1.ett_id = par_1 );
    set par_10 = ( select kozasas.fact_table_1_part_1.propagation_process_id from ko-
    zasas.fact_table_1_part_1 where kozasas.fact_table_1_part_1.ett_id = par_6 );
    set par_11 = ( select kozasas.fact_table_1_part_1.ps_name from kozasas.fact_table_1_part_1
    where kozasas.fact_table_1_part_1.ett_id = par_1 );
    set par_12 = ( select kozasas.fact_table_1_part_1.ps_name from kozasas.fact_table_1_part_1
    where kozasas.fact_table_1_part_1.ett_id = par_6 );
    set par_13 = microsecond (par_8 - par_5);

IF par_11 = par_12 THEN
    IF par_9 = par_10 THEN
        INSERT INTO KOZASAS.fact_table_1_part_2 (event_id_first, event_id_next,
        ps_name,propagation_process_id, event_name_first, event_name_next, duration, date_f , time_f)
        VALUES (par_1, par_6, par_11, par_9, par_4, par_7, par_13,
        case when par_4 = 'START' THEN par_date
            else par_date END ,
        case when par_4 = 'START' THEN par_time
            else par_time END );
    end if;
end if;
set par_1 = par_6;
set par_6 = par_6 + 1;
set par_3 = par_3 + 1;
end while;
end;
```

Schemata

1. Schema zu PMLOGDB (Operational Source System)

```
-----  
-- DDL-Anweisungen für Tabelle "KOZASAS"."LAST_PS_ETT_ID_STAGING_AREA"  
-----
```

```
CREATE TABLE "KOZASAS"."LAST_PS_ETT_ID_STAGING_AREA" (  
  "LAST_ETT_ID" INTEGER NOT NULL )  
IN "USERSPACE1" ;
```

```
-----  
-- DDL-Anweisungen für Tabelle "KOZASAS"."LAST_ETT_ID_CURRENT_LOAD"  
-----
```

```
CREATE TABLE "KOZASAS"."LAST_ETT_ID_CURRENT_LOAD" (  
  "LAST_CURRENT_ETT_ID" INTEGER NOT NULL )  
IN "USERSPACE1" ;
```

```
-----  
-- DDL-Anweisungen für Tabelle "KOZASAS"."LAST_SOURCE_ETT_ID_STAGING_AREA"  
-----
```

```
CREATE TABLE "KOZASAS"."LAST_SOURCE_ETT_ID_STAGING_AREA" (  
  "LAST_ETT_ID" INTEGER )  
IN "USERSPACE1" ;
```

```
-----  
-- DDL-Anweisungen für Tabelle "KOZASAS"."LAST_DEST_ETT_ID_STAGING_AREA"  
-----
```

```
CREATE TABLE "KOZASAS"."LAST_DEST_ETT_ID_STAGING_AREA" (  
  "LAST_ETT_ID" INTEGER )  
IN "USERSPACE1" ;
```

```
-----  
-- DDL-Anweisungen für Tabelle "KOZASAS"."LAST_LOADED_PS_ETT_ID"  
-----
```

```
CREATE TABLE "KOZASAS"."LAST_LOADED_PS_ETT_ID" (  
  "LOAD_DATE" DATE ,  
  "FIRST_ETT_ID" INTEGER ,  
  "LAST_ETT_ID" INTEGER ) IN "USERSPACE1" ;
```

2. Schema zu STAGE_DB (Data Staging Area)

```
-----  
-- DDL-Anweisungen für Tabelle "KOZASAS"."DESTINATION_ID_1"  
-----
```

```
CREATE TABLE "KOZASAS"."DESTINATION_ID_1" (  
  "DESTINATION_SYSTEM" VARCHAR(30) NOT NULL ,  
  "EVT_ID" INTEGER NOT NULL ,  
  "EVD_ID" INTEGER NOT NULL ,  
  "ETT_ID" INTEGER NOT NULL ,  
  "DESCRIPTION" VARCHAR(200) NOT NULL ) IN "USERSPACE1" ;
```

```
-----  
-- DDL-Anweisungen für Tabelle "KOZASAS"."DESTINATION_ID_2"  
-----
```

```
CREATE TABLE "KOZASAS"."DESTINATION_ID_2" (  
  "SCHEMA" VARCHAR(30) NOT NULL ,  
  "DESCRIPTION" VARCHAR(200) NOT NULL ,  
  "EVT_ID" INTEGER NOT NULL ,  
  "EVD_ID" INTEGER NOT NULL ,  
  "ETT_ID" INTEGER NOT NULL )  
IN "USERSPACE1" ;
```

```
-----  
-- DDL-Anweisungen für Tabelle "KOZASAS"."DESTINATION_ID_3"  
-----
```

```
CREATE TABLE "KOZASAS"."DESTINATION_ID_3" (  
  "DESTINATION_SYSTEM" VARCHAR(30) NOT NULL ,  
  "DESTINATION_SCHEMA" VARCHAR(30) NOT NULL ,  
  "DESCRIPTION_SYSTEM" VARCHAR(200) NOT NULL ,  
  "DESCRIPTION_SCHEMA" VARCHAR(200) NOT NULL )  
IN "USERSPACE1" ;
```

```
-----  
-- DDL-Anweisungen für Tabelle "KOZASAS"."DESTINATION_SURROGATE_ID"  
-----
```

```
CREATE TABLE "KOZASAS"."DESTINATION_SURROGATE_ID" (  
  "SURROGATE_DESTINATION_ID" INTEGER NOT NULL ,  
  "DESTINATION_SYSTEM" VARCHAR(30) NOT NULL ,  
  "DESTINATION_SCHEMA" VARCHAR(30) NOT NULL )  
IN "USERSPACE1" ;
```

```
-----  
-- DDL-Anweisungen für Tabelle "KOZASAS"."DIMENSION_DATE"  
-----
```

```
CREATE TABLE "KOZASAS"."DIMENSION_DATE" (  
  "DATE_ID" INTEGER NOT NULL ,  
  "DAY" INTEGER NOT NULL ,  
  "WEEKDAY" VARCHAR(20) NOT NULL ,  
  "WEEKDAY_ID" INTEGER NOT NULL ,  
  "WEEK" INTEGER NOT NULL ,  
  "MONTH" VARCHAR(20) NOT NULL ,  
  "MONTH_ID" INTEGER NOT NULL ,  
  "QUARTER" VARCHAR(20) NOT NULL ,  
  "QUARTER_ID" INTEGER NOT NULL ,  
  "YEAR" INTEGER NOT NULL ,  
  "SEASON" VARCHAR(20) NOT NULL ,  
  "DATE_D" DATE NOT NULL ) IN "USERSPACE1" ;
```

```
-----  
-- DDL-Anweisungen für Tabelle "KOZASAS"."DETAIL_DIMENSION"  
-----
```

```
CREATE TABLE "KOZASAS"."DETAIL_DIMENSION" (  
  "SURROGATE_ID" INTEGER NOT NULL ,  
  "SOURCE_DETAIL_ID" INTEGER NOT NULL ,  
  "DETAIL" VARCHAR(30) NOT NULL ,  
  "DETAIL_DESCRIPTION" VARCHAR(200) NOT NULL ,  
  "DATA_TYPE" VARCHAR(20) NOT NULL ,  
  "STATUS" VARCHAR(20) NOT NULL ) IN "USERSPACE1" ;
```

```
-----  
-- DDL-Anweisungen für Tabelle "KOZASAS"."DESTINATION_DIMENSION"  
-----
```

```
CREATE TABLE "KOZASAS"."DESTINATION_DIMENSION" (  
  "DESTINATION_SURROGATE_ID" INTEGER NOT NULL ,  
  "DESTINATION_SYSTEM" VARCHAR(30) NOT NULL ,  
  "DESTINATION_SCHEMA" VARCHAR(30) NOT NULL ,  
  "DESCRIPTION_SYSTEM" VARCHAR(200) NOT NULL ,  
  "DESCRIPTION_SCHEMA" VARCHAR(200) NOT NULL )  
IN "USERSPACE1" ;
```

```
-----  
-- DDL-Anweisungen für Tabelle "KOZASAS"."DATE_DIMENSION_1"  
-----
```

```
CREATE TABLE "KOZASAS"."DATE_DIMENSION_1" (  
  "DAY" INTEGER ,  
  "WEEKDAY" VARCHAR(20) ,  
  "WEEKDAY_ID" INTEGER ,  
  "WEEK" INTEGER ,  
  "MONTH" VARCHAR(20) ,  
  "MONTH_ID" INTEGER ,  
  "QUARTER" VARCHAR(20) ,  
  "QUARTER_ID" INTEGER ,  
  "YEAR" INTEGER ,  
  "SEASON" VARCHAR(20) ,  
  "DATE_D" DATE NOT NULL ) IN "USERSPACE1" ;
```

```
-----  
-- DDL-Anweisungen für Tabelle "KOZASAS"."EVENT_DIMENSION"  
-----
```

```
CREATE TABLE "KOZASAS"."EVENT_DIMENSION" (  
  "EVENT_ID" INTEGER NOT NULL ,  
  "EVENT_NAME" VARCHAR(30) NOT NULL ,  
  "EVENT_DESCRIPTION" VARCHAR(200) NOT NULL ,  
  "EVENT_TYPE" VARCHAR(30) NOT NULL ,  
  "STATUS" VARCHAR(20) NOT NULL ,  
  "FATHER_ID" INTEGER ,  
  "FATHER_NAME" VARCHAR(30) ,  
  "SOURCE_EVT_ID" INTEGER NOT NULL ) IN "USERSPACE1" ;
```

```
-----  
-- DDL-Anweisungen für Tabelle "KOZASAS"."ETT_ID_LAST_STAGE_DB"  
-----
```

```
CREATE TABLE "KOZASAS"."ETT_ID_LAST_STAGE_DB" (  
  "LAST_ETT_ID" INTEGER NOT NULL ) IN "USERSPACE1" ;
```

```
-----  
-- DDL-Anweisungen für Tabelle "KOZASAS"."FACT_TABLE_1_PART_1"  
-----
```

```
CREATE TABLE "KOZASAS"."FACT_TABLE_1_PART_1" (  
  "PS_NAME" VARCHAR(30) ,  
  "PROPAGATION_PROCESS_ID" VARCHAR(100) ,  
  "ETT_ID" INTEGER NOT NULL ,  
  "EVT_ID" INTEGER NOT NULL ,  
  "EVENT_NAME" VARCHAR(30) NOT NULL ,  
  "ETT_TIMESTAMP" TIMESTAMP NOT NULL ) IN "USERSPACE1" ;
```

```
-----  
-- DDL-Anweisungen für Tabelle "KOZASAS"."FACT_TABLE_1_PART_2"  
-----
```

```
CREATE TABLE "KOZASAS"."FACT_TABLE_1_PART_2" (  
  "EVENT_ID_FIRST" INTEGER NOT NULL ,  
  "EVENT_ID_NEXT" INTEGER NOT NULL ,  
  "PS_NAME" VARCHAR(30) ,  
  "PROPAGATION_PROCESS_ID" VARCHAR(100) ,  
  "EVENT_NAME_FIRST" VARCHAR(30) ,  
  "EVENT_NAME_NEXT" VARCHAR(30) ,  
  "DURATION" INTEGER ,  
  "DATE_F" DATE ,  
  "TIME_F" TIME )  
IN "USERSPACE1" ;
```

```
-----  
-- DDL-Anweisungen für Tabelle "KOZASAS"."FACT_TABLE_1"  
-----
```

```
CREATE TABLE "KOZASAS"."FACT_TABLE_1" (  
  "PS_ID" INTEGER NOT NULL ,  
  "EVENT_ID_FIRST" INTEGER NOT NULL ,  
  "EVENT_ID_NEXT" INTEGER NOT NULL ,  
  "TIME_ID" INTEGER NOT NULL ,  
  "DATE_ID" INTEGER NOT NULL ,  
  "DURATION" INTEGER NOT NULL ,  
  "NUMBER_OF_PROCESS" INTEGER NOT NULL ,  
  "NUMBER_OF_RESTART" INTEGER NOT NULL ,  
  "PROPAGATION_PROCESS_ID" VARCHAR(100) NOT NULL )  
IN "USERSPACE1" ;
```

```
-----  
-- DDL-Anweisungen für Tabelle "KOZASAS"."LOAD_DESTINATION_INFORMATION_1"  
-----
```

```
CREATE TABLE "KOZASAS"."LOAD_DESTINATION_INFORMATION_1" (  
  "ETT_ID" INTEGER NOT NULL ,  
  "LOAD_DATE" TIMESTAMP NOT NULL )  
IN "USERSPACE1" ;
```

```
-----  
-- DDL-Anweisungen für Tabelle "KOZASAS"."LOAD_DESTINATION_INFORMATION_2"  
-----
```

```
CREATE TABLE "KOZASAS"."LOAD_DESTINATION_INFORMATION_2" (  
  "ETT_ID" INTEGER NOT NULL ,  
  "LOAD_DATE" TIMESTAMP NOT NULL )  
IN "USERSPACE1" ;
```

```
-----  
-- DDL-Anweisungen für Tabelle "KOZASAS"."LOAD_DESTINATION_INFORMATION_3"  
-----
```

```
CREATE TABLE "KOZASAS"."LOAD_DESTINATION_INFORMATION_3" (  
  "ETT_ID" INTEGER NOT NULL ,  
  "LOAD_DATE" TIMESTAMP NOT NULL )  
IN "USERSPACE1" ;
```

```
-----  
-- DDL-Anweisungen für Tabelle "KOZASAS"."LOAD_DESTINATION_INFORMATION"  
-----
```

```
CREATE TABLE "KOZASAS"."LOAD_DESTINATION_INFORMATION" (  
  "LOAD_DATE" TIMESTAMP NOT NULL ,  
  "FIRST_ETT_ID" INTEGER NOT NULL ,  
  "LAST_ETT_ID" INTEGER NOT NULL )  
IN "USERSPACE1" ;
```

```
-----  
-- DDL-Anweisungen für Tabelle "KOZASAS"."FACT_TABLE_2_PART_2"  
-----
```

```
CREATE TABLE "KOZASAS"."FACT_TABLE_2_PART_2" (  
  "PS_NAME" VARCHAR(30) NOT NULL ,  
  "PROPAGATION_PROCESS_ID" VARCHAR(100) NOT NULL ,  
  "DESTINATION_SYSTEM" VARCHAR(50) NOT NULL ,  
  "DATE_D" DATE NOT NULL ,  
  "TIME_D" TIME NOT NULL ,  
  "ETT_ID" INTEGER NOT NULL ) IN "USERSPACE1" ;
```

```
-----  
-- DDL-Anweisungen für Tabelle "KOZASAS"."FACT_TABLE_2_PART_1"  
-----
```

```
CREATE TABLE "KOZASAS"."FACT_TABLE_2_PART_1" (  
  "PS_NAME" VARCHAR(30) NOT NULL ,  
  "PROPAGATION_PROCESS_ID" VARCHAR(100) NOT NULL ,  
  "SOURCE_SYSTEM" VARCHAR(50) NOT NULL ,  
  "DATE_S" DATE NOT NULL ,  
  "TIME_S" TIME NOT NULL ,  
  "ETT_ID" INTEGER NOT NULL ) IN "USERSPACE1" ;
```

```
-----  
-- DDL-Anweisungen für Tabelle "KOZASAS"."FACT_TABLE_JOIN_2"  
-----
```

```
CREATE TABLE "KOZASAS"."FACT_TABLE_JOIN_2" (  
  "PS_NAME" VARCHAR(30) NOT NULL ,  
  "DATE_F" DATE NOT NULL ,  
  "TIME_F" TIME NOT NULL ,  
  "SOURCE_SYSTEM" VARCHAR(50) NOT NULL ,  
  "DESTINATION_SYSTEM" VARCHAR(50) NOT NULL ,  
  "PROPAGATION_PROCESS_ID" VARCHAR(100) NOT NULL ) IN "USERSPACE1" ;
```

```
-----  
-- DDL-Anweisungen für Tabelle "KOZASAS"."FACT_TABLE_3_PART_1"  
-----
```

```
CREATE TABLE "KOZASAS"."FACT_TABLE_3_PART_1" (  
  "ETT_PROPAGATION_PROCESS_ID" VARCHAR(100) NOT NULL ,  
  "DETAIL_NAME" VARCHAR(30) NOT NULL ,  
  "EVENT_NAME" VARCHAR(30) NOT NULL ,  
  "DETAIL_VALUE" VARCHAR(30) NOT NULL ,  
  "ETT_PROPAGATION_SCRIPT_NAME" VARCHAR(30) NOT NULL ,  
  "TIME_F" TIME NOT NULL ,  
  "DATE_F" DATE NOT NULL ,  
  "ETT_ID" INTEGER NOT NULL ,  
  "EVD_ID" INTEGER NOT NULL ,  
  "EVT_ID" INTEGER NOT NULL ) IN "USERSPACE1" ;
```

-- DDL-Anweisungen für Tabelle "KOZASAS"."FACT_TABLE_1_PART_3"

```
CREATE TABLE "KOZASAS"."FACT_TABLE_1_PART_3" (  
  "PS_ID" INTEGER NOT NULL ,  
  "EVENT_ID_FIRST" INTEGER NOT NULL ,  
  "TIME_ID" INTEGER NOT NULL ,  
  "DATE_ID" INTEGER NOT NULL ,  
  "DURATION" INTEGER NOT NULL ,  
  "PROPAGATION_PROCESS_ID" VARCHAR(100) NOT NULL ,  
  "ETT_ID_1" INTEGER NOT NULL ,  
  "ETT_ID_2" INTEGER NOT NULL )  
IN "USERSPACE1" ;
```

-- DDL-Anweisungen für Tabelle "KOZASAS"."FACT_TABLE_1_PART_4"

```
CREATE TABLE "KOZASAS"."FACT_TABLE_1_PART_4" (  
  "PS_ID" INTEGER NOT NULL ,  
  "EVENT_ID_NEXT" INTEGER NOT NULL ,  
  "TIME_ID" INTEGER NOT NULL ,  
  "DATE_ID" INTEGER NOT NULL ,  
  "DURATION" INTEGER NOT NULL ,  
  "PROPAGATION_PROCESS_ID" VARCHAR(100) NOT NULL ,  
  "ETT_ID_1" INTEGER NOT NULL ,  
  "ETT_ID_2" INTEGER NOT NULL )  
IN "USERSPACE1" ;
```

-- DDL-Anweisungen für Tabelle "KOZASAS"."FACT_TABLE_2"

```
CREATE TABLE "KOZASAS"."FACT_TABLE_2" (  
  "TIME_ID" INTEGER NOT NULL ,  
  "DATE_ID" INTEGER NOT NULL ,  
  "PS_ID" INTEGER NOT NULL ,  
  "DESTINATION_ID" INTEGER NOT NULL ,  
  "SOURCE_ID" INTEGER NOT NULL ,  
  "CONNECTION" INTEGER NOT NULL ,  
  "PROPAGATION_PROCESS_ID" VARCHAR(100) NOT NULL )  
IN "USERSPACE1" ;
```

-- DDL-Anweisungen für Tabelle "KOZASAS"."FACT_TABLE_3"

```
CREATE TABLE "KOZASAS"."FACT_TABLE_3" (  
  "TIME_ID" INTEGER NOT NULL ,  
  "DATE_ID" INTEGER NOT NULL ,  
  "PS_ID" INTEGER NOT NULL ,  
  "DETAIL_ID" INTEGER NOT NULL ,  
  "EVENT_ID" INTEGER NOT NULL ,  
  "PROPAGATION_PROCESS_ID" VARCHAR(100) NOT NULL ,  
  "VALUE" VARCHAR(50) NOT NULL )  
IN "USERSPACE1" ;
```

```
-----  
-- DDL-Anweisungen für Tabelle "KOZASAS"."LAST_LOADED_DATE"  
-----
```

```
CREATE TABLE "KOZASAS"."LAST_LOADED_DATE" (  
  "LAST_DATE" DATE )  
IN "USERSPACE1" ;
```

```
-----  
-- DDL-Anweisungen für Tabelle "KOZASAS"."LOAD_SOURCE_INFORMATION_1"  
-----
```

```
CREATE TABLE "KOZASAS"."LOAD_SOURCE_INFORMATION_1" (  
  "ETT_ID" INTEGER NOT NULL ,  
  "LOAD_DATE" TIMESTAMP NOT NULL )  
IN "USERSPACE1" ;
```

```
-----  
-- DDL-Anweisungen für Tabelle "KOZASAS"."LOAD_PS_INFORMATION_1"  
-----
```

```
CREATE TABLE "KOZASAS"."LOAD_PS_INFORMATION_1" (  
  "LOAD_DATE" DATE NOT NULL ,  
  "LAST_ETT_ID" INTEGER NOT NULL )  
IN "USERSPACE1" ;
```

```
-----  
-- DDL-Anweisungen für Tabelle "KOZASAS"."LOAD_PS_INFORMATION"  
-----
```

```
CREATE TABLE "KOZASAS"."LOAD_PS_INFORMATION" (  
  "LOAD_DATE" DATE NOT NULL ,  
  "FIRST_ETT_ID" INTEGER NOT NULL ,  
  "LAST_ETT_ID" INTEGER NOT NULL )  
IN "USERSPACE1" ;
```

```
-----  
-- DDL-Anweisungen für Tabelle "KOZASAS"."SOURCE_DETAIL_ID"  
-----
```

```
CREATE TABLE "KOZASAS"."SOURCE_DETAIL_ID" (  
  "DETAIL" VARCHAR(30) NOT NULL ,  
  "DETAIL_ID" INTEGER NOT NULL ,  
  "DESCRIPTION" VARCHAR(200) NOT NULL ,  
  "DATA_TYPE" VARCHAR(30) NOT NULL )  
IN "USERSPACE1" ;
```

```
-----  
-- DDL-Anweisungen für Tabelle "KOZASAS"."PS_DIMENSION"  
-----
```

```
CREATE TABLE "KOZASAS"."PS_DIMENSION" (  
  "PS_NAME" VARCHAR(30) NOT NULL ,  
  "PS_SURROGATE_ID" INTEGER NOT NULL ,  
  "NUMBER_OF_SOURCES" INTEGER NOT NULL ,  
  "NUMBER_OF_DESTINATION" INTEGER NOT NULL ,  
  "RELATIONSHIP" VARCHAR(20) NOT NULL )  
IN "USERSPACE1" ;
```

```
-----  
-- DDL-Anweisungen für Tabelle "KOZASAS"."SOURCE_DIMENSION"  
-----
```

```
CREATE TABLE "KOZASAS"."SOURCE_DIMENSION" (  
  "SOURCE_SURROGATE_ID" INTEGER NOT NULL ,  
  "SOURCE_SYSTEM" VARCHAR(30) NOT NULL ,  
  "DESCRIPTION_SYSTEM" VARCHAR(200) NOT NULL ,  
  "SOURCE_SCHEMA" VARCHAR(30) NOT NULL ,  
  "DESCRIPTION_SCHEMA" VARCHAR(200) NOT NULL )  
IN "USERSPACE1" ;
```

```
-----  
-- DDL-Anweisungen für Tabelle "KOZASAS"."NUMBER_OF_DESTINATION"  
-----
```

```
CREATE TABLE "KOZASAS"."NUMBER_OF_DESTINATION" (  
  "NUMBER_OF_DESTINATION" INTEGER NOT NULL ,  
  "PS_NAME" VARCHAR(20) NOT NULL )  
IN "USERSPACE1" ;
```

```
-----  
-- DDL-Anweisungen für Tabelle "KOZASAS"."NUMBER_OF_SOURCES"  
-----
```

```
CREATE TABLE "KOZASAS"."NUMBER_OF_SOURCES" (  
  "NUMBER_OF_SOURCES" INTEGER NOT NULL ,  
  "PS_NAME" VARCHAR(20) NOT NULL )  
IN "USERSPACE1" ;
```

```
-----  
-- DDL-Anweisungen für Tabelle "KOZASAS"."SOURCE_ID_1"  
-----
```

```
CREATE TABLE "KOZASAS"."SOURCE_ID_1" (  
  "EVT_ID" INTEGER NOT NULL ,  
  "SOURCE_SYSTEM" VARCHAR(30) NOT NULL ,  
  "EVD_ID" INTEGER NOT NULL ,  
  "ETT_ID" INTEGER NOT NULL ,  
  "DESCRIPTION" VARCHAR(200) NOT NULL )  
IN "USERSPACE1" ;
```

```
-----  
-- DDL-Anweisungen für Tabelle "KOZASAS"."LOAD_SOURCE_INFORMATION_2"  
-----
```

```
CREATE TABLE "KOZASAS"."LOAD_SOURCE_INFORMATION_2" (  
  "ETT_ID" INTEGER NOT NULL ,  
  "LOAD_DATE" TIMESTAMP NOT NULL )  
IN "USERSPACE1" ;
```

```
-----  
-- DDL-Anweisungen für Tabelle "KOZASAS"."LOAD_SOURCE_INFORMATION_3"  
-----
```

```
CREATE TABLE "KOZASAS"."LOAD_SOURCE_INFORMATION_3" (  
  "ETT_ID" INTEGER NOT NULL ,  
  "LOAD_DATE" TIMESTAMP NOT NULL )  
IN "USERSPACE1" ;
```

```
-----  
-- DDL-Anweisungen für Tabelle "KOZASAS"."SOURCE_EVENT_ID"  
-----
```

```
CREATE TABLE "KOZASAS"."SOURCE_EVENT_ID" (  
  "EVENT_NAME" CHAR(30) ,  
  "SOURCE_ID" INTEGER NOT NULL ,  
  "EVENT_DESCRIPTION" CHAR(200) ,  
  "EVENT_TYPE" CHAR(30) ,  
  "FATHER_ID" INTEGER )  
IN "USERSPACE1" ;
```

```
-----  
-- DDL-Anweisungen für Tabelle "KOZASAS"."LOAD_SOURCE_INFORMATION"  
-----
```

```
CREATE TABLE "KOZASAS"."LOAD_SOURCE_INFORMATION" (  
  "LOAD_DATE" TIMESTAMP ,  
  "FIRST_ETT_ID" INTEGER ,  
  "LAST_ETT_ID" INTEGER )  
IN "USERSPACE1" ;
```

```
-----  
-- DDL-Anweisungen für Tabelle "KOZASAS"."TIME_DIMENSION"  
-----
```

```
CREATE TABLE "KOZASAS"."TIME_DIMENSION" (  
  "TIME_ID" INTEGER NOT NULL ,  
  "HOUR_T" INTEGER ,  
  "MINUTE_T" INTEGER ,  
  "DAYTIME" VARCHAR(20) ,  
  "TIME_D" TIME NOT NULL )  
IN "USERSPACE1" ;
```

```
-----  
-- DDL-Anweisungen für Tabelle "KOZASAS"."SURROGATE_DETAIL_ID"  
-----
```

```
CREATE TABLE "KOZASAS"."SURROGATE_DETAIL_ID" (  
  "SOURCE_DETAIL_ID" INTEGER NOT NULL ,  
  "SURROGATE_ID" INTEGER NOT NULL )  
IN "USERSPACE1" ;
```

```
-----  
-- DDL-Anweisungen für Tabelle "KOZASAS"."SOURCE_PS_ID_1"  
-----
```

```
CREATE TABLE "KOZASAS"."SOURCE_PS_ID_1" (  
  "ETT_EVT_ID" INTEGER NOT NULL ,  
  "ETT_ID" INTEGER NOT NULL ,  
  "PS_NAME" VARCHAR(30) NOT NULL )  
IN "USERSPACE1" ;
```

```
-----  
-- DDL-Anweisungen für Tabelle "KOZASAS"."SOURCE_PS_ID_JOIN_1_2"  
-----
```

```
CREATE TABLE "KOZASAS"."SOURCE_PS_ID_JOIN_1_2" (  
  "NUMBER_SOURCES" INTEGER NOT NULL ,  
  "PS_NAME" VARCHAR(30) NOT NULL ,  
  "NUMBER_DESTINATION" INTEGER NOT NULL ,  
  "RELATIONSHIP" VARCHAR(20) NOT NULL ) IN "USERSPACE1" ;
```

```
-----  
-- DDL-Anweisungen für Tabelle "KOZASAS"."SURROGATE_PS_ID"  
-----
```

```
CREATE TABLE "KOZASAS"."SURROGATE_PS_ID" (  
  "PS_NAME" VARCHAR(30) NOT NULL ,  
  "SURROGATE_PS_ID" INTEGER NOT NULL ) IN "USERSPACE1" ;
```

```
-----  
-- DDL-Anweisungen für Tabelle "KOZASAS"."SOURCE_ID_2"  
-----
```

```
CREATE TABLE "KOZASAS"."SOURCE_ID_2" (  
  "DESCRIPTION" VARCHAR(200) NOT NULL ,  
  "SCHEMA" VARCHAR(30) NOT NULL ,  
  "EVT_ID" INTEGER NOT NULL ,  
  "EVD_ID" INTEGER NOT NULL ,  
  "ETT_ID" INTEGER NOT NULL ) IN "USERSPACE1" ;
```

```
-----  
-- DDL-Anweisungen für Tabelle "KOZASAS"."SOURCE_ID_3"  
-----
```

```
CREATE TABLE "KOZASAS"."SOURCE_ID_3" (  
  "SOURCE_DESCRIPTION" VARCHAR(200) NOT NULL ,  
  "SOURCE_SYSTEM" VARCHAR(30) NOT NULL ,  
  "SOURCE_SCHEMA" VARCHAR(30) NOT NULL ,  
  "SCHEMA_DESCRIPTION" VARCHAR(200) NOT NULL ) IN "USERSPACE1" ;
```

```
-----  
-- DDL-Anweisungen für Tabelle "KOZASAS"."SOURCE_SURROGATE_ID"  
-----
```

```
CREATE TABLE "KOZASAS"."SOURCE_SURROGATE_ID" (  
  "SOURCE_SYSTEM" VARCHAR(30) NOT NULL ,  
  "SURROGATE_SOURCE_ID" INTEGER NOT NULL ,  
  "SOURCE_SCHEMA" VARCHAR(30) NOT NULL )  
IN "USERSPACE1" ;
```

```
-----  
-- DDL-Anweisungen für Tabelle "KOZASAS"."SURROGATE_EVENT_ID"  
-----
```

```
CREATE TABLE "KOZASAS"."SURROGATE_EVENT_ID" (  
  "SOURCE_ID" INTEGER NOT NULL ,  
  "SURROGATE_ID" INTEGER NOT NULL ) IN "USERSPACE1" ;
```

3. Schema zu DWLOGDB (Data Presentation Area)

```
-----  
-- DDL-Anweisungen für Tabelle "KOZASAS"."EVENT_DIMENSION"  
-----
```

```
CREATE TABLE "KOZASAS"."EVENT_DIMENSION" (  
  "EVENT_ID" INTEGER NOT NULL ,  
  "SOURCE_EVT_ID" INTEGER NOT NULL ,  
  "EVENT_NAME" CHAR(30) ,  
  "EVENT_DESCRIPTION" CHAR(200) ,  
  "EVENT_TYPE" CHAR(30) ,  
  "STATUS" CHAR(15) ,  
  "FATHER_ID" INTEGER ,  
  "FATHER_NAME" CHAR(30) ) IN "USERSPACE1" ;
```

```
-----  
-- DDL-Anweisungen für Tabelle "KOZASAS"."DIMENSION_DATE"  
-----
```

```
CREATE TABLE "KOZASAS"."DIMENSION_DATE" (  
  "DATE_ID" INTEGER NOT NULL ,  
  "DAY" INTEGER ,  
  "WEEKDAY" VARCHAR(20) ,  
  "WEEKDAY_ID" INTEGER ,  
  "WEEK" INTEGER ,  
  "MONTH" VARCHAR(20) ,  
  "MONTH_ID" INTEGER ,  
  "QUARTER" VARCHAR(20) ,  
  "QUARTER_ID" INTEGER ,  
  "YEAR" INTEGER ,  
  "SEASON" VARCHAR(20) ,  
  "DATE_D" DATE ) IN "USERSPACE1" ;
```

```
-----  
-- DDL-Anweisungen für Tabelle "KOZASAS"."FACT_TABLE_1"  
-----
```

```
CREATE TABLE "KOZASAS"."FACT_TABLE_1" (  
  "PS_ID" INTEGER NOT NULL ,  
  "EVENT_ID_FIRST" INTEGER NOT NULL ,  
  "EVENT_ID_NEXT" INTEGER NOT NULL ,  
  "DATE_ID" INTEGER NOT NULL ,  
  "TIME_ID" INTEGER NOT NULL ,  
  "DURATION" INTEGER NOT NULL ,  
  "NUMBER_OF_PROCESS" INTEGER NOT NULL ,  
  "NUMBER_OF_RESTART" INTEGER NOT NULL ,  
  "PROPAGATION_PROCESS_ID" VARCHAR(100) NOT NULL ) IN "USERSPACE1" ;
```

```
-----  
-- DDL-Anweisungen für Tabelle "KOZASAS"."FACT_TABLE_2"  
-----
```

```
CREATE TABLE "KOZASAS"."FACT_TABLE_2" (  
  "TIME_ID" INTEGER NOT NULL ,  
  "DATE_ID" INTEGER NOT NULL ,  
  "PS_ID" INTEGER NOT NULL ,  
  "DESTINATION_ID" INTEGER NOT NULL ,  
  "SOURCE_ID" INTEGER NOT NULL ,  
  "CONNECTION" INTEGER NOT NULL ,  
  "PROPAGATION_PROCESS_ID" VARCHAR(100) NOT NULL ) IN "USERSPACE1" ;
```

```
-----  
-- DDL-Anweisungen für Tabelle "KOZASAS"."FACT_TABLE_3"  
-----
```

```
CREATE TABLE "KOZASAS"."FACT_TABLE_3" (  
  "TIME_ID" INTEGER NOT NULL ,  
  "DATE_ID" INTEGER NOT NULL ,  
  "PS_ID" INTEGER NOT NULL ,  
  "DETAIL_ID" INTEGER NOT NULL ,  
  "EVENT_ID" INTEGER NOT NULL ,  
  "PROPAGATION_PROCESS_ID" VARCHAR(100) NOT NULL ,  
  "VALUE" VARCHAR(50) NOT NULL ) IN "USERSPACE1" ;
```

```
-----  
-- DDL-Anweisungen für Tabelle "KOZASAS"."DESTINATION_DIMENSION"  
-----
```

```
CREATE TABLE "KOZASAS"."DESTINATION_DIMENSION" (  
  "DESTINATION_ID" INTEGER NOT NULL ,  
  "DESTINATION_SYSTEM" VARCHAR(30) NOT NULL ,  
  "DESCRIPTION_SYSTEM" CHAR(200) NOT NULL ,  
  "DESCRIPTION_SCHEMA" CHAR(200) NOT NULL ,  
  "DESTINATION_SCHEMA" CHAR(30) NOT NULL ) IN "USERSPACE1" ;
```

```
-----  
-- DDL-Anweisungen für Tabelle "KOZASAS"."SOURCE_DIMENSION"  
-----
```

```
CREATE TABLE "KOZASAS"."SOURCE_DIMENSION" (  
  "SOURCE_ID" INTEGER NOT NULL ,  
  "SOURCE_SYSTEM" VARCHAR(30) NOT NULL ,  
  "SOURCE_SCHEMA" VARCHAR(30) NOT NULL ,  
  "DESCRIPTION_SYSTEM" VARCHAR(200) NOT NULL ,  
  "DESCRIPTION_SCHEMA" VARCHAR(200) NOT NULL ) IN "USERSPACE1" ;
```

```
-----  
-- DDL-Anweisungen für Tabelle "KOZASAS"."PS_DIMENSION"  
-----
```

```
CREATE TABLE "KOZASAS"."PS_DIMENSION" (  
  "PS_ID" INTEGER NOT NULL ,  
  "PS_NAME" VARCHAR(30) NOT NULL ,  
  "NUMBER_OF_SOURCES" INTEGER NOT NULL ,  
  "NUMBER_OF_DESTINATION" INTEGER NOT NULL ,  
  "RELATIONSHIP" VARCHAR(30) NOT NULL ) IN "USERSPACE1" ;
```

```
-----  
-- DDL-Anweisungen für Tabelle "KOZASAS"."DETAIL_DIMENSION"  
-----
```

```
CREATE TABLE "KOZASAS"."DETAIL_DIMENSION" (  
  "DETAIL_ID" INTEGER NOT NULL ,  
  "SOURCE_DETAIL_ID" INTEGER NOT NULL ,  
  "DETAIL" VARCHAR(30) NOT NULL ,  
  "DETAIL_DESCRIPTION" VARCHAR(200) NOT NULL ,  
  "DATA_TYPE" VARCHAR(20) NOT NULL ,  
  "STATUS" VARCHAR(20) NOT NULL ) IN "USERSPACE1" ;
```

```
-----  
-- DDL-Anweisungen für Tabelle "KOZASAS"."PROPAGATION_PROCESS_ID"  
-----
```

```
CREATE TABLE "KOZASAS"."PROPAGATION_PROCESS_ID" (  
  "PROPAGATION_PROCESS_ID" VARCHAR(100) NOT NULL ,  
  "PS_NAME" VARCHAR(30) NOT NULL ) IN "USERSPACE1" ;
```