Fachhochschule Köln
Abteilung Gummersbach
Fachbereich Informatik
Studiengang Wirtschaftsinformatik

Diplomarbeit
zur Erlangung
des Diplomgrades
Diplomwirtschaftsinformatiker

ROLAP

Data Warehousing

mit

Oracle und MicroStrategy

eingereicht von:
Andreas Trodtfeld
Matrikelnummer 10 23 17 90
Föhrenweg 39
51491 Overath

Gutachter:
Frau Prof. Dr. Faeskorn-Woyke
Herr Prof. Dr. Victor

Gummersbach, 09. November 1998

0 Inhaltsverzeichnis

1 EINLEITUNG

1.1 Die Information als Wettbewerbsfaktor

Durch die zunehmende Globalisierung der Märkte und dem damit immer härter werdenden Konkurrenzkampf ist es für ein Unternehmen absolut notwendig, die im Unternehmen vorhandenen Datenmengen optimal zu nutzen. Diese Daten ermöglichen es dem Unternehmen das laufende Geschäft besser zu durchleuchten und zu analysieren, um so die Wettbewerbsfähigkeit zu sichern.

Der Umfang, der einem Unternehmen zur Verfügung stehenden Daten, steigt rapide an. Unternehmen, die in der Lage sind diese immer größer werdende Datenmenge effektiv zu verwalten und zur Unterstützung von Geschäftsentscheidungen zu nutzen, werden einen deutlichen Wettbewerbsvorteil vor ihrer Konkurrenz erzielen.

Die Informationen sind in jedem Unternehmen vorhanden. Das Unternehmen muß nur in der Lage sein relevante Informationen herauszufiltern, um auf sich immer schneller ändernde Kundenwünsche und Marktgegebenheiten reagieren zu können. 'Schneller und besser' zu sein als die Konkurrenz ist immer mehr die Devise der heutigen Unternehmen und genau dazu benötigt man schnell und aktuell interne wie auch externe Daten zur Entscheidungsunterstützung.

Effektives Kostenmanagement ist ein weiteres Schlagwort durch das Unternehmen ihre Marktposition verstärken wollen. Unnötige Kosten müssen aufgedeckt und eliminiert werden. Auch das erreicht ein Unternehmen nur, wenn es in der Lage ist alle im Unternehmen vorkommenden Daten zu Informationen aufzubereiten.

Die Notwendigkeit sich ein "gläsernes Unternehmen" zu schaffen wird immer bedeutender. Jede noch so kleine Unternehmensoptimierung ist enorm wichtig, um sich auf dem Markt zu behaupten.

1.2 Der Informationsnutzer

Durch immer flachere Hierarchien im Unternehmen ist die Anzahl der Entscheidungsträger und Datennutzer enorm angestiegen. Durch die immer einfacher zu handhabende Hard- und Software sind es die Endbenutzer gewohnt den PC zur Datenverarbeitung und Entscheidungsunterstützung zu nutzen. Sie wollen direkt, und vor allem selbständig, Daten von ihrem Arbeitsplatz aus abrufen, analysieren und für den jeweiligen Problemfall nutzen.

Mitarbeiter unterschiedlichster Hierarchieebenen nutzen die Datenanalyse als Hilfsmittel für Ihre tägliche Arbeit. Es ist nicht mehr nur der Top-Manager des Unternehmens, der Daten für Entscheidungsprozesse benötigt. Auch Mitarbeiter der unteren Hierarchieebene, wie z.B. Einkäufer, Vertriebsleute, Außendienstler, Personalsachbearbeiter, usw., wollen in ihrer Arbeit durch Berichte und Analysen unterstützt werden.

Die für ihn relevanten Unternehmensdaten müssen für den jeweiligen Endbenutzer schnell und vor allem dynamisch (**Data-Surfing**) zur Verfügung stehen. Dynamisch bedeutet, daß der Entscheidungsträger am Anfang vielleicht noch gar nicht so genau weiß wonach er eigentlich sucht und erst durch das Surfen durch den Datenbestand wichtige Zusammenhänge und Gegebenheiten erkennt und daraus eventuell neue Analysen und Berichte ableitet.

Die klassische Form der Berichtserstellung durch die zentrale DV-Abteilung ist bei weitem nicht so effektiv wie es erforderlich ist. Ein Data-Surfing wie

oben beschrieben, ist aufgrund der zentralen Abarbeitung der Berichtswünsche und der damit verbundenen großen Anzahl von Datenauswertungen kaum zu bewältigen. Es kommt zu sogenannten **Report-Staus**. Endanwender müssen mehrere Tage, oder sogar Wochen, auf die angeforderten Berichte warten, nur um dann festzustellen, daß der Bericht vielleicht doch nicht die gewünschte Sicht bzw. die gewünschten Zusammenhänge auf die Daten ermöglicht.

Analysen und Berichte müssen dort erstellt werden wo der Bedarf nach Information entsteht. Nur so kann man sicherstellen, daß die Entscheidungsträger mit den richtigen Informationen zur richtigen Zeit versorgt werden.

Aufgabe der DV-Abteilung sollte es nur noch sein ein gewisses Daten-Umfeld für den Endbenutzer zu schaffen, aus dem sich dieser selbständig bedienen kann, bzw. selbständig, **ad-hoc** (= zur gewünschten Zeit, in der gewünschten Form) Analysen und Berichte erstellen kann.

1.3 Das operative System als Analyseumgebung

Das **operative System** (also die EDV-gestütze Umgebung zur Abhandlung des Tagesgeschäftes) speichert sämtliche Unternehmensdaten die für die Informationsgewinnung notwendig sind, denn dort entstehen die Unternehmensdaten durch die täglichen Aktivitäten des Unternehmens, wie z. B. durch das Auftragswesen, die Lagerhaltung, das Personalwesen, usw. Es gibt allerdings verschiedene Gründe, warum es weniger sinnvoll ist Analysen und Berichte direkt auf das operative System aufzusetzen:

(1) Die Datenverteilung

Das operative System hat sich mit dem Bestehen des Unternehmens im Laufe der Zeit weiterentwickelt. Neue Systeme, Datenbanken, Applikationen sind hinzugekommen. Die Daten des Unternehmens werden meist am Ort der Entstehung (z. B. in den entsprechenden Abteilungen, in den Filialen) bzw. applikationsabhängig (Datenbank für das Personalwesen, Datenbank für das Auftragswesen, usw.) gespeichert und sind aus diesem Grund im ganzen Unternehmen auf den unterschiedlichsten Speichersystemen verteilt (zentrale Hostsysteme, Datenbanken einzelner Abteilungen, Datenbanken einzelner Applikationen, Festplatten einzelner PC's, archivierte Daten, externe Daten, Datenbanken der einzelnen Filialen, Disketten, usw.).

Aus diesem Grund ist es nahezu unmöglich Analysen, Berichte auf den gesamten Datenbestand des operativen Systems durchzuführen, bzw. Zusammenhänge zwischen Daten aus den einzelnen Bereichen des operativen Systems herzuleiten. Natürlich ist es möglich die Anzahl der Krankmeldungen pro Monat darzustellen. Auch ist es möglich Lieferverzögerungen der einzelnen Aufträge zu ermitteln. Schwieriger wird es jedoch zwischen diesen beiden Berichten pc-gestützt einen Zusammenhang herzustellen, da diese Daten aus unterschiedlichen Systemen kommen und erst in eine Beziehung gesetzt, sprich in geeigneter Form zusammengefaßt werden müssen. Dieser Analysevorgang ist über das operative System als sehr aufwendig zu bezeichnen, geschweige denn innerhalb weniger Minuten zu realisieren.

(2) Integrationsprobleme

Möchte man Analysen auf Daten durchführen, die in unterschiedlichen Applikationen erstellt wurden, so muß man beachten, daß z. B. Kennzeichen, auf die sich die Analyse bezieht, durch die Applikationen unterschiedlich gespeichert sein können.
Beispiel: Das Geschlechtskennzeichen[1]

[1] W. H: Inmon (Building the Data Warehouse, 1996) S. 75

Applikation A	**m, w**
Applikation B	**1, 0**
Applikation C	**x, y**
Applikation D	**männlich, weiblich**

Weiterhin kann es sein, daß z.B. 'Umsätze' in Applikation A eine ganz andere Definition besitzt als in Applikation B. In A sind es die 'Umsätze' in Stck, in B sind es die 'Umsätze' in DM.

Oder aber wir haben die Bezeichnung 'Umsatz' in Applikation A und 'Absatz' in der Applikation B - zwei verschiedene Bezeichnungen, die aber das gleiche meinen: 'Umsatz in DM'.

Gerade die Integration von Daten aus unterschiedlichen Applikationen, bzw. Bereichen des operativen Systems erfordern von dem Endbenutzer eine genaue Kenntnis über die Datenstruktur. Wie sind die Daten abgespeichert, welche Definition verbirgt sich hinter den Tabellen- und Spaltennamen, usw.

D. h., bevor die eigentliche Analyse stattfinden kann, muß der Endbenutzer erst einmal die in Frage kommenden Datenquellen zeitaufwendig untersuchen, um nicht Äpfel mit Birnen zu vergleichen.

(3) Historische Datenhaltung

In dem operativen System werden Daten mit einem Zeithorizont von nur wenigen Monaten gehalten, damit eine zu große Datenmenge das System nicht blockiert und eine effektive Administration verhindert. Nicht aktuelle Daten werden in Archive ausgelagert. D. h., möchte man eine Analyse auf die Umsätze der letzten 5 Jahre starten, so muß man die schon archivierten Daten wieder in das System einspielen, um die Analyse überhaupt durchführen zu können.

Weiterhin müssen **Datenaggregationen** (= Summierung einer Kennzahl über ein bestimmtes Attribut wie z.B. die Zeit) wie der Umsatz pro Monat bei jedem Bericht wieder neu errechnet werden, da diese Aggregation so nicht in dem operativen System vorkommt und gespeichert werden kann (in dem operativen System ist der Umsatz meistens pro Tag bzw. pro Transaktion gespeichert).

(4) Unterschiedliche Auslastung der Hardware

Bei dem Tagesgeschäft werden sehr viele Transaktionen gleichzeitig in dem operativen System angestoßen. Jede dieser Transaktionen betrifft meistens nur eine kleine Menge an Daten und verwendet kaum CPU-intensive Rechenoperationen, wie z. B. die Abfrage einer aktuellen Kundenadresse oder des aktuellen Lagerbestand eines Artikels, so daß die Hardwareauslastung des operativen Systems normalerweise als ausgewogen zu bezeichnen ist.

In Abbildung 1 erkennt man, daß natürlich zu bestimmten Zeiten des Tages die Auslastung des Systems weniger hoch ist, wie z.B. zur Mittagszeit, oder zum Ende des Arbeitstages. Der Auslastungsverlauf wird aber normalerweise jeden Tag gleich aussehen. Vielleicht gibt es noch Unterschiede zum Wochenende hin, bzw. Unterschiede in der Jahreszeit, aber ansonsten ist dieser typische Verlauf als konstant zu bezeichnen.

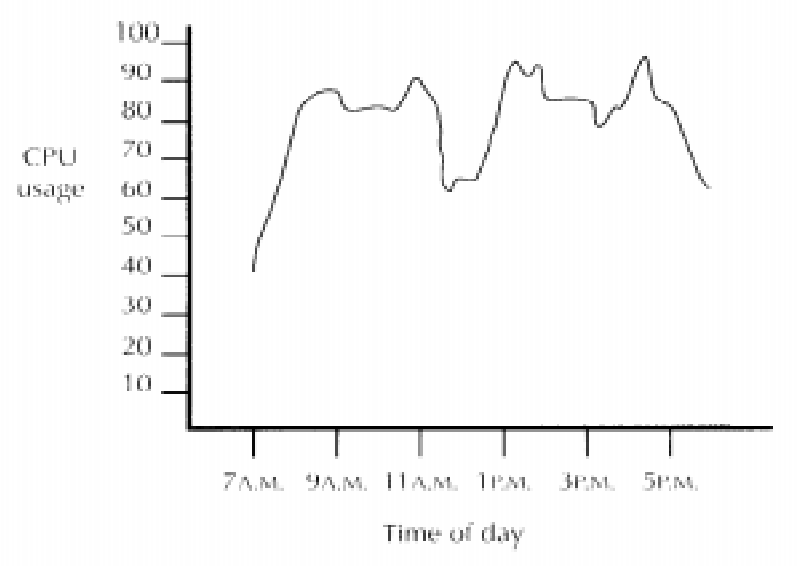


Abbildung 1: Auslastungsverlauf bei Transaktions-Prozessen[2]

Bei analytischen Abfragen werden nur wenige Abfragen gleichzeitig durchgeführt, jedoch betreffen diese Abfragen meistens eine große Datenmenge (gerade bei historischen Analysen über mehrere Jahre) oder aber komplexe Rechenoperationen, wie z.B. die Top 10 der Umsätze pro Produktgruppe, die sehr CPU-intensiv sind.

Die Auslastung der Hardware für solche analytische Prozesse ist als unausgewogen zu bezeichnen. Zeitweise ist die CPU voll ausgelastet und zeitweise gar nicht. Einen täglich gleichen Auslastungsverlauf wie bei der Transaktionsverarbeitung wird es hier nie geben, da die Analyseanforderungen sehr sporadisch verlaufen und somit von Tag zu Tag verschieden sein können.

[2] M. J. Corey (ORACLE Data Warehousing, 1997), S. 9

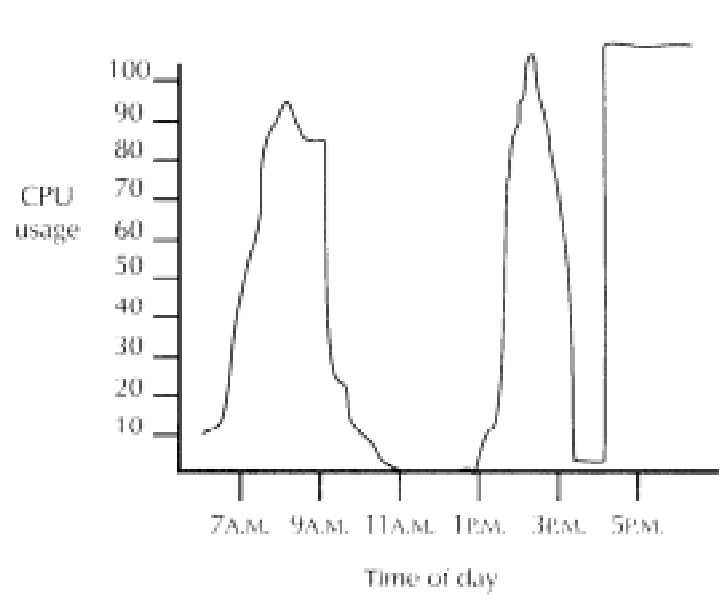


Abbildung 2: Auslastungsverlauf bei Analyse-Prozessen[3]

Möchte man nun zusätzlich zu dem Tagesgeschäft auch noch die Analysen auf dem operativem System durchführen, so muß damit gerechnet werden, daß die Ressourcen diesen Anforderungen nicht mehr gewachsen sind und die reibungslose Durchführung des Tagesgeschäftes gefährdet ist. Geschwindigkeitseinbußen bis hin zu Systemausfällen können die Folge sein.

Eine Optimierung der Hardwareressourcen ist nur in eine Richtung möglich, entweder zum Vorteil der Transaktionsprozesse (und gleichzeitig zum Nachteil der Analyseprozesse), oder umgekehrt.

Schlußfolgerung:

Wenn man sich die oben aufgeführten Einschränkungen bei der Nutzung des operativen Systems als Basis für analytische Auswertungsprozesse betrachtet, so muß man einfach zu dem Entschluß kommen, daß eine andere Architektur, ein getrenntes System notwendig wird, um fehlerfreie Analysen

[3] M. J. Corey (ORACLE Data Warehousing, 1997), S. 10

und Berichte mit schnellen Antwortzeiten zu erstellen, ohne daß das Tagesgeschäft in irgend einer Form eingeschränkt wird. 'Data Warehousing' bietet genau die Lösung zu diesem Problem.

2 Data Warehousing

Nach **W. H. Inmon** (der Ur-Vater des Data Warehousing) definiert sich ein Data Warehouse folgendermaßen:

"Ein Data Warehouse bezeichnet eine themenorientierte, integrierte, dauerhafte und zeitbezogene Sammlung von Informationen zur Entscheidungsunterstützung des Managements."

Unter einem Data Warehouse versteht man also nichts anderes als ein neben dem operativen System bestehenden großen Datenpool, in dem die Unternehmensdaten speziell aufbereitet zur Verfügung gestellt werden. Lieferant der Unternehmensdaten ist das operative System, in dem die Daten des Tagesgeschäftes gespeichert werden, sowie externe Quellen, wie z. B. Nachrichtendienste, Wetterdienste, usw.

Durch sogenannte **Transformations- und Extraktions-Programme** (siehe Kapitel 6.2) werden die Daten in gewünschter Form in das Data Warehouse überführt (Daten-Load). Dort stehen sie für Analysen und Berichte zur Verfügung.

Ein Data Warehouse stellt in erster Linie nur eine Datenbasis in geeigneter Form bereit, auf die mit zusätzlichen Analyse-Tools zugegriffen werden kann. Durch das Data Warehouse wird im Gegensatz zum operativen System eine einzige, zentrale Datenquelle geschaffen, auf die der Endbenutzer zur Analyse- und Berichtserstellung wesentlich einfacher und schneller zugreifen kann.

2.1 Data Warehouse Architekturen

Folgende Data Warehouse Architekturen sind am häufigsten vorzufinden.

2.1.1 Zentrales Data Warehouse

Hauptmerkmal eines zentralen Data Warehouses ist die zentrale Data Warehouse Datenbank die zusätzlich zu den operativen Datenbeständen existiert.

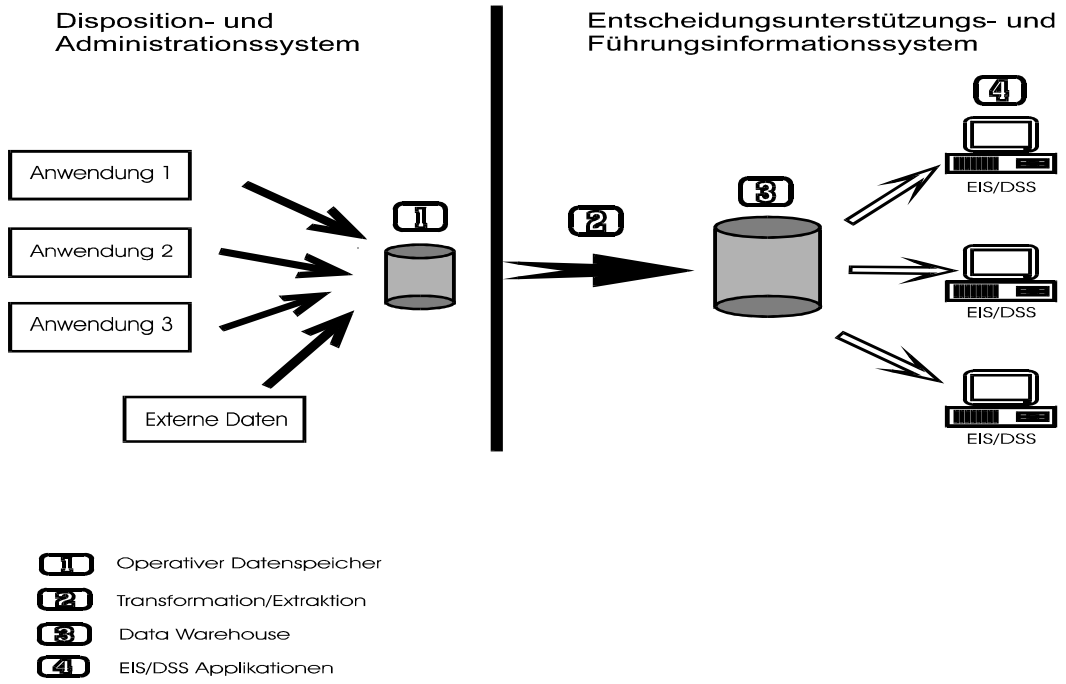


Abbildung 3: Architektur des zentralen Data Warehouses[4]

Durch das Tagesgeschäft entstehen Daten, wie z. B. Lagerbestände, Personaldaten, Verkaufsdaten, usw., die in dem operativen Datenspeicher (1) gesammelt werden. Das zentrale Data Warehouse (3) wird mittels **Extraktion und Transformation** (2) der Daten aus dem operativen System ge-

füllt.Wenn sich die Daten im zentralen Data Warehouse befinden ist es möglich mit zusätzlichen **DSS/EIS Tools** auf den zentralen Datenbestand zuzugreifen und Analysen und Berichte zu erstellen (4).

Bei EIS und DSS Tools handelt es sich um EDV gestützte, interaktive, endbenutzerfreundliche Managementunterstützungs-Werkzeuge.

Mit Hilfe der **DSS** (**D**ecision **S**upport **S**ystem) Tools und dem Datenbestand des Data Warehouses wird der Endbenutzer in komplexen, fachbezogenen Planungs- und Entscheidungssituationen mit Hilfe von Problemlösungsmodellen und -funktionen unterstützt. Das Tool fungiert dabei als eine Art Assistent.

EIS (**E**xecutive **I**nformation **S**ystem) Tools haben sich im Gegensatz zu den DSS Tools auf die Datenpräsentation spezialisiert und liefern weder Modelle noch Funktionen zur Problemlösung, sondern ausschließlich unterschiedlichste Darstellungsarten der zu analysierenden Daten.

2.1.2 Data Marts

Data Marts sind dezentrale Data Warehouse Datenbanken, die im Gegensatz zum zentralen Data Warehouse jeweils nur einen Teilbereich des Gesamtdatenpools bereitstellen. Der Einsatz von Data-Marts kann unterschiedliche Gründe haben[5]:

(1) Data Marts als Data Warehouse Startpunkt

[4] in Anlehnung an H. Mucksch, W. Behme (Das Data Warehouse-Konzept, 1996) S. 94
[5] Schinzer, Bange, Wehner, Zeile (Management mit Maus und Monitor, 1997) S. 23
J. Sweeney, M. Griese (The Role of a Data Mart, Winter 1996) Artikel in 'Inside Decisions'

Bevor mit der Realisierung eines unternehmensweiten, zentralen Data Warehouses begonnen wird, entschließen sich viele Firmen erst einmal mit einem oder mehreren Data Marts zu beginnen. Data Mart Projekte sind wesentlich kostengünstiger und schneller zu realisieren, da hier nur ein Teilbereich der gesamten Unternehmendaten in ein dezentrales Data Warehouse übernommen wird. "***Think big, start small***" ist eine bewährte Devise des Data Warehouse Geschäftes. Der Bereich für den das Data Mart erstellt wird könnte z.B. eine einzelne Abteilung sein, oder aber bei örtlich verteilten Unternehmen, ein einzelner Standort.

Hat man mehrere Data Marts erfolgreich installiert, so macht es eventuell zu einem späteren Zeitpunkt Sinn, die einzelnen Data Marts in ein zentrales Data Warehouse zu überführen, um anstelle mehrerer verteilter Datenpools nur noch einen zentralen Datenpool zu verwalten und zu pflegen. Weiterhin können durch das zentrale Data Warehouse sämtliche Teilbereiche des Unternehmens über nur eine Datenquelle verfügen, anstatt auf eine Vielzahl von Data Marts zugreifen zu müssen.

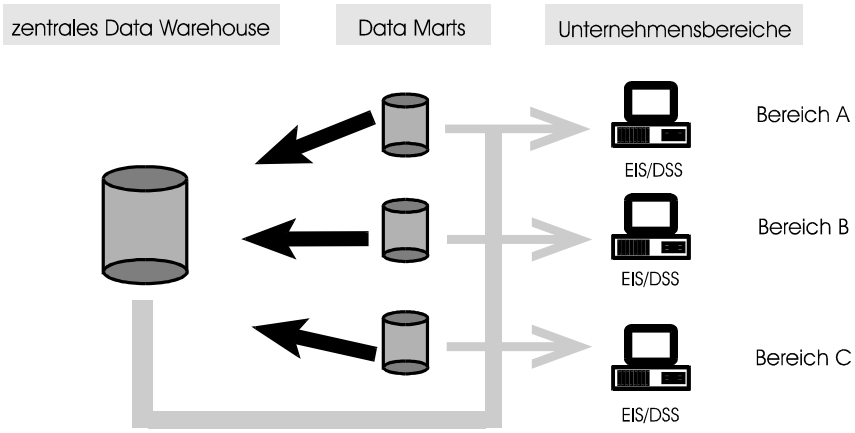


Abbildung 4: Data Mart Architektur als Startpunkt für zentrales Data Warehouse

Gelingen wird dieser Wechsel in die zentrale Data Warehouse Architektur aber nur, wenn schon zu Beginn der Installation der ersten Data Marts ein solcher Wechsel in einem einheitlichen Design der dezentralen Data Warehouses bedacht wurde ("***Think big****, start small*"). Beachtet man die über-

greifende Planung nicht, so besteht die Gefahr der Bildung von Insellösungen, die gar nicht oder nur sehr schwer in eine zentrale Warehouse Architektur zu überführen sind.

(2) Data Marts als schneller Zugriff auf Data Warehouse Daten

Nachdem ein zentrales Data Warehouse erstellt wurde, macht es eventuell Sinn zusätzlich Data Marts in den einzelnen Bereichen des Unternehmens (Abteilungen, Standorte, usw.) zu installieren, um den jeweils notwendigen Teilbereich des zentralen Datenpools näher an den eigentlichen Benutzerkreis zu bringen. Dadurch kann der Zugriff auf die Daten in dem jeweiligen Teilbereich enorm beschleunigt werden.

Die Data Warehouse Daten sind somit an mehreren Stellen verfügbar (komplett im zentralen Datenpool und Teilbereiche des zentralen Datenpools direkt in den dezentralen Data Marts der einzelnen Unternehmensbereiche).

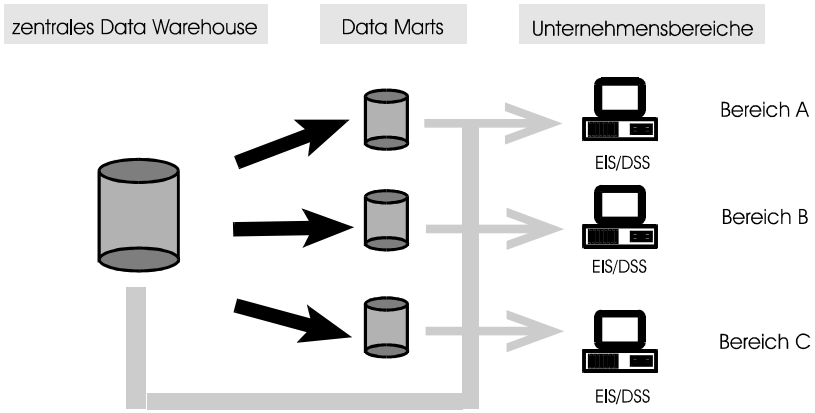


Abbildung 5: Data Mart Architektur zur Beschleunigung des Analysezugriffes

(3) Data Marts für Spezialdaten

Manche Unternehmensbereiche besitzen Daten, die auch wirklich nur für diesen Bereich relevant sind. Es wäre unlogisch auch solche Daten zentral

im Unternehmen zu speichern, da sie sowieso nur von einer begrenzten Benutzerzahl desselben Bereiches genutzt werden.

Auch bei solchen Spezialdaten macht es Sinn, direkt bei dem betroffenen Unternehmensbereich ein Data Mart mit diesen Spezialdaten zu installieren, ohne jedoch diese Daten zusätzlich in einem zentralen Datenpool zu speichern.

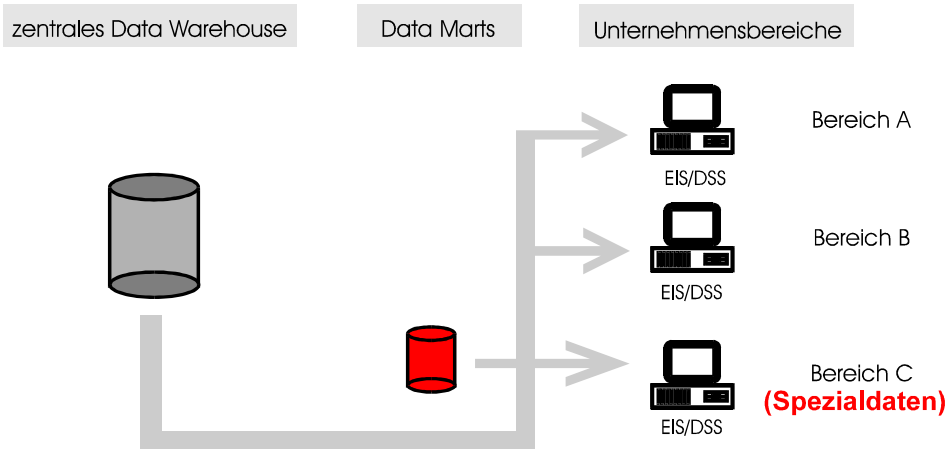


Abbildung 6: Data Mart Architektur für Spezialdaten

2.1.3 Virtuelles Data Warehouse

Bei einem virtuellen Data Warehouse verzichtet man auf eine separate Data Warehouse Datenbank. Analysen und Berichte werden direkt auf das operative System aufgesetzt.
Durch sogenannte Middleware-Programme werden Zugriffe auf das virtuelle Data Warehouse durch direktes Lesen von operationalen Datenbeständen so umgesetzt, daß für Benutzer der Eindruck entsteht direkt auf einem physikalischem Data Warehouse zu arbeiten[6].

[6] P. Mertens (Konzeption für ein Data Warehouse, 01/98), Artikel in 'Datenbank Fokus', S. 55

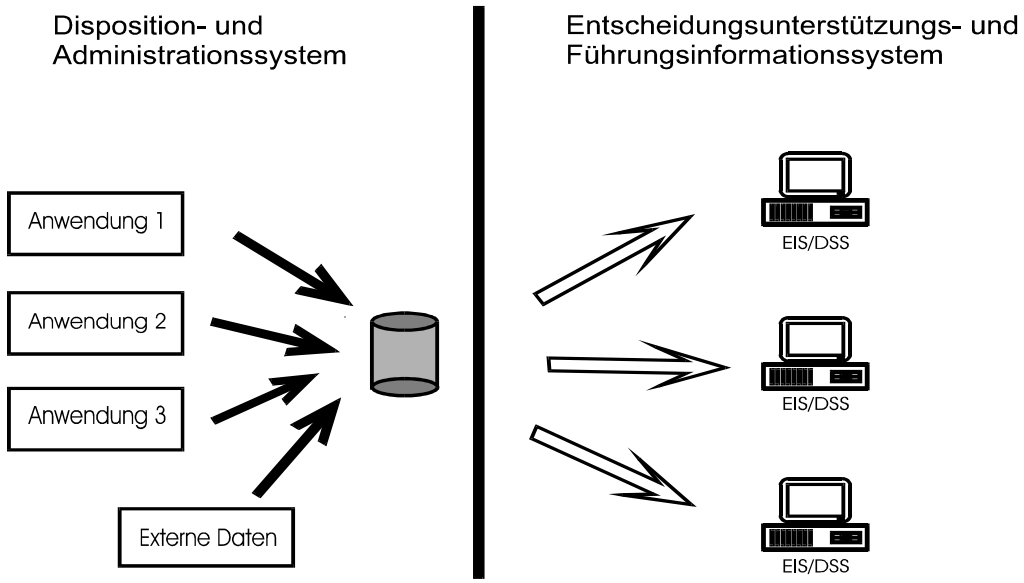


Abbildung 7: Architektur des virtuellen Data Warehouses

3 OLAP und Data Warehousing

Der Begriff OLAP (**O**n-**L**ine **A**nalytical **P**rocessing) wurde durch Edgar F. Codd, den geistigen Erfinder des relationalen Datenmodells, 1993 geprägt und beinhaltet im wesentlichen die konzeptionelle Basis für Lösungen zur Unterstützung einer dynamischen Datenanalyse in Unternehmen[7].

OLAP ermöglicht eine komplett neue, multidimensionale Sichtweise auf die Datenbestände des Unternehmens, die schnellere und flexiblere Abfragen ermöglicht. Im Gegensatz zum typischen OLTP (**O**n-**L**ine **T**ransaction **P**rocessing) der relationalen Datenbanken spricht man bei der Einteilung der Daten nicht mehr von Tabellen und Spalten, sondern von **Dimensionen**, **Dimensionselementen**, **Dimensionshierarchien** und **Datenwürfeln** (auch Measures oder Metriken genannt).

Ein Data Warehouse, bzw. die zugrunde liegende Datenbank, muß OLAP konform sein, um die multidimensionale Analysefunktionalität zur Verfügung stellen zu können. Aus diesem Grund haben sich zwei Data Warehouse Ansätze herauskristallisiert:

(1) ROLAP (relational OLAP)

Bei ROLAP wird das Data Warehouse auf einer herkömmlichen relationalen Datenbank erstellt. Relationale Datenbanken sind aber ursprünglich für **OLTP** (**O**n **L**ine **T**ransactional **P**rocessing) ausgelegt und speichern die Daten physikalisch in Tabellen, Zeilen und Spalten.

Um hier die OLAP Datensicht zu realisieren muß ein spezielles, multidimensionales Datenmodell erstellt werden, welches sich stark von dem herkömmlichen Entity-Relationship-Modell unterscheidet (siehe auch Kapitel 4).

[7] Schinzer, Bange, Wehner, Zeile (Management mit Maus und Monitor, 1997) S. 37

(2) MDOLAP (multidimensional OLAP)

Bei MDOLAP wird das Data Warehouse auf einer multidimensionalen Datenbank erstellt. Bei dieser Datenbank ist das OLAP Prinzip tatsächlich auch physikalisch implementiert. Daten werden nicht in Tabellen, Spalten und Zeilen gespeichert, sondern in OLAP Objekten, wie z. B. Dimensionen, Dimensionshierarchien und Measures.

Egal auf welcher Datenbank das Warehouse aufgebaut wird: Zusätzlich werden OLAP fähige Frontend-Tools benötigt, die das OLAP Potential auch wirklich für die Analyse und Berichterstellung ausnutzen.

Folgende Objekte sind für den OLAP Ansatz typisch:

(1) Dimension

Logische Einteilung von gleichartigen Daten in ein Zugehörigkeitsgebiet.

Beispiel: Dimension **Zeit**: 01.01.98, 1998, 02/98
Dimension **Geographie**: Nord, Süd, Hessen, Frankfurt, Filiale 8

(2) Dimensionselemente

Dimensionselemente sind die gespeicherten Daten einer Dimension.
Beispiel: Dimension Zeit: **01.01.98, 1998, 02/98**
Dimension Geographie: **Nord, Süd, Hessen, Frankfurt, Filiale 8**

(3) Dimensionhierarchien

Unterteilung einer Dimension in voneinander abhängige **Attribute**.

Beispiel: Eine **Region** beinhaltet ein oder mehrere **Bundesländer**
Ein Bundesland beinhaltet ein oder mehrere **Städte**
Eine Stadt beinhaltet eine oder mehrere **Filialen**

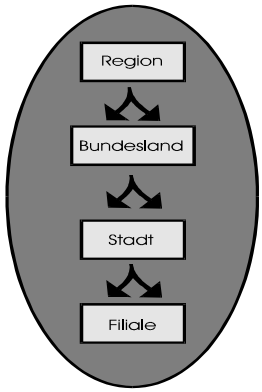


Abbildung 8: Dimensionshierarchie für die Dimension Geographie

(4) Datenwürfel (Measure, Metrik)

Beinhaltet die eigentlichen Kennzahlen, wie z. B. den Umsatz in DM, und wird normalerweise in Abhängigkeit mehrerer Dimensionen gesetzt. Dabei ist die Anzahl der Dimensionen, die diesen Datenwürfel beschreiben, nicht begrenzt.

Jede der Würfelzellen kann einen Kennzahlenwert aufweisen. Es kann aber auch vorkommen, daß z. B. im Monat 03/98 mit der Produktgruppe Geländewagen noch gar kein Umsatz gemacht wurde, weil die Produktgruppe erst in 04/98 auf dem Markt angeboten wurde. Hier würde die Würfelzelle nicht belegt werden, da keine Daten vorhanden sind.

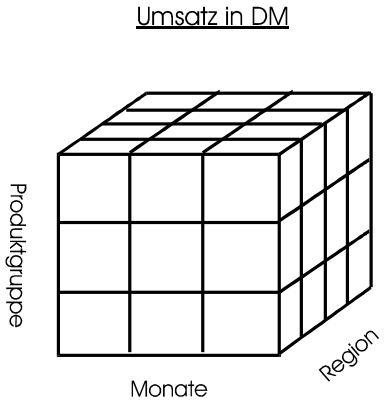


Abbildung 9: Datenwürfel

Durch die multidimensionale Sicht wird die menschliche Denkweise abgebildet und erleichtert somit das Analysieren des Datenbestandes.

Beispiel: Ein Manager formuliert eine Anfrage etwa folgendermaßen

"*Ich möchte gerne die **Umsätze in DM** aller **Produktgruppen** der einzelnen **Verkaufsregionen** für die **Monate 03/98, 04/98 und 05/98** sehen.*"

Durch diese Analysedefinition werden festgelegt:

1. die **Dimensionen** Zeit(Monat), Produkt(Produktgruppe), Geographie (Region)

2. Die **Dimensionshierarchien** könnten vorher schon folgendermaßen festgelegt worden sein:

   Dimension Zeit: Jahr → Monat → Tag
   Dimension Produkt: Abteilung → Produktgruppe → Artikel
   Dimension Geographie: Region → Bundesland → Stadt → Filiale

3. die **Measure '**Umsätze in DM' aufsummiert auf Produktgruppen-, Monats- und Regionsebene (eine sogenannte **Konsolidierung**)

Durch die Analyseanforderung ergibt sich der folgende Datenwürfel mit drei beschreibenden Dimensionen:

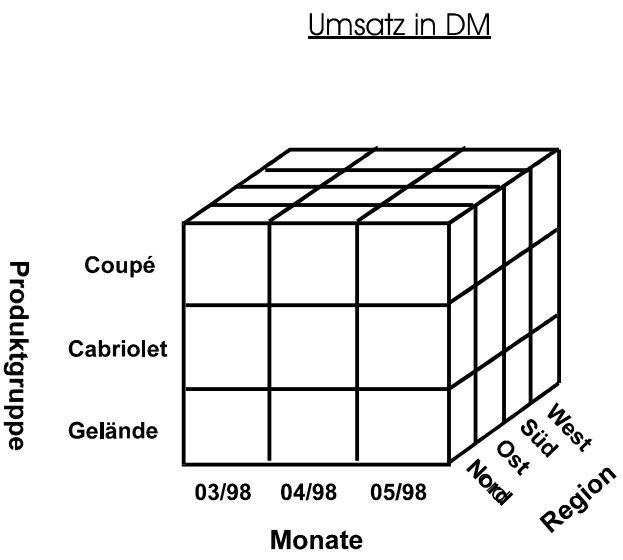


Abbildung 10: Datenwürfel Umsatz in DM pro Monat, pro Produktgruppe, pro Region

3.1 OLAP Analysefunktionalitäten

OLAP bietet verschiedene Funktionalitäten, mit deren Hilfe der Endbenutzer die Daten unterschiedlich betrachten und analysieren kann, vorausgesetzt das Frontend-Tool, welches der Benutzer für seine Berichte und Analysen einsetzt, ist OLAP fähig[8].

3.1.1 Data Dicing

Durch Drehen des Datenwürfels kann sich der Endbenutzer die unterschiedlichen Ansichten des Würfels betrachten (= **Daten Pivoting**)

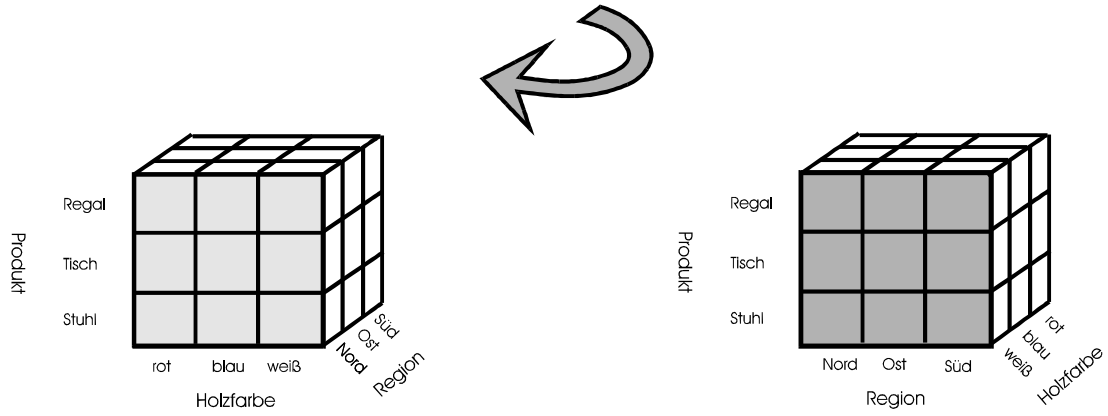


Abbildung 11: Data Dicing

Realisiert wird dieses Data Dicing meistens durch die Möglichkeit die Dimensionen innerhalb einer Brichtsschablone horizontal und vertikal zu verschieben. Die Berichtsschablone definiert genau wo welche Daten in dem Bericht zu erscheinen haben.

Die Anzahl der möglichen Ansichten des Würfels steigt mit der Anzahl der Dimensionen des Würfels (n Fakultät):

2 Dimensionen, 2 Ansichten
3 Dimensionen, 6 Ansichten
4 Dimensionen, 24 Ansichten
usw.

[8] Schinzer, Bange, Wehner, Zeile (Management mit Maus und Monitor, 1997) S. 42

Beispiel[9]**:** Dimensionen des Würfels sind **Produkt**, **Holzfarbe**, **Region**

1. Ansicht: Produkt nach Holzfarbe nach Region
2. Ansicht: Produkt nach Region nach Holzfarbe
3. Ansicht: Holzfarbe nach Produkt nach Region
4. Ansicht: Holzfarbe nach Region nach Produkt
5. Ansicht: Region nach Produkt nach Holzfarbe
6. Ansicht: Region nach Holzfarbe nach Produkt

3.1.2 Data Slicing

Durch das Data Slicing wird es dem Benutzer ermöglicht einzelne, individuell ausgewählte Scheiben des Datenwürfels zu betrachten, indem er bestimmte Filterkriterien entlang der Dimensionen definiert, wie z. B. Holzfarbe = weiß, oder Region = Süd und dementsprechend auch nur diesen entsprechenden Teilbereich des Würfels angezeigt bekommt.

[9] H. Mucksch, W. Behme (Das Data Warehouse-Konzept, 1996) S. 181

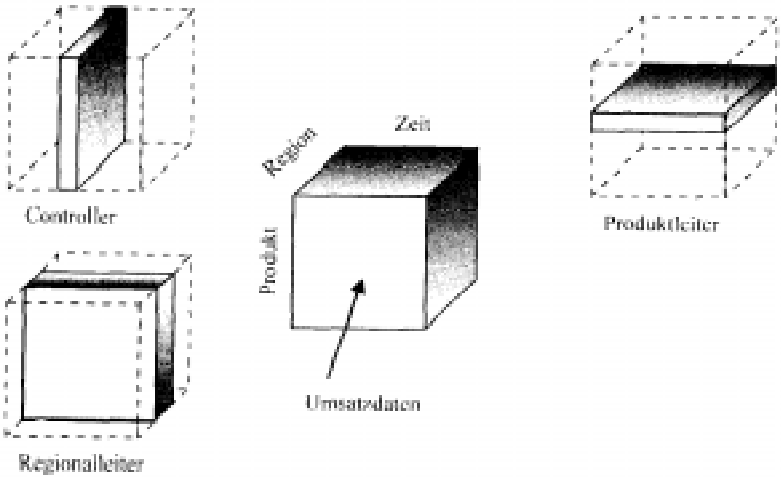


Abbildung 12: Data Slicing[10]

Ein Produktleiter könnte sich somit sehr einfach und schnell genau die Umsätze des Produktes anzeigen lassen für das er verantwortlich ist, ohne gleichzeitig unnötige Informationen über den Umsatz aller anderen Produkte zu erhalten.

Ein Regionalleiter filtert den Würfel genau auf die Region für die er verantwortlich ist, ein Controller hingegen möchte die Umsätze für bestimmte Zeitpunkte sehen.

D. h., durch Data Slicing wird eine **dynamische**, **anwenderspezifische Betrachtung** des statischen Datenmodells ermöglicht.

3.1.3 Drill Up, Drill Down, Drill Through

Durch diese Funktionalität kann der Benutzer ein dynamisches **Data Surfing** durch den Datenbestand durchführen, indem er beliebig in dem ihm zur Verfügung stehenden Datenraum navigiert.

Ermöglicht wird diese Navigation durch die Bildung von sogenannten **Konsolidationsebenen**. Diese Ebenen stellen **Konsolidierungen** (= Summierungen) von Kennzahlen auf den einzelnen Hierarchieebenen einer Dimension dar.

Die unterste Hierarchieebene der Dimension stellt die atomare Datenebene dar. Jede darüber liegende Ebene ist eine Summierung der Kennzahlen der direkt darunterliegenden Hierarchiebebene.

Beispiel: Die hierarchische Geographie-Dimension mit der Kennzahl Umsatz in DM

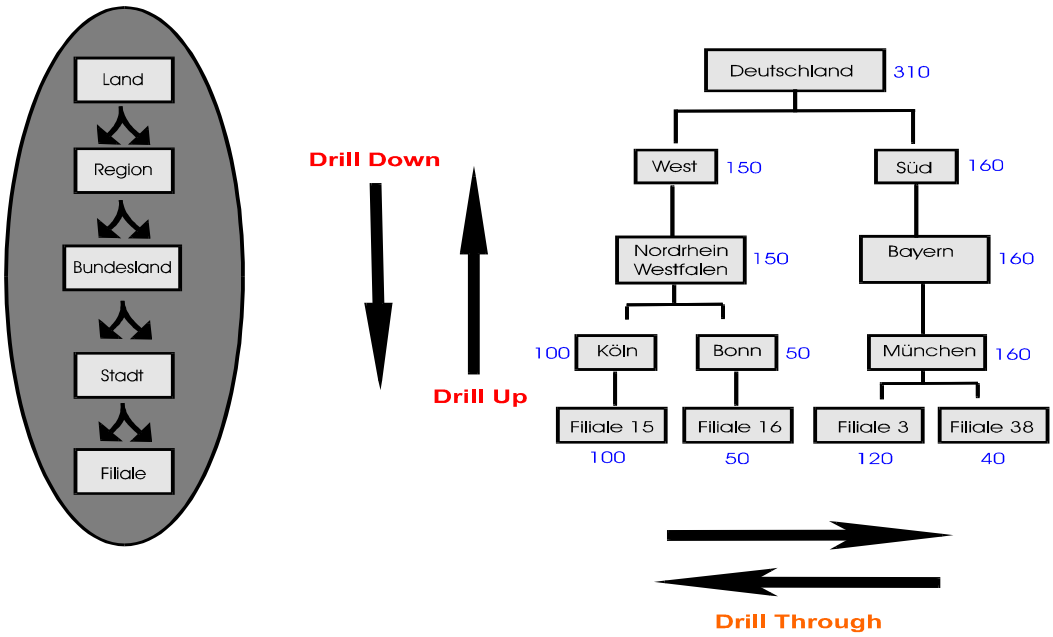


Abbildung 13: Drill Funktionalität

[10] Quelle: Schinzer, Bange, Wehner, Zeile (Management mit Maus und Monitor, 1997) S. 40

Mit Hilfe des **Drill Downs** kann der Benutzer immer detailliertere Kennzahlen ermitteln, indem er sich bei jedem Drill Down eine Hierarchieebene tiefer begibt.

Durch ein **Drill Up** bewegt sich der Benutzer wieder eine Hierarchieebene nach oben und erhält weniger detaillierte, konsolidierte Kennzahlen.

Der **Drill Through** ermöglicht es innerhalb einer Dimensionshierarchieebene von einem Dimensionselement zu einem anderen zu wechseln.

Natürlich funktioniert diese Drill Funktionalität nur, wenn überhaupt konsolidierte Daten vorhanden sind.

Konsolidierungen können direkt im Data Warehouses gespeichert werden, oder das Frontend-Tool, das zur Analyse eingesetzt wird, bietet die Möglichkeit aus atomaren Daten, welche im Data Warehouse gespeichert sind, Konsolidationen dynamisch zur Laufzeit zu errechnen.

4 Das ROLAP Data Warehouse Design

Wie schon angesprochen, handelt es sich bei **ROLAP** (**r**elational **OLAP**) um ein Data Warehouse, welches auf einer relationalen Datenbank aufgebaut wird.

Die OLAP Datensicht wird mittels spezieller Datenmodellierung auch auf einem sonst OLTP basierendem, relationalen Datenbanksystem zumindest **virtuell** ermöglicht. Physisch wird diese Datensicht natürlich immer noch über Tabellen, Spalten und Zeilen realisiert. Mit einem herkömmlichen, auf OLTP basierendem **ERM** (**E**ntity **R**elationship **M**odell), hat das dazu notwendige Datenmodell aber nur noch wenig zu tun (siehe Kapitel 4.3).

4.1 Die ROLAP Architektur

In erster Linie entspricht die ROLAP Architektur der Basis-Architektur eines Data Warehouses, mit der Ausnahme, daß sie um eine weitere Komponente, die **ROLAP Engine (auch SQL-Engine genannt)** ergänzt werden muß.

Auf der relationalen Datenbank existiert nur ein virtuelles OLAP Schema, welches mittels Tabellen, Spalten und Zeilen realisiert wird. Auf dieses Schema kann nur mittels SQL Befehlen zugegriffen werden. SQL ist eine OLTP Abfragesprache und kann demzufolge nur typische OLTP Objekte wie Tabellen, Spalten und Zeilen ansprechen.

OLAP Frontends verwenden zur Definition der Analyse typische OLAP Objekte wie z. B. Dimensionen, Metriken, Dimensionshierarchien, usw. Damit solche OLAP Anfragen auf der relationalen Datenbank bearbeitet werden können, müssen sie erst einmal in eine SQL-Abfrage umgewandelt werden. Diese Umwandlung übernimmt die ROLAP Engine.

Für den Endbenutzer ist nicht ersichtlich, daß die Daten in einer relationalen Form gespeichert sind. Für ihn existiert nur die multidimensionale Sichtweise.

Die ROLAP Engine kann entweder auf dem gleichen Server wie das Data Warehouse, oder aber auf einem eigenen Server liegen. Durch die Plazierung auf einen eigenen Server wird der Datenbankserver entlastet und bietet mehr Ressourcen für die eigentliche Datenauswertung, so daß Anfragen schneller bearbeitet werden können.

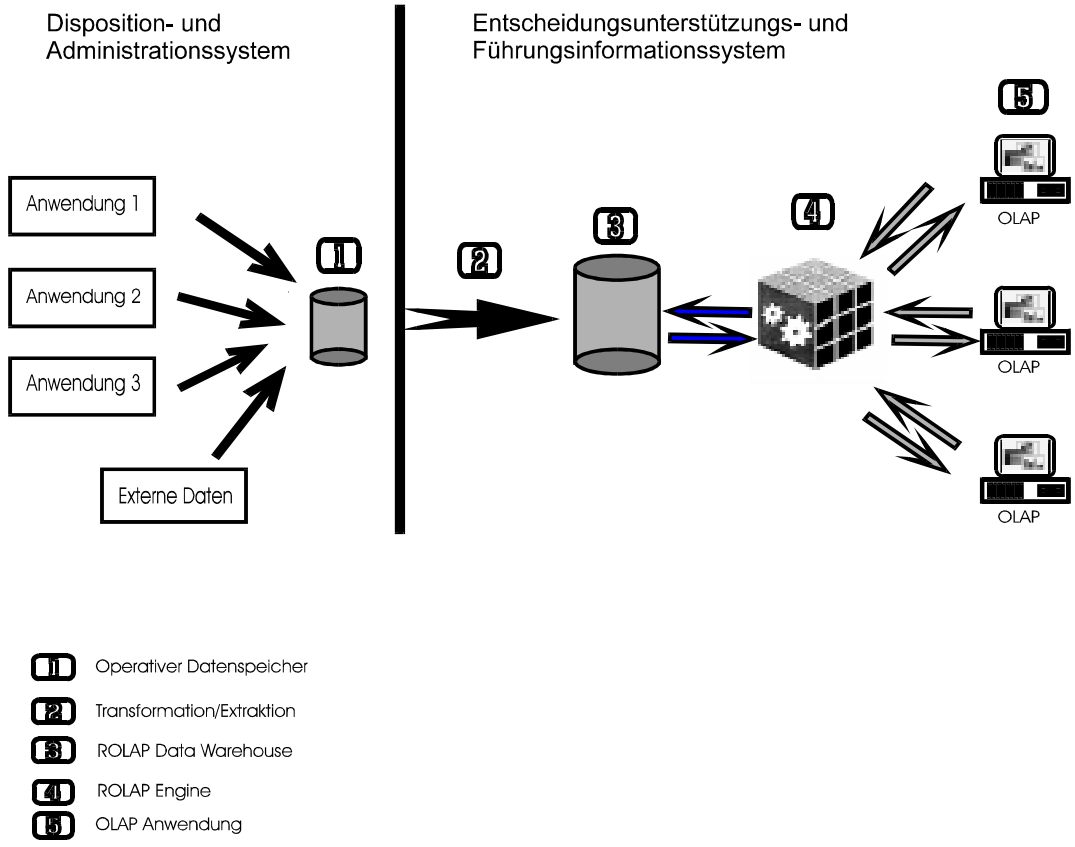


Abbildung 14: Die ROLAP Architektur

Zuerst werden natürlich auch hier wieder die Daten des operativen Systems (1) mittels Datenextraktion und -transformation (2) in das ROLAP Data Wa-

rehouse (3) kopiert. Anfragen der OLAP-Anwendungen (5) werden erst in der ROLAP-Engine (4) in optimiertes SQL umgewandelt, danach an die relationale Datenbank weitergereicht. Die Datenbank wertet die SQL Abfrage aus und liefert das Ergebnis zurück an die ROLAP-Engine. Dort wird das Ergebnis erst wieder OLAP konform umgewandelt und schließlich an die aufrufende OLAP Anwendung weitergereicht.

4.2 Metadaten

Metadaten spielen eine zentrale Rolle in der Welt des ROLAP Data Warehouses. Sämtliche Informationen über Inhalte und Aufbau des Data Warehouses werden in diesen Metadaten gespeichert und stehen den unterschiedlichsten Benutzern des Data Warehouses (Administratoren, Software-Entwickler, Analysten, Manager, usw.) zur Verfügung.

Jedes Software-Produkt, ob es nun für die Datenextraktion, -transformation, oder für Analysen und Berichtserstellung verwendet wird, benötigt und erstellt eigene Metadaten. Diese Metadaten stellen die Verbindung zwischen diesen Softwareprodukten und dem Data Warehouse mit den darin gespeicherten Daten her. Weiterhin speichern die Metadaten Informationen über Objekte die von dem Anwender mit Hilfe des Tools erstellt wurden.

Leider verwendet fast jedes Softwareprodukt seine eigene Metadatenstruktur, um die speziellen Funktionalitäten des Softwareproduktes zu unterstützen. Dadurch können für ein Data Warehouse durchaus mehrere Metadatenquellen existieren, die auch nur mit dem jeweiligen Softwaretool zu verwalten sind[11].

Grundlegend kann man folgende zwei Arten von Metadaten unterscheiden[12].

[11] R. Tanler, J. Frome (The Expanding Role of Enterprise Meta Data) Artikel in DM Review
[12] D. Marco (Managing Meta Daten) Artikel in DM Review

4.2.1 Technische Metadaten

Technische Metadaten helfen Administratoren und Data Warehouse Designern das Data Warehouse zu erstellen, zu verwalten und zu optimieren.

Beispiele für Technische Metadaten:

* Data Warehouse Tabellennamen, Primary- und Foreign-Keys, Indizes
* Transformations- und Extraktionsdaten, wie z. B. das Mapping (Zuordnung) zwischen den Datenqellen des operativen Systems und den Tabellen des Data- Warehouses
* Informationen über das physische Datenmodell
* Definition der Tabellenstrukturen, der Tabellenattribute, der Beziehungen zwischen Tabellen
* Auslastungsinformationen der Datenbank
* Speicherinformationen der Datenbank

Technische Meta-Daten werden über das Data-Dictionary der zugrunde liegenden relationalen Datenbank angelegt und gepflegt, sowie über Softwareprodukte, die speziell für den Aufbau und die Verwaltung des Data Warehouses benötigt werden, wie z. B. Datenextraktions- und transformations Tools.

4.2.2 Business Metadaten (semantische Metadaten)

Business Metadaten helfen weniger technisch versierten Benutzern (Analysten, Manager, usw.) sich innerhalb der Datenflut des Data Warehouses in der für sie gewohnten Sprache zurecht zu finden. Diesen Benutzerkreis interessiert es nicht, wie die tatsächliche physikalische Struktur des Data Wa-

rehouses aussieht, bzw. wie Anfragen technisch umgesetzt werden, geschweige denn möchten sie auf dieser komplexen Stufe Analysen und Berichte erstellen.

Beispiele für Business Metadaten:

* Welche Berichte liegen im Data Warehouse bereits vor
* Was für Dimensionen, Dimensionshierarchien, Kennzahlen gibt es
* Wie alt sind die Daten (wann wurden sie das letzte Mal aktualisiert)
* Wie ist eine Kennzahl definiert
* Aus welcher Datenquelle kommen die jeweiligen Daten
* Welche Konsolidierungen liegen vor

Immer wenn Änderungen im Data Warehouse vorgenommen werden, dann sind auch die Metadaten des Warehouses bzw. der Softwaretools von dieser Änderung betroffen.

Werden weitere Tabellen und Daten im Warehouse hinzugefügt, z. B. mit Hilfe eines Extraktions- und Transformationstools, dann werden einmal die Metadaten des Tools verändert und zum anderen die Metadaten der relationalen Datenbank. Benutzt man beim nächsten Mal wieder das Extraktions- und Transformationstool, so ist die vorher getätigte Änderung des Warehouses immer noch erkennbar, da das Tool den aktuellen Stand aus seinen Metadaten und denen des Datenbanksystems einliest.

Erstellt ein Manager mit Hilfe eines OLAP Analysetools einen neuen Bericht und speichert ihn ab, so wird dieser neu erstellte Bericht auch beim nächsten Starten des Analysetools aufgeführt, da auch das Analysetool den aktuellen Stand aus seinen Metadaten einliest.

In den Metadaten werden auch die einzelnen Bestandteile, die Definition des Berichtes gespeichert, so daß auch noch nachträglich Änderungen an diesem Bericht durchgeführt werden können.

4.3 Die ROLAP Datenmodellierung

Das typische **E**ntity-**R**elationship-**M**odelling (**ERM**) wie es im operativen System angewandt wird sorgt zwar für ein sauberes Datendesign; Folge dieses Designs sind aber viele kleine Tabellen mit einer geringen Anzahl von Attributen.

Typische multidimensionale Geschäftsabfragen, wie sie im Data Warehouse Umfeld gestellt werden, erfordern auf solch einem Datenmodell viele Tabellenverknüpfungen (**Joins**), welche sich negativ auf die Antwortzeiten, sowie die Systemlast auswirken.

Aus diesem Grund ist es notwendig ein spezielles Datenmodell für das Data Warehouse zu verwenden. Die folgenden Modelle haben sich im Warehouse Umfeld durchgesetzt[13].

4.3.1 Star-Schema

Ziel dieses Schemas ist es die Anzahl der Joins und die Komplexität des Datenmodells zu minimieren, so daß eine benutzerfreundliche Antwortzeit gewährleistet werden kann.

Hauptkomponenten des Star-Schemas sind die sogenannten **Fakttabellen** und **Dimensionstabellen**, wobei die Dimensionstabellen **sternförmig** um

[13] Schinzer, Bange, Wehner, Zeile (Management mit Maus und Monitor, 1997) S. 48

die Fakttabellen mittels Primary-Key / Foreign-Key Beziehungen angeordnet sind.

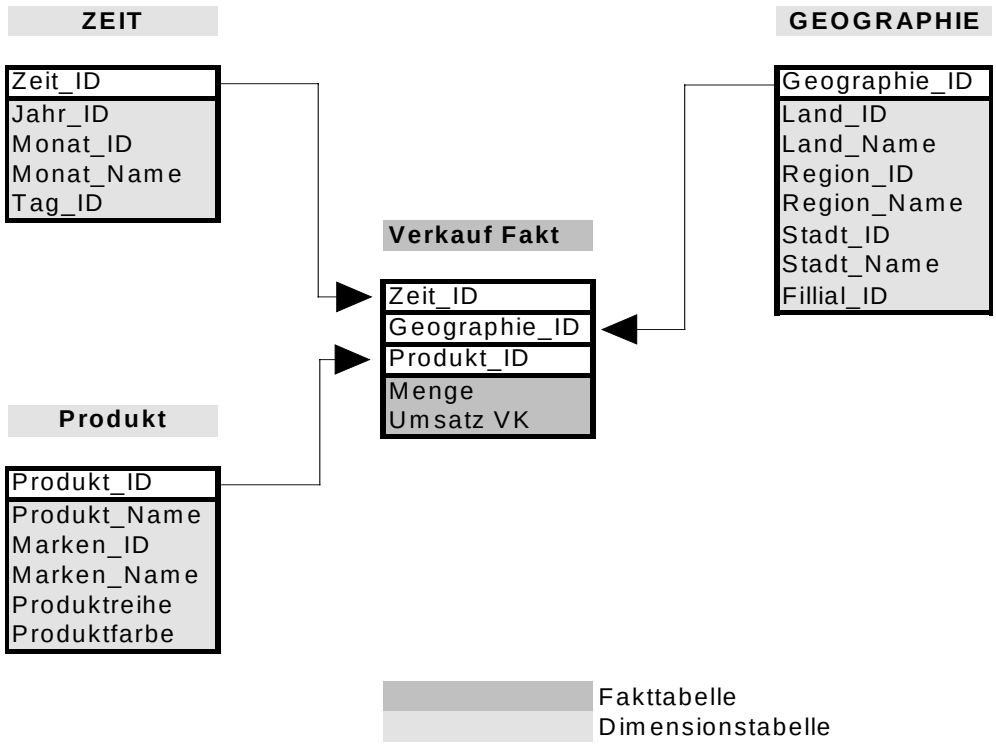


Abbildung 15: Star-Schema[14]

In der Fakttabelle werden die eigentlichen Analysedaten, wie z. B. Umsatz, Verkaufsmenge gespeichert. In den Dimensionstabellen hingegen werden nur beschreibende Daten, wie z. B. Produktbezeichnungen, Geographiedaten, usw. gespeichert.

Aus diesem Grund kann man auch sagen, daß die Fakttabelle ausschließlich numerische Daten beinhaltet und die Dimensionstabellen überwiegend Textdaten.

Dieses Datenmodell macht eigentlich nichts anderes als über die Fakttabelle und die Dimensionstabellen die typischen OLAP Objekte: Datenwürfel, Dimensionen, Dimensionshierarchien und Dimensionselemente abzubilden.

Jeder der Dimensionstabellen stellt genau eine Dimension dar. Innerhalb jeder Dimensionstabelle können Dimensionshierarchien abgebildet werden. Die einzelnen Datensätze der Dimensionstabellen stellen die Dimensionelemente dar.

Die Fakttabelle und die entsprechenden Dimensionstabellen zusammen bilden einen Datenwürfel, wobei die Datensätze der Fakttabelle die Zellen dieses Datenwürfels darstellen.

Strenggenommen bedeutet das Star-Schema einen Rückschritt in der Datenmodellierung, da hier zur Minimierung der Tabellen-Joins die Normalisierungsregeln außer Kraft gesetzt werden.

Folge davon ist eine **Denormalisierung** des Datenbestandes und somit die **redundante Datenhaltung** (siehe Abbildung 16).

[14] in Anlehnung an Schinzer, Bange, Wehner, Zeile (Management mit Maus und Monitor, 1997) S. 49

Normalisiertes Datenmodell

Atikelltabelle

Artikel_ID	Artikel_Name	WG_ID
1001	Cola	1
1002	Wasser	1
2001	Kölsch	2
2002	Alt	2
2002	Wodka	2

Warengruppentabelle

WG_ID	WG_Name
1	Alkoholfreie Getränke
2	Alkoholische Getränke

Filial-/Warengruppentabelle

Fillial_ID	WG_ID
115	1
115	2
91	1
88	2

Denormalisiertes Datenmodell

Produkttabelle

Artikel_ID	Artikel_Name	WG_ID	WG_Name
1001	Cola	1	Alkoholfreie Getränke
1002	Wasser	1	Alkoholfreie Getränke
2001	Kölsch	2	Alkoholische Getränke
2002	Alt	2	Alkoholische Getränke
2002	Wodka	2	Alkoholische Getränke

Filial-/Warengruppentabelle

Fillial_ID	WG_ID	WG_Name
115	1	Alkoholfreie Getränke
115	2	Alkoholische Getränke
91	1	Alkoholfreie Getränke
88	2	Alkoholische Getränke

redundante Daten

Abbildung 16: Normalisiertes vs. Denormalisiertes Datenmodell

Ein Kritikpunkt des denormalisierten Star-Schemas könnte die Pflegbarkeit des Datenbestandes sein.

Ergeben sich Änderungen, wie z. B. die Änderung einer Warengruppenbezeichnung, dann müssen sämtliche Datensätze mit der alten Warengrup-

penbezeichnung in den jeweiligen Tabellen abgeändert werden. Von dieser Änderung können leicht tausend oder mehr Datensätze betroffen sein.

Bei einem normalisierten Datenmodell würde man lediglich einen Datensatz in der Tabelle Warengruppe abändern. Diese Änderung würde sich direkt auf alle anderen abhängigen Datensätze in den entsprechenden Tabellen auswirken, da dort die Warengruppe nur über den Primary Key der Warengruppentabelle verknüpft ist.

Diese Pflegbarkeit des Datenmodells spielt aber in einem Data Warehouse keine Rolle, da die Daten in das Data Warehouse nur einmal kopiert werden und dann nur noch lesend auf diese Daten zugegriffen wird. D. h., eine Abänderung von Datensätzen findet gar nicht statt.

Dieses Merkmal wird auch als **Nicht-Volatilität** bezeichnet, wobei die Volatilität die Anzahl der Änderungen eines Datenbestandes beschreibt. Da sich die Daten in einem Data Warehouse nicht mehr ändern (kein update .... set Befehl) liegt eine Nicht-Volatilität vor.

Eine tatsächliche Auswirkung der redundanten Datenhaltung ist der höhere Speicherbedarf dieses Datenmodells. Anstatt von speichersparenden, numerischen Foreign-Keys werden hier redundant Textdaten gespeichert, die wesentlich mehr Speicherplatz erfordern.

Grundsätzlich sind aber die Fakttabellen die größten Tabellen eines Data Warehouses, da hier sämtliche Transaktionen des operativen Systems, wie z. B. der Verkauf eines Artikels an einem bestimmten Tag festgehalten werden. Die Fakttabellen sind aber nicht redundant. Nur die Dimensionstabellen werden redundant gespeichert. Allerdings werden die Dimensionstabellen in den seltensten Fällen so groß, daß der Aspekt des Speicherbedarfs ein anderes Datenmodell erfordert.

Sollte der Speicherplatz doch eine Rolle spielen, so ist das nachfolgende Modell zu bevorzugen.

4.3.2 Das Snowflake-Schema

Das Snowflake-Schema besteht ebenfalls aus den beiden Komponenten Fakt-Tabelle und Dimensions-Tabelle. Die Dimensions-Tabellen werden jedoch im Gegensatz zum Star-Schema normalisiert. Durch die Normalisierung entstehen zusätzliche Attribut-Tabellen.

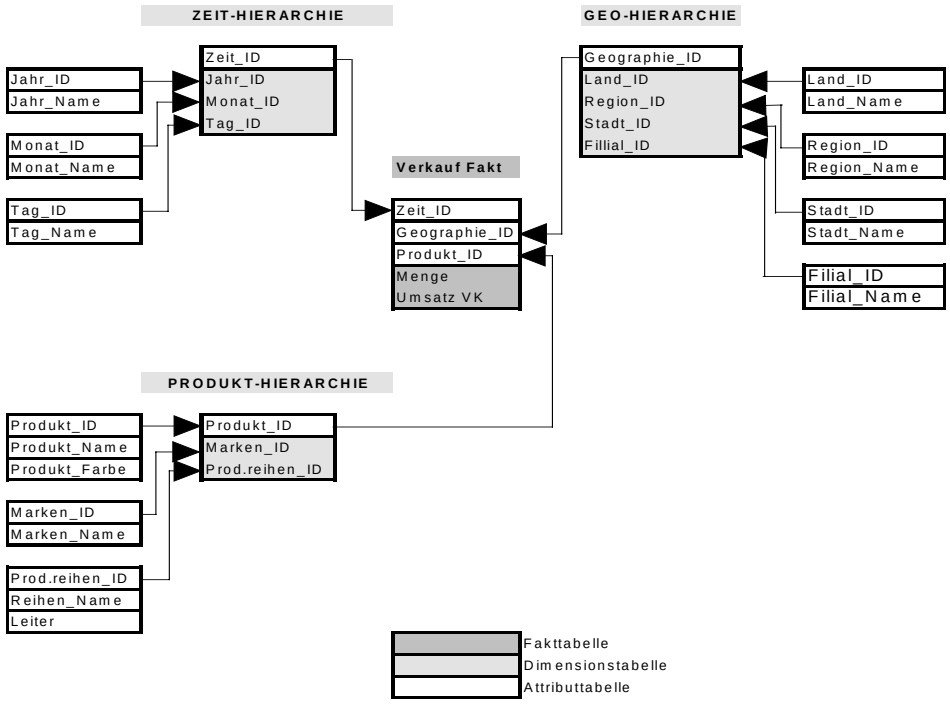


Abbildung 17: Snowflake-Schema[15]

Vorteil dieses Schemas ist die Vermeidung der redundanten Datenhaltung und somit die Einsparung von Speicherplatz. Nachteil dagegen ist die kom-

[15] in Anlehnung an Schinzer, Bange, Wehner, Zeile (Management mit Maus und Monitor, 1997) S. 50

plexere Modellstruktur und zusätzlich notwendige Tabellen-Joins, die wiederum die Abfrage-Performance verschlechtern.

Eine kombinierte Form von Star- und Snowflake-Schema stellt das sogenannte **Partial Snowflake-Schema** dar. Bei diesem Schema werden nur sehr große Dimensionen nach dem Snowflake-Schema normalisiert. Alle anderen Dimensionen bleiben in dem Star-Schema.

4.4 ROLAP Designaspekte

Da in einem Data Warehouse sehr schnell große Massen an Daten anfallen, ist es um so wichtiger mittels geeigneter Modellierungsverfahren den Speicherplatzbedarf sowie die **Abfrageperformance** (Antwortzeiten) zu optimieren.

Für diese Optimierungsmaßnahmen gibt es neben den schon genannten Modellierungsverfahren, Star- und Snowflake-Schema, die zusätzlichen Designaspekte der **Granularität** und der **Partitionierung**.

4.4.1 Granularität

Unter einer Granularität versteht man den Grad der Verdichtung von Data Warehouse Faktdaten (z. B. Umsatz) entlang der Ebenen der beteiligten Dimensionshierarchien. Dabei wird diese Verdichtung nicht zur Laufzeit einer Analyse temporär berechnet, sondern physikalisch in einer Fakttabelle gespeichert.

Bei der Verdichtung handelt sich in der Regel um eine Summierung detaillierterer Faktdaten (z. B. Umsatz pro Tag verdichtet auf Umsatz pro Monat).

Durchgeführt wird diese Verdichtung in der Regel während des Ladevorganges der Daten des operativen Systems in das Data Warehouse, so daß die verdichteten Daten direkt in den entsprechenden Fakttabellen gespeichert werden können.

Beispiel:

Folgende Dimensionshierarchien könnten im Data Warehouse vorliegen:

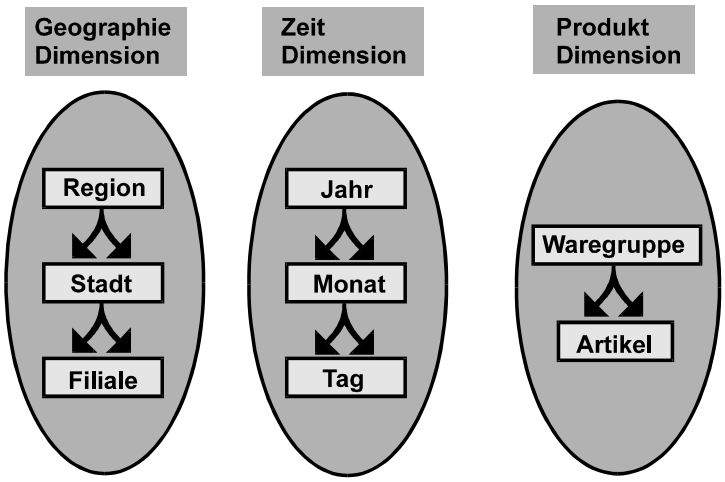


Abbildung 18: Dimensionhierarchien Geographie, Zeit, Produkt

Folgende **mehrstufige Granularitätsgrade** könnten für das Data Warehouse gewünscht sein (in dem operativen System werden die Verkaufstranskaktionen mit Datum, **Uhrzeit**, Artikelnummer, Filialnummer, Verkaufsmenge, Stückpreis, Gesamtbetrag gespeichert):

Fakt Umsatz_VK Granularitätgrad 1

Tag_ID	Filial_ID	Artikel_ID	Umsatz_VK in DM

Fakt Umsatz_VK Granularitätsgrad 2

Monat_ID	Stadt_ID	Warengruppen_ID	Umsatz_VK in DM

Fakt Umsatz_VK Granularitätsgrad 3

Jahr_ID	Region_ID	Warengruppen_ID	Umsatz_VK in DM

In der Granularitätsstufe 1 hat es nur eine leichte Verdichtung des **Umsatz_VK in DM** auf Tagesebene, pro Artikel, pro Filiale gegeben.

In der Granularitätsstufe 2 wurden die Ursprungsdaten des operativen Systems so verdichtet, daß in dieser Fakttabelle der **Umsatz_VK in DM** für die einzelnen Warengruppen, pro Stadt, pro Monat gespeichert ist.

In der Granularitätsstufe 3 wurden die Ursprungsdaten des operativen Systems so verdichtet, daß in dieser Fakttabelle der **Umsatz_VK in DM** pro Warengruppe, pro Region, pro Jahr gespeichert wird.

Das heißt, bei mehrstufigen Granularitätsgraden gibt es für eine Kennzahl mehrere unterschiedlich verdichtete Fakttabellen.

Entweder werden die unterschiedlich verdichteten Fakttabellen gleichzeitig im Data Warehouse gehalten, oder aber eine weniger verdichtete Fakttabelle wird durch eine höher verdichtete Fakttabelle ersetzt.

Um so höher die Verdichtung, desto geringer der Detaillierungsgrad und um so höher die Granularität.

Um so geringer die Verdichtung, desto höher der Detaillierungsgrad und um so niedriger die Granularität.

Eine besondere Form des mehrstufigen Granularitätsgrades stellt die **Rollende Summierung (Rolling Summary)** dar.

Durch diese Methode bestimmt man verschiedene Granularitätsgrade, in dem die Datenverdichtung **mit zunehmenden Alter** der Daten steigt. Jüngere Daten erhalten eine niedrige Granularität. Ältere Daten eine höhere Granularität. Nach angemessenem Zeitablauf werden die Daten auf einen sinnvolleren Granularitätsgrad verdichtet. In dieser verdichteten Form werden diese Daten dann entweder zusätzlich oder anstatt der detaillierteren Daten wieder zu Analysezwecken zur Verfügung gestellt.

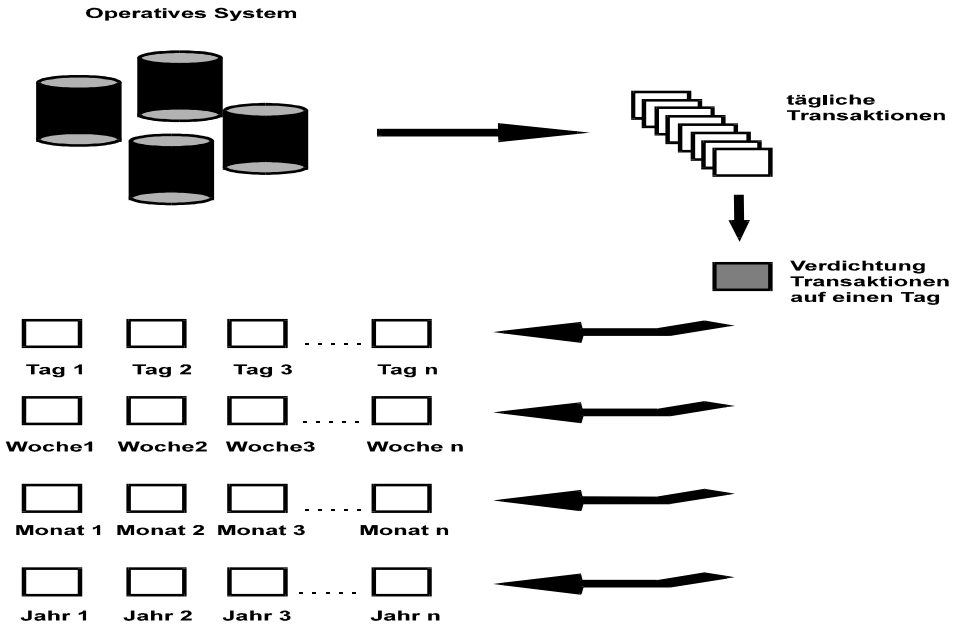


Abbildung 19: Rolling Summary[16]

Beispiel:

[16] Quelle: W. H. Inmon (Building the Data Warehouse, 1996) S. 62

> In einem Unternehmen werden die täglichen Umsätze pro Verkaufstransaktion gespeichert. Die Daten werden leicht verdichtet (Summierung der Umsätze einer Transaktion auf Umsatz pro Tag) in das Data Warehouse übernommen. Nach Ablauf einer Woche werden die Tagesumsätze der Woche auf Wochenumsatz verdichtetet. Nach Ablauf eines Monats werden die Daten auf Monatsumsatz verdichtet. Nach Ablauf eines Jahres werden die Daten auf Jahresumsatz verdichtet, usw.

Natürlich muß die Verdichtung bei der Rolling Summary nicht unbedingt über die Zeitdimension durchgeführt werden. Verdichtungen entlang der Hierarchieebenen anderer Dimensionen wie z. B. die Produktdimension sind ebenfalls denkbar, aber auch dort wird es wieder einen zeitlichen Verlauf geben, nachdem die Verdichtungen von einer Hierarchieebene zur nächsten durchgeführt werden.

Von der Wahl des Granularitätsgrades einer Fakttabelle hängt der Speicherplatzbedarf der Tabelle ab.

Um so höher der Granularitätsgrad, um so weniger Datensätze werden in der Fakttabelle gespeichert. Somit kann die Wahl eines höheren Granularitätsgrades den Speicherplatzbedarf des Data Warehouses optimieren.

> Als Beispiel[17] könnte man sich eine Telekommunikationsgesellschaft vorstellen, die die täglichen Anrufe ihrer Kunden registriert, um monatlich eine Abrechnung dieser Anrufe durchführen zu können.
>
> Zu jedem Telefonat wird das Datum, der Gesprächspartner, die Start-Uhrzeit, End-Uhrzeit, Gesprächsdauer, Gesprächstarif, usw. gespeichert. Pro Monat fallen dort etwa 200 Anrufe pro Kunde an.

[17] W. H. Inmon (Building the Data Warehouse, 1996) S. 47

Das bedeutet, daß pro Kunde 200 Datensätze pro Monat in dem operativen System der Gesellschaft gespeichert werden (entspricht 200.000 Datensätze pro Monat für 1000 Kunden).

Würde man die Daten des operativen Systems mit der gleichen niedrigen Granularität in das Data Warehouse übernehmen, so ergibt sich bei 1000 Kunden und etwa 200 Byte pro Datensatz ein Speicherplatzbedarf von ca. 38 Gbyte (1 Gbyte = 2^{20} Byte) pro Monat. Nach einem Jahr würden sich schon 456 Gbyte an Daten im Warehouse angesammelt haben.

Aufgrund des hohen Speicherplatzbedarfs solcher detaillierten Daten (niedrige Granularität) muß man sich überlegen, ob es nicht sinnvoller wäre diese Daten ab einem bestimmten Zeitpunkt nur noch mit einer höheren Granularität zu speichern.

Da bei der Telekommunikationsgesellschaft monatlich abgerechnet wird und in diesem Beispiel für den Warehouse-Benutzer nur Detaildaten des laufenden Monats für Analysen notwendig sind, wäre es vorteilhaft nach der Abrechnung den abgelaufenen Monat zu verdichten, so daß pro Kunde pro Monat nur noch ein Datensatz im Data Warehouse gespeichert werden muß (entspricht nur noch 1000 Datensätzen pro Monat anstatt 200.000). Aus dem Datensatz soll hervorgehen wie viele Anrufe der Kunde in dem letzten Monat getätigt hat, die durchschnittliche Gesprächsdauer, die Gesamtdauer aller Gespräche, der gesamte Rechnungsbetrag, usw.

Nach der Verdichtung des abgelaufenen Monats werden die detaillierten Datensätze pro Anruf des letzten Monats aus dem Data Warehouse gelöscht. Dadurch würde für ein abgelaufenes Jahr nur noch ein Speicherplatzbedarf von maximal 2,28 GByte entste-

hen, was eine deutliche Einsparung bedeutet (im Vergleich zu 456 Gbyte für die Speicherung jedes einzelnen Telefonates).

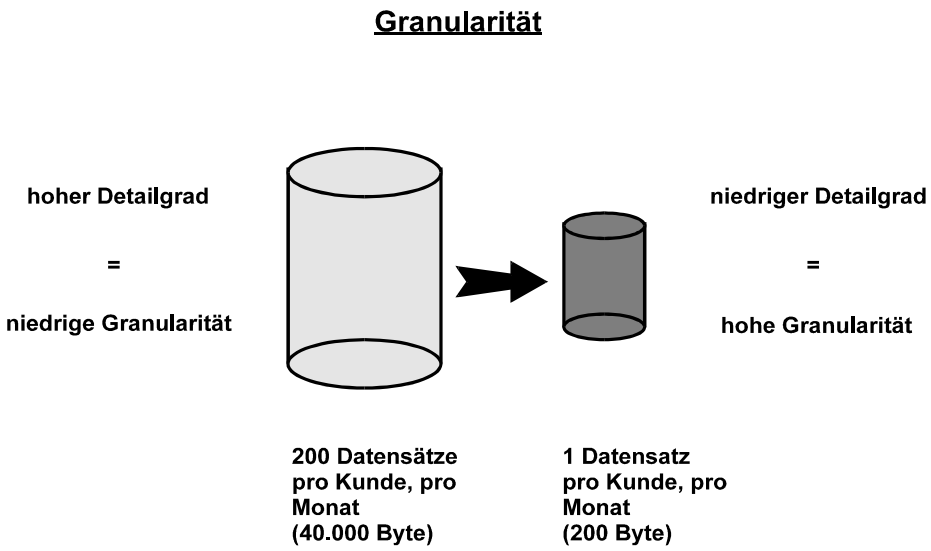


Abbildung 20: Niedrige vs. Hohe Granularität

Zusätzlich zur Einsparung von Speicherplatz werden Abfragen auf Fakttabellen mit höherem Granularitätsgrad schneller ausgeführt als Abfragen auf Fakttabellen mit niedrigerem Granularitätsgrad, da weniger Datensätze zu verarbeiten sind.

Somit kann durch die Wahl eines höheren Granularitätsgrades auch die Abfragegeschwindigkeit optimiert werden.

Durch die Verwendung von mehrstufigen Granularitätsgraden kann die zeitaufwendige, temporäre Summierung von Kennzahlen durch OLAP Frontend-Tools vermieden werden.

Stattdessen können die aufsummierten Kennzahlen direkt aus den entsprechenden Fakttabellen gelesen werden, vorausgesetzt die unterschiedlichen

Verdichtungen einer Kennzahl werden gleichzeitig in dem Data Warehaus gespeichert.

Werden allerdings zu Gunsten des Speicherplatzes nur noch Fakttabellen hoher Granularität gespeichert, so verliert das Data Warehouse an Aussagekraft. Die Folge ist, daß Abfragen evtl. nicht mehr in der gewünschten Detailtiefe durchgeführt werden können.

Somit ist schon beim Warehouse-Design genaustens abzuwägen, in welcher Granularität die einzelnen Fakttabellen vorliegen müssen, damit eine gewünschte Detailtiefe vorliegt, eine Speicherplatzgrenze nicht überschritten wird und die Antwortzeiten der Benutzeranalysen und -berichte in einem verträglichen Rahmen liegen.

Eine Möglichkeit alle drei genannten Ziele zu verwirklichen, stellt die **Dual Level of Granularity** Methode[18] von W. H. Inmon dar.

Nach dieser Methode werden die Daten des Data Warehouses nach ihrer Granularität zum einen direkt im Data Warehouse (Daten mit höherer Granularität) gespeichert und zum anderen in Data Warehouse Archiven (Daten mit niedriger Granularität bzw. die Ursprungsdaten des operativen Systems).

Dadurch wird es ermöglicht, daß die hauptsächlich auszuführenden Analysen, für die der Detailreichtum der leicht bis stark verdichteten Daten des Data Warehouses ausreichend ist, direkt über das Data Warehouse durchgeführt werden können (nach Inmon ca. 95 % aller Abfragen).

Eher seltene Abfragen auf detaillierte Daten können über das Data Warehouse Archiv verwirklichen werden (lt. Inmon nur 5 % aller Abfragen).

[18] W. H. Inmon (Building the Data Warehouse, 1996) S. 51

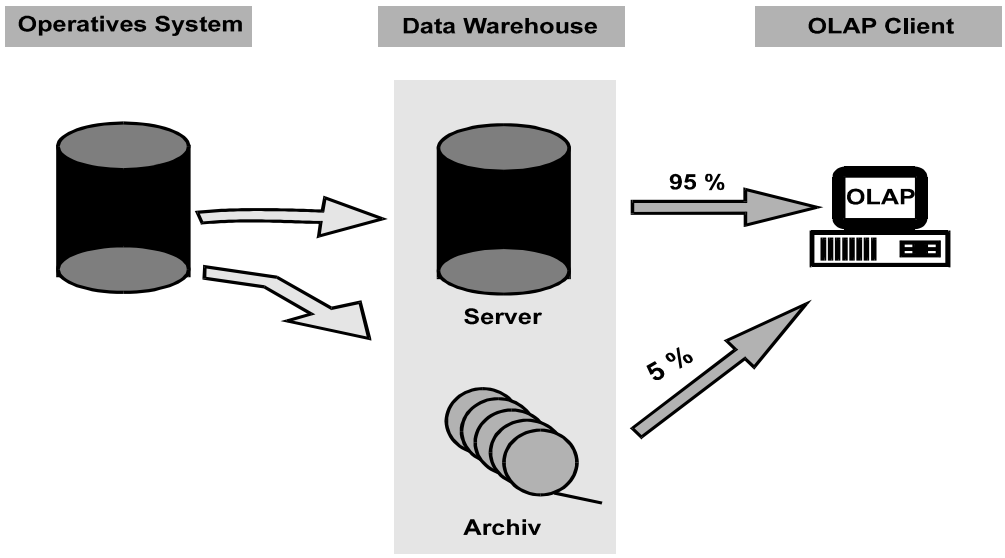


Abbildung 21: Dual Level of Granularity[19]

Natürlich ist der Zugriff auf die Archiv-Daten als sehr zeitaufwendig und somit kostspielig zu bezeichnen, da für diese Daten normalerweise kostengünstigere, aber dafür langsamere, optische Speichermedien verwendet werden.

Durch die Dual Level Methode ist es allerdings möglich einen hohen Detaillierungsgrad und gleichzeitig schnelle Antwortzeiten bei geringem Speicherplatzbedarf zu realisieren, da in dem eigentlichen Data Warehouse aufgrund der höheren Granularität geringere Datenmengen zu verwalten und zu bearbeiten sind.

Problem der Granularität ist es nicht, die unterschiedlichen Granularitätsgrade im Data Warehouse physikalisch zu realisieren, sondern zu erkennen, ab wann höhere Granularitätsgrade absolut notwendig für ein Warehouse sind.

Nach Inmon genügt das folgende Schaubild um zu erkennen, welcher Aufwand im Design des Data Warehouses betrieben werden muß[20]:

Planung 1 Jahr	Planung 5 Jahre	notwendiges Design
10.000.000	20.000.000	Dual Level Granularity und besonders sorgfältiges Warehouse Design
1.000.000	10.000.000	Dual Level Granularity
100.000	1.000.000	Besonders sorgfältiges Warehouse Design
10.000	100.000	Praktisch jedes Design funktioniert (keine besondere Sorgfalt notwendig)

Abbildung 22: Designkriterien [21]

Der Tabelle kann man das notwendige Warehouse-Design entnehmen, wenn nach Ablauf eines Jahres bzw. nach Ablauf von fünf Jahren eine bestimmte Anzahl von Datensätzen in dem Data Warehouse erwartet werden.

Z. B. bedeutet das für ein Data Warehouse bei einer erwarteten Größe von 1.000.000 Datensätze nach einem Jahr, daß man für dieses Warehouse direkt schon zu Beginn ein Design nach der Dual Level of Granulartiy Methode realisieren sollte, um die erwartete Datenmenge verwalten und effektiv verarbeiten zu können.
Zu erkennen ist, daß der Fünf-Jahres Horizont nicht einfach eine Multiplikation der Anzahl von Datensätzen des Ein-Jahres Horizont mit Fünf darstellt.

[19] In Anlehnung an W. H. Inmon (Building the Data Warehouse, 1996) S. 54
[20] W. H. Inmon (Building the Data Warehouse, 1996) S. 148

Inmon gibt hierfür folgende Gründe an:

* Im Laufe der Zeit von 5 Jahren wird die Verwaltung großer Datenmengen weniger Probleme bereiten, da innerhalb dieser Zeit enorme Fachkenntnisse ím Umgang mit dieser Datenmenge erworben werden.

* Die Hardwarekosten werden im hohen Maße sinken, so daß für das gleiche Geld mehr Hardwareressourcen zur Verfügung stehen und größere Mengen an Daten mit gleicher oder besserer Effektivität verarbeitet werden können.

* Leistungsstärkere Software-Tools werden erhältlich sein.

* Der End-User wird sich ebenfalls weiterentwickeln, und effektiver mit den Tools und dem Data Warehouse arbeiten.

Die Datensatzlänge spielt bei der Designauswahl keine Rolle. Es macht keinen Unterschied bei der Auswahl des Designs und der Granularität, ob ein Datensatz 25 oder 250 Byte lang ist.

Der Grund für die Vernachlässigung der Bytegröße liegt in der Indizierung der Datensätze.

Ein Index wird für jeden Primary-Key eines Datensatzes angelegt und ist als eine Art Inhaltsverzeichnis zu verstehen, aus dem direkt hervorgeht wo der Datensatz zu finden ist.

Abfragen über den Primary Key zur Ermittlung eines bestimmten Datensatzes werden durch den Index wesentlich schneller verarbeitet.

[21] Quelle: W. H. Inmon (Building the Data Warehouse, 1996) S. 148

Da für jeden Datensatz unabhängig von der Bytegröße ein Index erstellt wird und normalerweise Abfragen mit Hilfe des Indexes durchgeführt werden, kann man die Bytegröße als Kriterium zur Auswahl des geeigneten Designs getrost vernachlässigen.

Dagegen spielt die Anzahl der Datensätze eine wesentlich größere Rolle, da durch eine größere Anzahl von Datensätzen auch eine größere Anzahl von Indizes vorliegt und somit die Verarbeitungsgeschwindigkeit reduziert wird, da auch eine größere Anzahl von Indizes durchsucht werden muß.

Betrachtet man sich die oben aufgeführte Tabelle, so stellt man sich natürlich als erstes die Frage, wie man überhaupt die zu erwartende Datensatzanzahl bzw. die zu erwartende Warehousegröße in Byte ermittelt.

Auch hierzu bietet Inmon folgende Lösung:

Für jede bekannte Tabelle:
Wie groß (in Bytes) ist ein Datensatz
- maximale Vermutung
- minimale Vermutung

Für den 1 bzw. 5 Jahres Horizont
Welche maximale Anzahl an Datensätzen hält man für möglich
Welche minimale Anzahl an Datensätzen hält man für möglich
Für jeden Key einer Tabelle
Wie groß (in Bytes) ist der Key
Maximaler Speicherbedarf in Bytes = größter Datensatz in Bytes * max. Anzahl Datensätze
Minimaler Speicherbedarf in Bytes = kleinster Datensatz in Bytes * min. Anzahl Datensätze
plus Index Speicherplatz (Anzahl Indexeinträge * Keygröße in Bytes)

Abbildung 23: Speicherplatz-Kalkulation[22]

Das oben aufgeführte Ermittlungsverfahren ist getrennt für jede Tabelle des Data Warehouses durchzuführen, einmal für den Ein-Jahres Horizont und einmal für den Fünf-Jahres Horizont.

[22] Quelle: W. H. Inmon (Building the Datawarehouse, 1996) S. 146

Mit Hilfe dieses Verfahrens ist es nun möglich, die erwartete Anzahl an Datensätzen und den erwarteten Speicherplatzbedarf in Bytes pro Tabelle und somit für das gesamte Data Warehouse zu ermitteln.

Dabei erhält man zwei Werte, einmal einen maximalen Wert (der schlimmste anzunehmende Fall) und einen minimalen Wert (der günstigste anzunehmende Fall).

Diese zwei Einstufungen maximal und minimal ergeben sich aus der Tatsache, daß die Datensatzlänge in Byte nur sehr schlecht vorhersagbar ist, da in der Regel solch ein Datensatz aus Attributen mit einer statischen Bytelänge und Attributen mit einer dynamischen Bytelänge, z. B. Stringtypen wie VARCHAR2, besteht. Die dynamischen Attribute kann man somit nur mit einem minimal und maximal zu erwartenden Byte-Wert verrechnen.

Natürlich wird hier kein 100% iger Wert ermittelt, sondern auf Basis von Erwartungen eine grobe Berechnung durchgeführt. Um so besser die Einschätzung des Warehouse Designers, desto genauer wird auch die Berechnung.

Unterstützen läßt sich die Schätzung durch Informationsquellen wie z. B. interne, externe Marktanalysen, Trendberechnungen, Erwartungen der Geschäftsleitung, Vergangenheitswerte, usw.

Bei der Berechnung geht es in erster Linie nicht um die 100%ig genaue Berechnung der zu erwartenden Warehousegröße, sondern vielmehr um die Ermittlung einer groben Größenordnung anhand der man das notwendige Data Warehouse Design und die notwendige Hardware ausmachen kann.

4.4.2 Partitionierung

Zusätzlich zur Optimierung des Data Warehouse Designs mittels Granularitäten ist eine Optimierung durch eine sogenannte Partitionierung des Datenbestandes möglich.

Bei der Partionierung werden Fakttabellen des Data Warehouses redundanzfrei in mehrere separate, physikalische Teilbereiche (**Partitionen**) zerlegt. Die Teilbereiche können vollkommen unabhängig voneinander für Analysen genutzt werden.

Folgende Arten der Partitionierung sind möglich.

4.4.2.1 Horizontale Partitionierung

Bei der horizontalen Partitionierung wird eine Fakttabelle **zeilenweise** in entsprechende Partitionen zerlegt. Die Partitionierung kann z. B. nach Zeit, Region, Artikel, usw. durchgeführt werden.

Für alle zusammengehörigen Partitionen wird nur ein OLAP Objekt (Metrik) in dem OLAP Frontend-Tool definiert.

Alle Partitionen der Tabelle behalten die gleiche Tabellenstruktur (Attribute).

Beispiel:

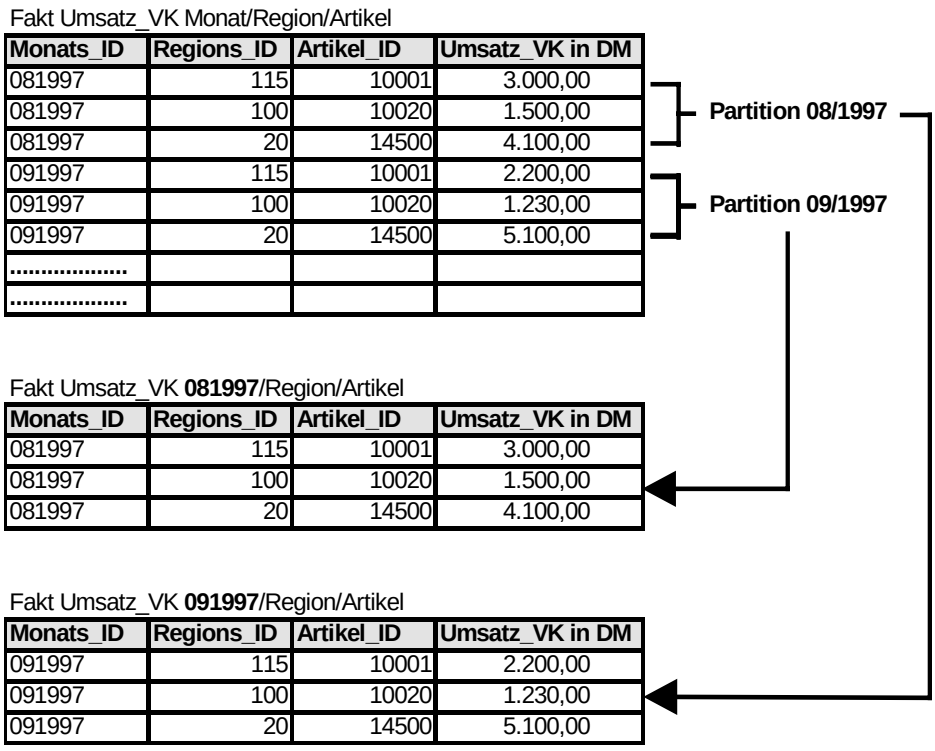


Abbildung 24: Horizontale Partitionierung nach Monat

In der Art, wie auf die horizontalen Partitionen zugegriffen wird unterscheidet man:

(1) Horizontale Partitionierung auf Datenbankebene

Bei der Partitionierung auf Datenbankebene ermöglicht das Datenbanksystem die Zerlegung einer Tabelle in mehrere Partitionen. ORACLE 8 ist eine der Datenbanken, die eine horizontale Partitionierung zuläßt.

Der Administrator der Datenbank gibt bei der Erstellung der Fakttabelle einen Wertebereich an, nach dem die Tabelle automatisch horizontal partitioniert wird (z. B. Partition 1 = 1997, Partition 2 = 1998, usw., siehe auch Kapitel 5.2.1) .

Der Endbenutzer spricht mit seinen Analysen immer nur ein OLAP Objekt (Metrik), welches genau eine Fakttabelle referenziert, an. Das Datenbanksystem ermittelt selbständig die notwendigen Partitionen der Fakttabelle und führt die Analyse auf genau diesen Partitionen durch.

(2) Horizontale Partitionierung auf Programm-Ebene

Bei der Partitionierung auf Programm-Ebene wird im Gegensatz zur Partitionierung auf Datenbank-Ebene für jede horizontale Partition eine eigene Tabelle erstellt.

Trotzdem wird jedoch innerhalb des OLAP Frontend-Tools nur ein Objekt für alle zusammengehörigen Partitionstabellen erstellt, welches selbständig (programmgesteuert) die Analyse auf die entsprechende Partitionstabellen lenkt.

Meistens ist dafür noch eine zusätzliche Tabelle im Data Warehouse notwendig, eine sogenannte **Partitions-Abbildungs-Tabelle**, aus der hervorgeht, welche Partitionstabelle welche Daten beinhaltet.

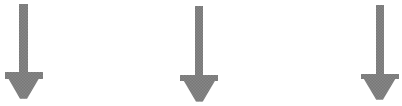
4.4.2.2 Vertikale Partitionierung

Bei der vertikalen Partitionierung wird die Fakttabelle **spaltenweise** in Teilbereiche zerlegt. Die Zerlegung wird z. B. in Anlehnung an unternehmensspezifische Sachverhalte bzw. an der unternehmenspezifischen Organisation durchgeführt.

Beispiel:

Fakt Tag/Artikel/Lagerbestand/Meldebestand/VK/EK

Tag_ID	Artikel_ID	Lagerbestand	Meldebestand	Stck-preis VK	Stck-Preis_EK
15.01.93	115	25000	8500	50	20
15.01.93	116	13000	4000	25	15
16.01.93	115	22000	8500	48	20
.....					



Fakt Lager

Tag_ID	Artikel_ID	Lagerbestand	Meldebestand
15.01.93	115	25000	8500
15.01.93	116	13000	4000
16.01.93	115	22000	8500
.....			

Fakt Vertrieb

Tag_ID	Artikel_ID	Lagerbestand	Stck-preis VK
15.01.93	115	25000	50
15.01.93	116	13000	25
16.01.93	115	22000	48
.....			

Fakt Einkauf

Tag_ID	Artikel_ID	Lagerbestand	Meldebestand	Stck-Preis_EK
15.01.93	115	25000	8500	20
15.01.93	116	13000	4000	15
16.01.93	115	22000	8500	20
.....				

Abbildung 25: Vertikale Partitionierung

Bei der vertikalen Partitionierung besitzen die einzelnen Teilbereiche unterschiedliche Tabellenstrukturen (Attribute). Für jedes vertikale Tabellenobjekt wird auch ein entsprechendes OLAP Objekt in dem OLAP Frontend-Tool erstellt.

Vorteile / Nachteile:

In erster Linie bedeutet die Partitionierung einen Zusatzaufwand beim Design des Data Warehouses. Weiterhin kann es sein, daß sich die Anzahl der benötigten Tabellen-Joins für eine Abfrage erhöht, da die Abfrage jetzt nicht mehr mit einer Fakttabelle, sondern nur über mehrere Fakttabellen zu beantworten ist. Folge davon ist eine höhere Antwortzeit des Data Warehouses. Den Nachteilen der Partitionierung stehen jedoch auch eine Vielzahl von Vorteilen gegenüber:

* Für den Data Warehouse Administrator bedeutet die Partitionierung eine wesentlich einfacher zu administrierende Datenbank. Backup, Reoganisation, Restrukturierungen beziehen sich immer auf relativ kleine Tabellen und sind dementsprechend schneller durchzuführen.

* Durch eine Vielzahl kleinerer Tabellen ist bei Abfragen das sogenannte Parallel Query, also die gleichzeitige Abfrage auf mehreren Tabellen (ab Oracle 7.1) möglich, so daß Join-Abfragen schneller durchgeführt werden können.

  Mit Oracle 8 ist zusätzlich sogar ein Parallel Load mit Hilfe eines speziellen SQL Befehlssatzes möglich, der gleichzeitig mehrere Tabellen durch INSERT, UPDATE, DELETE bearbeiten läßt, und somit den Ladevorgang der operativen Daten in das Data Warehouse beschleunigt.

* Für den Endbenutzer bedeutet die Partitionierung ein besseres Antwortzeitverhalten der partitionierten Tabellen, da Analysen auf wesentlich kleinere Tabellen und somit auch auf kleinere Indexbereiche, durchgeführt werden.

  Evtl. wird durch die Partitionierung auch der Personenkreis eingegrenzt, der Analysen auf bestimmte Fakttabellen durchführt (z.B. Lagerverwalter auf die Lager Fakttabelle, Einkäufer auf die Einkaufs-Fakttabelle, usw.). Auch dadurch können bessere Antwortzeiten erzielt werden, da je nach Architektur die I/O Zugriffe auf verschiedene Speichermedien verteilt werden, und somit gleichzeitig durchgeführt werden können.

Alles in Allem gehört die Partitionierung wie die Granularität zu den wichtigsten Design-Aspekten des Data Warehouses.

5 Data Warehouse Datenbanktuning

Neben der Optimierung des Designs des Datenmodells eines Data Warehouses besteht noch die Möglichkeit durch datenbankspezifische Optimierungen die **Performance** (= Verarbeitungsgeschwindigkeit) des Data Warehouses zu verbessern.

Bei den nachfolgenden Methoden beziehe ich mich allein auf ORACLE 7.3 und ORACLE 8 Datenbanken, da natürlich jeder Datenbankhersteller eigene, **proprietäre** (= herstellerspezifische) Lösungen anbietet, die hier nicht alle aufgeführt werden können.

Selbstverständlich handelt es sich aufgrund der Vielfalt des Tuningthemas auch nur um eine kleine Auswahl von Tuningmaßnahmen, auf die hier besonders eingegangen wird.

5.1 ORACLE 7.3 Data Warehouse Features

5.1.1 B*-tree und Bitmapped Index

Jeder Datensatz einer Tabelle ist mit einer datenbankweiten, einzigartigen ROWID versehen. Anhand dieser ROWID kann ein Datensatz direkt angesprochen werden.

Ein Tabellen-Index ist ein Datenbankobjekt, ähnlich einem Inhaltsverzeichnis, mit dessen Hilfe die einzelnen Datensätze über ihre ROWID's sehr schnell gefunden werden können.

Tabellen-Indizes beinhalten eine Liste von Einträgen. Jeder Eintrag besitzt einen Schlüsselwert und die dazugehörige ROWID. Der Schlüsselwert ist ein

Spaltenwert eines Datensatzes, bzw. die Kombination von Spaltenwerten eines Datensatzes.

Die Einträge von Tabellen-Indizes wurden in ORACLE 7 Datenbanken bisher mittels eines sogenannten B*-tree Mechanismus gespeichert, der einen schnellstmöglichen Zugriff auf den Schlüsselwert des Tabellen-Indexes ermöglichte. Wenn der Schlüsselwert gefunden wurde, kann über die damit erhaltene ROWID direkt auf den entsprechenden Datensatz zugegriffen werden.

Voraussetzung für die Nutzung von Indizes ist jedoch, daß der Schlüsselwert in der Abfrage (**where .... and ...**) auch benutzt wird. Erst dann wird die Abfrage über den entsprechenden Tabellen-Index durchgeführt.

Verwendet man den Schlüsselwert des Indexes nicht in der Abfrage, so muß eine zeitaufwendigere Suche über die eigentliche Tabelle durchgeführt werden, bei dem jeder Datensatz auf die 'where .... and ....' Klausel hin überprüft wird (**Full Table Scan**).

In Oracle 7.3 wird ein eindeutiger B*-tree Index automatisch erstellt, wenn bei einer Tabellendefinition ein UNIQUE bzw. ein PRIMARY KEY constraint verwendet wird.

Zusätzliche B*-tree Indexe können über den folgenden Befehl explizit erzeugt werden:

create index ind_emp_ename on emp(ename);

B*-tree Indizes sollten bevorzugt für die Optimierung von Abfragen eingesetzt werden[23]:

* auf Tabellen, die häufig mit der gleichen 'where .... and ...' Klausel abgefragt werden, wobei die Abfragen maximal 2 - 4 % der gesamten Anzahl an Datensätzen der Tabelle zurückliefern.

* auf Tabellenspalten mit einer hohen **Kardinalität**. Die Kardinalität gibt die Anzahl der unterschiedlichen Werte in einer Tabellenspalte wieder. Um so höher die Kardinalität, um so mehr **distincte** (unterschiedliche) Werte gibt es (besonders Primary Key oder Unique Spalten).

* auf Tabellen, deren Inhalte nicht ständig mittels UPDATE, DELETE, INSERT verändert werden, da durch die Indizierung zusätzlich zur Änderung einer Datensatzes nun auch noch eine Änderung des Indexes durchgeführt werden muß. Dadurch dauern Änderungen in indizierten Tabellen länger als Änderungen innerhalb von Tabellen die nicht indiziert wurden.

* auf Tabellen zur Optimierung von Abfragen die relativ simple 'where .... and ... ' Bedingungen aufweisen. Also möglichst nur einfache 'where ....' Abfragen.

* auf Tabellenspalten, die nicht mittels einer Funktion (andere Funktion als MIN oder MAX) innerhalb der 'where .... and ....' Klausel angesprochen werden, da solche Abfragen nicht über den Index abgearbeitet werden.

[23] ORACLE Online Dokumentation 7.3.3 Kapitel 'Data Access Methods'

Eine schnellere Alternative zu dem herkömmlichen B*-tree Index stellt unter ORACLE 7.3 der Bitmap Index dar. Dieser Indextyp wurde speziell für den Einsatz in Data Warehouse Umgebungen entworfen.

In einem B*-tree Index werden Schlüsselwerte mit dazugehörigen ROWID's gespeichert. Die ROWID's verweisen auf Datensätze die diesen Schlüsselwert in der entsprechenden indizierten Spalte aufweisen.

Ein Bitmap Index speichert anstatt einer Liste von ROWID's sogenannte **Bitspalten**. Jedes Bit eines Bitmaps korrespondiert zu einer möglichen ROWID. Ist das Bit gesetzt (also 1) , dann besitzt der Datensatz mit der korrespondierenden ROWID den Schlüsselwert (siehe Abbildung 26). Eine sogenannte **Mapping** Funktion der Oracle 7.3 Datenbank sorgt für die Konvertierung der Bitposition zu der entsprechenden ROWID. Für jeden distincten Spaltenwert der indizierten Tabellenspalte wird genau ein Bitmap innerhalb des Bitmap Indexes angelegt.

Bitmap Indizes müssen explizit über den folgenden Befehl angelegt werden:

create bitmap index *person_region on person (region);*

Wenn man sich vorstellt, daß die Tabelle 'person' eine Spalte 'region' verwendet, die 4 unterschiedliche Werte aufweisen kann (Nord, Ost, Süd, West) dann wird der Bitmap Index folgendermaßen erstellt:

(1) Die Tabelle 'person' wird als erstes gescanned und die Werte der Spalte 'region' werden sortiert. Nach der Sortierung wird entschieden, wieviele Bitmaps für den Index benötigt werden. Pro distinctem Spalten-Wert ein Bitmap. In diesem Fall werden 4 Bitmaps für den Index benötigt: Nord-Bitmap, Ost-Bitmap, Süd-Bitmap, West-Bitmap.
(2) Im zweiten Schritt wird der Bitmap Index für die Tabelle 'person' nach den Erkenntnissen aus (1) erstellt:

Row	REGION	Nord-Bitmap	Ost-Bitmap	Süd-Bitmap	West-Bitmap
1	Nord	1	0	0	0
2	Ost	0	1	0	0
3	West	0	0	0	1
4	West	0	0	0	1
5	Ost	0	1	0	0
6	West	0	0	0	1
7	Süd	0	0	1	0
8	Nord	1	0	0	0
9	Ost	0	1	0	0
10	Süd	0	0	1	0

Abbildung 26: Bitmap Index[24]

Bitmap Indizes sollten bevorzugt zur Optimierung von Abfragen eingesetzt werden:

* auf Tabellen einer Data Warehouse Umgebung

* zur Optimierung von Abfragen mit sehr umfangreichen, komplexen where .... and ....' Klauseln, da im Gegensatz zu dem B*-tree Index solche Klauseln über den Bitmap Index sehr einfach und effektiv aufgelöst werden können.

* auf Tabellenspalten, die eine geringe Kardinalität besitzen, also sehr wenige distincte Spaltenwerte aufweisen, da für jeden Spaltenwert ein Bitmap erzeugt werden muß.

* zur Optimierung von Abfragen, die eine große Menge an Datensätzen zurückliefern.
* auf Tabellen deren Inhalte gar nicht, bzw. nur selten geändert werden, da die zusätzliche Pflege des Bitmap Indizes noch mehr Zeit benötigt als die Pflege eines B*-tree Indexes. Durch Einsatz

[24] Quelle: M. J. Corey, M. Abbey (ORACLE Data Warehousing, 1997) S. 141

des Bitmap Indizes in einer Data Warehouse Umgebung fällt dieser Nachteil aufgrund der Nicht-Volatilität kaum ins Gewicht.

Der Vorteil des Bitmap Index gegenüber dem herkömmlichen B*-tree Index liegt einmal in dem schnelleren Zugriff des Indexes auf die gesuchte ROWID und zum anderen in der Einsparung von Speicherplatz.

Teilweise benötigt der Bitmap Index nur 1/100 des Speicherplatzes eines B*-tree Index[25].

Die Vorteile der Bitmap Indizes entfalten sich allerdings erst ab einer Kardinalität von 0,1 % gemessen an der gesamten Datensatzanzahl einer Tabelle. D. h. bei einer Tabelle mit 100.000 Datensätzen und einer zu indizierenden Spalte mit 1.000 unterschiedlichen Werten hätte man eine Kardinalität von 1 %. Bei dieser Kardinalität empfiehlt ORACLE den herkömmliche B*-tree Index anstatt des Bitmap Indexes zu verwenden.

5.1.2 Star Queries

Beim einem Star Query handelt es sich um eine von der ORACLE 7.3 Datenbank verwendeten Optimierungstechnik, die gezielt das Joinen von Fakttabellen mit Dimensionstabellen beschleunigt.

Da in herkömmlichen relationalen Datenbanken immer nur zwei Tabellen über einen Join verbunden werden können, müssen Abfragen mit mehr als zwei zu joinenden Tabellen künstlich in eine Serie paarweiser Joins mit Hilfe von dynamischen Zwischentabellen aufgebrochen werden.

[25] M. J. Corey, M. Abbey (ORACLE Data Warehousing, 1997) S. 143

Die richtige Reihenfolge in der Abarbeitung der künstlich erzeugten, paarweisen Joins ist für die Verarbeitungszeit in einem Data Warehouse enorm wichtig.

Beispiel:

Folgende Tabellen müssen über eine Abfrage gejoined werden:

* Fakttabelle Verkauf
* Dimensiontabellen Artikel, Kunde, Zeit

Die Abarbeitung der paarweisen Joins ohne einen Star Query würde etwa so aussehen:

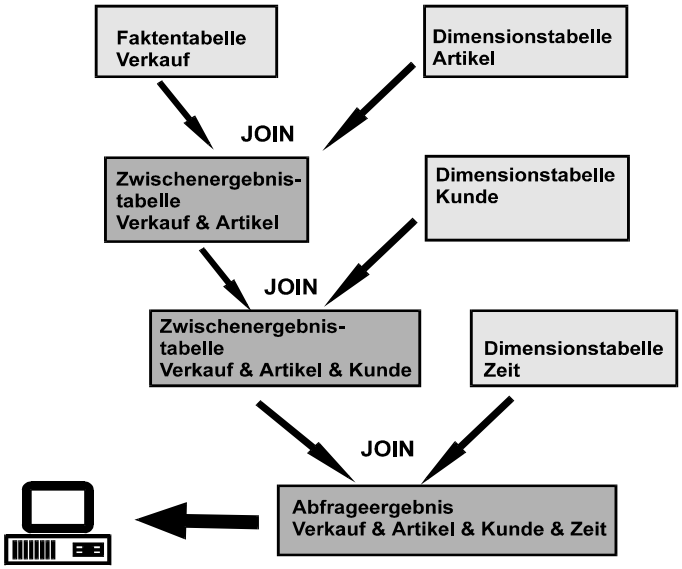


Abbildung 27: Abarbeitung von Abfragen ohne Star Query[26]

Bei der Abarbeitung dieses Joins ohne Star Query können die Tabellen mit den Zwischenergebnissen sehr groß werden. Im ungünstigsten Fall weisen sie ebenso viele Zeilen wie die Fakttabelle auf und benötigen damit erhebliche Systemressourcen. Der Aufbau solch einer umfangreichen Zwischener-

gebnistabelle kann sich zusätzlich als sehr zeitaufwendiges Unterfangen herausstellen.

Anders als bei der herkömmlichen Vorgehensweise wird beim Star Query die Abfrage vor Abarbeitung durch den sogenannten **ORACLE Optimizer** (ein integriertes Programm welches die optimale Ausführung von SQL-Abfragen sicherstellt) automatisch auf Dimensions- und Fakttabellen untersucht.

Danach wird zuerst das kartesische Produkt der an der Abfrage beteiligten Dimensionstabellen unter Beachtung der jeweiligen einschränkenden Bedingungen gebildet.

Bei einem **kartesischen Produkt** handelt es sich um einen Tabellen-Join zwischen zwei Tabellen, bei dem keine Join-Bedingung in der Form:

where tabelle1.spalte_x = tabelle2.spalte_y

angegeben worden ist. D. h., jeder Datensatz der Tabelle 1 wird mit jedem Datensatz der Tabelle 2 gejoined.

Erst nach der Bildung des kartesischen Produktes wird die entstandene Zwischenergebnistabelle mit der Fakttabelle gejoined (siehe Abbildung 28).

Voraussetzung für das Gelingen dieses Star-Queries ist die Gewährleistung, daß sich der Primary-Key der Fakttabelle aus den Primary-Keys der beteiligten Dimensionstabellen zusammensetzen läßt, um den abschließenden Join durchführen zu können.

[26] Quelle: P. Gluchowski (Antwortzeit als Erfolgsfaktor, 1998) Artikel in Datenbank Fokus 03/98

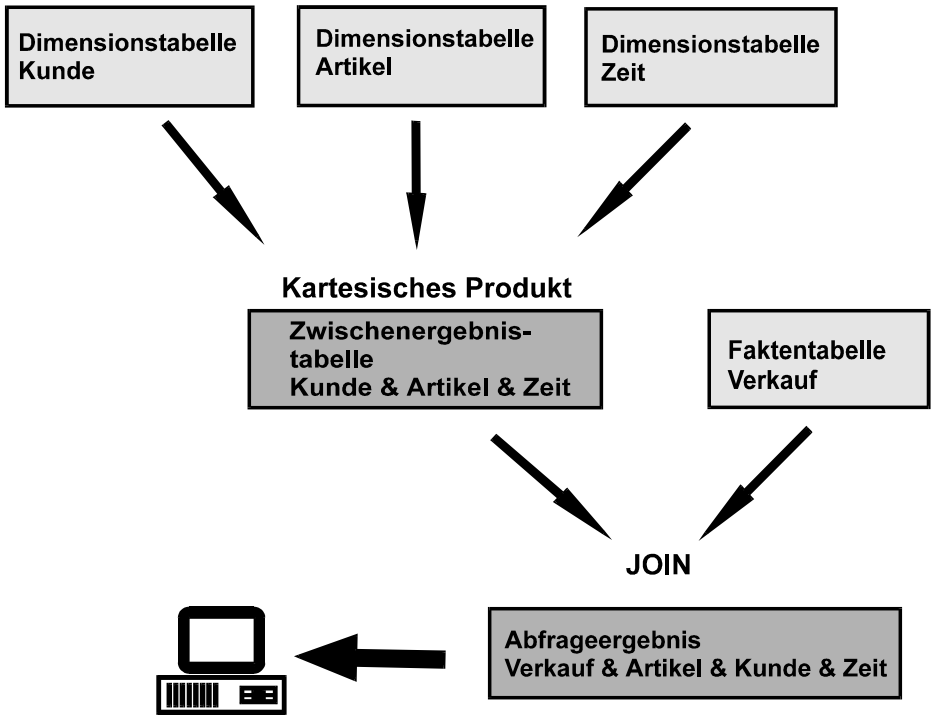


Abbildung 28: Abarbeitung von Abfragen mit Star Query Technik[27]

Im Idealfall kann man durch den Einsatz des Star-Queries bis zu 90 % der Bearbeitungszeit einsparen, vorausgesetzt, das kartesische Produkt wird durch die Abfragebedingungen ausreichend eingegrenzt. Ansonsten gestaltet sich die Bildung des Zwischenergebnisses über das kartesische Produkt als ebenfalls sehr aufwendig.

5.1.3 Parallel Query

[27] Quelle: P. Gluchowski (Antwortzeit als Erfolgsfaktor, 1998) Artikel in Datenbank Fokus 03/98

Ohne das Parallel Query Feature wird eine SQL-Abfrage (Query) durch einen einzelnen Server-Prozeß abgearbeitet.

Ein **Server-Prozeß** ist ein datenbankspezifischer Prozeß, der das SQL Statement von einem Benutzer-Prozeß entgegennimmt, das Statement auf Fehlerfreiheit überprüft und ausführt. Nach Ausführung liefert der Server-Prozeß das Ergebnis wieder an den aufrufenden Benutzer-Prozeß zurück.

Der **Benutzer-Prozeß** ist ebenfalls ein datenbankspezifischer Prozeß der erstellt wird, wenn ein Benutzer ein Anwendungsprogramm ausführt und mittels der Anwendung auf die Datenbank zugreift. Durch den Zugriff werden SQL-Statements an einen freien Server-Prozeß übergeben[28].

Beispiel:

Die Abarbeitung eines Full-Table Scans (z. B. select * from emp) würde ohne das Parallel Query Feature folgendermaßen aussehen:

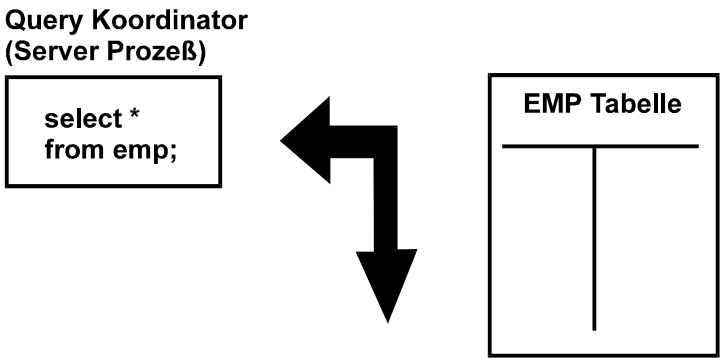


Abbildung 29: Full Table Scan ohne Parallel Query Feature[29]

Das Parallel Query Feature dagegen erlaubt die gleichzeitig (parallele) Abarbeitung einer SQL-Abfrage durch mehrere sogenannter Query-Prozesse.

[28] Oracle Schulungsunterlagen 'DBA ORACLE 7.3' Kapitel 1 Seite 23
[29] Quelle: Oracle Dokumentation 7.3.3 Kapitel 'Parallel Query Concepts'

Dabei übernimmt ein **Query-Koordinator** die Steuerung der parallelen Abarbeitung.

Bei dem **Query-Koordinator** handelt es sich eigentlich um einen normalen Server-Prozeß, der jedoch zusätzlich zu seinen normalen Aufgaben die Zerlegung der gesamten Abarbeitung der SQL-Abfrage in Teilaufgaben, die Zuweisung der entstandenen Teilaufgaben zu den einzelnen Query-Prozessen und das Zusammenfassen der Teilergebnisse zu dem Gesamtergebnis übernimmt.

Die **Query Server** übernehmen hier die eigentliche Abarbeitung der SQL-Abfrage.

Beispiel:

Die Abarbeitung des gleichen Full-Table Scans (z. B. select * from emp) würde mit dem Parallel Query Feature folgendermaßen aussehen:

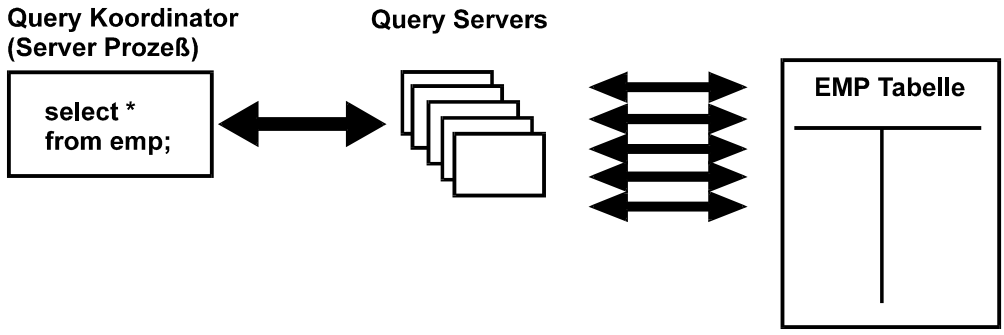


Abbildung 30: Full Table Scan mit dem Parallel Query Feature[30]

Die Anzahl der für die Abarbeitung verwendeten Query-Server wird auch als **degree of parallelism** bezeichnet, also der Grad, in welchem die SQL-Abfrage in parallele Teilaufgaben zerlegt wird.

[30] Quelle: Oracle Dokumentation 7.3.3 Kapitel 'Parallel Query Concepts'

Der Oracle Datenbank Server kann folgende SQL-Statements parallel abarbeiten:

* SELECT Statements
* Subqueries in CREATE TABLE, UPDATE, INSERT, DELETE Statements
  (z. B. CREATE TABLE emp_copy **as select * from emp**;)
* CREATE INDEX

Wie schon vorher erwähnt, ist der Oracle Optimizer für die optimale Abarbeitung eines SQL-Statements verantwortlich.

Nachdem der Optimizer den optimalsten Abarbeitungsplan erstellt hat, überprüft der Query-Koordinator zusätzlich das SQL Statement auf Operationen, die parallel verarbeitet werden können.

Danach zerlegt der Query-Koordinator die Gesamtverarbeitung in Teilaufgaben, legt eine Abarbeitungsreihenfolge der parallel zu verarbeitenden Operatoren fest und weist jeweils eine Teilaufgabe einem Query-Server zu.

Zu beachten ist jedoch, daß der Query-Koordinator seine Arbeit erst dann aufnimmt, wenn der Optimizer mindestens eine Full Table Scan Operation in dem SQL-Statement registriert.

Folgende Operationen können parallel ausgeführt werden:

* Sortierungen (z. B. order by .....)
* Tabellen-Joins

(z. B. select ..... from a, b where a.spalte_x = b.spalte_y)

* Tabellen Scans (z. B. select .... from ....)
* Tabellen Population (z. B. insert ... as select ...)
* Index Erstellung (z. B. create index ....)

Die Zerlegung der gesamten Abarbeitung durch den Query-Koordinator sieht so aus, daß für jede durchzuführende Operation die von der Operation betroffenen Datensätze horizontal partitioniert werden und die so entstandenen Partitionen auf die einzelnen Query-Server Prozesse zur Abarbeitung der Operationen verteilt werden.

Dabei werden immer nur zwei Operationen eines SQL-Statements gleichzeitig abgearbeitet. Danach die nächsten zwei, usw., bis sämtliche Operationen des Statements verarbeitet worden sind.

Beispiel:

```
select dname, max(sal), avg(sal)
from emp, dept
where emp.deptno = dept.deptno
group by dname;
```

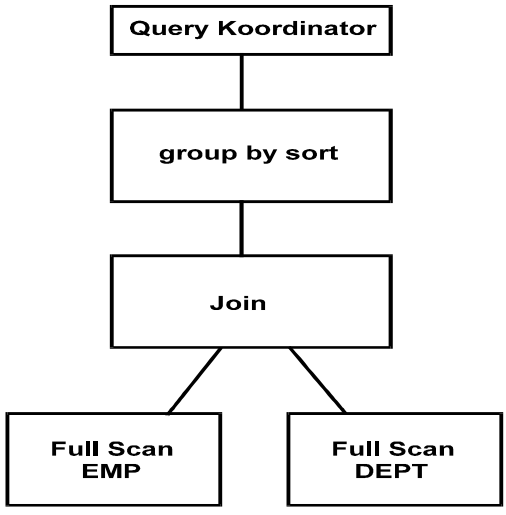


Abbildung 31: Flußdiagramm für die Abarbeitung eines SELECT-Statements[31]

Operationen die erst auf das Resultat der Abarbeitung anderere Operationen warten müssen bezeichnet man als **Parent Operations**. Die vorher zu erledigenden Operationen nennt man logischerweise **Child Operations.**

Anhand des Flußdiagramms erkennt man, daß die zwei Full Table Scan Operationen gleichzeitig durchgeführt werden. Erst danach wird die Join-Operation und die Group by Operation durchgeführt.

Der Degree of Parallelism, der für die Verarbeitung eines SQL-Statements verwendet wird, kann explizit festgelegt werden über:

* einen sogenannten Hint. Ein **Hint** ist nichts anderes als ein Kommentar in der SQL-Anweisung. Dieser Kommentar wird von dem ORACLE Optimizer ausgewertet und bei der Erstellung des optimalen Abarbeitungsplans berücksichtigt.
  z. B.: select **/*+PARALLEL(emp, 12) */** from emp ....

* über die Tabellendefinition in dem Data Dictionary, wenn eine Tabelle mittels Parallel Hint erzeugt wurde.
  z. B.: create table .... **parallel (degree 12)** as select .....

* über Parametereintragungen in der init.ora. Die **init.ora** ist ein Textfile, welches beim Start der ORACLE Datenbank gelesen wird und die darin aufgeführten Parameteranweisungen zur Einrichtung der Datenbankumgebung benutzt werden.
  z. B.: **PARALLEL_DEFAULT_SCANSIZE = 12**

Sind keinerlei Eintragungen zum Degree of Parallelism zu finden, dann betrachtet ORACLE die Anzahl der CPU'S des Datenbankservers und die An-

[31] Quelle: ORACLE Dokumentation 7.3.3 Kapitel 'Parallel Query Concepts'

zahl der Festplatten, auf denen die zu verarbeitende Tabelle verteilt ist. Der kleinere Wert beider Zahlen wird als Degree of Parallelism festgelegt.

Vorteile / Nachteile:

Das Parallel Query Feature der Oracle Datenbank ist besonders nützlich für SQL-Abfragen welche eine große Anzahl von Datensätzen betreffen, für Abfragen, die umfangreiche Join-Operationen durchführen, für die Erstellung großer Indizes, für umfangreiche Datenloads, usw. Diese Merkmale treffen alle auf ein Data Warehouse zu.

Trotzdem gibt es einige Bedingungen, die erfüllt sein müssen, damit das Parallel Query Feature den gewünschten Performanceschub leistet[32]:

* Multiprozessor Server
* Verteilung der Datenfiles der Datenbank auf viele Festplatten zur Verteilung des I/O Aufwandes
* Möglichst gering ausgelastete CPU-Resourcen (lt. Oracle Systeme mit einer typischen CPU Auslastung von weniger als 30 %)
* Umfangreicher Arbeitsspeicher um zusätzliche, speicherintensive Operationen wie z. B. Sortierungen durchführen zu können

Sollte eine oder mehrere dieser Bedingungen nicht erfüllt werden können, so kann man davon ausgehen, daß das Parallel Query Feature nicht die erhoffte Performancesteigerung bringen wird, evtl. sogar zu einer Performance-Reduzierung führt.

Ein weiterer nicht zu vernachlässigender Punkt ist die Tatsache, daß um so mehr User gleichzeitig mit dem Data Warehouse arbeiten, desto mehr

[32] ORACLE Dokumentation 7.3.3 Kapitel 'Parallel Query Tuning'

Query-Server benötigt werden, um die größere Anzahl von SQL-Statements parallel zu verarbeiten.

Dabei muß man natürlich die Grenzen der Hardwareressourcen beachten. Besonders der vorhandene Arbeitsspeicher und die vorhandenen I/O Schnittstellen.

Um so mehr User gleichzeitig SQL-Statements abschicken, desto mehr Arbeitsspeicher wird durch Query-Server belegt, desto mehr Lese-Operationen sind von den I/O Schnittstellen zu bewerkstelligen und desto weniger profitiert das Data Warehouse von einem Performancezuwachs.

Ab einer best. Anzahl von Usern wird die Performance sogar niedriger sein, als bei einem Einsatz des Data Warehouses ohne Parallel Query Feature.

5.2 ORACLE 8.0 Data Warehouse Features

Im Folgenden werden einige zusätzliche Data Warehouse Features der objekt-relationalen ORACLE 8.0 Datenbank erläutert.

Die schon erwähnten Features der ORACLE 7.3 Datenbank sind auch unter ORACLE 8.0 verfügbar.

5.2.1 Partitionierung

Unter ORACLE 8.0 ist eine horizontale Partitionierung von Tabellen und auch von Indizes mittels einem eigens dafür zur Verfügung stehenden Befehlssatzes möglich.

5.2.1.1 Partitionierung von Tabellen[33]

Die horizontale Partitionierung einer Tabelle wird durch folgenden **DDL** (**D**ata **D**efinition **L**anguage) Befehl durchgeführt:

```
create table Umsatz_1997_Fakt(
  Tag_ID        date,
  Kunde_ID number,
  Artikel_ID number,
  Umsatz_VK  number(*,2)
)
partition by range(Tag_ID)(
  partition Ums_199701          values less than (to_date('97-02-01', 'YY-MM-DD'))
    tablespace TS_199701
  partition Ums_ 199702         values less than (to_date('97-03-01', 'YY-MM-DD'))
    tablespace TS_199702,
            .....
  partition Ums_ 199712         values less than (to_date('98-01-01', 'YY-MM-DD'))
    tablespace TS_199712);
```

Durch diesen DDL-Befehl wird eine tagesbasierende Fakt-Tabelle 'Umsatz_1997_Fakt' mit 12 Partitionen ' von '**Ums**_199701' bis '**Ums**_199712' angelegt.

Die Einteilung der Partitionierung wird über die Tabellen-Spalte 'Tag_ID' durchgeführt. Datensätze mit einer Tag_ID < 97-02-01 werden automatisch der Partition 'Um**s**_199701' zugewiesen, Datensätze 97-01-30 < Tag_ID < 97-03-01 werden automatisch der Partition '**Ums**_199702' zugewiesen, usw.

[33] L. Leverrenz, R. Mateosian, S.Bobrowski ('Oracle8 Server Concepts', 1997) S. 8.1

Abfragen auf die partitionierte Fakttabelle können zum einen direkt über den Tabellennamen durchgeführt werden. Der Oracle Optimizer überprüft selbständig, welche Partitionen von der Abfrage betroffen sind. Dadurch werden unnötige Partitionen von dem Table-Scan ausgeschlossen (**pruning**):

```
select * from Umsatz_1997_Fakt
where Tag_ID between '97-01-01' and '97-02-01';
```

Zusätzlich können die Abfragen auch direkt auf die entsprechende Partition durchgeführt werden, da man den einzelnen Partitionen einer Tabelle eindeutige Namen vergeben kann:

```
select * from Umsatz_1997_Fakt partition (Monat_199701);
```

Vorteile / Nachteile:

Ein großer Vorteil der ORACLE Partitionierung liegt in der Zuweisung eines **Tablespaces** (ein logischer Speicherbereich für Objekte der Datenbank) zu einzelnen Partitionen.

Dadurch ist es möglich jede Partition in einem eigenem Tablespace zu speichern. Da jeder Tablespace wiederum auf verschiedenen Festplatten gespeichert werden kann ist somit eine effektive I/O Verteilung auf die Ressourcen möglich.

Aus diesem Grund kann ein enormer Performancevorteil für das Data Warehouse erzielt werden, da somit mehrere Schreib- und Lesevorgänge gleichzeitig getätigt werden können, vorausgesetzt sie beziehen sich auf unterschiedliche Partitionen einer Tabelle, bzw. eines Indexes.

Weiterhin sind Ad-hoc Abfragen, wie sie im Data Warehouse Umfeld sehr häufig durchgeführt werden, wesentlich schneller durchzuführen. Ohne eine Partitionierung müßte die komplette Fakttabelle 'Umsatz_1997_Fakt' mittels eines Full-Table-Scans durchlaufen werden.

Mittels der Partitionierung kann der Full-Table-Scan bei einer entsprechenden Abfrage (z. B. Umsätze_VK pro Artikel, pro Kunde für den **Monat 1997/02**) auf eine Partition begrenzt werden, was natürlich gerade bei tagesbasierenden Fakttabellen einen enormen Geschwindigkeitsvorteil bedeutet.

Die Pflege der erstellten Tabellen-Partitionen ist über den von ORACLE 8.0 zur Verfügung gestellten SQL-Befehlssatz möglich.

Durch einfache SQL-Befehle wird der Datenbankadministrator darin unterstützt zusätzliche Partitionen zu erzeugen, eine Partition in zwei Partitionen aufzusplitten, Partitionen zu löschen, Partitionen auf andere Tablespaces zu verlagern usw.

Das sind Tätigkeiten, die unter ORACLE 7.3 teilweise nur mit größerem Aufwand zu bewerkstelligen waren, da dort noch keine partitionierten Tabellen bzw. kein eigener Befehlssatz für Partitionierungen existierte.

Partitionierungen von Tabellen konnten unter ORACLE 7.3 durch zusätzliche Partition-Basis-Tabellen und einer sogenannten **Partition-View** erzeugt werden.

Dabei wurde die Partition-View über alle Partition-Basis-Tabellen gelegt. Gelesen wurde nur von der Partition-View und somit, da eine View nur eine Referenz auf Tabellen darstellt, von der entsprechenden Partition-Basis-Tabelle.

Leider war die Pflege solch einer Partition-View relativ aufwendig und teilweise konnten entsprechende Operationen und Abfragen nur mit sehr schlechter Performance auf diese View durchgeführt werden.

5.2.1.2 Partitionierung von Indizes[34]

Zusätzlich zu den Tabellen ist es auch möglich Indexe zu partitionieren. Man unterscheidet hier folgende Index-Typen:

(1) Lokale Indizes

Sie werden lokal für eine Tabellen-Partition erstellt. D. h. eine lokale Index-Partition beinhaltet nur Schlüsselwerte einer bestimmten Tabellen-Partition.

Man spricht hier auch von **equipartitionierten** Tabellen und Indizes.

Jede Index-Partition bezieht sich auf genau eine Tabellenpartition, wobei beide Objekte (Index sowie auch die Tabelle) nach dem gleichen Attribut partitioniert worden sind.

Änderungen der Partitionseinteilung der zugrunde liegenden Tabelle bewirken eine automatische Anpassung der dazugehörigen Index-Partitionen durch ORACLE.

Über den folgenden DDL-Befehl können lokale Indizes angelegt werden:

create index idx_umsatz_1997_fakt **on** umsatz_1997_fakt(Tag_ID, Kunde_ID, Artikel_ID)

```
local (   partition idx_ums_199701 tablespace ITS_199701,
      partition idx_ums_199702 tablespace ITS_199702,
      .....
      partition idx_ums_199712 tablespace ITS_199712);
```

Dabei ist zu beachten, daß die Anzahl und die Reihenfolge der Index-Partitionen mit der Anzahl und der Reihenfolge der Tabellenpartitionen übereinstimmen muß.

Läßt man die Angabe der Index-Partitionen weg, so übernimmt ORACLE die Einteilung und Bezeichnung der Index-Partitionen gemäß der zugrunde liegenden partitionierten Tabelle.

(2) Globale Indizes

sind **user-defined** partitionierte Indizes und werden nicht mit der zugrunde liegenden Tabelle equipartitioned. Globale Indizes können somit unabhängig von der zugrunde liegenden Tabelle partitioniert werden.

Über den folgenden DDL-Befehl werden Globale Indizes angelegt:

```
create index idx_umsatz_1997_fakt on umsatz_1997_fakt(Tag_ID, Kunde_ID,
Artikel_ID)
global partition by range(Tag_ID)(
 partition idx_Ums_1997_1  values less than (to_date('97-07-01', 'YY-MM-DD'))
    tablespace ITS_199701
  partition idx_Ums_1997_2 values less than (to_date('98-01-01', 'YY-MM-DD'))
    tablespace ITS_199702);
```

Partitioniert man den globalen Index gemäß der zugrunde liegenden Tabellen-Partitionierung, so liegt zwar eine Equi-Partitionierung zwischen Index und Tabelle vor, da der Index aber 'global' und nicht 'local' definiert wurde erkennt ORACLE die Equi-Partitionierung nicht. Folge davon ist,

[34] L. Leverrenz, R. Mateosian, S.Bobrowski ('Oracle8 Server Concepts', 1997) S. 8-19

daß Änderung an der Partitionierungsstruktur der zugrunde liegenden Tabelle nicht automatisch von ORACLE in dem dazugehörigen, global partitionierten, Index gepflegt werden.

Natürlich ist es auch möglich Bitmap Indizes anstatt der oben aufgeführten B*-tree Indizes zu partitionieren. Die Syntax zur Erstellung eines Bitmapped Indexes ist gleich der oben aufgeführten Syntax für B*-tree Indizes außer, daß dem Schlüsselwort INDEX das BITMAP Schlüsselwort vorangestellt wird. Weiterhin muß der partitionierte Bitmap Index unbedingt als ‚**local'** definiert werden. Einen ‚global' Bitmap Index unterstützt ORACLE 8.0 nicht.

Zur Erstellung eines partitionierten **Unique** Indexes muß das zusätzliche Schlüsselwort UNIQUE vor dem INDEX Schlüsselwort eingefügt werden. Durch einen Unique Index kann man sicherstellen, daß Werte in der indizierten Spalte der zugrunde liegenden Tabelle nicht doppelt vorkommen.

D. h. bei Eintragung eines neuen Datensatzes in die Tabelle bzw. bei einem Update eines indizierten Spaltenwertes eines Datensatzes wird der entsprechende Index durch ORACLE gepflegt und zusätzlich auf doppelte Einträge überprüft, wenn es sich um einen Unique Index handelt.

Vorteile / Nachteile:

Die Vorteile die für die Partitionierung von Tabellen gelten, beziehen sich auch auf die Partitionierung der Indizes. Auch dadurch wird zum einen die Abfrage-Performance im Data Warehouse gesteigert, und zum anderen ist es möglich, über Verteilung der Index-Partitionen auf eine Vielzahl von Festplatten, die I/O Last zu verteilen.

Auch bei den partitionierten Indizes wird der Datenbankadministrator durch einfache SQL-Befehle unterstützt, zusätzliche Partitionen zu erzeugen, eine

Partition in zwei Partitionen aufzusplitten, Partitionen zu löschen, Partitionen auf andere Tablespaces zu verlagern usw.

5.2.2 Parallel Execution

Das Parallel Execution Feature der ORACLE 8.0 Datenbank ist eine Erweiterung der Parallel Query Option der ORACLE 7.3 Datenbank.

Zusätzlich zu der Parallelisierung von SQL Queries (SELECT) ist es nun auch möglich DDL-Befehle (**CREATE TABLE** AS SELECT) und DML-Befehle (**INSERT, UPDATE, DELETE**) parallel zu verarbeiten.

Weiterhin können die meisten SQL-Statements für partitionierte wie auch für unpartitionierte Tabellen und Indexe parallel ausgeführt werden.

Der Vorgang der parallelen Verarbeitung hat sich im Vergleich zur ORACLE 7.3 Datenbank kaum verändert. Auch hier werden zur Verarbeitung ein **Parallel Koordinator** (bei 7.3 'Query Koordinator') und mehrere **Parallel Server Prozesse** (bei 7.3 'Query Server') verwendet.

Die Operation die parallelisiert werden soll (z. B. ein Table Scan, ein Table Update, die Erstellung eines Indexes, usw.) wird in sogenannte **Granules** (Teilstücke) zerlegt. Jeder Parallel Server Prozeß führt die entsprechende Operation nur auf ein Granule gleichzeitig aus.

Bei ORACLE 8.0 werden folgende zwei Granule-Arten unterschieden:

(1) Block Range Granules

Block Range Granules werden für die meisten Query Operationen (Table Scan) verwendet. Dabei wird eine Tabelle, bzw. ein Index in Blöcke zerlegt, die durch einen oberen und unteren ROWID Wert begrenzt werden.

D. h. jeder Parallel Server Prozeß führt die entsprechende Operation (z. B. einen Table Scan) nur auf eine begrenzte Anzahl von ROWID's innerhalb des ihm zugewiesenen Blockes parallel zu den anderen Parallel Server Prozessen durch.

Folgende Operationen können von ORACLE 8.0 per Block Range Parallelisierung durchgeführt werden:

* Queries die Table Scans verwenden (auch Subqueries in DML und DDL Befehlen wie z. B. INSERT INTO emp **SELECT ....**)
* der MOVE PARTITION Befehl
* der SPLIT PARTITION Befehl
* der REBUILD INDEX PARTITION Befehl
* der CREATE INDEX Befehl (nur bei nichtpartitionierten Indexen)
* der CREATE TABLE .... AS SELECT Befehl (nur bei nicht partitionierten Tabellen)

(2) Partition Granules

Partition Granules werden für die parallele Ausführung von Operationen verwendet, bei denen mehrere Partitionen einer Tabelle oder eines Indexes zu bearbeiten sind.

Bei dieser Form der Parallelisierung führt jeder Parallel Server Prozeß die entsprechende Operation immer nur auf eine komplette Partition durch. Jede Partition wird immer nur durch einen Parallel Server Prozeß gleichzeitig bearbeitet. Jeder Parallel Server Prozeß kann jedoch mehrere Parti-

tionen nacheinander bearbeiten. Das ist dann notwendig, wenn weniger Parallel Server Prozesse als Partitionen zur Verfügung stehen.

Dadurch wird natürlich die Flexibilität der parallelen Bearbeitung von Partitionen eingeschränkt, da, auch wenn mehr Parallel Server Prozesse als Partitionen zur Verfügung stehen, immer nur ein Parallel Server Prozeß eine Partition bearbeiten darf.

Folgende Operationen werden mittels Partition Granules parallel durchgeführt:

* UPDATE Befehl (nur partitionierte Tabelle)
* DELETE Befehl (nur partitionierte Tabelle)
* CREATE INDEX Befehl (nur partitionierte Indexe)
* CREATE TABLE AS SELECT Befehl (nur partitionierte Tabellen)
* INSERT .... SELECT (nur insert in partitionierte Tabellen)

Die Entscheidung des ORACLE Optimizers ob ein SQL Statement parallel ausgeführt wird oder nicht und mit welchem Degree of Parallelism, hängt bei ORACLE 8.0 von folgenden drei Möglichkeiten ab:

* **Parallel Clause**
  Das DDL SQL-Statement beinhaltet das PARALLEL Schlüsselwort:

```
create table emp(
   empno          number,
   ename          char(30))
PARALLEL (DEGREE 4);
```

* **Parallel Declaration**

Die das SQL-Statement betreffende Tabelle wurde mit dem PARALLEL Schlüsselwort erstellt:

create table emp ..... **PARALLEL (DEGREE 5);**

select * from **emp**;

* **Parallel Hint**
  Ein sogenannter Hint (Kommentar an den Optimizer) wird in dem SQL-Statement verwendet:

  select **/*+PARALLEL(emp, 12) */** from emp ....

Zu beachten ist zusätzlich, daß Queries (SELECT....) nur dann parallel verarbeitet werden, wenn der Optimizer in der Query mindestens einen Full Table Scan registriert.

UPDATE und DELETE Operationen können nur parallel abgearbeitet werden, wenn sie sich auf eine partitionierte Tabelle beziehen.

Bei INSERT ..... SELECT und CREATE ..... AS SELECT Operationen werden die Teile unabhängig voneinander auf die Parallelisierung hin überprüft und dann auch unabhängig von einander parallel ausgeführt. Allerdings wird ein einheitlicher Degree of Paralelism (Wahl des maximalen Degree Wertes der Teile) für die parallele Abarbeitung der Teile verwendet.

Die nachfolgende Abbildung 32 zeigt auf, welche SQL Statements in ORACLE 8 parallel verarbeitet werden können und wie diese Statements auf Parallelisierung durch den Optimizer überprüft werden.

Ist ein SQL-Statement durch mehrere Parallel-Execution Merkmale (Hint, Clause, Declaration) gekennzeichnet, dann geht der ORACLE Optimizer nach bestimmten Prioritäten bei der Entscheidung, ob das Statement parallel durchgeführt werden soll oder nicht vor.

Die Prioritätsstufen sind in Abbildung 32 durch die Zahlen (1) - (3) gekennzeichnet. Wobei Merkmale mit der Prioritätsstufe 1 als erstes Kriterium gewählt werden, Merkmale mit Prioritätsstufe 2 als zweites Kriterium, usw.
Z. B. würde der Optimizer bei einem UPDATE Statement als erstes nach einem Parallel Hint in der Form /*PARALLEL(emp, 4) */ Ausschau halten. Kommt der Parallel Hint in dem Statement nicht vor, dann wird der Optimizer die Tabellendeklaration betrachten. Wurde die Tabelle auch nicht mit dem PARALLEL Clause deklariert, dann wird das SQL-Statement nicht parallel abgearbeitet.

Parallel Operation	Parallelized by Clause, Hint, or Underlying Table/Index Declaration (priority order: 1, 2, 3)		
	Parallel Clause	Parallel Hint	Parallel Declaration
Parallel query table scan (partitioned or nonpartitioned table)		(1) PARALLEL	(2) of table
Parallel query index range scan (partitioned index)		(1) PARALLEL_INDEX	(2) of index
Parallel UPDATE/DELETE (partitioned table only)		(1) PARALLEL	(2) of table being updated or deleted from
Insert operation of parallel INSERT... SELECT (partitioned or nonpartitioned table)		(1) PARALLEL of insert	(2) of table being inserted into
Select operation of parallel INSERT... SELECT (partitioned or nonpartitioned table)		(1) PARALLEL of select	(2) of selecting table
Create operation of parallel CREATE TABLE... AS SELECT (partitioned or nonpartitioned table)	(1)	(Note: Hint in select clause does not affect the create operation.)	
Select operation of parallel CREATE TABLE ... AS SELECT (partitioned or nonpartitioned table)	(2)	(1) PARALLEL/ PARALLEL_INDEX	(3) of querying tables/ partitioned indexes
Parallel CREATE INDEX (parti-tioned or nonpartitioned index)	(1)		
Parallel REBUILD INDEX (nonpartitioned index)	(1)		
REBUILD INDEX(partitioned index)	Never parallelized		
Parallel REBUILD INDEX partition	(1)		
Parallel MOVE/SPLIT partition	(1)		

Abbildung 32: SQL-Statements für Parallel Execution[35]

Zu beachten ist, daß DML-Befehle (INSERT, UPDATE, DELETE) nur dann parallel durchgeführt werden, wenn die Datenbank Umgebung (**session**) mittels folgendem Befehl für dieses Feature eingestellt worden ist:

alter session enable PARALLEL DML;

Vorteile / Nachteile:

[35] Quelle: L. Leverenz, R. Mateosian, S. Bobrowski ('Oracle8 Server Concepts', 1997) S. 21-23

Vorteil des ORACLE 8.0 Parallel Execution Features ist zum einen, daß Abfragen parallel und somit wesentlich schneller zu verarbeiten sind. Dabei können nicht nur unpartitionierte Tabellen und Indizes parallel bearbeitet werden, sonder auch partitionierte.

Zum anderen ist es möglich den teilweise zeitaufwendigen Daten-Load von Daten des operativen Systems in das Data Warehouse durch die parallel durchzuführenden INSERT, UPDATE und DELETE Befehle wesentlich zu beschleunigen, besonders wenn für die Durchführung des Daten-Loads nur ein kleines Zeitfenster (maximaler Zeitraum, in der der Load durchgeführt werden kann) zur Verfügung steht. Dabei ist allerdings zu beachten, daß UPDATE und DELETE Statements nur auf partitionierte Tabellen parallel ausführbar sind.

Aber auch in dieser Version der ORACLE Datenbank gelten für den effektiven Einsatz des Parallel Execution Features die gleichen hohen Hardware-Voraussetzungen wie für die Version 7.3, was den effektiven Einsatz dieses Features einschränkt.

6 Die Datenübernahme in das Data Warehouse

Nachdem das Datawarehouse physikalisch und auch logisch modelliert, implementiert und vor allem optimiert wurde, müssen die entsprechenden Daten des operativen Systems "nur" noch in das Data Warehouse geladen werden.

Data Warehouse Spezialisten beziffern den Aufwand für die Implementation dieser Ladevorgänge auf etwa 80 % des gesamten Projektaufwandes.

Die erste Stufe der Datenübernahme in das Data Warehouse stellt die sogenannte Datenextraktion dar. Hier werden die gewünschten Daten für das Data Warehouse aus dem operativen Datenbestand **extrahiert** (herausgefiltert). Danach müssen diese Daten in der zweiten Stufe der Datenübernahme **transformiert** (umgewandelt) werden. Die Transformation kann entweder schon bei der Datenextraktion, oder aber erst in der dritten Stufe der Datenübernahme, der Integration der transformierten Daten in das Data Warehouse, stattfinden.

Dabei basiert die Datenübernahme immer auf dem sogenannten Snapshot Konzept.

6.1 Das Snapshot-Konzept

Bei einem Snapshot handelt es sich um einen Datenauszug aus dem operativen Datenbestand zu einem bestimmten Zeitpunkt. D. h. es wird ein Teil des operativen Datenbestandes bzw. der ganze operative Datenbestand in das Data Warehouse bzw. erst einmal in Datenfiles kopiert, wenn eine direkte Übernahme in das Warehouse technisch nicht möglich ist. Die Daten-

sätze der Datenfiles werden dann zu einem späteren Zeitpunkt in das Data Warehouse übernommen.

Ein Snapshot kann auf zwei Arten ausgelöst werden:

(1) Ein ereignisgesteuerter Snapshot

Dieser Snapshot wird durch eine Transaktion im operativen System ausgelöst. Solch eine Transaktion könnte z. B. ein Warenverkauf sein, eine Änderung einer Kundenanschrift, eine Buchung eines Wareneinganges, usw. Dabei wird der entstandene Datensatz des operativen Systems um eine Zeitangabe erweitert und sofort in das Data Warehouse oder in ein Datenfile kopiert. Dadurch wird durch ein Snapshot immer nur ein Datensatz abgebildet.

(2) Ein zeitgesteuerter Snapshot

Bei einem zeitgesteuerten Snapshot wird das Abbild des operativen Datenbestandes regelmäßig zu einem bestimmten Zeitpunkt erstellt. Z. B. könnte jeden Abend nach Beendigung des Tagesgeschäftes ein Snapshot über die Tagesverkäufe erstellt werden.

D. h., ein zeitgesteuerter Snapshot betrifft in der Regel mehrere Datensätze die in einem bestimmten Zeitraum durch Transaktionen erstellt wurden.

Ein **Snapshotdatensatz** weist in der Regel folgende Bestandteile auf:

* eine Zeitangabe (z. B. 14.04.98 14:05:35)
* einen Schlüssel (z.B. die Transaktionsnummer eines Warenverkaufes)
* Primäre Daten die in Beziehung zum Schlüssel stehen (z. B. Artikel, Verkaufsmenge, Einzelpreis, usw.)

* Sekundäre Daten die keine direkte Beziehung zum Schlüssel besitzen
  (z. B. Lagerbestand des Artikels zum Zeitpunkt des Verkaufs)

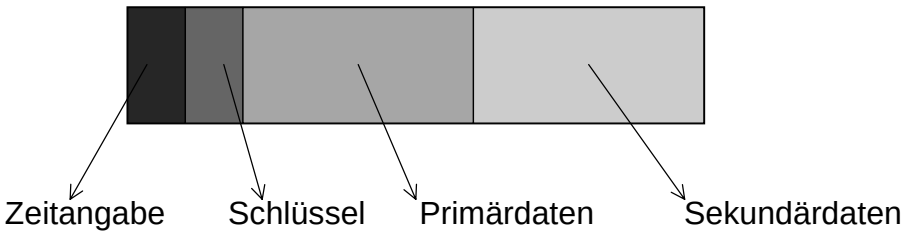


Abbildung 33: Aufbau eine Snapshot Datensatzes[36]

Die Zeitangabe kann entweder der Zeitpunkt des auslösenden Ereignisses sein, oder aber der Zeitpunkt der Datenübernahme aus dem operativen Datenbestand in ein Datenfile bzw. in das Data Warehouse.

Der Schlüssel des Snapshot Datensatz kann aus einem einzelnen Feld oder aus mehreren Feldern bestehen und identifiziert eindeutig die Primärdaten.

Die Primärdaten enthalten Daten die sich direkt auf den Schlüssel beziehen wie z. B. Artikelbeschreibung, Artikelfarbe, Artikeleinheit zu der Artikelnummer.

Die Sekundärdaten hingegen haben keine direkte Beziehung zu dem Snapshotschlüssel wie z. B. der Lagerbestand des verkauften Artikels zu der Transaktionsnummer der Verkaufstransaktion und sind als optional zu bezeichnen.

Vorteile / Nachteile

[36] W. H. Inmon (Building the Data Warehouse, 1996) S. 120

Ereignisgesteuerte Snapshots stellen erst einmal eine zusätzliche Belastung für das operative System dar, da zusätzlich zur Verarbeitung der eigentlichen Transaktion auch noch der Snapshot erstellt werden muß.

Ein besonderer Vorteil der ereignisgesteuerten Snapshots liegt in der Tatsache, daß durch diese Snapshots auch immer nur die neuesten Änderungen des operativen Datenbestandes betroffen sind und direkt im Data Warehouse bzw. erst in einem Datenfile integriert werden. Eine Überprüfung, ob der Datensatz schon im Data Warehouse eingepflegt wurde, entfällt hier.

Zeitgesteuerte Snapshots können hingegen zu einem Zeitpunkt durchgeführt werden, in dem sie das operative System nicht zusätzlich behindern (z. B. Nachts).

Man spricht hier meistens von sogenannten **'Zeitfenstern'**, also die Zeiträume, in denen Verarbeitungen über das operative System laufen können, ohne das operative Tagesgeschäft zu stören.

Auf der anderen Seite kann es problematisch werden die Datensätze zu identifizieren, die noch nicht in das Data Warehouse durch einen vorherigen Snapshot integriert wurden. Folge davon sind zeitaufwendige Snapshot-Vergleiche bzw. Überprüfungen, ob bestimmte Datensätze schon im Data Warehouse existieren.

Die meisten der heutigen relationalen Datenbanken besitzen zur Unterstützung des Snapshotverfahrens einen eigenen Befehlssatz, durch den die Datenextraktion vereinfacht und automatisiert werden kann.

Der eigentliche Anwendungsfall für die Snapshottechnik in relationalen Datenbanken liegt allerdings in der Bereitstellung von **Tabellenreplikaten** (doppelte Speicherung einer Tabelle) innerhalb einer verteilten operativen Datenbank. D. h., daß operative Daten mehrfach an unterschiedlichen Stel-

len des Netzwerkes gespeichert werden, um für Leseoperationen die Antwortzeiten bei Anfragen nach eigentlich auf einem entfernten Netzknoten liegende Daten zu verringern[37].

Mit Hilfe der Snapshot-Technik werden also Daten des operativen Datenbestandes extrahiert, sprich einzelne Datensätze werden kopiert. Dieser Vorgang wird als **Datenextraktion** bezeichnet und stellt die erste Stufe der Datenübernahme in das Data Warehouse dar.

6.2 Die Datenextraktion

Die Datenextraktion stellt den eigentlichen Datenload, das Extrahieren der notwendigen operativen Daten für die Übernahme in das Data Warehouse, dar. Extrahierte Daten werden in der Regel nicht direkt in den entsprechenden Data Warehouse Tabellen gespeichert, sondern entweder erst in separate Tabellen des Data Warehouses, bzw., wenn das technisch nicht möglich ist, in separate Datenfiles.

Durch diese Zwischenspeicherung ist es möglich fehlerhafte Datenübernahmen einfacher zu wiederholen und den Fehlergrund herzuleiten. Weiterhin können die nachfolgenden Datenübernahmestufen, Datentransformation und -integration, außerhalb des operativen Systems durchgeführt werden, so daß die Ressourcen des operativen Systems wieder frei für das Tagesgeschäft sind.
Man unterscheidet grundsätzlich zwei Arten des Datenloads:

(1) den Initialisierungs-Load (= Initial Load)

Bei einem Initialisierungs-Load wird die erste Datenübernahme in das bis dahin noch leere Data Warehouse durchgeführt. In der Regel wird dieser

[37] P. Chamoni, P. Gluchowski (Analytische Informationssysteme, 1998) S. 95

Load kurz vor Produktivsetzung des Data Warehouses ausgeführt. Dabei werden alle betroffenen Daten des operativen Datenbestandes, sowie evtl. archivierte, operative Daten in das Data Warehouse übernommen.

Dieser Load ist eine sehr umfangreiche Datenübernahme, da hier teilweise Daten mit einem Zeithorizont von mehreren Jahren übernommen werden müssen.

Trotz des hohen Datenvolumens ist dieser Load als eher unproblematisch zu bezeichnen.

Die Daten des operativen Systems werden in Datenfiles oder in separate Tabellen des Data Warehouses gespeichert und können dort in der zweiten Stufe der Datenübernahme transformiert werden, ohne das operative System zu stören.

Dieser Datenload wird in der Regel nur einmal in dem Lebenszyklus des Data Warehouses durchgeführt, so daß auf aufwendige und somit teuere Vorgehensweisen zur effektiven Verarbeitung des Loads meistens verzichtet werden kann.

(2) der zyklische und permanente Load (= Refresh Load)

Der zyklische Load findet regelmäßig ab dem Zeitpunkt der Produktivsetzung des Data Warehouses statt.
Im Laufe der Zeit wird der operative Datenbestand durch das Tagesgeschäft erweitert und modifiziert. Die Änderungen und Erweiterungen müssen natürlich regelmäßig in das Data Warehouse übertragen werden, um auch diese Daten analysieren zu können.

Bei dem zyklischen Load wird nach dem zeitgesteuerten Snapshot Konzept vorgegangen. Der permanente Load basiert auf dem ereignisgesteuerten Snapshot Konzept (siehe auch Kapitel 6.1).

Im Gegensatz zu dem Initialisierungs-Load muß hier ein wesentlich größerer Aufwand investiert werden, um diese Datenübernahme zu automatisieren und sie so effektiv wie möglich zu gestalten, da diese Form der Datenübernahme regelmäßig durchgeführt werden muß.

Gerade bei dem zyklischen Load besteht allerdings das große Problem festzustellen, welche der Datensätze noch nicht in dem Data Warehouse vorliegen und somit für die nächste Datenübernahme extrahiert werden müssen.

Inmon unterscheidet fünf Techniken mit deren Hilfe man solche Änderungsdaten ermitteln kann, um das Volumen der zu übernehmenden Daten so gering wie möglich zu halten[38]:

(1) Ermittlung über Timestamps

Wird ein Datensatz der gerade neu erzeugt bzw. geändert wurde mit einer zusätzlichen Zeitangabe (Datum, Uhrzeit) wann diese Transaktion durchgeführt wurde versehen, dann spricht man von einem Timestamp (Zeitstempel).

Wenn Applikationen des operativen Systems ihre Daten mit einem Timestamp versehen, ist es sehr einfach nur die Datensätze zu ermitteln, die seit dem letzten Datenload ergänzt bzw. geändert wurden. Dabei wird bei einem neuen Datenload das Datum und die Uhrzeit des letzten Datenloads mit den Timestamps der Datensätze verglichen. Nur die Datensätze die einen jüngeren Timestamp als das Vergleichsdatum aufweisen müssen noch in das Data-Warehous übernommen werden.

[38] W. H. Inmon (Building the Data Warehouse, 1996) S. 77

(2) Ermittlung mit Hilfe von Delta Files bzw. Protokoll-Tabellen

Ein Delta File ist ein File in das eine Applikation neue Datensätze, bzw. Änderungen von Datensätzen, zusätzlich zu der Speicherung in einer Datenbank, schreibt. Das gleiche gilt für Protokoll-Tabellen, nur daß hier diese relevanten Datensätze in gesonderten Datenbanktabellen gespeichert werden (siehe auch Kapitel 6.1 Snapshot Konzept).

D. h., diese Files oder Tabellen enthalten nur die Datensätze, die bei der nächsten Datenübernahme in das Data Warehouse transportiert werden müssen.

(3) Ermittlung mit Hilfe von Logdateien

Zur Wahrung der Datensicherung führen die meisten Datenbanksysteme sogenannte Logdateien. In diesen Logdateien werden sämtliche Transaktionen der Datenbank gespeichert, so daß die Datenbank im Falle eines Systemabsturzes bzw. eines Hardwaredefektes auf den letzten, vollständigen (**konsistenten**) Zustand zurückgesetzt werden kann. In diesen Files werden also neue Datensätze und auch Änderungen von Datensätzen gespeichert.

Leider werden diese Files in der Regel in einem herstellerspezifischen (**proprietären**), nicht standardisierten Format gespeichert, so daß ein einfacher Zugriff auf diese Daten nicht möglich ist.

Weiterhin werden in diesen Files eine Vielzahl von zusätzlichen Daten gespeichert, die für den eigentlichen Datenload vollkommen irrelevant sind.

(4) Ermittlung durch Änderung der operativen Applikationen

Eine weitere Möglichkeit die relevanten Daten für die nächste Datenübernahme kenntlich zu machen besteht in der Anpassung der Applikationen des operativen Systems.

Dabei könnte z. B. eine Applikation so abgeändert werden, daß sie die Daten mit einem Timestamp versieht, oder daß die Daten in ein Delta File geschrieben werden.

Dies ist allerdings nicht gerade die optimalste Lösung, da die meisten Applikationen sehr alt und unübersichtlich aufgebaut sind. Eine Änderung solcher Applikationen kann sehr aufwendig werden und ist in erster Linie zu vermeiden.

(5) Selektion durch Vergleich eines 'Vorher' und eines 'Nachher' Snapshots

Hierbei werden bei jedem Datenload zwei Snapshots der operativen Datenbasis miteinander verglichen. Ein aktueller Snapshot zum Zeitpunkt der Datenübernahme (**Nachher Snapshot**) und ein alter Snapshot zum Zeitpunkt des vorherigen Datenloads (**Vorher Snapshot**).

Durch den Vergleich dieser Snapshots kann man die in der Zwischenzeit durchgeführten Transaktionen und somit die relevanten Datensätze für den aktuell durchzuführenden Datenload ermitteln.

Natürlich ist diese Lösung sehr aufwendig, da hier zwei komplette operative Datenbestände miteinander verglichen werden müssen. Aufgrund dieses hohen Aufwandes empfiehlt es sich, erst einmal nach einer effektiveren Lösung zu suchen.

Hat man die relevanten Daten erst einmal extrahiert, dann beginnt die zweite Stufe der Datenübernahme, die sogenannte Transformation dieser Daten.

6.3 Die Datentransformation

Natürlich ist es nicht damit getan, die Daten des operativen Systems zu extrahieren. Dadurch daß die operativen Daten in unterschiedlichen Anwendungen erzeugt und dementsprechend **anwendungsspezifisch** gespeichert werden, ist bei dem Daten-Load eine Vereinheitlichung der Daten aus den unterschiedlichsten operativen Anwendungen unerläßlich.

Als die einzelnen Anwendungen des operativen Systems erstellt wurden hat sich noch niemand Gedanken darüber gemacht, wie man die Daten aus all diesen verschiedenen Anwendungen miteinander kombinieren kann. Diese Forderung war damals auch noch nicht notwendig, da die mit einer Anwendung erzeugten Daten normalerweise auch nur mit dieser Anwendung weiter bearbeitet wurden.

Dadurch wurden natürlich Daten erstellt, die eigentlich die gleiche Bedeutung haben, aber in den unterschiedlichen Anwendungen unterschiedlich codiert wurden.
Beispiel: Das Geschlechtskennzeichen[39]

Applikation A	**m, w**
Applikation B	**1, 0**
Applikation C	**x, y**
Applikation D	**männlich, weiblich**

Weiterhin wurden Tabellenspalten in den verschiedenen Anwendungen unterschiedlich bezeichnet, obwohl sie die gleichen Daten enthalten, bzw. die

[39] W. H: Inmon (Building the Data Warehouse, 1996) S. 75

Tabellenspalten erhielten die gleiche Bezeichnung, obwohl sie unterschiedliche Daten beinhalten.

Ebenso kann es sein, daß die gleichen Kennzahlen in den verschiedenen Anwendungen in unterschiedlichen Maßeinheiten und Währungseinheiten gespeichert werden. Z. B. werden die Umsätze in der einen Anwendung in DM und in der anderen Anwendung in Dollar gespeichert. Produktlängen werden einmal in Zentimeter und einmal in Meter gespeichert.

Zusätzlich zu diesen semantischen Unterschieden zwischen den einzelnen operativen Anwendungen kommen noch die technischen Unterschiede hinzu. Gleiche Tabellenattribute können in den verschiedenen operativen Anwendungen unterschiedliche Datentypen aufweisen. Z. B. wird die Produktnummer in Applikation A als NUMBER Wert gespeichert, und in der Applikation B als VARCHAR2. Oder aber die Datentypen werden in unterschiedlicher Genauigkeit gespeichert. Denkbar ist auch, daß Daten mit unterschiedlichen Zeichensätzen, z. B. ASCII und EBCDIC, gespeichert wurden.

Alle diese **Integrationsprobleme**, also Probleme die auftauchen wenn man Daten aus unterschiedlichen operativen Systemen miteinander kombiniert, um sie als einheitliche Datenbasis in dem Data Warehouse zu speichern, müssen mittels geeigneter **Transformation** (Umwandlung) behoben werden, damit die Daten miteinander vergleichbar und verrechenbar werden.

Die Transformation kann entweder direkt während der ersten Stufe der Datenübernahme durchgeführt werden. D. h., die relevanten Daten werden extrahiert, transformiert und dann in Datenfiles oder Tabellen gespeichert. Oder die Transformation findet in der dritten Stufe, der Datenintegration, statt. Dabei werden die relevanten Daten aus den Datenfiles bzw. den entsprechenden Tabellen gelesen, transformiert und in dem Data Warehouse integriert.

Bei der Transformation legt man z. B. fest, daß die Geschlechtsausprägung nur noch durch 'm' und 'w' im Data Warehouse repräsentiert wird. Alle Geschlechtskennzeichen des operativen Systems die dieser Codierung nicht entsprechen werden in ihrer alten Codierung eingelesen und nach einer bestimmten, programmierten Logik in der neuen Codierung im Data Warehouse gespeichert (**Transformation**). D. h., die Daten werden selbstverständlich nur im Data Warehouse einheitlich gespeichert. Die Daten des operativen Systems bleiben von dieser Vereinheitlichung unberührt, so daß der Transformationsprozeß bei jedem neuen Daten-Load notwendig wird.

Die Transformation dient zusätzlich dazu, die geplante Granularität im Data Warehouse schon während des Lade-Vorganges der Daten sicherzustellen. D. h., Daten aus dem operativen System, welche normalerweise in einer sehr niedrigen Granularität vorliegen (z. B. Umsatz pro Tag), werden während des Ladevorganges auf höhere Granulariäten verdichtet (z. B. Umsatz pro Monat) und in den entsprechenden Data Warehouse Fakttabellen gespeichert.

Ist der Vorgang der Datentransformation abgeschlossen, wird mit der letzten Stufe der Datenübernahme, der Datenintegration begonnen.

6.4 Die Datenintegration

Bei der Datenintegration werden die schon extrahierten und evtl. transformierten Daten nur noch in den entsprechenden Data Warehouse Tabellen gespeichert.

Der Vorgang der Datenintegration findet vollständig in der Data Warehouse Umgebung statt, so daß das operative System durch diese Stufe der Datenübernahme nicht mehr betroffen ist.

Als Datenquelle für die Datenintegration dienen entweder separate Datenfiles oder separate Tabellen des Data Warehouses in die die relevanten Daten während der Datenextraktion gespeichert wurden.

Sind diese Daten noch nicht transformiert, so muß diese Transformation zeitgleich mit der Datenintegration durchgeführt werden. D. h. die Daten werden aus den Datenfiles oder Tabellen gelesen, transformiert und sofort in das Data Warehouse integriert.

Mit dem Abschluß der Datenintegration ist auch gleichzeitig die Datenübernahme abgeschlossen. Das Data Warehouse ist aktualisiert und steht wieder für Analysen zur Verfügung.

7 Data Warehouse Prototyp (Fa. Oppenhoff & Rädler)

7.1 Aufgabenstellung

Ziel ist der Aufbau eines ROLAP Data Warehouse Prototyps durch den zum einen die ROLAP Data Warehouse Technik dargestellt werden soll, und zum anderen die ROLAP Data Warehouse Entwicklungsumgebung der Firma MicroStrategy vorgestellt wird.

Bei einem **Prototyp** handelt es sich nur um eine Vorablösung, mit deren Hilfe die Lösbarkeit eines Problems ermittelt und die Art der Lösung möglichst schnell dargestellt werden kann. Dabei werden normalerweise Testdaten des Auftragebers verwendet, so daß ein direkter Bezug zu dem Problem hergestellt wird und für den Auftraggeber erkennbar ist, wie eine endgültige Lösung aussehen wird. Dabei wird natürlich nur ein kleinerer Ausschnitt des Gesamtproblems behandelt, der aber repräsentativ für den restlichen, noch zu erledigenden Aufwand steht.

Für die Realisierung des Prototyps wird eine objekt-relationale Datenbank der Firma ORACLE Version 8.0.5 verwendet, sowie die Windows NT Produkte DSS Architect und DSS Agent der Firma MicroStragtegy, Version 5.1. Dabei handelt es sich bei dem Tool **DSS Architect** um eine 16-Bit Applikation mit deren Hilfe das multidimensionale Modell und somit die notwendigen semantischen Metadaten (siehe Kapitel 4.2.2) erstellt werden. Auf dieses Modell wird mittels des Tools **DSS Agent** zugegriffen (ebenfalls 16-Bit Applikation), welches zum einen für die Entwicklung der OLAP Analysen verwendet wird und zum anderen als Auswertungstool dieser OLAP Analysen von dem Endbenutzer verwendet wird. Dabei verwendet dieses Tool eine spezielle **SQL–Engine** der Firma MicroStrategy, die es ermöglicht aus den OLAP Analysen optimierte SQL-Statements zu generieren, welche auf der Daten-

bank die gewünschten Analysen ermöglichen. Die SQL Erzeugung geschieht durch Nutzung der mittels DSS Architect erstellten Metadaten automatisch, so daß der Endbenutzer keinerlei SQL-Kenntnisse benötigt (siehe auch Kapitel 4.1).

Für den Prototypen werden OLAP Berichte zur Lösung der nachfolgenden betriebswirtschaftlichen Fragestellungen gewünscht:

- Für welche Aktensachgebiete werden die meisten Zeitaufwände verbucht und welche Aufwandssachgebiete beinhalten diese Aktensachgebiete ?
- Auf welche Userarten wurden die meisten Zeitaufwände gebucht?
- Welche Umsätze wurden wann und mit welchen Aktensachgebieten erzielt ?
- Wie (nach welchen Stati) teilen sich die Zeitaufwände bzw. die Umsätze auf ?

Für die Erstellung des Oppenhoff & Rädler Prototyps sind folgende Arbeitsschritte zu erledigen:

- Analyse des operativen Datenmodells (**Kapitel 7.4**)
- Planung des multidimensionalen Modells (**Kapitel 7.5**)
- Planung des notwendigen ROLAP Datenmodells (**Kapitel 7.6**)
- Aufbau der Warehouse Datenbank (**Kapitel 7.7**)
- Implementierung der notwendigen Datenextraktions und -transformations Prozeduren (**Kapitel 7.8**)
- Erstellung des multidimensionalen Modells (**Kapitel 7.9**)
- Erstellung der OLAP Berichte (**Kapitel 7.10**)

Folgende Software wird für den Prototypen verwendet:

- Betriebssystem Windows NT 4.0 Service Pack 3
- ORACLE 8.0.5 für Win NT 4.0 (Data Warehouse Datenbank)
- ORACLE 7.3.4 für Win NT 4.0 (Datenbank des Operativen System)
- DSS Architect (Erstellung des multidimensionalen Modells) und DSS Agent (Erstellung der OLAP Berichte) der Firma MicroStrategy in der Version 5.1
- ORACLE 2.5.3.1.0 ODBC (16 Bit)
- ORACLE SQL*Net 2.3.3.0.1 (16 Bit)
- ORACLE SQL*Net 2.3.4.0.0 (32 Bit)
- ORACLE SQL*Net 8 (32 Bit)

Folgende Hardwareumgebung steht für den Prototypen zur Verfügung:

Server 1(Operatives System):

- Pentium Pro 200 MHz
- 98 MB RAM
- Windows NT 4.0 Service Pack 3
- ORACLE 7.3.4
- ORACLE SQL*Net 2.3.4.0.0 (32 Bit)

Server 2 (Data Warehouse + Warehouse Entwicklungsumgebung)

- Pentium II 300 MHz
- 128 MB RAM
- 4 GB SCSI Festplatte, 1 GB EIDE Festplatte
- Windows NT 4.0 Service Pack 3
- ORACLE 8.0.5
- ORACLE 2.5.3.1.0 ODBC (16 Bit)

- ORACLE SQL*Net 2.3.3.0.1 (16 Bit)
- ORACLE SQL*Net 8 (32 Bit)
- MicroStrategy 5.1 (DSS Architect 16 Bit, DSS Agent 16 Bit)

Die beiden Server sind in einem Netzwerk mittels dem üblichen TCP/IP Protokoll verbunden. Der Datenaustausch zwischen den Servern erfolgt über das proprietäre ORACLE Protokoll SQL*Net 8 (32 Bit) welches auf das TCP/IP Protokoll aufsetzt und die Datenübertragung um ORACLE spezifische Fuktionalitäten erweitert.

Da es sich bei den beiden MicroStrategy Produkten DSS Architect und DSS Agent um 16-Bit Applikationen handelt, ist es notwendig den Zugriff dieser Tools auf das Data Warehouse durch den 16-Bit ORACLE 2.5.3.1.0 ODBC Treiber und dem 16-Bit ORACLE SQL*Net2 2.3.3.0.1 zu ermöglichen. Nur diese Versionen des ODBC Treibers und des SQL*Nets sind durch MicroStrategy zertifiziert und demzufolge zu verwenden. Anfang nächsten Jahres wird MicroStrategy 32-Bit Versionen des DSS Architects und des DSS Agent veröffentlichen, so daß eine zusätzliche Installation des 16-Bit ORACLE ODBC Treibers und des 16-Bit ORACLE SQL*Net Protokols überflüssig wird.

Bei den Daten die in dem operativen System vorliegen, handelt es sich nur um eine kleinere Auswahl von Testdaten der Firma Oppenhoff & Rädler. Die Daten wurden zudem für diese Diplomarbeit verfälscht, so daß keine Rückschlüsse auf die Originaldaten möglich sind.

7.2 Firmenprofil Oppenhoff & Rädler

Bei der Firma Oppenhoff & Rädler handelt es sich um eine internationale Anwaltskanzlei mit 558 Mitarbeitern, 111 Partnern und 163 weiteren Berufsträgern.

Die Kanzlei ist in Berlin, Frankfurt/Main, Köln, Leipzig, München, Alicante, Brüssel, London, New York, Prag und Warschau vertreten.

Oppenhoff & Rädler praktizieren deutsches und EU-Recht mit Schwerpunkt Handels-, Steuer- , Bank- und Wettbewerbsrecht.

7.3 Das operative System der Fa. Oppenhoff & Rädler

Die Testdaten für den Prototypen stammen aus dem Mandatsverwaltungssystem "**ORSYS**". Über dieses System wird das operative Tagesgeschäft der Firma Oppenhoff & Rädler in folgenden Bereichen unterstützt:

- Mandatsverwaltung (**Mandat** = Rechtsdienstleistung)
- Personenverwaltung
- Userverwaltung (**User** = Mitarbeiter)
- Zeit-/Kostenerfassung
- Zahlungsanweisungen
- Rechnungserstellung
- Geldein-/ausgang
- Schnittstelle zu der Finanzbuchhaltung (FIBU)
- Beteiligtenverwaltung

ORSYS wird dezentral an allen Standorten der Firma Oppenhoff & Rädler eingesetzt, wobei die Daten ebenfalls dezentral bei jedem Standort in einer

ORACLE 7.3.3 Datenbank gespeichert werden. Eine zentrale Datenhaltung ist nicht vor Anfang 1999 geplant.

7.4 Die Analyse des operativen Datenmodells

Bei der Analyse des operativen Datenmodells ist zu ermitteln, welche Entitäten und somit Tabellen des operativen Systems für den Data Warehouse Prototypen eine Rolle spielen, und wie diese Tabellen zusammenhängen (1:1, 1:n, n:m Beziehungen).

Interessant sind solche Tabellen des operativen Datenmodells, welche Daten für die Erstellung der geforderten betriebswirtschaftlichen OLAP Analysen liefern (siehe Kapitel 7.1).

In der Abbildung 34 auf der nächsten Seite ist ein Ausschnitt des **E**ntity **R**elationship **D**iagramms (**ERD**) des ORSYS – Systems zu sehen, welcher die Entitäten beinhaltet, die alle notwendigen Daten für den ROLAP Prototypen liefern.

Neben der Auswahl der notwendigen Entitäten ist zudem zu analysieren:

- Der Aufbau der dazugehörigen Tabellen (Definition der Tabellenattribute)
- Die Beziehungen der Tabellen untereinander
- Die Anzahl der Datensätze pro Tabelle
- Der Inhalt der Tabellen (Analyse der Daten)

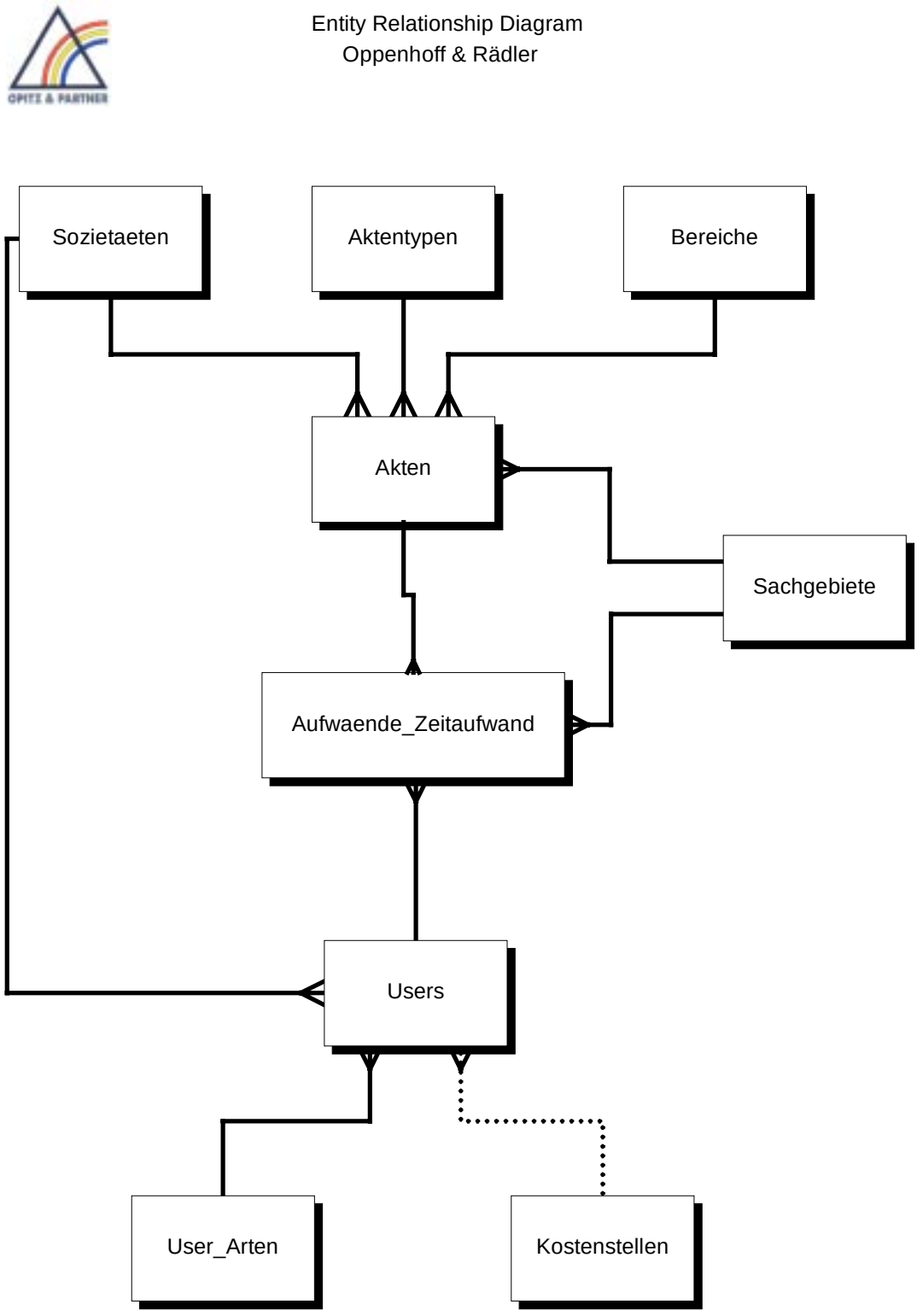


Abbildung 34: Relevanter ERD-Ausschnitt von ORSYS

Folgende Tabellendefinition existiert für das aufgeführte ERD Modell in ORSYS:

Tabelle ‚SOZIETAETEN' (12 Datensätze):

In der Tabelle ‚Sozietaeten' werden Daten über die unterschiedlichen Oppenhoff & Rädler Geschäftsstellen gespeichert. Diese Daten beinhalten z. B. den Namen der Geschäftsstelle, Standort, Interner Schlüssel der Geschäftsstelle, usw.:

```
Name                           Null?    Type
 ------------------------------ -------- ----
 SORT_ID                        NOT NULL NUMBER(15)
 SORT_CODE                      NOT NULL VARCHAR2(5)
 SORT_BEZEICHNUNG               NOT NULL VARCHAR2(60)
 SORT_STANDORT                  NOT NULL VARCHAR2(20)
 SORT_RECHTSFORM                         VARCHAR2(10)
 SORT_FIBU_MANDANT                       VARCHAR2(4)
 SORT_KURZWAHL                           VARCHAR2(10)
 SORT_ERFASST_VON               NOT NULL VARCHAR2(30)
 SORT_ERFASST_AM                NOT NULL DATE
 SORT_GEAENDERT_VON                      VARCHAR2(30)
 SORT_GEAENDERT_AM                       DATE
 SORT_BEMERKUNG                          VARCHAR2(2000)
 SORT_ACIS_NR                            NUMBER(10)
 SORT_FILESERVER                         VARCHAR2(300)
 SORT_MAP_LAUFWERK                       VARCHAR2(1)
```

Tabelle ‚AKTENTYPEN' (13 Datensätze):

Hier werden die Arten eines Mandats mit Schlüssel und Beschreibung gespeichert. Solche Aktentypen sind zum Beispiel Mahnvefahren, Beratung, Schiedsgericht, usw.:

```
Name                           Null?    Type
 ------------------------------ -------- ----
 ATYP_ID                        NOT NULL NUMBER(15)
 ATYP_CODE                      NOT NULL VARCHAR2(5)
 ATYP_BEZEICHNUNG               NOT NULL VARCHAR2(60)
 ATYP_ACIS_CHECK_KNZ            NOT NULL VARCHAR2(1)
 ATYP_ERFASST_VON               NOT NULL VARCHAR2(30)
 ATYP_ERFASST_AM                NOT NULL DATE
```

```
 ATYP_GEAENDERT_VON                              VARCHAR2(30)
 ATYP_GEAENDERT_AM                               DATE
 ATYP_BEMERKUNG                                  VARCHAR2(2000)
 ATYP_KLASSE                            NOT NULL VARCHAR2(1)
```

Tabelle ‚Bereiche' (14 Datensätze):

In dieser Tabelle werden die einzelnen Bereiche mit Schlüssel und Beschreibung festgehalten zu dem ein **Mandat** (eine Rechtsdienstleistung) zuzuordnen ist. Solche Bereiche sind z. B. Banken, Versicherung, Automobilindustrie, usw.:

```
Name                           Null?    Type
 ------------------------------ -------- ----
 BRCH_ID                        NOT NULL NUMBER(15)
 BRCH_CODE                      NOT NULL VARCHAR2(5)
 BRCH_BEZEICHNUNG               NOT NULL VARCHAR2(60)
 BRCH_ERFASST_VON               NOT NULL VARCHAR2(30)
 BRCH_ERFASST_AM                NOT NULL DATE
 BRCH_GEAENDERT_VON                      VARCHAR2(30)
 BRCH_GEAENDERT_AM                       DATE
 BRCH_BEMERKUNG                          VARCHAR2(2000)
```

Tabelle ‚Akten' (19858 Datensätze):

Bei einer Akte handelt es sich um ein Mandat. D. h. hinter jeder Aktennummer verbirgt sich eine Rechtsberatungsdienstleistung der Fa. Oppenhoff & Rädler gegenüber einem Kunden.

Zusätzlich zur Aktennummer werden Daten wie z. B. der Streitwert, eine Beschreibung des Mandats, ein Festpreis, usw. gespeichert:

```
Name                           Null?    Type
 ------------------------------ -------- ----
 AKTE_ID                        NOT NULL NUMBER(15)
 AKTE_SACH_ID                   NOT NULL NUMBER(15)
 AKTE_ATYP_ID                   NOT NULL NUMBER(15)
 AKTE_BRCH_ID                   NOT NULL NUMBER(15)
 AKTE_SORT_ID                   NOT NULL NUMBER(15)
 AKTE_USER_ID                   NOT NULL NUMBER(15)
```

AKTE_ANLAGE_DATUM	NOT NULL	DATE
AKTE_RUBRUM	NOT NULL	VARCHAR2(150)
AKTE_ERFASST_AM	NOT NULL	DATE
AKTE_ERFASST_VON	NOT NULL	VARCHAR2(30)
AKTE_AKTE_ID		NUMBER(15)
AKTE_AKTENNR		NUMBER(8)
AKTE_PROJEKTNR		NUMBER(8)
AKTE_SPERR_KZ		VARCHAR2(1)
AKTE_ALTAKTENNR		VARCHAR2(20)
AKTE_AKTE_STUNDENSATZ		NUMBER(5)
AKTE_STICHWORT		VARCHAR2(1000)
AKTE_ABLAGE_DATUM		DATE
AKTE_BRAGO_KZ		VARCHAR2(1)
AKTE_ABSCHLUSS_DATUM		DATE
AKTE_STREITWERT		NUMBER(14,2)
AKTE_FESTPREIS		NUMBER(13,2)
AKTE_FREMDGELD_SOLL		NUMBER(15,2)
AKTE_FREMDGELD_HABEN		NUMBER(15,2)
AKTE_GEAENDERT_AM		DATE
AKTE_GEAENDERT_VON		VARCHAR2(30)
AKTE_GEBUEHRENTEILUNG_KZ		VARCHAR2(1)
AKTE_ACIS_KZ		VARCHAR2(1)
AKTE_LAGERORT		VARCHAR2(10)
AKTE_ACIS_NR		NUMBER(10)
AKTE_DV_ARCHIV		VARCHAR2(40)
AKTE_NOTAR_AKTE_KZ		VARCHAR2(1)
AKTE_STRATEGIE_KZ		VARCHAR2(1)
AKTE_ABRECHNUNG_MODUS		VARCHAR2(1)
AKTE_SUMME_NICHT_FREI		NUMBER(15,2)
AKTE_SUMME_FREI_KOSTEN		NUMBER(15,2)
AKTE_SUMME_FREI_BRUTTO		NUMBER(15,2)
AKTE_SUMME_FREI_NETTO		NUMBER(15,2)
AKTE_SUMME_FREI_INTKOSTEN		NUMBER(15,2)
AKTE_SUMME_FREI_ZEITEN		NUMBER(15,2)
AKTE_SUMME_PREBILL_KOSTEN		NUMBER(15,2)
AKTE_SUMME_PREBILL_ZEITEN		NUMBER(15,2)
AKTE_SUMME_PREBILL_INTKOSTEN		NUMBER(15,2)
AKTE_SUMME_PREBILL_NETTO		NUMBER(15,2)
AKTE_SUMME_PREBILL_BRUTTO		NUMBER(15,2)
AKTE_SUMME_PREBILL_RAK		NUMBER(15,2)
AKTE_SUMME_IN_RG_KOSTEN		NUMBER(15,2)
AKTE_SUMME_IN_RG_BRUTTO		NUMBER(15,2)
AKTE_SUMME_IN_RG_NETTO		NUMBER(15,2)
AKTE_SUMME_IN_RG_INTKOSTEN		NUMBER(15,2)
AKTE_SUMME_IN_RG_ZEITEN		NUMBER(15,2)
AKTE_SUMME_IN_RG_RAK		NUMBER(15,2)
AKTE_SUMME_BEZAHLT_KOSTEN		NUMBER(15,2)
AKTE_SUMME_BEZAHLT_BRUTTO		NUMBER(15,2)
AKTE_SUMME_BEZAHLT_NETTO		NUMBER(15,2)
AKTE_SUMME_BEZAHLT_INTKOSTEN		NUMBER(15,2)
AKTE_SUMME_BEZAHLT_ZEITEN		NUMBER(15,2)
AKTE_SUMME_BEZAHLT_RAK		NUMBER(15,2)
AKTE_SUMME_DAUER		NUMBER(15,2)
AKTE_SUMME_EINNAHMEN		NUMBER(15,2)
AKTE_FREMDGELD_SALDO		NUMBER(15,2)
AKTE_SUMME_PREBILL_VORSCHUSS		NUMBER(15,2)
AKTE_SUMME_IN_RG_VORSCHUSS		NUMBER(15,2)
AKTE_SUMME_FG_VORSCHUSS		NUMBER(15,2)
AKTE_BEMERKUNG		VARCHAR2(2000)
AKTE_SUMME_BEZAHLT_VORSCHUSS		NUMBER(15,2)
AKTE_SUMME_FREI_HONORAR		NUMBER(15,2)
AKTE_SUMME_PREBILL_HONORAR		NUMBER(15,2)
AKTE_SUMME_IN_RG_HONORAR		NUMBER(15,2)

```
AKTE_SUMME_BEZAHLT_HONORAR                NUMBER(15,2)
AKTE_KOLL_ABSTIMMUNG                      VARCHAR2(50)
AKTE_SUMME_DAUER_NICHT_ABRECH             NUMBER(15,2)
AKTE_SUMME_DAUER_AUSGEBUCHT               NUMBER(15,2)
AKTE_ARCHIV_KZ                            VARCHAR2(1)
AKTE_BEARBEITER                           VARCHAR2(200)
AKTE_BEARBEITER_REPO                      VARCHAR2(2000)
AKTE_VERANTWORTLICHER_REPO                VARCHAR2(2000)
AKTE_PARTNER_REPO                         VARCHAR2(2000)
AKTE_SUMME_DAUER_PARTNER                  NUMBER(15,2)
AKTE_SUMME_DAUER_PARTNER_N_ABR            NUMBER(15,2)
```

Tabelle ‚Sachgebiete' (63 Datensätze):

In dieser Tabelle werden die unterschiedlichen Rechtsgebiete, wie z. B. Strafrecht, Handelsrecht, Bankrecht usw., sowie deren interne Schlüssel gespeichert.

Eine Akte bezieht sich immer auf ein bestimmtes Sachgebiet, aber auch ein einzelner Aufwand für eine Akte (Tabelle ‚Aufwaende_Zeitaufwand') kann wiederum einem bestimmten Sachgebiet zugewiesen werden, der von dem eigentlichem Sachgebiet der Akte abweicht.

```
Name                              Null?    Type
 ------------------------------- -------- ----
 SACH_ID                           NOT NULL NUMBER(15)
 SACH_CODE                         NOT NULL VARCHAR2(5)
 SACH_BEZEICHNUNG                  NOT NULL VARCHAR2(60)
 SACH_ERFASST_VON                  NOT NULL VARCHAR2(30)
 SACH_ERFASST_AM                   NOT NULL DATE
 SACH_GEAENDERT_VON                         VARCHAR2(30)
 SACH_GEAENDERT_AM                          DATE
 SACH_BEMERKUNG                             VARCHAR2(2000)
```

Tabelle ‚Aufwaende_Zeitaufwand' (336203 Datensätze):

Hier werden die Aufwände (in DM, in Stunden) für eine Akte gespeichert. Man erhält zusätzlich zu den Aufwänden auch Informationen darüber, wer den Aufwand verursacht hat (User), welchem Sachgebiet dieser Aufwand zuzuweisen ist, usw.

Für die Erstellung der gewünschten OLAP Analysen (siehe Kapitel 7.1) stellt diese Tabelle die notwendigen Daten zur Verfügung.

```
Name                           Null?    Type
 ------------------------------ -------- ----
 AZEI_ID                        NOT NULL NUMBER(15)
 AZEI_DAUER                     NOT NULL NUMBER(5,2)
 AZEI_AKTENNR                            NUMBER(8)
 AZEI_TAETIGKEITS_CODE                   VARCHAR2(5)
 AZEI_BETRAG_BEZAHLT                     NUMBER(11,2)
 AZEI_BETRAG_NETTO                       NUMBER(11,2)
 AZEI_ROTER_ZETTEL_DATUM                 DATE
 AZEI_STUNDENSATZ                        NUMBER(6,2)
 AZEI_ABRECHENBAR_KZ            NOT NULL VARCHAR2(1)
 AZEI_TRANSFER_KZ                        VARCHAR2(1)
 AZEI_AUFWAND_VOM               NOT NULL DATE
 AZEI_BETRAG                             NUMBER(11,2)
 AZEI_STATUS_KZ                 NOT NULL VARCHAR2(1)
 AZEI_AUFWAND_JAHR                       NUMBER(4)
 AZEI_AUFWAND_MONAT                      NUMBER(2)
 AZEI_AUFWAND_QUARTAL                    NUMBER(1)
 AZEI_UMBUCHUNG_KZ              NOT NULL VARCHAR2(1)
 AZEI_KOMMENTAR                          VARCHAR2(240)
 AZEI_RPOS_ID                            NUMBER(15)
 AZEI_AKTE_ID                   NOT NULL NUMBER(15)
 AZEI_TAET_ID                   NOT NULL NUMBER(15)
 AZEI_ZAPR_ID                            NUMBER(15)
 AZEI_USER_ID                   NOT NULL NUMBER(15)
 AZEI_KPOS_ID                            NUMBER(15)
 AZEI_ERFASST_VON               NOT NULL VARCHAR2(30)
 AZEI_ERFASST_AM                NOT NULL DATE
 AZEI_GEAENDERT_VON                      VARCHAR2(30)
 AZEI_GEAENDERT_AM                       DATE
 AZEI_BEMERKUNG                          VARCHAR2(2000)
 AZEI_SACH_ID                   NOT NULL NUMBER(15)
 AZEI_MANU_KZ                   NOT NULL VARCHAR2(1)
```

Tabelle ‚Users' (441 Datensätze):

Hier werden die Mitarbeiter des Unternehmens Oppenhoff & Rädler mit Namen, Kennung, externer Stundensatz, usw. gespeichert. Nur ‚User' können Aufwandsbuchungen in der Tabelle ‚Aufwaende_Zeitaufwand' vornehmen.

```
Name                           Null?    Type
 ------------------------------ -------- ----
 USER_ID                        NOT NULL NUMBER(15)
 USER_SORT_ID                   NOT NULL NUMBER(15)
 USER_UART_ID                   NOT NULL NUMBER(15)
```

```
 USER_KENNUNG                    NOT NULL VARCHAR2(10)
 USER_PERS_IDENT                 NOT NULL VARCHAR2(15)
 USER_NACHNAME                   NOT NULL VARCHAR2(40)
 USER_STUNDENSATZ_EXTERN         NOT NULL NUMBER(4)
 USER_ERFASST_AM                 NOT NULL DATE
 USER_ERFASST_VON                NOT NULL VARCHAR2(30)
 USER_KOST_ID                             NUMBER(15)
 USER_VORNAME                             VARCHAR2(30)
 USER_TITEL                               VARCHAR2(20)
 USER_CAL_NAME                            VARCHAR2(93)
 USER_PARTNER_KZ                          VARCHAR2(1)
 USER_RA_KZ                               VARCHAR2(1)
 USER_FREIGABESUMME                       NUMBER(9)
 USER_STUNDENSATZ_INTERN                  NUMBER(4)
 USER_RAUM_NR                             VARCHAR2(4)
 USER_DURCHWAHL                           VARCHAR2(8)
 USER_BEMERKUNG                           VARCHAR2(2000)
 USER_GEAENDERT_AM                        DATE
 USER_GEAENDERT_VON                       VARCHAR2(30)
 USER_INFORMATION                         VARCHAR2(100)
 USER_ACIS_ADMIN                          VARCHAR2(1)
 USER_SOLVER                              VARCHAR2(1)
 USER_AUSGESCHIEDEN_KNZ                   VARCHAR2(1)
```

Tabelle ‚User_ Arten' (21 Datensätze):

Hier werden die unterschiedlichen Userarten mit internem Schlüssel, Bezeichnung, usw. gespeichert. Durch eine Userart wird ein User genauer spezifiziert. So kann man z. B. erkennen ob es sich bei einem User um einen Rechtsanwalt, einem Mitarbeiter aus der Buchhaltung oder um eine Aushilfe handelt.

```
Name                            Null?    Type
 ------------------------------ -------- ----
 UART_ID                         NOT NULL NUMBER(15)
 UART_CODE                       NOT NULL VARCHAR2(5)
 UART_BEZEICHNUNG                NOT NULL VARCHAR2(60)
 UART_ERFASST_VON                NOT NULL VARCHAR2(30)
 UART_ERFASST_AM                 NOT NULL DATE
 UART_GEAENDERT_VON                       VARCHAR2(30)
 UART_GEAENDERT_AM                        DATE
 UART_BEMERKUNG                           VARCHAR2(2000)
```

Tabelle ‚Kostenstellen' (79 Datensätze):

Diese Tabelle beinhaltet die Bezeichnungen, interne Nummern, usw. von Kostenstellen, auf die die Aufwände der Tabelle ‚Aufwaende_Zeitaufwand' in der Finanzbuchhaltung (FIBU) gebucht werden.

```
Name                           Null?    Type
 ------------------------------ -------- ----
 KOST_ID                        NOT NULL NUMBER(15)
 KOST_CODE                      NOT NULL NUMBER(6)
 KOST_BEZEICHNUNG               NOT NULL VARCHAR2(60)
 KOST_ERFASST_VON               NOT NULL VARCHAR2(30)
 KOST_ERFASST_AM                NOT NULL DATE
 KOST_GEAENDERT_VON                      VARCHAR2(30)
 KOST_GEAENDERT_AM                       DATE
 KOST_BEMERKUNG                          VARCHAR2(2000)
 KOST_USER_ID                            NUMBER(15)
```

Die Beziehungen (1:1, 1:n, n:m) der Tabellen untereinander ist dem ERD Modell zu entnehmen.

7.5 Die Planung des multidimensionalen Modells

Nachdem durch die vorhergegangene Analyse bekannt ist, welche Daten existieren und wo diese Daten zu finden sind, wird nun ein multidimensionales Modell erstellt, welches den OLAP Anforderungen entspricht (siehe Kapitel 3). Dabei werden Dimensionen, Dimensionshierarchien und Datenwürfel gemäß den gewünschten OLAP Analysen (siehe Kapitel 7.1) berücksichtigt.

Folgendes Modell wird für den Oppenhoff & Rädler Prototypen benötigt:

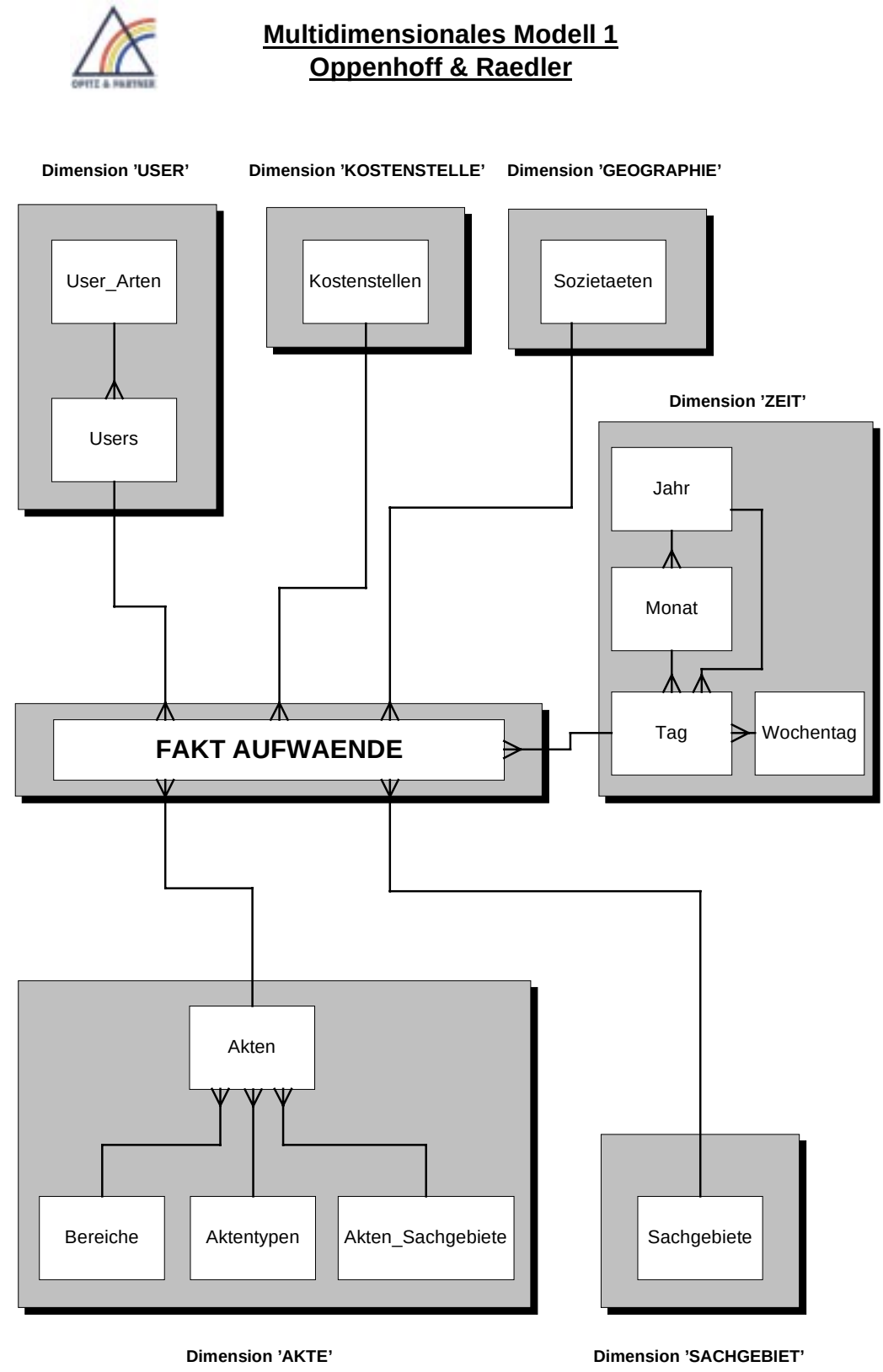


Abbildung 35: Oppenhoff & Rädler Multidimensionales Modell 1

Bei diesem multidimensionalen Modell wird ein Datenwürfel, der die Zeitaufwände in Stunden, sowie die Umsätze in DM beinhaltet, durch sechs Dimensionen definiert, wobei drei der Dimensionen als hierarchisch zu bezeichnen sind.

Die Hierarchien der Dimensionen entsprechen dem Drill Down und Drill Up Weg (siehe Kapitel 3.1.3), den der Endbenutzer bei seiner OLAP Analyse benutzen kann.

Umsätze in DM, oder Zeitaufwände in Stunden können somit in einer beliebigen Kombination von Dimensionen analysiert werden.

Zu beachten ist, daß das Sachgebiet von Aufwänden und das Sachgebiet von Akten durch zwei unterschiedliche Dimensionen repräsentiert wird, obwohl in dem operativen Datenmodell nur eine Tabelle Sachgebiet besteht, die mit beiden Tabellen **‚**Akten**‘** und **‚**Aufwaende_Zeitaufwand**‘** in einer 1 zu n Beziehung steht. Nur so ist eine Analyse der Zeitaufwände oder Umsätze für ein bestimmtes Aktensachgebiet und die zu diesem Aktensachgebiet dazughörigen Aufwandssachgebiete möglich (siehe Kapitel 7.1 Anforderungen für OLAP Analysen), da zwischen diesen Sachgebieten eine n zu m Beziehung besteht, welche über den Datenwürfel und somit über die Fakttabelle, die diesen Datenwürfel repräsentiert, aufgelöst wird. Dadurch wird eine flexible Analyse der Umsätze und der Zeitaufwände über die Aktensachgebiete, über die Aufwandssachgebiete oder aber über beide Sachgebiete in Abhängigkeit zueinander ermöglicht.

Um den Datenwürfel zusätzlich in einer höheren Granularität darzustellen (siehe Kapitel 4.4.1), so daß schnellere Analysen auf niedrigerer Detailebene durchführbar sind, ist folgendes multidimensionales Modell notwendig:

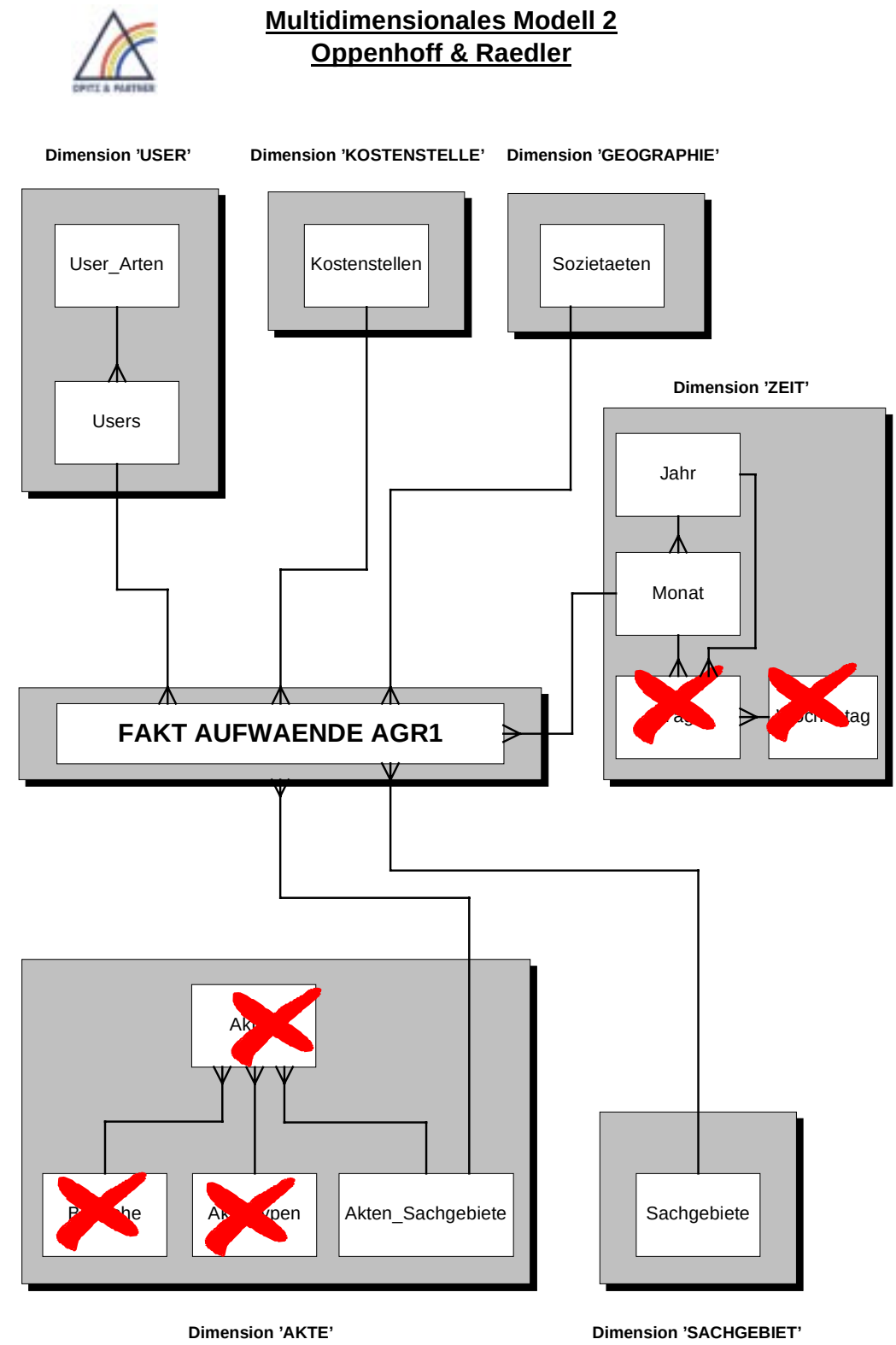


Abbildung 36: Oppenhoff & Rädler Multidimensionales Modell 2

Im Gegensatz zu dem ersten multidimensionalen Modell werden in dem aggregierten Datenwürfel die Umsätze und Zeitaufwände nicht auf Tages- und Aktenbasis gehalten, sondern nur auf **Monats- und Aktensachgebietsbasis**. Dadurch wird zum einen der Detailgrad der Daten verringert, zum anderen aber die Anzahl der Datensätze der zu Grunde liegenden Fakttabelle ebenfalls drastisch reduziert, so daß schnellere Analysen auf diese höher aggregierten Daten ausgeführt werden können.

Zu beachten ist jedoch, daß somit keine Analysen auf Akten- oder Tagesbasis über diesen Datenwürfel realisierbar sind. Auch Analysen über damit in Beziehung stehenden Dimensionsattributen wie z. B. der Wochentag, der Aktentyp, oder der Aktenbereich sind nicht mehr möglich, da auch hierfür die notwendigen Basisdaten (Tag und Akte) in der Fakttabelle nicht mehr existieren. Solche Analysen müssen dementsprechend über den Datenwürfel mit der niedrigen Granularität durchgeführt werden.

Natürlich sind auch weitere aggregierte Datenwürfel denkbar, jedoch reichen die vorgegestellten beiden multidimensionalen Modelle für die Erfüllung der in Kapitel 7.1 genannten Ziele aus.

7.6 Die Planung des ROLAP Datenmodells

Bei der Planung des ROLAP Datenmodells sind die Entwicklungs- und Analysewerkzeuge der Firma MicroStrategy zu berücksichtigen, da diese Tools nur bestimmte ROLAP Datenmodelle unterstützen. Hält man diese unterstützten Modelle bei der Entwicklung der Data Warehouse Datenbank nicht ein, so sind diese Tools für das Warehouse nicht einsetzbar, da sie die Struktur des Warehouses nicht erkennen. Folge davon ist, daß die SQL Engine (siehe Kapitel 4.1) keine korrekten SQL Statements für die ROLAP Datenbank erstellen kann. Die unterstützten Modelle orientieren sich jedoch stark an das übliche Star- bzw. an das Snowflakeschema (siehe Kapitel

4.3.1/4.3.2) wobei insgesamt sechs Ausprägungen dieser beiden Basismodelle möglich sind.

Anhand des folgenden multidimensionalen Modells sollen die sechs einsetzbaren ROLAP Datenmodelle beschrieben werden[40]:

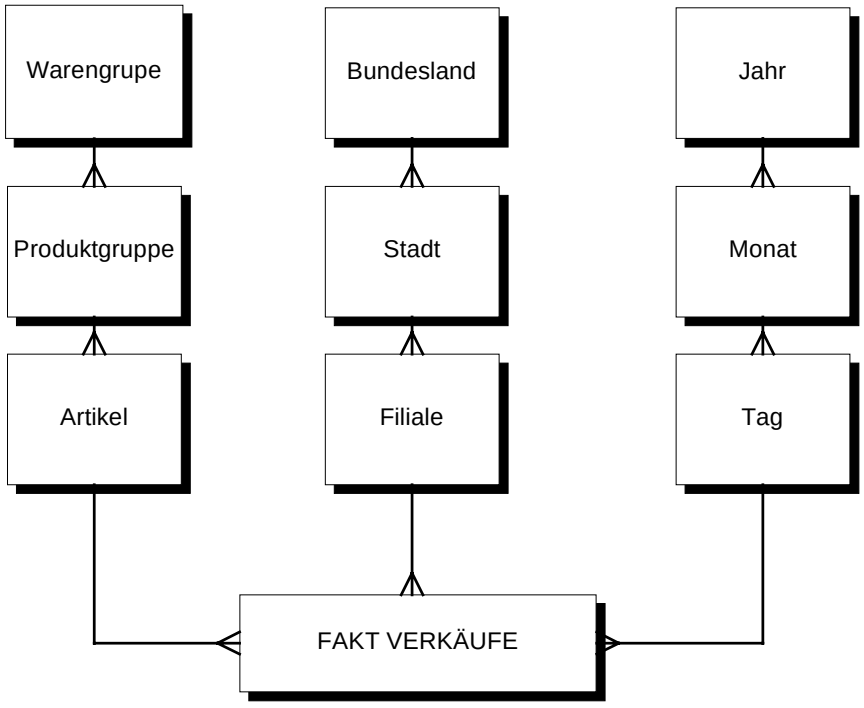


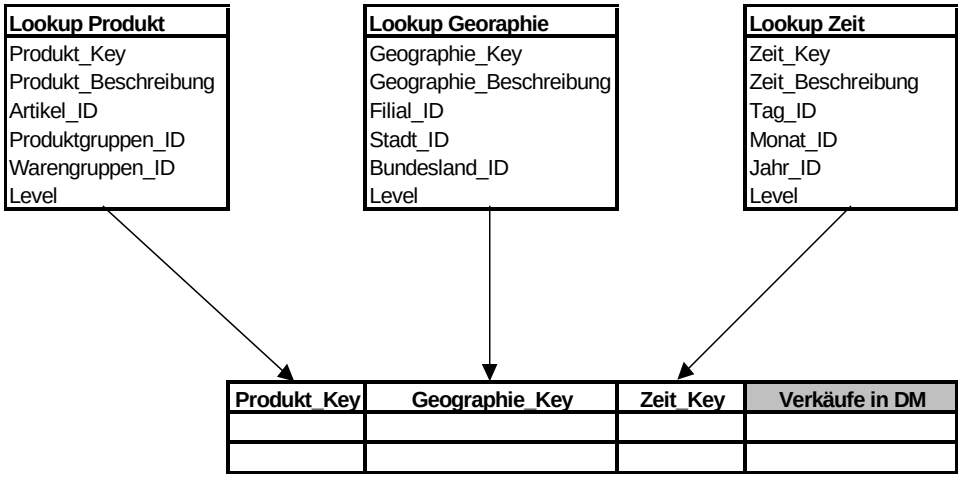
Abbildung 37: Beispielmodell (multidimensional)

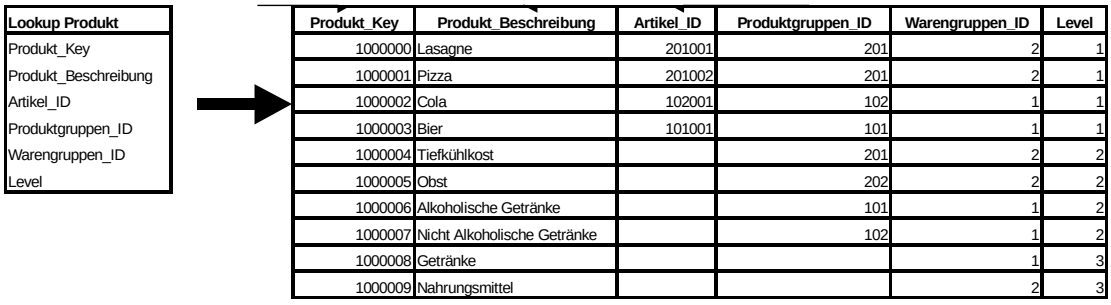
(1) Consolidated Star Schema #1

- Für jede Dimension wird eine Lookup-Tabelle erstellt
- Für jeden Datenwürfel wird eine Fakt-Tabelle erstellt

[40] MicroStrategy Schulungsunterlagen Kapitel „DSS Schemas“

- Jede Dimensions-Lookup-Tabelle besitzt einen generierten Dimensionsschlüssel, über den eine Beziehung zu der Fakttabelle hergestellt wird.
- Es existiert nur eine Beschreibungsspalte für alle Dimensionsattribute einer hierarchischen Dimension.
- Zusätzliche Level-Spalte, durch die die Hierarchiestufe sichtbar gemacht wird.





Lookup Produkt

Lookup Produkt
Produkt_Key
Produkt_Beschreibung
Artikel_ID
Produktgruppen_ID
Warengruppen_ID
Level

Produkt_Key	Produkt_Beschreibung	Artikel_ID	Produktgruppen_ID	Warengruppen_ID	Level
1000000	Lasagne	201001	201	2	1
1000001	Pizza	201002	201	2	1
1000002	Cola	102001	102	1	1
1000003	Bier	101001	101	1	1
1000004	Tiefkühlkost		201	2	2
1000005	Obst		202	2	2
1000006	Alkoholische Getränke		101	1	2
1000007	Nicht Alkoholische Getränke		102	1	2
1000008	Getränke			1	3
1000009	Nahrungsmittel			2	3

Abbildung 38: Consolidated Star Schema #1

Vorteile:

- Innerhalb einer Fakttabelle können Kennzahlen, dank der Level-Spalte in den Dimensionstabellen, in verschiedenen Aggregationsstufen vorkommen (= **IN-TABLE** Aggregation). D. h., zusätzlich zu den Kennzahlen einer Hierarchiestufe (z. B. Umsatz pro Tag) können diese Kennzahlen in höheren Aggregationsstufen in der gleichen Fakttabelle vorliegen (z. B. Umsatz pro Jahr) und somit über die gleiche Fakttabelle direkt ausgewertet werden, ohne eine temporäre Aufsummierung der Tagesumsätze auf Jahresebene durchzuführen.

Produkt_Key	Geographie_Key	Zeit_Key	Verkäufe in DM
Artikel	Filiale	Tag	
Artikel	Filiale	Monat	
Artikel	Filiale	Jahr	
Produktgruppe	Filiale	Tag	
Produktgruppe	Filiale	Monat	
Produktgruppe	Filiale	Jahr	
Warengruppe	Filiale	Tag	
Warengruppe	Filiale	Monat	
Warengruppe	Filiale	Jahr	
Artikel	Stadt	Tag	
Artikel	Stadt	Monat	
Artikel	Stadt	Jahr	
Artikel	Bundesland	Tag	
Artikel	Bundesland	Monat	
Artikel	Bundesland	Jahr	

Abbildung 39: Fakttabelle mit IN-TABLE Aggregation

Beispiel: Ausgabe der gesamten Umsätze pro Warengruppe (Level3)

```
select b.produkt_beschreibung, a.umsatz ‚UMSATZ IN DM‘
from umsatz_f a, lookup_produkt b
where a.produkt_key = b.produkt_key
and b.level = 3;
```

Der notwendige Level in dem SQL-Befehl wird automatisch von der SQL-Engine der Firma MicroStrategy ermittelt.

- Maximal ein Join zwischen Fakt- und Dimensionstabelle notwendig, um alle notwendigen Daten zu erhalten (außer bei Darstellung mehrerer Beschreibungen einer Dimensionshierarchie, siehe Nachteile).
- Geringe Anzahl Tabellen notwendig

Nachteile:

- Möchte man in einer OLAP Analyse meherere Attributbezeichnungen einer Dimensionshierarchie darstellen, so müssen zeitaufwendige, komplexe Self-Joins auf die Lookup Tabelle der Dimension durchgeführt werden. Mit der Anzahl der Dimensionsattribute steigt auch die Anzahl der Self-Joins, die notwendig sind um alle Attributbeschreibungen in einer OLAP Analyse darzustellen.

Beispiel: Darstellung der Bundesland-, Stadt, Filialbeschreibung und der Umsatz pro Filiale

```
Select sum(a.umsatz), bundesland.geographie _beschreibung,
stadt.geographie _beschreibung, filiale.geographie_beschreibung
from umsatz_f a,
lookup_geographie b,
lookup_geographie bundesland,
lookup_geographie stadt,
lookup_geopraphie filiale
where a.geographie_key = b.geographie_key
and b.bundesland_id = bundesland.bundesland_id
and b.stadt_id = stadt.stadt_id
and b.filial_id = filiale.filial_id
group by bundesland.geographie _beschreibung,
stadt.geographie _beschreibung,
filiale.geographie_beschreibung;
```

- Komplexere Load-Prozeduren, da Fakt- und Dimensionstabellen über einen extra generierten Schlüssel in Beziehung gesetzt werden.

(2) Consolidated Star Schema #2

- Beinahe identischer Aufbau wie Consolidated Star Schema #1
- Jedes Attribut einer Dimensionshierarchie wird durch eine Beschreibung repräsentiert, dafür entfällt allerdings die ID-Nummer des Attributs
- Die gemeinsame Beschreibungsspalte für alle Attribute der Dimensionshierarchie entfällt
- Hierarchiestufe des Datensatzes wird wiederum durch Level-Spalte dargestellt.

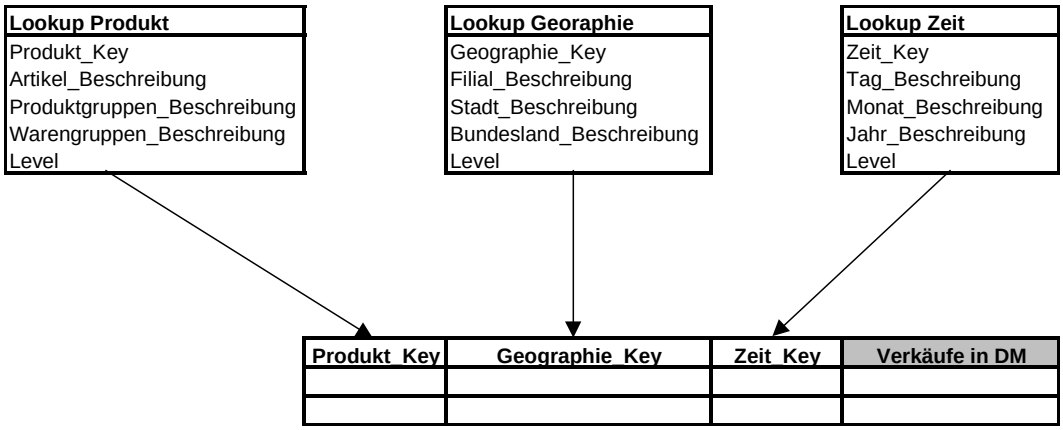


Abbildung 40: Consolidated Star Schema #2

Vorteile:

- Gleiche Vorteile wie Consolidated Star Schema #1
- Gleichzeitige Darstellung aller Attribute einer Dimensionshierarchie ohne aufwendige Self-Joins möglich

Nachteile:

- Gleiche Nachteile wie Consolidated Star Schema #1
- Nur ein Schlüssel für die gesamte Dimension. Analysen welche ein bestimmtes Dimensionsattribut suchen (z. B. where Stadt_Beschreibung = ‚KÖLN') gestalten sich als Charaktervergleich langsamer, als eine Suche über eine Attribut-Nummer (z. B. where Stadt_ID = 5).

(3) Consolidated Star Schema #3

- Ähnlicher Aufbau wie Consolidated Star Schema #1 und #2
- Auch hier existiert für jedes Dimensionattribut eine eigene Beschreibungsspalte.

- Zusätzlich zur Beschreibungsspalte enthält jedes Dimensionsattribut eine ID-Spalte.
- Auch hier entfällt die gemeinsame Beschreibungsspalte für alle Dimensionsattribute
- Wiederum wird eine Level-Spalte angelegt, um die Hierarchiestufe darzustellen

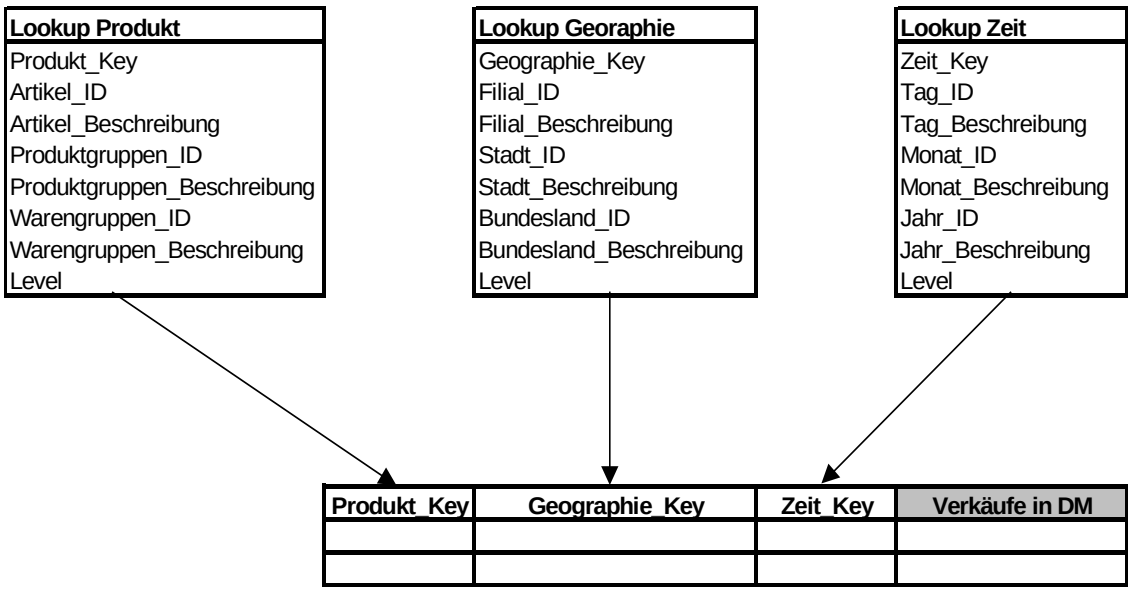


Abbildung 41: Consolidated Star Schema #3

Vorteile:

- Vorteile wie Consolidated Star Schema #2
- Kein langsamer Character-Vergleich bei Analysen wie z. B. Stadt = ‚KÖLN', da auch Suche über die ID-Nummer möglich ist.

Nachteile:

- Nachteile wie Consolidated Star Schema #2

(4) Normalised Star Schema #1

- Aufgebaut wie das typische Snowflake Schema (siehe Kapitel 4.3.2).
- Für jedes Dimensionsattribut und jeden Datenwürfel wird eine Lookup- bzw. eine Fakt-Tabelle erstellt.
- Jede Dimensionsattributtabelle besitzt einen Dimensionsattribut-schlüssel sowie eine Beschreibung.
- Jedes niedrigere Dimensionsattribut ist durch den Schlüssel des direkt darüberliegenden Dimensionsattributes verbunden.
- Nur das niedrigste Dimensionsattribut einer Dimension wird über den Schlüssel dieses Attributs mit der Fakttabelle verbunden.

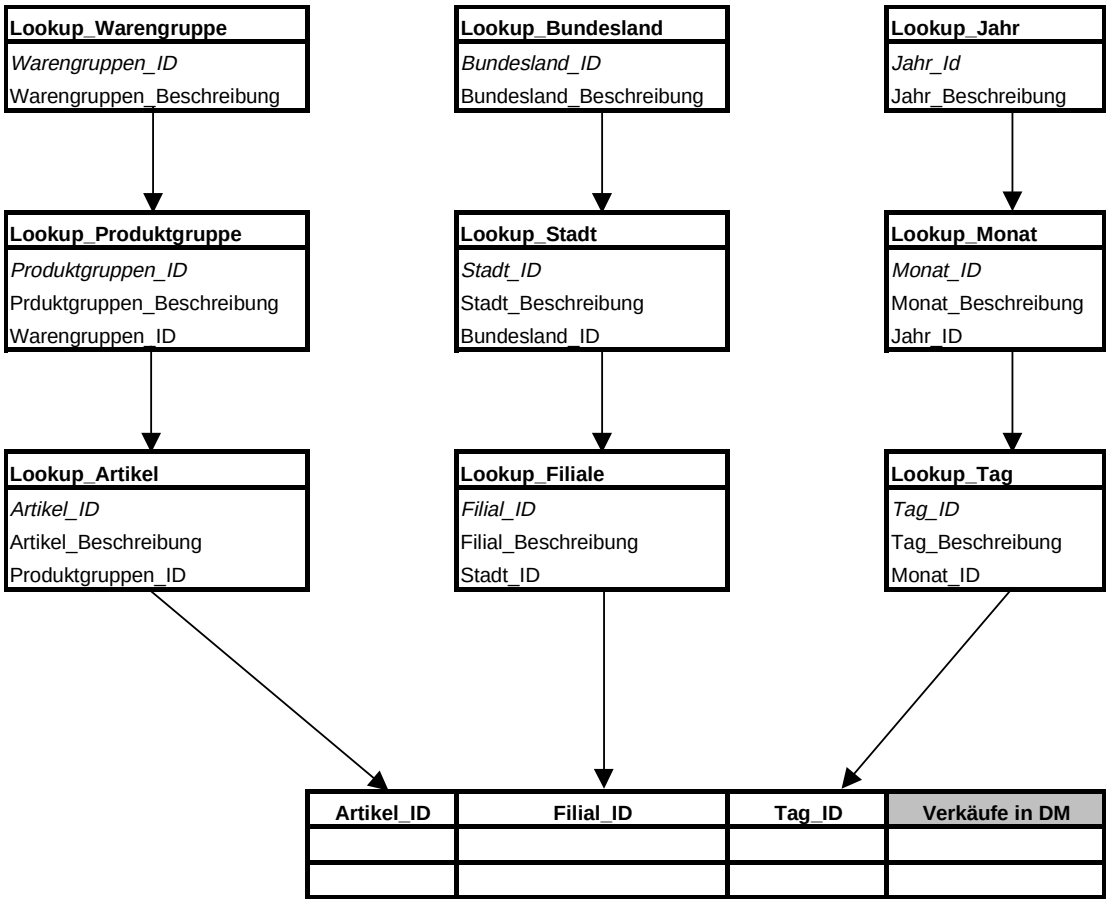


Abbildung 42: Normalised Star Schema #1 (Snowflake Schema)

Vorteile:

- Geringer Speicherplatzbedarf, da vollkommen normalisiert (3. Normalform)
- Kein extra zu generierender Dimensionsschlüssel

Nachteile:

- Höhere Anzahl Joins um an Daten höherer Dimensionshierarchie-stufen zu gelangen.

- Nur ein Aggregationsstufe pro Fakttbelle darstellbar (keine **IN-TABLE** Aggregation möglich). D. h., möchte man z. B. die Umsätze nicht tagesweise, sondern pro Monat in einer Fakttabelle aufführen, so muß eine zusätzliche Fakttabelle erstellt werden, die nur diese Aggregationsstufe beinhaltet.

  Bei einem Join zwischen dieser höher aggregierten Fakttabelle und der entsprechenden Lookup-Tabelle muß allerdings auch nur eine geringere Anzahl von Datensätzen der Fakttabelle gescanned werden.

  Im Gegensatz dazu muß bei dem Consolidated Star Schema #3 immer die komplette Fakttabelle gescanned werden, auch wenn die Analyse über eine höhere Aggregationsstufe verläuft und nur eine geringere Anzahl Datensätze betrifft, da die Level-Information nur in den Dimensionstabellen existiert (siehe SQL Statement Seite 125).

  Einschränkungen ergeben sich somit in der Erstellung der OLAP Analyse, da genau beachtet werden muß auf welche Fakttabelle und somit auf welche Aggregationsstufe sich diese Analyse beziehen soll. Die Analyse einer höher aggregierten Fakttabelle kann somit nicht auch dafür verwendet werden, um detailliertere Daten zu untersuchen. Dafür muß eine zusätzliche Analyse, basierend auf einer niedriger aggregierten Fakttabelle, erstellt werden.

(5) Normalised Star Schema #2

- Aufbau ähnlich wie Normalised Star Schema #1

- Jede untergeordnete Lookup-Tabelle enthält die ID's aller darüber liegenden Lookup-Tabellen der gleichen Dimension.

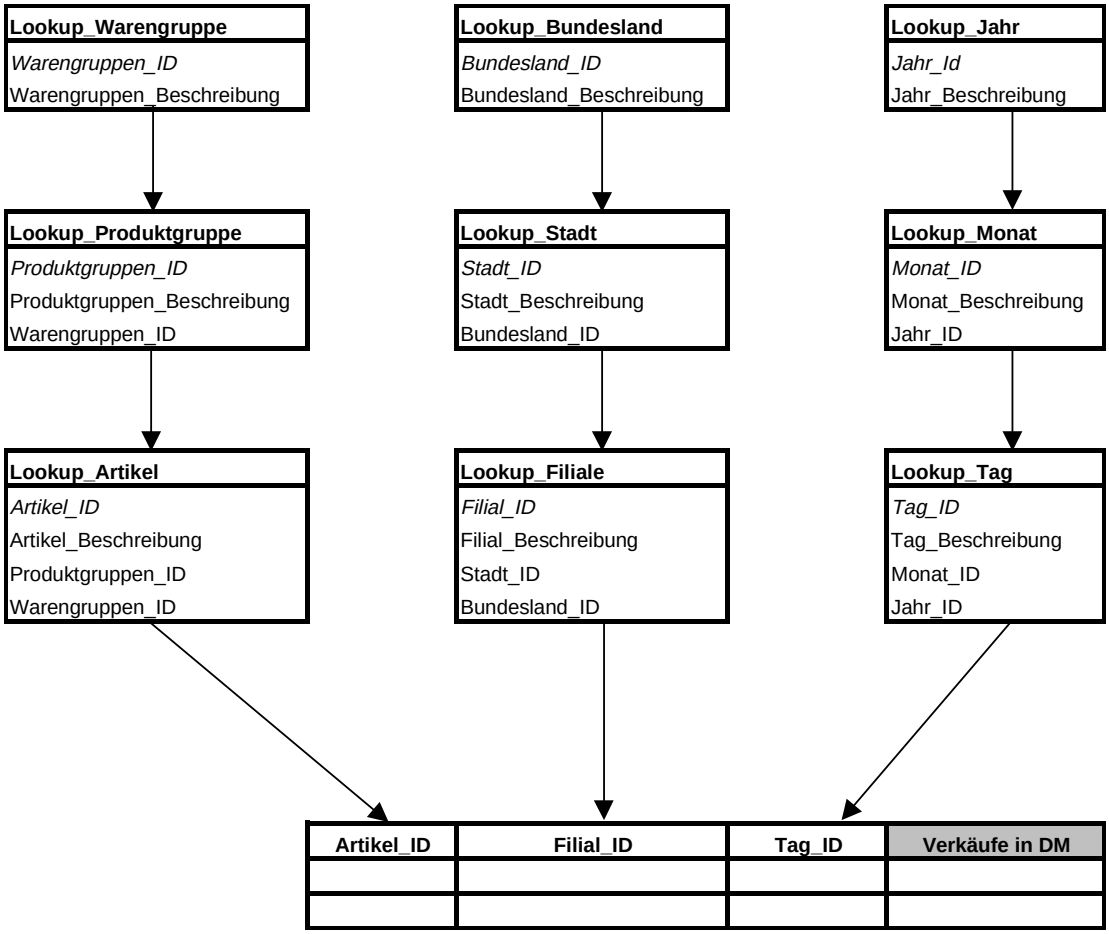


Abbildung 43: Normalised Star Schema #2

Vorteile:

- Wie Normalised Star Schema #1
- Bei Analysen über die Attributschlüssel (wie z. B. where Stadt_ID = 5) ist maximal nur noch ein Join notwendig.

Bei Analysen über die Attributbeschreibung (wie z. B. where Stadt_Beschreibung = ‚KÖLN') werden wiederum mehr Joins benötigt.

Nachteile:

- Wie Normalised Star Schema #1
- Höherer Speicherplatzbedarf, da alle Attributschlüssel der höheren Lookup-Tabellen in untergeordenten Lookup-Tabellen der gleichen Dimension gespeichert werden.

(6) Normalised Star Schema #3

- Aufbau wie Normalised Star Schema #1 und #2
- Jede untergeordnete Lookup-Tabelle speichert zusätzlich zu den Schlüsseln auch sämtliche Beschreibungen aller übergeordneten Lookup- Tabellen einer Dimension.
  ➔ vollkommene Denormalisation

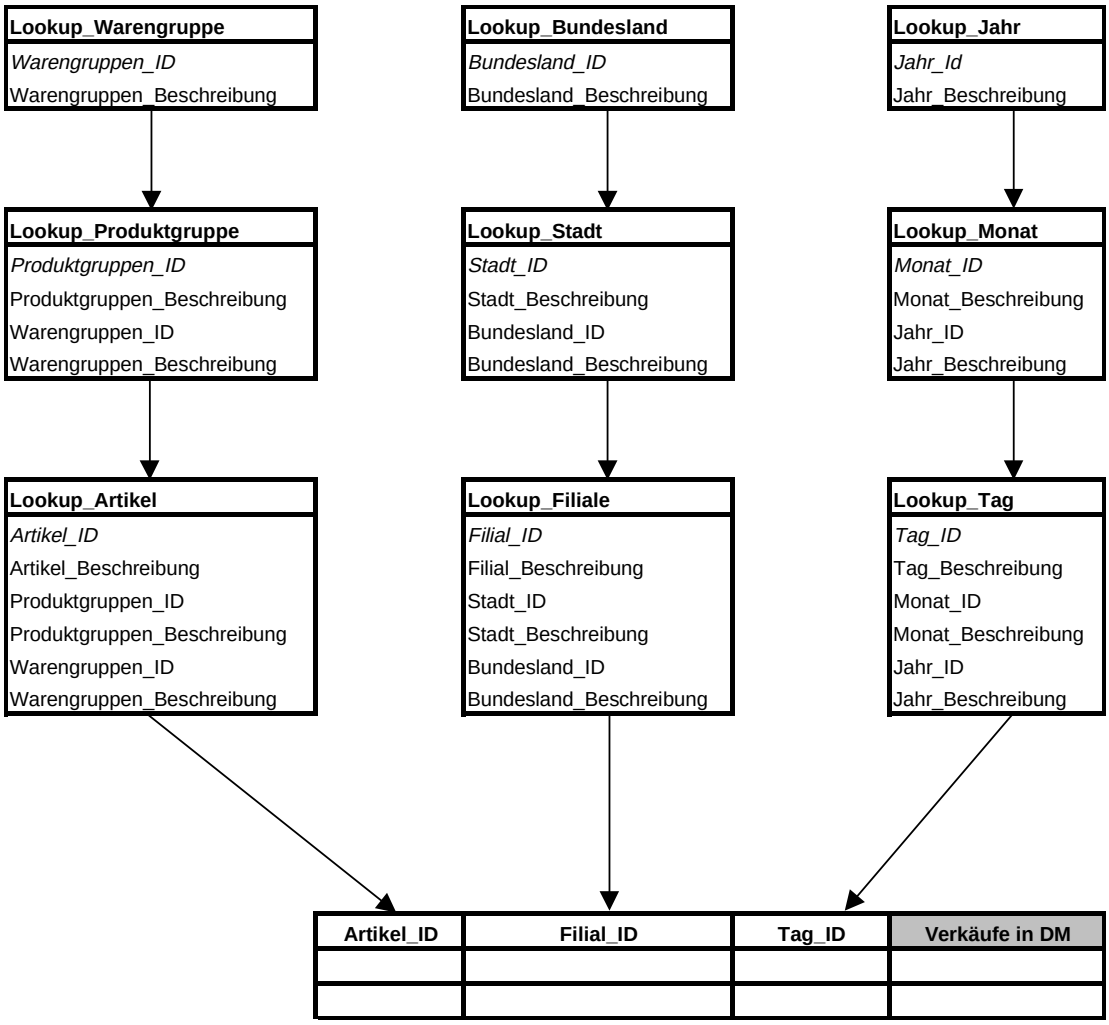


Abbildung 44: Normalised Star Schema #3

Vorteile:

- Wie Normalised Star Schema #1 und #2
- Durch vollkommene Denormalisierung immer maximal ein Join zwischen Fakt- und Dimensionstabelle notwendig.

  Lookup-Tabellen höherer Dimensionsattribute müssen trotzdem bestehen bleiben, da Fakttabellen mit einer höheren Granularität (siehe Kapitel 4.4.1) über die höheren Lookup-Tabellen einer Di-

mension gejoined werden müssen, um zusätzliche Informationen (außer der ID-Nummer) in der Analyse darstellen zu können.

Joined man solche Fakttabellen mit der untersten Lookup-Tabelle einer Dimension, so erhält man zwar auch die gewünschten Zusatzdaten, jedoch kommt es bei der Ermittlung der Kennzahlen zu einer Mehrfachzählung.

Beispiel:

Abbildung 45: Beispiel Double-Count Problem (Normalised Star Schema #3)

Double Count Select (FEHLER):

select b.warengruppen_beschreibung, a.umsatz ‚UMSATZ IN DM'
from umsatz_f a, ***lookup_artikel*** b
where a.warengruppen_id = b.warengruppen_id
and b.warengruppen_id = 15;

Die Ausgabe könnte etwa folgendermaßen aussehen:

Warengruppen Beschreibung	**UMSATZ IN DM**
Nahrungsmittel	50000,-
Nahrungsmittel	50000,-
Nahrungsmittel	50000,-
Nahrungsmittel	50000,-
.....	

Für jeden Artikel der Warengruppe 15 wird ein Join mit dem Datensatz der aggregierten Fakttabelle, der den Umsatz in DM für die Warengruppe 15 enthält, erzeugt. Somit enthält die Analyse n-Artikel-mal den Umsatz der Warengruppe 15 (die Warengruppe 15 enthält n Artikel).

Der richtige Select Befehl muß lauten:

select b.warengruppen_beschreibung, a.umsatz ‚UMSATZ IN DM'
from umsatz_f a, ***lookup_warengruppe*** b
where a.warengruppen_id = b.warengruppen_id
and b.warengruppen_id = 15;

Die SQL-Engine von MicroStrategy ist allerdings so intelligent, den richtigen SQL-Befehl abzusetzten, so daß ein Double-Count vermieden wird.

Nachteile:

- Gleiche Nachteile wie Normalised Star Schema #2
- Benötigt von allen ROLAP Modellen den meisten Speicherplatz durch die vollkommene Denormalisierung und der Notwendigkeit für höher aggregierte Fakttabellen die übergeordneten Lookup-Tabellen beizubehalten.

Bei allen Modellen ist darauf zu achten, daß die Verbindung der Lookup-Tabellen untereinander und mit der Fakttabelle über gleiche Schlüsselspaltenbezeichnungen geschieht.

Die Bildung von Primary- und Foreign-Key Beziehungen ist nicht notwendig, da die MicroStrategy Tools die Beziehungen der Tabellen untereinander nur über die gleichen Spaltenbezeichnungen erkennen. Eine Auswertung der Tabellen-Constraints erfolgt nicht.

Für den Prototypen der Fa. Oppenhoff & Rädler wird das **Normalised Star Schema #6** aus folgenden Gründen angewendet :

- Maximal ein Join für alle Analysen zwischen der Fakttabelle und jeder Dimensions Lookup-Tabelle notwendig.

- Alle Attribute einer Dimension können ohne Self-Joins in einer Analyse dargestellt werden.

- Es muß nicht extra ein Dimensionsschlüssel generiert werden. Diese Schlüssel sind alle schon im operativen System enthalten und müssen nur noch übernommen werden. Dadurch sind die notwendigen Load-Prozeduren weniger komplex aufzubauen.

- Bei Verwendung von höher aggregierten Fakttabellen muß zwar jeweils eine eigene Fakttabelle erstellt werden, die Auswertung dieser Tabelle ist allerdings performanter, da auch nur eine geringere Anzahl von Datensätzen zu scannen ist.

- Das Modell benötigt zwar den meisten Speicherplatz, da aber das geplante Data Warehouse der Firma Oppenhoff & Rädler aus maximal 10 Gbyte an Rohdaten besteht, macht sich dieser Nachteil des Modells kaum bemerkbar.

Auf den nächsten beiden Seiten werden die entsprechenden ROLAP Datenmodelle dargestellt.

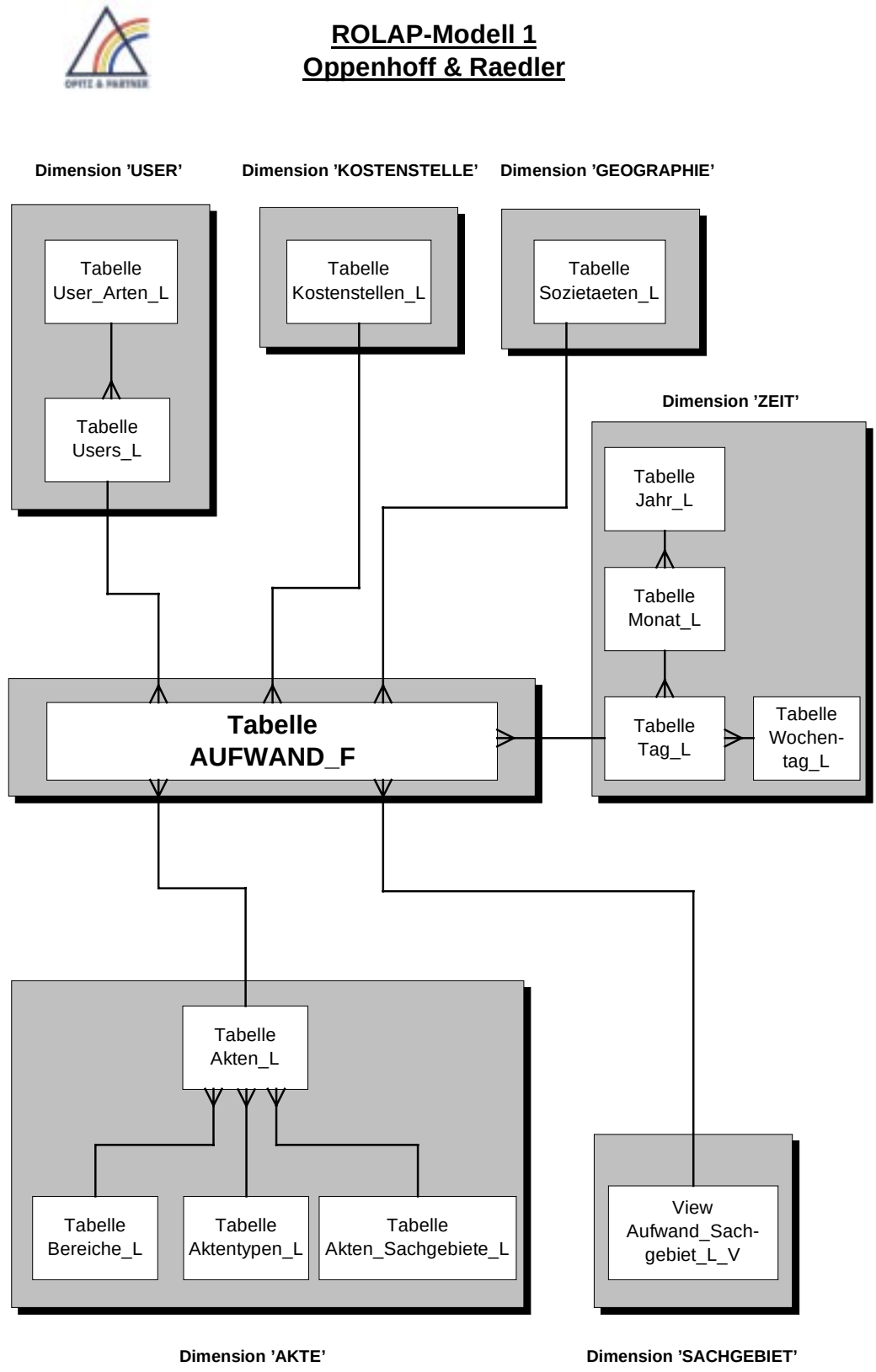


Abbildung 46: ROLAP-Modell 1 (Fa. Oppenhoff & Rädler)

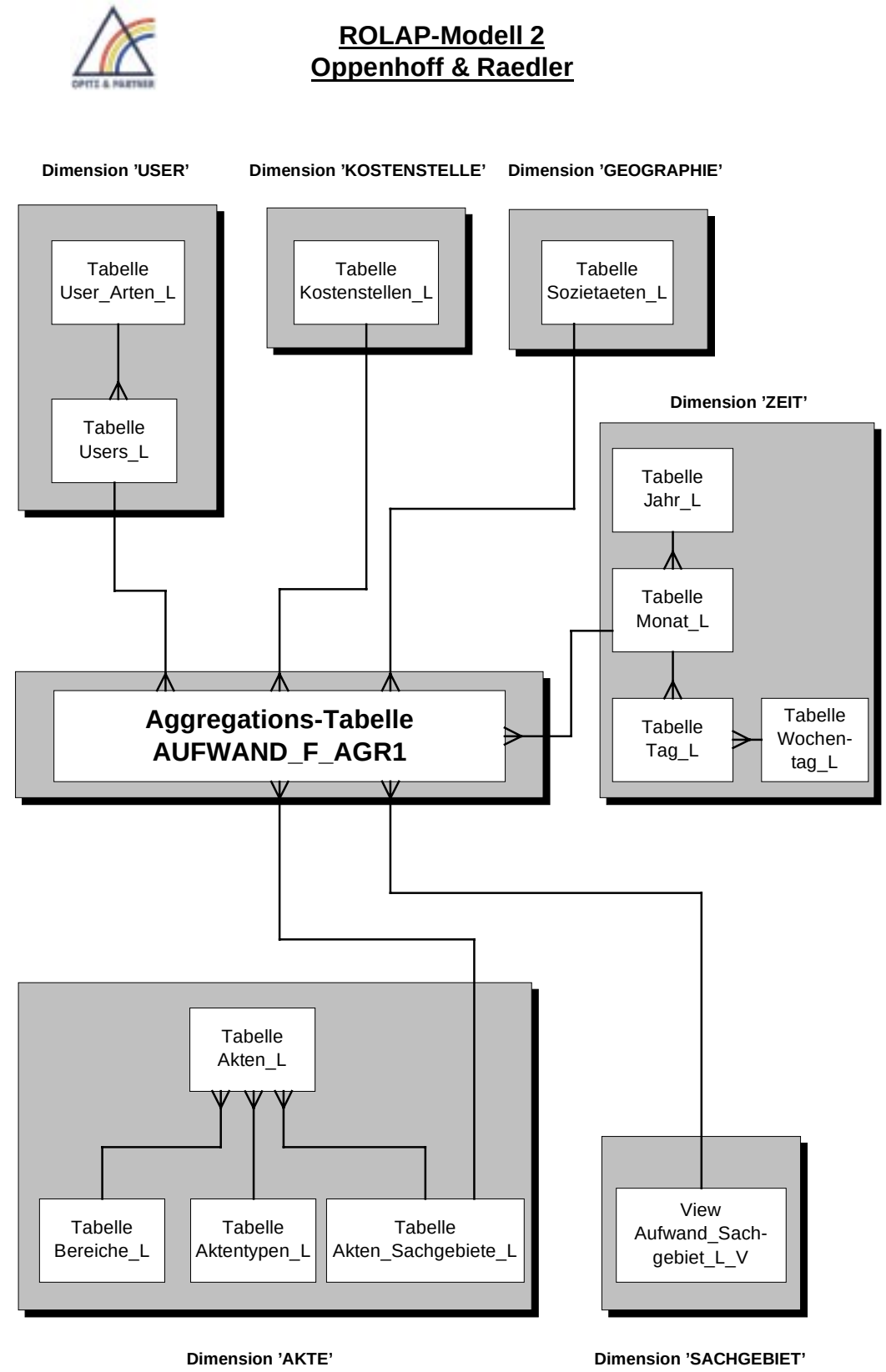


Abbildung 47: ROLAP-Modell 2 (Fa. Oppenhoff & Rädler)

7.7 Der Aufbau der Data Warehouse Datenbank

7.7.1 Einleitung

Bei der zu verwendenden ORACLE 8.0.5 Datenbank handelt es sich um eine sogenannte objekt-relationale Datenbank. Also eine relationale Datenbank, welche um objektorientierte Ansätze erweitert wurde. Leider können die objektorientierten Funktionalitäten, wie z. B. der Datentyp ‚table' (= **Nested Table**), nicht verwendet werden, da sie von den Tools der Firma MicroStrategy nicht erkannt werden, bzw. die zusätzlich notwendige SQL-Syntax zur Nutzung dieser Objekte nicht erzeugt werden kann. Für den Prototypen der Fa. Oppenhoff & Rädler bedeutet dies, daß nur der relationale Teil der Datenbank verwendet wird, der sich kaum von dem Funktionsumfang der ORACLE 7.3.4 Datenbank unterscheidet. Allerdings bietet die ORACLE 8.0.5 Datenbank eine zusätzliche nützliche Funktionalität, die horizontale Partitionierung (siehe Kapitel 4.4.2), welche für den Prototypen genutzt wird.

Der Aufbau der Data Warehouse Datenbank vollzieht sich durch folgende Schritte:

(1) Aufbau der eigentlichen Datenbank, also der Speicherstruktur in der Datenbankobjekte gespeichert und verwaltet werden.
(2) Aufbau der notwendigen Datenbankobjekte wie Tabellen, Views, und Indizes.

Die für den Punkt (1) notwendigen Arbeitsschritte sind nicht Bestandteil dieser Diplomarbeit, da sie den Rahmen der Diplomarbeit sprengen würden. Für den Punkt (2) soll zuerst erläutert werden, wie in einer ORACLE Datenbank Daten gespeichert werden und wie auf diese Daten zugegriffen wird.

Dieses Wissen ist notwendig um die notwendigen Datenbankobjekte wie Tabellen und Indizes optimal einzurichten.

7.7.2 Die ORACLE Speicherstruktur

Grundsätzlich gibt es folgende **logische** Speicherobjekte in denen die Daten einer Datenbank verwaltet werden[41]:

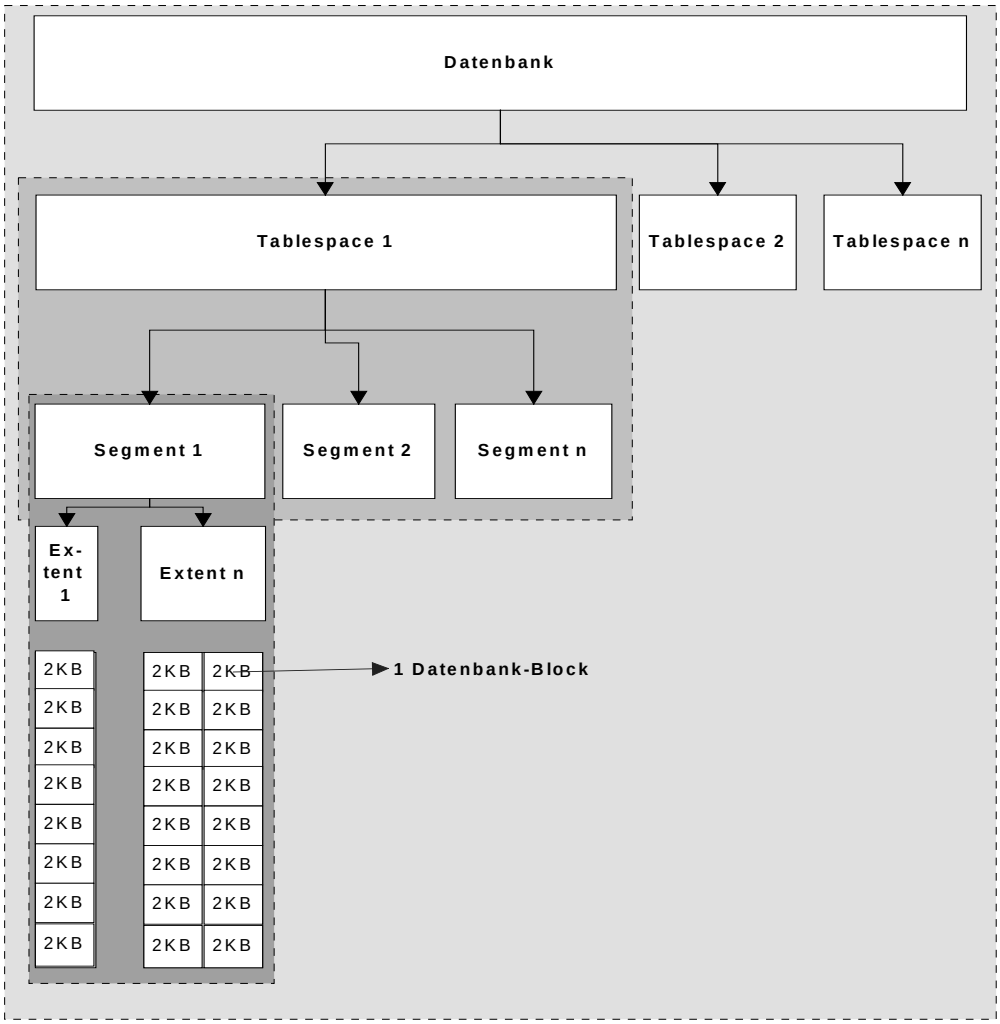


Abbildung 48: ORACLE Speicherstruktur

[41] ORACLE Online Dokumentation ‚Data Blocks, Extents, and Segments'

(1) Tablespace

Eine ORACLE Datenbank besteht in erster Linie aus sogenannten Tablespaces. Für jeden Tablespace wird mindestens ein physikalisches Datenfile angelegt, in das sämtliche Daten des Tablespace gespeichert werden. Ein Tablespace selber ist nochmals in ein oder mehrere Segmente zerlegt.

(2) Segment

Ein ORACLE Segment ist Bestandteil eines Tablespaces und kann in drei Formen existieren (je nach Art der Daten, die diesem Segment logisch zugeordnet werden):

(a) Table Segment

Ein Table Segment ist nichts anderes als eine Tabelle, zu der Daten, die zu dieser Tabelle gehören, logisch zugewiesen werden. Die Daten einer Tabelle werden auch nur einem Table Segment logisch zugewiesen.

(b) Index Segment

Im Gegensatz zu einem Table Segment werden einem Index Segment die Daten eines Tabellenindexes logisch zugeordnet.

(c) Rollback Segment

In einem Rollback Segment werden geänderte Daten einer Datenbank gehalten und zwar Änderungen, welche noch nicht durch ein COMMIT in der Datenbank festgeschrieben wurden und evtl. durch einen ROLLBACK Befehl wieder rückgängig gemacht werden könnten. In diesem Fall können die Ur-

sprungsdaten durch das Rollback Segment wiederhergestellt werden. Das Rollback Segment kann nur in einem bestimmten ROLLBACK Tablespace angelegt werden.

Jedes Segement ist wiedrum in ein oder mehrere Extents aufgeteilt.

(3) Extent

Ein Extent ist ein logischer Speicherbereich, der wiedrum in benachbarte Datenbank Blöcke zerlegt ist, die die Datenbank Daten, genauer die Daten einer Tabelle oder eines Indizes, speichern. Bei der Erstellung eines Indexes bzw. einer Tabelle kann genau festgelegt werden, in wieviele Extents die Daten gesplittet werden und wie groß die einzelnen Extents sein sollen.

Natürlich ist es besonders vorteilhaft, wenn die Daten einer Tabelle, bzw. eines Indexes, genau in einem Extent gespeichert werden, da ein Extent wiederum aus benachbarten Datenbank Blöcken besteht, die somit sequentiell eingelesen werden können. Bei Verteilung der Daten auf mehrere Extents kann es sein, daß die Daten auf dem Speichermedium wild verteilt werden, so daß der Zugriff auf diese Daten wesentlich langsamer erfolgt.

(4) Datenbank Block

Ein Datenbank Block stellt die letzte logische Speicherstufe der ORACLE Datenbank dar. Jeder Datenbank Block besteht aus einer plattformspezifischen Anzahl Bytes, die auf einem physikalischen Speichermedium, wie z. B. der Festplatte, innerhalb des für den übergeordneten Tablespace angelegten Datenfiles belegt werden und in denen die Daten der Datenbank physikalisch gespeichert werden. Ein Datenbank Block ist die kleinste I/O Einheit (Input/Output Einheit) die es in der Datenbankspeicherstruktur gibt. Folge davon ist, daß Daten immer nur in kompletten Datenbank Blöcken von der Festplatte gelesen werden und dementsprechend Daten auch immer nur in

kompletten Datenbank Blöcken auf die Festplatte geschrieben werden. Die Größe eines Datenbankblockes hängt von der unterstützten Blockgröße des Betriebssystems ab, auf dem die ORACLE Datenbank läuft.

Jeder Datenbank Block besitzt sogenannte **Header Informationen** welche für die Verwaltung der Daten, die in dem Datenbank Block gespeichert werden, benötigt werden. Die Größe des Headers beträgt etwa 80 – 110 Bytes und steht nicht zur Speicherung von Datenbank Daten zur Verfügung.

Zwei ORACLE Speicherverwaltungs Parameter ermöglichen es, den freien Speicherplatz eines Datenbank Blockes für INSERTS und UPDATES von Datensätzen in diesem Datenbank Block zu kontrollieren:

- **PCTFREE** gibt den Prozentsatz an, den ORACLE an Speicherplatz in einem Datenbank Block für einen neu eingefügten Datensatz für Update Operationen freihält.

  Dadurch können Änderungen an diesem Datensatz (UPDATE), durch den der Speicherplatzbedarf für diesen Datensatz erhöht wird, trotzdem in dem gleichen Datenbank Block eingefügt werden.

  Mit Hilfe dieser Technik wird das sogenannte **CHAINING** vermieden, also die Verteilung eines Datensatzes auf mehrere Datenbank Blöcke. Chaining sollte vermieden werden, da dadurch die Abfrageperformance rapide verschlechtert wird.

  Ein hoher PCTFREE Wert vermeidet zwar Chaining führt aber auch dazu, daß für die Speicherung der Tabelle mehrere Datenbank Blöcke benötigt werden, da der PCTFREE Speicherplatz nicht für neue Datensätze zur Verfügung steht. Durch eine Verteilung der Tabelle auf zu viele Datenbank Blöcke leidet ebenfalls die Abfrageperformance, da mehr Datenbank

Blöcke eingelesen werden müssen um z. B. einen Full Table Scan zu realisieren.

PCTFREE gibt zusätzlich das prozentuale Limit an, bis zu dem neue Datensätze in den Datenbank Block eingetragen werden können. Wird dieses Limit erreicht, so wird der Datenbank Block als ‚nicht verfügbar' eingestuft. Ein Eintrag eines neuen Datensatzes ist in diesen Datenbank Block erst dann wieder möglich, wenn der Prozentsatz des genutzten Speicherplatzes des Datenbank Blocks wieder unter dem Wert von PCTUSED fällt. Der Datenbank Block bekommt dann wieder den Status ‚frei'.

- **PCTUSED** gibt das prozentuale Limit des genutzten Speicherplatzes an, der unterschritten werden muß, bevor nach Erreichen von PCTFREE neue Datensätze eingetragen werden können.

Beispiel: PCTUSED 40%; PCTFREE 20%

Bei diesen Parameterwerten kann der Datenblock bis zu 80 % mit neuen Datensätzen gefüllt werden. Bei Erreichen dieses Limits wird der Datenbank Block als ‚**nicht verfügbar**' gekennzeichnet, neue Datensätze können nicht mehr eingetragen werden. Erst wenn der genutzte Speicherplatz dieses Datenbank Blockes auf unter 40 % sinkt wird der Datenbank Block als ‚**frei**' gekennzeichnet und neue Datensätze können wieder bis zum Limit PCTFREE eingetragen werden.

Ziel bei der Planung und Einrichtung der Datenbanktabellen und –indizes ist es nun, den zu erwartenden Speicherbedarf jeder Tabelle und jedes Indexes zu ermitteln, um somit die optimale Extentgröße der Tabelle bzw. eines Indexes festlegen zu können.

7.7.3 Planung der notwendigen Data Warehouse Tabellen, Views und Indizes

Bei der Erstellung der notwendigen Tabellen und Indizes ist, wie im vorherigen Kapitel festgestellt, unbedingt die optimale Extent-Größe zu ermitteln.

Da es am effektivsten ist, wenn der erwartete Inhalt einer Tabelle (Index) komplett in dem ersten Extent der Tabelle (Index) gespeichert wird, muß der erwartete Speicherplatz der Tabelle (Index) ermittelt werden. Die Byte Zahl die man dort erhält ist zugleich die optimale Größe des ersten Extents der Tabelle (Index).

Folgende Schritte sind zur Ermittlung des erwarteten Speicherplatzbedarfs einer Tabelle durchzuführen[42]:

Schritt 1: Tabellendesign

Design der einzelnen Tabellen (welche Spalten mit welchem Datentyp)

Schritt 2: Kalkulierung der Größe des Datenbank Block-Headers

Space after headers (hsize) = DB_BLOCK_SIZE
- KCBH
- UB4
- KTBBH
- ((INITRANS – 1) * KTBIT)
- KDBH

- **DB_BLOCK_SIZE** Wert erhält man aus der init.ora oder der Data Dictionary Tabelle V$PARAMETER

[42] Oracle Online Dokumentation ‚Space Estimations for Schema Objects'

(Normalerweise 2048 Byte bei WIN NT, aber durch entsprechende Formatierung des Datenträgers ist auch ein Vielfaches dieses Wertes realisierbar)

- **KCBH, UB4, KTBBH, KTBIT, KDBH** Werte erhält man aus der Data Dictionary Tabelle V$TYPE_SIZE
  (sie geben Auskunft über Basiswerte gemäß der verwendeten Plattform)

- **INITRANS** Wert gibt Auskunft darüber, wieviele Transaktionen simultan einen Datenblock updaten können. Dieser Wert ist natürlich abhängig von der Anzahl der User, die gleichzeitig auf einen Block (auf eine Tabelle) zugreifen könnten. Jeder Eintrag für eine Transaktion, die gerade auf einen Block zugreift belegt 24 Bytes freien Speicher des Blockes. Bei der Erstellung eines Indexes, bzw. einer Tabelle wird gleichzeitig auch der INITRANS Wert festgelegt. Default 1 für Tabellen, 2 für Indizes.

Schritt 3: Kalkulierung des verfügbaren Speicherplatzes für Daten pro Datenbank Block

Available data space (availspace) = AUFRUNDEN(hsize * (1 – PCTFREE/100))
- KDBT

Schritt 4: Kalkulierung des Speicherplatzbedarfes in Byte pro Tabellenspalte

Column size including byte length = column size + (1, if column size < 250, else 3)

Die eigentliche Column Size hängt von dem verwendeten Datentyp ab:

NUMBER 1 Byte
+ ABRUNDEN (**p** / 2) + 1Bytes

+ 1 Byte (nur bei negativen NUMBER Werten, bei denen die signifikante Stellenanzahl kleiner als 38 ist.)

= fixe Bytelänge, welche auch belegt wird, wenn kein Wert eingetragen ist

Definition: Number (**p**recision, scale)

DATE fixe Bytelänge von 7 Byte

CHAR fixe Bytelänge, abhängig von der definierten, zu belegenden Byteanzahl, z. B. CHAR(20) → 20 Byte

VARCHAR2 variable Bytelänge, abhängig von der definierten, maximal zu belegenden Byteanzahl und der tatsächlichen Anzahl an Bytes die benötigt weden, wenn ein bestimmter Wert gespeichert wird, z. B.:

VARCHAR2(20) → benötigt maximal 20 Byte

Bei einer Belegung mit ‚TEST' würden jedoch nur 4 Byte (bei einem single Byte Character Set) an Speicherplatz belegt.

5. Schritt: Kalkulierung der Row Size

Rowsize = 3 * UB1 (Row Header)
+ Summe des Speicherplatzbedarfes aller Spalten einer Tabelle (incl. Length Byte)

6. Schritt: Kalkulierung des Platzes der per Row benutzt wird

Rowspace = Maximum (UB1 * 3 + UB4 + SB2, Rowsize) + SB2

7. Schritt: Kalkulierung der Anzahl Rows die in einen Datenbank Block gespeichert werden können

Number of rows in block = ABRUNDEN (availspace / rowspace)

Beachten:

Die oben aufgeführten Schritte können den zu erwartenden Speicherplatz nur annähernd bestimmen. Durch die ermittelten Kennzahlen kann die INITIAL EXTENT Größe bestimmt werden, so daß eine Tabelle komplett in dem 1. Extent gespeichert werden kann.

Folgende Schritte sind zur Ermittlung des erwarteten Speicherplatzbedarfs eines Indexes durchzuführen:

1. Schritt: Kalkulierung der gesamten Datenbank Block-Header Größe

Block header = 113 Bytes (fixed Header)
\+ (24 * INITRANS) Bytes
= Header Größe in Bytes

2. Schritt: Kalkulierung des freien Speicherplatzes in einem Block für den Index

Available data space per block = (block size – block header)
\- ((block size – block header) * (PCTFREE /100)
= available date space in Byte

- Hier wird unter der Berücksichtigung von PCTFREE der frei verfügbare Speicherplatz eines Blockes ermittelt. PCTFREE wird prozentual von dem Block, abzüglich dem Header, ermittelt.

3. Schritt: Kalkulierung der Bytezahl pro Index-Spalte

4. Schritt: Kalkulierung der gesamten Index Größe

Bytes pro Indexeintrag = 2 Bytes (entry Header)
+ 6 Bytes (ROWID length)
+ gesamte Anzahl Length Bytes (1 byte pro Spalte < = 127 Byte)
+ gesamte Anzahl Length Bytes (2 byte pro Spalte > 127 Byte)
+ summierte Spaltenbytezahl
= Bytes pro Indexeintrag

5. Schritt: Kalkulierung der benötigten Anzahl Datenbank Blöcke pro Index

Anzahl Blöcke = AUFRUNDEN(
Anzahl Einträge in entsprechenden Spalten (ohne Null Werte)
/ ABRUNDEN(available Data Space per Block / Byte pro Indexeintrag))
= Anzahl benötigter Blöcke pro Index

6. Schritt: Kalkulierung der optimalen Extentgröße des 1. Extents

optimale Extentgröße in Byte = Anzahl Blöcke * Datenbank Blockgröße in Byte

Beachten:

Die oben aufgeführten Schritte können den zu erwartenden Speicherplatz nur annähernd bestimmen. Durch die ermittelten Kennzahlen kann die INITIAL EXTENT Größe bestimmt werden, so daß ein Index komplett in dem 1. Extent gespeichert werden kann.

Die Kalkulation des benötigten Speicherplatzbedarfs wird dadurch erleichtert, daß die Anzahl der zu übernehmenden historischen Datensätze des operativen Systems bekannt ist. Die Anzahl der noch hinzukommenden Datensätze des laufenden Geschäftsjahres kann selbstverständlich nur geschätzt werden und beruht in der Regel auf Vergangenheitswerten bzw. auf Aussagen der Geschäftsleitung (siehe auch Kapitel 4.4.1).

Die Wahl der Größe des ersten Extents einer Tabelle, bzw. eines Indexes, hat nur vorübergehenden Charakter. Nach Ablauf einer gewissen Zeitspanne (normalerweise mehrere Monate) ist eine Reorganisation der Tabellen und Indizes unvermeidbar, da durch neu hinzugefügte Daten in das Data Warehouse eine neue Berechnung der Speicherparameter aus Performancegründen notwendig wird.

Die Wahl der geeigneten Datenblock Parameter PCTFREE und PCTUSED gestaltet sich etwas schwieriger, da die Auswirkungen dieser Parameter nur sehr schlecht überprüft werden können. Generell gilt aber für diesen Data Warehouse Prototypen, daß der PCTFREE Wert nicht sehr hoch angesetzt werden muß, da Daten in das Data Warehouse generell nur geinserted werden (**Nicht-Volatilität** der Daten). Ein Update dieser eingefügten Datensätze findet nur in Ausnahmefällen statt. Das sogenannte Chaining der Datensätze tritt somit bei einem niedrigen PCTFREE Wert nur äußerst selten auf, so daß dieser Nachteil vernachlässigt werden kann. Stattdessen kann dementsprechend der PCTUSED Wert höher gesetzt werden, damit der Datenbank Block möglichst schnell wieder auf den Status ‚frei' gesetzt wird.

In der nachfolgenden Tabelle werden die angelegten Data Warehousetabellen und ihr Aufbau aufgeführt. Dabei wird für jede Tabellenspalte, die Bestandteil eines Indexes ist, der Name des angelegten Tabellenindexes aufgeführt. Zusätzlich wird die berechnete Größe des 1. und 2. Extents der Tabellen aufgeführt.

Generell kann man zu den angelegten Indizes sagen, daß es im Vorfeld sehr schwer abzuschätzen ist, auf welche Tabellenspalte, bzw. Kombination von Tabellenspalten, ein Index gelegt werden sollte, da noch nicht bekannt ist, welche Spalten durch die OLAP Analysen in der ‚where' Klausel häufig verwendet werden (siehe auch Kapitel 5.1.1). Grundsätzlich entsteht aber für diesen ROLAP Prototypen kein Nachteil darin im Vorfeld möglichst viele Indizes auf Tabellenspalten zu legen, da zeitaufwendigere Updates, welche ja nicht nur die eigentlichen Daten sondern auch den Index betreffen, im Data Warehouse Umfeld keine Rolle spielen. Lediglich der für die Indizes benötigte Speicherplatz und der größere zeitliche Aufwand für Insert Operationen bei Refresh-Ladevorgängen (siehe Kapitel 6.2) muß beachtet werden.

Für den Prototypen werden grundsätzlich nur B*tree Indizes angelegt. Einzig die Indizes ‚IDX8_AUFWAND_F' und ‚IDX7_AUFWAND_F_AGR1' werden als Bitmap Indizes angelegt, da hier die Kardinalität unter 0,1 % liegt (siehe Kapitel 5.1.1).

Tabelle JAHR_L (INITIAL 1 K; NEXT 2 K)

Name	Null?	Type	Index
JAHR_ID		VARCHAR2(4)	

Tabelle MONAT_L (INITIAL 1 K; NEXT 2 K)

Name	Null?	Type	Index
MONA_ID		VARCHAR2(4)	IDX1_MONAT_L
MONA_BEZ		VARCHAR2(9)	IDX2_MONAT_L
JAHR_ID		VARCHAR2(4)	IDX3_MONAT_L

Tabelle TAG_L (INITIAL 55 K; NEXT 83 K)

Name	Null?	Type	Index
TAG_ID		DATE	IDX1_TAG_L
WOCHEN_TAG_ID		NUMBER(1)	IDX2_TAG_L
WOCHEN_TAG_BEZ		VARCHAR2(10)	IDX3_TAG_L
MONA_ID		VARCHAR2(4)	IDX4_TAG_L
MONA_BEZ		VARCHAR2(9)	IDX5_TAG_L
JAHR_ID		VARCHAR2(4)	IDX6_TAG_L

Tabelle WOCHEN_TAG_L (INITIAL 1 K; NEXT 2 K)

Name	Null?	Type	Index
WOCHEN_TAG_ID		NUMBER(1)	
WOCHEN_TAG_BEZ		VARCHAR2(10)	

Tabelle KOSTENSTELLEN_L (INITIAL 6 K; NEXT 9 K)

Name	Null?	Type	Index
KOST_CODE_ID		NUMBER(6)	IDX1_KOSTENSTELLEN_L
KOST_BEZEICHNUNG_BEZ		VARCHAR2(60)	IDX2_KOSTENSTELLEN_L

Tabelle USER_ARTEN_L (INITIAL 2 K; NEXT 3 K)

Name	Null?	Type	Index
UART_CODE_ID		VARCHAR2(5)	IDX1_USER_ARTEN_L
UART_BEZEICHNUNG_BEZ		VARCHAR2(60)	IDX2_USER_ARTEN_L

Tabelle SOZIETAETEN_L (INITIAL 1 K; NEXT 2 K)

Name	Null?	Type	Index
SORT_CODE_ID		VARCHAR2(5)	IDX1_SOZiETAETEN_L
SORT_BEZEICHNUNG_BEZ		VARCHAR2(60)	IDX2_SOZiETAETEN_L

Tabelle USERS_L (INITIAL 74 K; NEXT 111 K)

Name	Null?	Type	Index
USER_KENNUNG_ID		VARCHAR2(10)	IDX1_USERS_L
USER_NACHNAME_L		VARCHAR2(40)	IDX2_USERS_L
USER_VORNAME_L		VARCHAR2(30)	IDX3_USERS_L
UART_CODE_ID		VARCHAR2(5)	IDX4_USERS_L
UART_BEZEICHNUNG_BEZ		VARCHAR2(60)	IDX5_USERS_L

Tabelle BEREICHE_L (INITIAL 2 K; NEXT 3 K)

Name	Null?	Type	Index
BRCH_CODE_ID		VARCHAR2(5)	IDX1_BEREICHE_L
BRCH_BEZEICHNUNG_BEZ		VARCHAR2(60)	IDX2_BEREICHE_L

Tabelle AKTENTYPEN_L (INITIAL 2 K; NEXT 3 K)

Name	Null?	Type	Index

ATYP_CODE_ID	VARCHAR2(5)	IDX1_AKTENTYPEN_L
ATYP_BEZEICHNUNG_BEZ	VARCHAR2(60)	IDX2_AKTENTYPEN_L

Tabelle AKTEN_SACHGEBIETE_L (INITIAL 6 K; NEXT 9 K)

Name	Null?	Type	Index
SACH_CODE_ID		VARCHAR2(5)	IDX1_AKTEN_SACHGEBIETE_L
SACH_BEZEICHNUNG_BEZ		VARCHAR2(60)	IDX2_AKTEN_SACHGEBIETE_L

Tabelle AKTEN_L (INITIAL 4965 K; NEXT 7448 K)

Name	Null?	Type	Index
AKTE_AKTENNR_ID		NUMBER(8)	IDX1_AKTEN_L
BRCH_CODE_ID		VARCHAR2(5)	IDX2_AKTEN_L
BRCH_BEZEICHNUNG_BEZ		VARCHAR2(60)	IDX3_AKTEN_L
ATYP_CODE_ID		VARCHAR2(5)	IDX4_AKTEN_L
ATYP_BEZEICHNUNG_BEZ		VARCHAR2(60)	IDX5_AKTEN_L
SACH_CODE_ID		VARCHAR2(5)	IDX6_AKTEN_L
SACH_BEZEICHNUNG_BEZ		VARCHAR2(60)	IDX7_AKTEN_L

View AUFWAND_SACHGEBIETE_L_V

Name	Null?	Type
SACH_CODE_ID_V		VARCHAR2(5)
SACH_BEZEICHNUNG_BEZ_V		VARCHAR2(60)

Tabelle AUFWAND_F (siehe CREATE TABLE Statement S. 156)

Name	Null?	Type	Index
AZEI_ID		NUMBER(15)	IDX1_AUFWAND_F

TAG_ID	DATE	IDX2_AUFWAND_F
AKTE_AKTENNR_ID	NUMBER(8)	IDX3_AUFWAND_F
USER_KENNUNG_ID	VARCHAR2(10)	IDX4_AUFWAND_F
SACH_CODE_ID_V	VARCHAR2(5)	IDX5_AUFWAND_F
SORT_CODE_ID	VARCHAR2(5)	IDX6_AUFWAND_F
KOST_CODE_ID	NUMBER(6)	IDX7_AUFWAND_F
AZEI_STATUS_KZ_Q	VARCHAR2(1)	IDX8_AUFWAND_F
AZEI_DAUER_F	NUMBER(5,2)	IDX9_AUFWAND_F
AZEI_STUNDENSATZ_F	NUMBER(6,2)	IDX10_AUFWAND_F
AZEI_BETRAG_F	NUMBER(11,2)	IDX11_AUFWAND_F
AZEI_BETRAG_BEZAHLT_F	NUMBER(11,2)	IDX12_AUFWAND_F

Tabelle AUFWAND_F_AGR1 (INITIAL 1927 K; NEXT 2891 K)

Name	**Null?**	**Type**	**Index**
MONA_ID		VARCHAR2 (4)	IDX1_AUFWAND_F_AGR1
SACH_CODE_ID		VARCHAR2(5)	IDX2_AUFWAND_F_AGR1
USER_KENNUNG_ID		VARCHAR2(10)	IDX3_AUFWAND_F_AGR1
SACH_CODE_ID_V		VARCHAR2(5)	IDX4_AUFWAND_F_AGR1
SORT_CODE_ID		VARCHAR2(5)	IDX5_AUFWAND_F_AGR1
KOST_CODE_ID		NUMBER(6)	IDX6_AUFWAND_F_AGR1
AGR1_AZEI_STATUS_KZ_Q		VARCHAR2(1)	IDX7_AUFWAND_F_AGR1
AGR1_AZEI_DAUER_F		NUMBER(5,2)	IDX8_AUFWAND_F_AGR1
AGR1_AZEI_BETRAG_F		NUMBER(11,2)	IDX9_AUFWAND_F_AGR1
AGR1_AZEI_BETRAG_BEZAHLT_F		NUMBER(11,2))	IDX10_AUFWAND_F_AGR1

Die Create Table und Create Index Statements erfolgen in der üblichen Form:

Beispiel:

```
create table MONAT_L(
MONA_ID                    varchar2(4),
MONA_BEZ                   varchar2(9),
```

```
JAHR_ID                          varchar2(4))
tablespace WAREHOUSE1
storage(initial                      1K
    next                             1K
    minextents                       1
    maxextents                       2
    pctincrease                      0)
pctfree 5
pctused   75;

create index IDX1_MONAT_L on MONAT_L(MONA_ID)
tablespace INDEX2
storage(initial                      2K
    next                             1K
    minextents                       1
    maxextents                       2
    pctincrease                      0)
pctfree 5;
```

Eine Besonderheit stellt jedoch die Tabelle ‚Aufwand_F' dar. Diese Tabelle wird als **horizontal partitionierte Tabelle** erstellt, da diese Tabelle über 300.000 Datensätze aufweist und durch die Partitionierung ein performanterer Zugriff ermöglicht wird.

Zur Auswahl der geeigneten Partitionierung mußte der Inhalt der zu Grunde liegenden operativen Tabelle ‚Aufwaende_Zeitaufwand' genauer analysiert werden:

Ermittlung der Anzahl Aufwendungen pro Jahr:

```
select to_char(azei_aufwand_vom, 'YYYY') JAHR, count(azei_aufwand_vom) AN-
ZAHL_AUFWENDUNGEN
from aufwaende_zeitaufwand
group by to_char(azei_aufwand_vom, 'YYYY');
```

JAHR	ANZAHL_AUFWENDUNGEN
1886	4
1887	2
1965	2

1966	1
1968	2
1969	1
1976	4
1977	2
1988	4
1989	2
1994	18
1995	30643
1996	167519
1997	117337
1998	20663

Bei der Auswahl der Partitionseinteilung ist zu beachten, daß relativ kleine Partitionen aus einer kleineren Anzahl Datensätzen bestehen, und somit schneller auf Datensätze dieser Partitionen zugegriffen werden kann. Auf der anderen Seite wächst allerdings die Anzahl der notwendigen Partitionen, um alle Datensätze der Tabelle abzubilden. Dadurch werden Scans über die gesamte Tabelle langsamer, da viel mehr Partitionen gejoined werden müssen.

Aufgrund dieser Erkenntnisse und der Vereinbarungen mit der Firma Oppenhoff & Rädler, nur Aufwandsdaten der Jahre 1996, 1997 und 1998 in dem Prototypen aufzunehmen, entstand folgende Partitionierungseinteilung:

```
create table AUFWAND_F(
AZEI_ID                 number(15),
TAG_ID                  date,
AKTE_AKTENNR_ID         number(8),
USER_KENNUNG_ID         varchar2(10),
SACH_CODE_ID_V          varchar2(5),
SORT_CODE_ID            varchar2(5),
KOST_CODE_ID            number(6),
AZEI_STATUS_KZ_Q        varchar2(1),
AZEI_DAUER_F            number(5,2),
AZEI_STUNDENSATZ_F      number(6,2),
AZEI_BETRAG_F           number(11,2),
AZEI_BETRAG_BEZAHLT_F   number(11,2))
pctfree 5
pctused 75
```

```
partition by range(TAG_ID)
        (partition AUFWAND_1996_1 values less than (to_date('01-JUL-1996','DD-MON-
YYYY'))
        tablespace warehouse1
        storage(initial         6856K
            next                3428K
            minextents 1
            maxextents          2
            pctincrease         0),
        partition AUFWAND_1996_2 values less than (to_date('01-JAN-1997','DD-MON-
YYYY'))
        tablespace warehouse2
        storage(initial         9099K
            next                4550K
            minextents 1
            maxextents          2
            pctincrease         0),
        partition AUFWAND_1997_1 values less than (to_date('01-JUL-1997','DD-MON-
YYYY'))
        tablespace warehouse1
        storage(initial         7347K
            next                3674K
            minextents 1
            maxextents          2
            pctincrease         0),
        partition AUFWAND_1997_2 values less than (to_date('01-JAN-1998','DD-MON-
YYYY'))
        tablespace warehouse2
        storage(initial         3829K
            next                1915K
            minextents 1
            maxextents          2
            pctincrease         0),
        partition AUFWAND_1998 values less than (to_date('01-JAN-1999','DD-MON-
YYYY'))
        tablespace warehouse1
        storage(initial         1966K
            next                983K
            minextents 1
```

```
        maxextents      2
        pctincrease     0)
);
```

Zusätzlicher Aufwand bei der Wahl der horizontalen Partitionierung besteht in der Erstellung der Indizes auf diese partitionierte Tabelle, da es natürlich nur Sinn macht auch diese Indizes horizontal zu partitionieren und zwar einheitlich zu dem Partitionierungsformat der basierenden Tabelle (= **Equipartitionierung**, siehe auch Kapitel 5.2.1.2).

Beispiel: IDX1_AUFWAND_F

```
create index IDX1_AUFWAND_F on AUFWAND_F(TAG_ID)
local (     partition IDX_AUFWAND_1996_1
            tablespace INDEX2
            storage(initial     1350K
                next            675K
                minextents 1
                maxextents      2
                pctincrease     0)
            pctfree 5,
            partition IDX_AUFWAND_1996_2
            tablespace INDEX1
            storage(initial     1792K
                next            896K
                minextents 1
                maxextents      2
                pctincrease     0)
            pctfree 5,
            partition IDX_AUFWAND_1997_1
            tablespace INDEX2
            storage(initial     1448K
                next            724K
```

```
    minextents 1
    maxextents 2
    pctincrease 0)
pctfree 5,
partition IDX_AUFWAND_1997_2
tablespace INDEX1
storage(initial 754K
    next 377K
    minextents 1
    maxextents 2
    pctincrease 0)
pctfree 5,
partition IDX_AUFWAND_1998
tablespace INDEX2
storage(initial 388K
    next 194K
    minextents 1
    maxextents 2
    pctincrease 0)
pctfree 5
);
```

7.7.4 Planung der notwendigen Load Tabellen

Der Daten-Load des Data Warehouse Prototyps vollzieht sich in zwei Stufen:

Stufe 1:

Nur Datensätze die sich seit dem letzten Datenload geändert haben, bzw. neu eingetragen wurden, werden von dem Server des operativen Systems auf den Server des Data Warehouses in die entsprechenden Load-Tabellen geladen. Bei der 1. Loadstufe wird weder eine Datentransformation, noch eine Datenaggregation durchgeführt.

Stufe 2:

Sämtliche Datensätze die sich in der Load-Tabelle befinden, werden in das Data Warehouse übernommen. Dabei werden alle notwendigen Datentransformationen und –aggregationen vorgenommen. Nach erfolgreicher Übernahme eines Datensatzes wird dieser aus der entsprechenden Load-Tabelle gelöscht.

Gründe für den zweistufigen Load:

(1) Dadurch, daß die zeitaufwendigen Datentransformationen und –aggregationen auf dem Data Warehouse Server durchgeführt werden, ist das operative System für eine wesentlich kürzere Zeit nicht für das operative Tagesgeschäft verfügbar.

(2) Sollte bei der Datenübertragung in die Load-Tabellen ein Fehler auftreten, der den Loadprozeß abbricht, so kann der Fehler behoben werden und der Load-Prozeß ohne Probleme erneut ausgeführt werden, da sich noch keine der neuen Daten im eigentlichen Data Warehouse befinden. Sollte bei der Datenübertragung zwischen Load-Tabellen und Data Warehousetabellen ein Fehler auftreten, so kann nach der Behebung des Fehlers ein erneuter Ladevorgang für die noch verbliebenen Datensätze durchgeführt werden.

Die Planung der Load-Tabellen und der entsprechenden Indizes erfolgt nach den gleichen Kriterien wie die Planung der Data Warehouse Tabellen (siehe Kapitel 7.3.3).

In der nachfolgenden Tabelle werden die angelegten Load Tabellen und ihr Aufbau aufgeführt. Dabei wird für jede Tabellenspalte, die Bestandteil eines

Indexes ist, der Name des angelegten Tabellenindexes aufgeführt. Zusätzlich wird die berechnete Größe des 1. Extents der Tabellen aufgeführt. Das zweite Extent ist unnötig da das 1. Extent die maximale Größe besitzt, die für den Initial-Load (siehe Kapitel 6.2) der Daten notwendig ist. Da die Daten nach der 2. Load-Stufe wieder aus den Load-Tabellen gelöscht werden, werden die Tabellen auch nicht über das erste Extent hinaus anwachsen:

Tabelle LOAD_KOSTENSTELLEN (INITIAL 7 K)

Name	Null?	Type	Index
KOST_CODE		NUMBER(6)	
KOST_BEZEICHNUNG		VARCHAR2(60)	
KOST_DSATZ_STAT		VARCHAR2(1)	

Tabelle LOAD_USER_ARTEN (INITIAL 2 K)

Name	Null?	Type	Index
UART_CODE		VARCHAR2(5)	
UART_BEZEICHNUNG		VARCHAR2(60)	
UART_DSATZ_STAT		VARCHAR2(1)	

Tabelle LOAD_SOZIETAETEN (INITIAL 1 K)

Name	Null?	Type	Index
SORT_CODE		VARCHAR2(5)	
SORT_BEZEICHNUNG		VARCHAR2(60)	
SORT_DSATZ_STAT		VARCHAR2(1)	

Tabelle LOAD_USERS (INITIAL 47 K)

Name	Null?	Type	Index
USER_KENNUNG		VARCHAR2(10)	
USER_NACHNAME		VARCHAR2(40)	
USER_VORNAME		VARCHAR2(30)	
UART_CODE		VARCHAR2(5)	
USER_DSATZ_STAT		VARCHAR2(1)	

Tabelle LOAD_BEREICHE (INITIAL 2 K)

Name	Null?	Type	Index
BRCH_CODE		VARCHAR2(5)	
BRCH_BEZEICHNUNG		VARCHAR2(60)	
BRCH_DSATZ_STAT		VARCHAR2(1)	

Tabelle LOAD_AKTENTYPEN (INITIAL 2 K)

Name	Null?	Type	Index
ATYP_CODE		VARCHAR2(5)	
ATYP_BEZEICHNUNG		VARCHAR2(60)	
ATYP_DSATZ_STAT		VARCHAR2(1)	

Tabelle LOAD_SACHGEBIETE (INITIAL 6 K)

Name	Null?	Type	Index
SACH_CODE		VARCHAR2(5)	
SACH_BEZEICHNUNG		VARCHAR2(60)	
SACH_DSATZ_STAT		VARCHAR2(1)	

Tabelle LOAD_AKTEN (INITIAL 685 K)

Name	Null?	Type	Index
AKTE_AKTENNR		NUMBER(8)	
BRCH_CODE		VARCHAR2(5)	
ATYP_CODE		VARCHAR2(5)	
SACH_CODE		VARCHAR2(5)	
AKTE_DSATZ_STAT		VARCHAR2(1)	

Tabelle LOAD_AUFWAENDE_ZEITAUFWAND (INITIAL 30552 K)

Name	Null?	Type	Index
AZEI_ID		NUMBER(15)	IDX1_LOAD_AZEI
TAG_ID		DATE	
AKTE_AKTENNR		NUMBER(8)	

USER_KENNUNG	VARCHAR2(10)	
SACH_CODE	VARCHAR2(5)	
SORT_CODE	VARCHAR2(5)	
KOST_CODE	NUMBER(6)	
AZEI_STATUS_KZ	VARCHAR2(1)	
AZEI_DAUER	NUMBER(5,2)	
AZEI_STUNDENSATZ	NUMBER(6,2)	
AZEI_BETRAG	NUMBER(11,2)	
AZEI_BETRAG_BEZAHLT	NUMBER(11,2)	
AZEI_DSATZ_STAT	VARCHAR2(1)	IDX2_LOAD_AZEI

Bei dem Index ‚IDX2_LOAD_AZEI' handelt es sich um einen Bitmap Index, da die Kardinalität dieser Spalte unter 0,1 % liegt.

7.7.5 Planung der notwendigen Data Warehouse Tablespaces

Da für den Prototypen zwei Festplatten zur Verfügung stehen, empfiehlt es sich die Daten des Data Warehouses auf diese beiden Festplatten zu verteilen, um evtl. einen Performancevorteil durch die Verteilung der I/O Zugriffe zu erzielen. Natürlich handelt es sich bei diesem Prototypen um eine eher schwache Hardwareausstattung, so daß eine Verteilung der Daten kaum sinnvoll erscheint. Für die Erleuterung des Prinzips der I/O Verteilung reicht sie aber vollkommen aus.

Bei der Verteilung der Data Warehouse Daten auf die Beiden zur Verfügung stehenden Festplatten mußte folgendes beachtet werden:

- Trennung der Index- und der Tabellendaten, so daß bei Nutzung eines Indexes schneller auf die eigentlichen Tabellendaten zugegriffen werden kann.
- Trennung der einzelnen Tabellen- (Index-)partitionen (siehe Kapitel 7.3.4), so daß Joins über mehrere Partitionen schneller vollzogen werden können.

- Trennung der Load-Tabellen der Stufe 1 von den Data Warehouse Tabellen (Ziel der Loadstufe 2), um zumindest einen Teil der Datenübernahme der Stufe 2 zu beschleunigen.

Die notwendige Größe des Datenfiles jedes Tablespaces hängt selbstverständlich davon ab, welche Tabellen und Indizes mit welchen Extentgrößen dem jeweiligen Tablespace zugewiesen werden.

Die nachfolgende Tabelle gibt die notwendigen Tablespaces und die Zuordnung der Datenbankobjekte zu den Tablespaces wieder:

TBS 'WAREHOUSE1'			
	1. Extent	1.+ 2. Extent	
Tabelle 'JAHR_L'	1	2	Kbyte
Tabelle 'MONAT_L'	1	2	Kbyte
Tabelle 'TAG_L'	55	83	Kbyte
Tabelle 'WOCHEN_TAG_L'	1	2	Kbyte
Tabelle 'KOSTENSTELLEN_L'	6	9	Kbyte
Tabelle 'USER_ARTEN_L'	2	3	Kbyte
Tabelle 'SOZIETAETEN_L'	1	2	Kbyte
Tabelle 'USERS_L'	74	111	Kbyte
Tabelle 'BEREICHE_L'	2	3	Kbyte
Tabelle 'AKTENTYPEN_L'	2	3	Kbyte
Tabelle 'AKTEN_SACHGEBIETE_L'	6	9	Kbyte
Tabelle 'AKTEN_L'	4965	7448	Kbyte
View 'AUFWAND_SACHGEBIETE_L_V'	0	0	Kbyte
Tabelle 'Aufwand_F_AGR1'	1927	2891	Kbyte
Tabelle 'AUFWAND_F' Part 'AUFWAND_1996_1'	6856	10284	Kbyte
Tabelle 'AUFWAND_F' Part 'AUFWAND_1997_1'	7347	11021	Kbyte
Tabelle 'AUFWAND_F' Part 'AUFWAND_1998'	1966	2949	Kbyte
Speicherplatzbedarf für TBS 'WAREHOUSE1'	23212	34822	Kbyte

TBS 'WAREHOUSE2'			
	1. Extent	1.+ 2. Extent	
Tabelle 'AUFWAND_F' Part 'AUFWAND_1996_2'	9099	13649	Kbyte
Tabelle 'AUFWAND_F' Part 'AUFWAND_1997_2'	3829	5744	Kbyte
Speicherplatzbedarf für TBS 'WAREHOUSE2'	12928	19393	Kbyte

TBS 'INDEX1'			
	1. Extent	1.+ 2. Extent	
Index 'IDX1_LOAD_AZEI'	6112	6112	Kbyte
Index 'IDX2_LOAD_AZEI'	3226	3226	Kbyte
Index 'IDX1_AUFWAND_F' Part 'IDX_AUFWAND_96_2'	1792	2688	Kbyte
Index 'IDX1_AUFWAND_F' Part 'IDX_AUFWAND_97_2'	754	1131	Kbyte
Index 'IDX2_AUFWAND_F' Part 'IDX_AUFWAND_96_2'	1686	2529	Kbyte
Index 'IDX2_AUFWAND_F' Part 'IDX_AUFWAND_97_2'	710	1065	Kbyte
Index 'IDX3_AUFWAND_F' Part 'IDX_AUFWAND_96_2'	2136	3204	Kbyte
Index 'IDX3_AUFWAND_F' Part 'IDX_AUFWAND_97_2'	900	1350	Kbyte
Index 'IDX4_AUFWAND_F' Part 'IDX_AUFWAND_96_2'	1568	2352	Kbyte
Index 'IDX4_AUFWAND_F' Part 'IDX_AUFWAND_97_2'	660	990	Kbyte
Index 'IDX5_AUFWAND_F' Part 'IDX_AUFWAND_96_2'	1010	1515	Kbyte
Index 'IDX5_AUFWAND_F' Part 'IDX_AUFWAND_97_2'	426	639	Kbyte
Index 'IDX6_AUFWAND_F' Part 'IDX_AUFWAND_96_2'	1466	2199	Kbyte
Index 'IDX6_AUFWAND_F' Part 'IDX_AUFWAND_97_2'	618	927	Kbyte
Index 'IDX7_AUFWAND_F' Part 'IDX_AUFWAND_96_2'	1568	2352	Kbyte
Index 'IDX7_AUFWAND_F' Part 'IDX_AUFWAND_97_2'	660	990	Kbyte
Index 'IDX8_AUFWAND_F' Part 'IDX_AUFWAND_96_2'	1792	2688	Kbyte
Index 'IDX8_AUFWAND_F' Part 'IDX_AUFWAND_97_2'	754	1131	Kbyte
Index 'IDX9_AUFWAND_F' Part 'IDX_AUFWAND_96_2'	1792	2688	Kbyte
Index 'IDX9_AUFWAND_F' Part 'IDX_AUFWAND_97_2'	754	1131	Kbyte
Index 'IDX10_AUFWAND_F' Part 'IDX_AUFWAND_96_2'	1912	2868	Kbyte
Index 'IDX10_AUFWAND_F' Part 'IDX_AUFWAND_97_2'	804	1206	Kbyte
Index 'IDX11_AUFWAND_F' Part 'IDX_AUFWAND_96_2'	1568	2352	Kbyte
Index 'IDX11_AUFWAND_F' Part 'IDX_AUFWAND_97_2'	660	990	Kbyte
Index 'IDX12_AUFWAND_F' Part 'IDX_AUFWAND_96_2'	1568	2352	Kbyte
Index 'IDX12_AUFWAND_F' Part 'IDX_AUFWAND_97_2'	660	990	Kbyte

Speicherplatzbedarf für TBS 'INDEX1'	37556	51665	Kbyte

TBS 'INDEX2'

	1. Extent	1.+ 2. Extent	
Index 'IDX1_MONAT_L'	2	3	Kbyte
Index 'IDX2_MONAT_L'	2	3	Kbyte
Index 'IDX3_MONAT_L'	2	3	Kbyte
Index 'IDX1_TAG_L'	22	33	Kbyte
Index 'IDX2_TAG_L'	14	21	Kbyte
Index 'IDX3_TAG_L'	26	39	Kbyte
Index 'IDX4_TAG_L'	18	27	Kbyte
Index 'IDX5_TAG_L'	24	36	Kbyte
Index 'IDX1_KOSTENSTELLEN_L'	0	0	Kbyte
Index 'IDX2_KOSTENSTELLEN_L'	8	12	Kbyte
Index 'IDX1_USER_ARTEN_L'	2	3	Kbyte
Index 'IDX2_USER_ARTEN_L'	2	3	Kbyte
Index 'IDX1_SOZIETAETEN_L'	2	3	Kbyte
Index 'IDX2_SOZIETAETEN_L'	2	3	Kbyte
Index 'IDX1_USERS_L'	10	15	Kbyte
Index 'IDX2_USERS_L'	26	39	Kbyte
Index 'IDX3_USERS_L'	8	12	Kbyte
Index 'IDX4_USERS_L'	38	57	Kbyte
Index 'IDX5_USERS_L'	22	33	Kbyte
Index 'IDX1_BEREICHE_L'	2	3	Kbyte
Index 'IDX2_BEREICHE_L'	2	3	Kbyte
Index 'IDX1_AKTENTYPEN_L'	2	3	Kbyte
Index 'IDX2_AKTENTYPEN_L'	2	3	Kbyte
Index 'IDX1_AKTEN_SACHGEBIETE_L'	2	3	Kbyte
Index 'IDX2_AKTEN_SACHGEBIETE_L'	6	9	Kbyte
Index 'IDX1_AKTEN_L'	352	528	Kbyte
Index 'IDX2_AKTEN_L'	326	489	Kbyte
Index 'IDX3_AKTEN_L'	1670	2505	Kbyte
Index 'IDX4_AKTEN_L'	326	489	Kbyte
Index 'IDX5_AKTEN_L'	1670	2505	Kbyte
Index 'IDX6_AKTEN_L'	326	489	Kbyte
Index 'IDX7_AKTEN_L'	1670	2505	Kbyte
Index 'IDX1_AUFWAND_F_AGR1'	400	600	Kbyte
Index 'IDX2_AUFWAND_F_AGR1'	428	642	Kbyte
Index 'IDX3_AUFWAND_F_AGR1'	582	873	Kbyte
Index 'IDX4_AUFWAND_F_AGR1'	428	642	Kbyte
Index 'IDX5_AUFWAND_F_AGR1'	428	642	Kbyte
Index 'IDX6_AUFWAND_F_AGR1'	276	414	Kbyte
Index 'IDX7_AUFWAND_F_AGR1'	400	600	Kbyte
Index 'IDX8_AUFWAND_F_AGR1'	488	732	Kbyte
Index 'IDX9_AUFWAND_F_AGR1'	488	732	Kbyte
Index 'IDX1_AUFWAND_F' Part 'IDX_AUFWAND_96_1'	1350	2025	Kbyte
Index 'IDX1_AUFWAND_F' Part 'IDX_AUFWAND_97_1'	1448	2172	Kbyte
Index 'IDX1_AUFWAND_F' Part 'IDX_AUFWAND_98'	388	582	Kbyte
Index 'IDX2_AUFWAND_F' Part 'IDX_AUFWAND_96_1'	1272	1908	Kbyte
Index 'IDX2_AUFWAND_F' Part 'IDX_AUFWAND_97_1'	1362	2043	Kbyte
Index 'IDX2_AUFWAND_F' Part 'IDX_AUFWAND_98'	366	549	Kbyte
Index 'IDX3_AUFWAND_F' Part 'IDX_AUFWAND_96_1'	1610	2415	Kbyte
Index 'IDX3_AUFWAND_F' Part 'IDX_AUFWAND_97_1'	1724	2586	Kbyte
Index 'IDX3_AUFWAND_F' Part 'IDX_AUFWAND_98'	442	663	Kbyte

Index 'IDX4_AUFWAND_F' Part 'IDX_AUFWAND_96_1'	1182	1773	Kbyte
Index 'IDX4_AUFWAND_F' Part 'IDX_AUFWAND_97_1'	1266	1899	Kbyte
Index 'IDX4_AUFWAND_F' Part 'IDX_AUFWAND_98'	340	510	Kbyte
Index 'IDX5_AUFWAND_F' Part 'IDX_AUFWAND_96_1'	760	1140	Kbyte
Index 'IDX5_AUFWAND_F' Part 'IDX_AUFWAND_97_1'	816	1224	Kbyte
Index 'IDX5_AUFWAND_F' Part 'IDX_AUFWAND_98'	220	330	Kbyte
Index 'IDX6_AUFWAND_F' Part 'IDX_AUFWAND_96_1'	1104	1656	Kbyte
Index 'IDX6_AUFWAND_F' Part 'IDX_AUFWAND_97_1'	1184	1776	Kbyte
Index 'IDX6_AUFWAND_F' Part 'IDX_AUFWAND_98'	318	477	Kbyte
Index 'IDX7_AUFWAND_F' Part 'IDX_AUFWAND_96_1'	1182	1773	Kbyte
Index 'IDX7_AUFWAND_F' Part 'IDX_AUFWAND_97_1'	1266	1899	Kbyte
Index 'IDX7_AUFWAND_F' Part 'IDX_AUFWAND_98'	340	510	Kbyte
Index 'IDX8_AUFWAND_F' Part 'IDX_AUFWAND_96_1'	1350	2025	Kbyte
Index 'IDX8_AUFWAND_F' Part 'IDX_AUFWAND_97_1'	1448	2172	Kbyte
Index 'IDX8_AUFWAND_F' Part 'IDX_AUFWAND_98'	388	582	Kbyte
Index 'IDX9_AUFWAND_F' Part 'IDX_AUFWAND_96_1'	1350	2025	Kbyte
Index 'IDX9_AUFWAND_F' Part 'IDX_AUFWAND_97_1'	1448	2172	Kbyte
Index 'IDX9_AUFWAND_F' Part 'IDX_AUFWAND_98'	388	582	Kbyte
Index 'IDX10_AUFWAND_F' Part 'IDX_AUFWAND_96_1'	1440	2160	Kbyte
Index 'IDX10_AUFWAND_F' Part 'IDX_AUFWAND_97_1'	1544	2316	Kbyte
Index 'IDX10_AUFWAND_F' Part 'IDX_AUFWAND_98'	414	621	Kbyte
Index 'IDX11_AUFWAND_F' Part 'IDX_AUFWAND_96_1'	1182	1773	Kbyte
Index 'IDX11_AUFWAND_F' Part 'IDX_AUFWAND_97_1'	1266	1899	Kbyte
Index 'IDX11_AUFWAND_F' Part 'IDX_AUFWAND_98'	340	510	Kbyte
Index 'IDX12_AUFWAND_F' Part 'IDX_AUFWAND_96_1'	1182	1773	Kbyte
Index 'IDX12_AUFWAND_F' Part 'IDX_AUFWAND_97_1'	1266	1899	Kbyte
Index 'IDX12_AUFWAND_F' Part 'IDX_AUFWAND_98'	340	510	Kbyte

Speicherplatzbedarf für TBS 'INDEX2'	45790	68685	Kbyte

TBS 'FIRST_LOAD'

	1. Extent	
Tabelle 'LOAD_KOSTENSTELLEN'	7	Kbyte
Tabelle 'LOAD_USER_ARTEN'	2	Kbyte
Tabelle 'LOAD_SOZIETAETEN'	1	Kbyte
Tabelle 'LOAD_USERS'	47	Kbyte
Tabelle 'LOAD_BEREICHE'	2	Kbyte
Tabelle 'LOAD_AKTENTYPEN'	2	Kbyte
Tabelle 'LOAD_SACHGEBIETE'	6	Kbyte
Tabelle 'LOAD_AKTEN'	685	Kbyte
Tabelle 'LOAD_AUFWAENDE_ZEITAUFWAND'	30552	Kbyte

Speicherplatzbedarf für TBS 'FIRST_LOAD'	31304	Kbyte

Die Einrichtung eines Tablespace wird mit folgendem SQL-Statement durchgeführt:

```
create tablespace WAREHOUSE1
datafile 'D:\orant\database\or\wh1_dwh1.ora' size 35M
default storage(pctincrease 0)
online;
```

7.8 Die Implementierung der notwendigen Datenextraktions- und –transformations Prozeduren

Bei dem Load Prozeß wird in einem zweistufigen Load vorgegangen (siehe Kapitel 7.7.4) bei dem die beiden Load Stufen unterschiedlich zu implementieren sind. Für die Implementierung wird die ORACLE proprietäre Programmiersprache PL/SQL verwendet, bei der es sich um eine prozedurale Spracherweiterung der Datenbanksprache SQL handelt. Der Load Vorgang soll einmal täglich in einem Nachtlauf durchgeführt werden.

7.8.1 Die Implementierung der Load Stufe 1

Die Load-Routinen für die Load Stufe 1 sind relativ einfach aufgebaut, da hier die notwendigen Daten des operativen Systems ohne Aggregations- bzw. Transformationsaktionen in die Load Tabellen des Data Warehouses übernommen werden müssen.

Dabei werden nur solche operative Daten berücksichtigt, die seit dem letzten Datenload geändert oder neu hinzugefügt wurden. Da die Datensätze des operativen Systems der Fa. Oppenhoff & Rädler mit einem **Timestamp** (siehe Kapitel 6.2) bei Neueintragungen bzw. bei Änderungen versehen werden, ist es relativ einfach die notwendigen Daten zu selektieren.

Zusätzlich werden nur Aufwandsdaten der Tabelle ‚Aufwaende_Zeitaufwand' berücksichtigt, die den Jahren 1996, 1997 oder 1998 zuzuordnen sind.

Damit überprüfbar ist, wann der letzte Datenload stattgefunden hat, wird eine zusätzliche Metadaten Tabelle benötigt, in der der Name der durchgeführten Load-Prozedur, das Startdatum, die Startuhrzeit, das Stopdatum, die Stopuhrzeit sowie die Anzahl der übernommenen Datensätze für jede erfolgreich durchgeführte Load-Prozedur gespeichert wird. Das Stopdatum ist der ausschlaggebende Parameter auf den, in den Load-Prozeduren der Stufe 1, die operativen Daten auf Notwendigkeit zur Übernahme überprüft werden:

```
create table META$LOAD_SUCCESS(
LOAD_NAME     VARCHAR2(30),
LOAD_STUFE    NUMBER(1),
START_DATE    DATE,
START_TIME    VARCHAR2(20),
STOP_DATE     DATE,
STOP_TIME     VARCHAR2(20),
LINES         NUMBER)
tablespace DWH_META;
```

Da es natürlich, aus welchen Gründen auch immer, zu Fehlern bei der Datenübernahme kommen kann, muß zusätzlich eine Error Metadatentabelle existieren, aus der zu entnehmen ist, in welcher Load-Prozedur der Fehler aufgetreten ist und welche Datenbank-Fehlermeldung ausgelöst wurde:

```
create table META$ERRORS(
ERR_DATUM     VARCHAR2(30),
PRG_TYP       VARCHAR2(20),
PRG_NAME      VARCHAR2(30),
ERR_CODE      INTEGER,
ERR_MSG       VARCHAR2(255))
tablespace DWH_META;
```

Zusätzlich zu der Registrierung von erfolgreich durchgeführten Datenloads, bzw. fehlerhaft durchgeführten Datenloads, in Metadaten-Tabellen, wird ein LOG-FILE für den Daten-Load gepflegt, indem die oben aufgeführten Load-Daten wie Fehler, bzw. Erfolg einer Loadoperation zusätzlich festgehalten werden.

Für jede zu überprüfendeTabelle des operativen Systems wird eine entsprechende PL/SQL Load Routine implementiert. Genauer gesagt zwei Routinen: eine, die sämtliche Datensätze der operativen Tabelle verarbeitet, die in das Data Warehouse neu eingetragen werden müssen, und eine Routine die nur solche Datensätze verarbeitet, die seit dem letzten Datenload geändert wurden.

Natürlich könnte man solche geänderten Datensätze mit einem Zeitstempel versehen und zusätzlich zu den Ursprungsdaten in dem Data Warehouse aufnehmen. So wäre es zum Beispiel möglich über das Data Warehouse zu ermitteln, wie sich die Wohnadresse eines Kunden im Laufe der Jahre verändert hat. Für diesen Prototypen war diese Analysierbarkeit allerdings nicht gefordert, so daß solche Änderungen von operativen Daten einfach im Data Warehouse nachgepflegt werden.

Generell sind die PL/SQL Prozeduren der Load Stufe 1 immer gleich aufgebaut:

- Aufbau eines PL/SQL Cursors, eine Art Container, in den sämtliche, zu übernehmende Datensätze einer Tabelle des operativen Systems, temporär übernommen werden.
- Auslesen des gefüllten PL/SQL Cursors, wobei Datensatz für Datensatz in einer Schleife verarbeitet wird. Bei der Verarbeitung wird jeder Datensatz mittels INSERT Befehl in die entsprechende Load-Tabelle des Data Warehouses eingetragen.

- Gleichzeitige Überprüfung, ob Fehler während der Ausführung auftreten (= **Exception Handling**). Bei Auftreten eines Fehlers wird eine Eintragung in die Tabelle **META$ERRORS** und in das Log-File durchgeführt.
- Sollte die Load-Prozedur fehlerfrei durchlaufen, dann wird der erfolgreiche Datenload in der Tabelle **META$LOAD_SUCCESS,** sowie in dem Log-File registriert.

Folgende PL/SQL Routinen sollen beispielhaft das Schema der Load-Stufe 1 Implementierung darstellen:

(1) Laden der Datensätze, die seit letzten Datenload neu eingetragen wurden

```
CREATE OR REPLACE PROCEDURE load_aufwaende_zeitaufwand_l IS

-- VARIABLEN DECLARIERUNG
v_errcode   integer;
v_errmsg    varchar2(255);
v_errdat    varchar2(30);
v_load_stufe    constant number(1) := 1;
v_start_date    date;
v_start_time    varchar2(20);
v_stop_date     date;
v_stop_time     varchar2(20);
v_lines         number := 0;
v_load_name     constant varchar2(30) := 'LOAD_AUFWAENDE_ZEITAUFWAND_l';
v_kost_code     number(6);

-- Variablen zur Erstellung eines LOG-Files
v_file_handle    utl_file.file_type;
v_file_location constant varchar2(20) := 'e:\database\or';
v_file_name     constant varchar2(15) := 'load_log.txt';
-- Alle V_COMMIT ein COMMIT WORK STATEMENT
v_commit    constant number(6) :=5000;


-- Erstellung eines PL/SQL Cursors, der nur Datensätze aufnimmt, die seit dem letzten LOAD
-- geINSERTed wurden
cursor c1 is      select
                  a.azei_id                     ID,
```

```
                a.azei_aufwand_vom          AUFWAND_VOM,
                b.akte_aktennr              AKTENNR,
                c.user_kennung              USERS,
                c.user_kost_id              KOST,
                d.sach_code                 SACH,
                e.sort_code                 SORT,
                a.azei_status_kz            STATUS_KZ,
                a.azei_dauer                DAUER,
                a.azei_stundensatz          STUNDENSATZ,
                a.azei_betrag               BETRAG,
                a.azei_betrag_bezahlt       BETRAG_BEZAHLT
        from aufwaende_zeitaufwand@oppe.nochen a,
             akten@oppe.nochen b,
             users@oppe.nochen c,
              sachgebiete@oppe.nochen d,
              sozietaeten@oppe.nochen e
              where a.azei_erfasst_am       > (select max(stop_date)
                                                from meta$load_success
                                                where load_name = 'LOAD_AUFWAENDE_ZEITAUFWAND_I'
                                                group by load_name)
              and a.azei_aufwand_vom >= to_date('01.01.1996', 'DD.MM.YYYY')
              and a.azei_akte_id = b.akte_id
              and a.azei_user_id = c.user_id
              and a.azei_sach_id = d.sach_id
              and c.user_sort_id = e.sort_id;

rec_c1 c1%rowtype;

BEGIN

-- INITIALISIERUNG V_START_DATE
select sysdate into v_start_date from dual;
select to_char(sysdate, 'HH24:MI:SS') into v_start_time from dual;

open c1;
v_file_handle := UTL_FILE.FOPEN(v_file_location, v_file_name, 'a');

        loop
        fetch c1 into rec_c1;
        exit when c1%notfound;

        -- Wenn in dem Tabellenattribut ‚AZEI_KOST_ID' kein Wert eingetragen ist, dann muß Default Wert
        -- ‚999' eingetragen werden
        if rec_c1.kost is null then
             insert into load_aufwaende_zeitaufwand     (azei_id,
                                                         tag_id,
                                                         akte_aktennr,
                                                         user_kennung,
```

```
                                                    sach_code,
                                                    sort_code,
                                                    kost_code,
                                                   azei_status_kz,
                                                   azei_dauer,
                                                   azei_stundensatz,
                                                   azei_betrag,
                                                   azei_betrag_bezahlt,
                                                   azei_dsatz_stat)

                values                             (rec_c1.id,
                                                    rec_c1.AUFWAND_VOM,
                                                    rec_c1.AKTENNR,
                                                    rec_c1.USERS,
                                                    rec_c1.SACH,
                                                    rec_c1.SORT,
                                                    999,
                                                    rec_c1.STATUS_KZ,
                                                    rec_c1.DAUER,
                                                    rec_c1.STUNDENSATZ,
                                                    rec_c1.BETRAG,
                                                    rec_c1.BETRAG_BEZAHLT,
                                                    'I');
else

    select kost.kost_code into v_kost_code
    from kostenstellen@oppe.nochen kost
    where rec_c1.kost = kost.kost_id;


        insert into load_aufwaende_zeitaufwand     (azei_id,
                                                    tag_id,
                                                    akte_aktennr,
                                                    user_kennung,
                                                    sach_code,
                                                    sort_code,
                                                    kost_code,
                                                    azei_status_kz,
                                                    azei_dauer,
                                                    azei_stundensatz,
                                                    azei_betrag,
                                                    azei_betrag_bezahlt,
                                                    azei_dsatz_stat)
        values                                     (rec_c1.id,
                                                    rec_c1.AUFWAND_VOM,
                                                    rec_c1.AKTENNR,
                                                    rec_c1.USERS,
                                                    rec_c1.SACH,
```

```
                                                    rec_c1.SORT,
                                                    v_kost_code,
                                                    rec_c1.STATUS_KZ,
                                                    rec_c1.DAUER,
                                                    rec_c1.STUNDENSATZ,
                                                    rec_c1.BETRAG,
                                                    rec_c1.BETRAG_BEZAHLT,
                                                    'I');
end if;

v_lines := v_lines + 1;


-- Alle V_COMMIT ein COMMIT_WORK;
if MOD(v_lines, v_commit) = 0 then

     commit work;

end if;

end loop;

select sysdate into v_stop_date from dual;
select to_char(sysdate, 'HH24:MI:SS') into v_stop_time from dual;

-- Registrierung des erfolgreichen Daten Loads in Metadaten Tabelle
insert into meta$load_success values ( v_load_name,
                                       v_load_stufe,
                                        v_start_date,
                                       v_start_time,
                                        v_stop_date,
                                       v_stop_time,
                                        v_lines);

commit work;


-- Registrierung des Load Erfolgs in LOG-FILE
UTL_FILE.PUT_LINE(v_file_handle, v_stop_date||' '||v_load_name||' fehlerfrei abgeschlossen.');
UTL_FILE.PUT_LINE(v_file_handle, v_lines||' Datensätze geINSERTed');
UTL_FILE.NEW_LINE(v_file_handle);
UTL_FILE.FCLOSE(v_file_handle);

-- Ausgabe des Load Erfolges am Bildschirm
dbms_output.put_line('**** LOAD '||v_load_name||' FEHLERFREI DURCHGEFÜHRT ****');

close c1;
```

```
-- Exception Handling (= Fehlerhandling)
EXCEPTION

WHEN others THEN

        rollback work;

        close c1;

        select to_char(sysdate, 'DD.MM.YYYY HH24:MI:SS') into v_errdat from dual;
        v_errcode := SQLCODE;
        v_errmsg  := SQLERRM(v_errcode);

        -- Registrierung ein Datenübernahme Fehlers in Metadaten Tabelle
        insert into meta$errors values(v_errdat,
                                       'PROCEDURE',
                                       v_load_name,
                                       v_errcode,
                                       v_errmsg);
        commit work;

        -- Registrierung eines Fehlers in dem LOG-FILE
        UTL_FILE.PUT_LINE(v_file_handle, v_errdat||' '||v_load_name||' durch Fehler beendet. Bitte überprüfen
        Sie die Tabelle META$ERRORS !!!!!');
        UTL_FILE.NEW_LINE(v_file_handle);
        UTL_FILE.FCLOSE(v_file_handle);
        raise_application_error(-20000,'*** Fehler beim Laden der Daten in PROCEDURE '||v_load_name||'
        ***',false);

END;
/
show errors
```

Bemerkung: *Eingeladene Datensätze werden mit dem Kennzeichen ‚I' (für INSERT) versehen, damit in der zweiten Load-Stufe erkennbar ist, ob ein Datensatz in dem Data Warehouse geinserted oder upgedated werden muß.*

(2) Laden der Datensätze, die seit dem letzten Datenload geändert wurden

```
CREATE OR REPLACE PROCEDURE load_aufwaende_zeitaufwand_U IS

-- VARIABLEN DECLARIERUNG
v_errcode   integer;
v_errmsg    varchar2(255);
v_errdat    varchar2(30);
```

```
v_load_stufe    constant number(1) := 1;
v_start_date    date;
v_start_time    varchar2(20);
v_stop_date     date;
v_stop_time     varchar2(20);
v_lines         number := 0;
v_load_name     constant varchar2(30) := 'LOAD_AUFWAENDE_ZEITAUFWAND_U';
v_kost_code     number(6);

-- Variablen zur Erstellung eines LOG-Files
v_file_handle   utl_file.file_type;
v_file_location constant varchar2(20) := 'e:\database\or';
v_file_name     constant varchar2(15) := 'load_log.txt';


-- Alle V_COMMIT ein COMMIT WORK STATEMENT
v_commit    constant number(6) := 5000;


-- Erstellung eines Cursors, der nur Datensätze aufnimmt, die seit dem letzten LOAD
-- geINSERTed wurden
cursor c1 is select
                a.azei_id                   ID,
                a.azei_aufwand_vom          AUFWAND_VOM,
                b.akte_aktennr              AKTENNR,
                c.user_kennung              USERS,
                c.user_kost_id              KOST,
                d.sach_code                 SACH,
                e.sort_code                 SORT,
                a.azei_status_kz            STATUS_KZ,
                a.azei_dauer                DAUER,
                a.azei_stundensatz          STUNDENSATZ,
                a.azei_betrag               BETRAG,
                a.azei_betrag_bezahlt       BETRAG_BEZAHLT
           from aufwaende_zeitaufwand@oppe.nochen a,
                akten@oppe.nochen b,
                users@oppe.nochen c,
                sachgebiete@oppe.nochen d,
                sozietaeten@oppe.nochen e
           where a.azei_geaendert_am is not null
           and a.azei_erfasst_am       < (select max(stop_date)
                                          from meta$load_success
                                          where load_name = 'LOAD_AUFWAENDE_ZEITAUFWAND_U'
                                          group by load_name)
           and a.azei_geaendert_am > (select max(stop_date)
                                          from meta$load_success
                                          where load_name = 'LOAD_AUFWAENDE_ZEITAUFWAND_U'
                                          group by load_name)
```

```
                and a.azei_aufwand_vom >= to_date('01.01.1996', 'DD.MM.YYYY')
                and a.azei_akte_id = b.akte_id
                and a.azei_user_id = c.user_id
                and a.azei_sach_id = d.sach_id
                and c.user_sort_id = e.sort_id;

rec_c1 c1%rowtype;

BEGIN

-- INITIALISIERUNG V_START_DATE
select sysdate into v_start_date from dual;
select to_char(sysdate, 'HH24:MI:SS') into v_start_time from dual;


open c1;
v_file_handle := UTL_FILE.FOPEN(v_file_location, v_file_name, 'a');

            loop

            fetch c1 into rec_c1;
            exit when c1%notfound;

            -- Wenn das Tabellenattribut ‚AZEI_KOST_ID' null ist → Eintrag des Defaultwertes ‚999'
            if rec_c1.kost is null then

                insert into load_aufwaende_zeitaufwand      (azei_id,
                                                             tag_id,
                                                             akte_aktennr,
                                                             user_kennung,
                                                             sach_code,
                                                             sort_code,
                                                             kost_code,
                                                             azei_status_kz,
                                                             azei_dauer,
                                                             azei_stundensatz,
                                                             azei_betrag,
                                                             azei_betrag_bezahlt,
                                                             azei_dsatz_stat)
            values
                                                             rec_c1.AUFWAND_VOM,
                                                             rec_c1.AKTENNR,
                                                             rec_c1.USERS,
                                                             rec_c1.SACH,
                                                             rec_c1.SORT,
                                                             999,
                                                             rec_c1.STATUS_KZ,
                                                             rec_c1.DAUER,
```

```
                                                    rec_c1.STUNDENSATZ,
                                                    rec_c1.BETRAG,
                                                    rec_c1.BETRAG_BEZAHLT,
                                                    'U');

else

    select kost.kost_code into v_kost_code
    from kostenstellen@oppe.nochen kost
    where rec_c1.kost = kost.kost_id;


        insert into load_aufwaende_zeitaufwand      (azei_id,
                                                    tag_id,
                                                    akte_aktennr,
                                                    user_kennung,
                                                    sach_code,
                                                    sort_code,
                                                    kost_code,
                                                    azei_status_kz,
                                                    azei_dauer,
                                                    azei_stundensatz,
                                                    azei_betrag,
                                                    azei_betrag_bezahlt,
                                                    azei_dsatz_stat)

        values                                      (rec_c1.id,
                                                    rec_c1.AUFWAND_VOM,
                                                    rec_c1.AKTENNR,
                                                    rec_c1.USERS,
                                                    rec_c1.SACH,
                                                    rec_c1.SORT,
                                                    v_kost_code,
                                                    rec_c1.STATUS_KZ,
                                                    rec_c1.DAUER,
                                                    rec_c1.STUNDENSATZ,
                                                    rec_c1.BETRAG,
                                                    rec_c1.BETRAG_BEZAHLT,
                                                    'U');
        end if;

        v_lines := v_lines + 1;

        -- Alle V_COMMIT ein COMMIT_WORK;
        if MOD(v_lines, v_commit) = 0 then

            commit work;
```

```
            end if;

            end loop;

select sysdate into v_stop_date from dual;
select to_char(sysdate, 'HH24:MI:SS') into v_stop_time from dual;

-- Registrierung des Load Erfolgs in Metadatentabelle
insert into meta$load_success values( v_load_name,
                                      v_load_stufe,
                                      v_start_date,
                                      v_start_time,
                                      v_stop_date,
                                      v_stop_time,
                                      v_lines);

commit work;

-- Registrierung des Load Erfolgs in LOG-FILE
UTL_FILE.PUT_LINE(v_file_handle, v_stop_date||' '||v_load_name||' fehlerfrei abgeschlossen.');
UTL_FILE.PUT_LINE(v_file_handle, v_lines||' Datensätze geINSERTed');
UTL_FILE.NEW_LINE(v_file_handle);
UTL_FILE.FCLOSE(v_file_handle);

-- Ausgabe des Load Erfolges am Bildschirm
dbms_output.put_line('**** LOAD '||v_load_name||' FEHLERFREI DURCHGEFÜHRT ****');

close c1;



-- Excption Handling (= Fehlerhandling)
EXCEPTION

WHEN others THEN

            rollback work;

            close c1;

            select to_char(sysdate, 'DD.MM.YYYY HH24:MI:SS') into v_errdat from dual;
            v_errcode := SQLCODE;
            v_errmsg  := SQLERRM(v_errcode);

            -- Registrierung des Fehlers in Metadaten Tabelle
            insert into meta$errors values(v_errdat,
                                           'PROCEDURE',
                                           v_load_name,
```

```
                                        v_errcode,
                                        v_errmsg);
        commit work;

        -- Registrierung des Fehlers in LOG-FILE
        UTL_FILE.PUT_LINE(v_file_handle, v_errdat||' '||v_load_name||' durch Fehler beendet. Bitte überprüfen
        Sie die Tabelle META$ERRORS !!!!!');
        UTL_FILE.NEW_LINE(v_file_handle);
        UTL_FILE.FCLOSE(v_file_handle);
        raise_application_error(-20000,'*** Fehler beim Laden der Daten in PROCEDURE '||v_load_name||'
        ***',false);

END;
/
show errors
```

Bemerkung: *Eingeladene Datensätze werden mit dem Kennzeichen ‚U' (für UPDATE) versehen, damit in der zweiten Load-Stufe erkennbar ist, ob ein Datensatz in dem Data Warehouse geinserted oder upgedated werden muß.*

7.8.2 Die Implementierung der Load Stufe 2

Nachdem die Load Stufe 1 erfolgreich abgeschlossen wurde, kann das operative System direkt wieder für das operative Tagesgeschäft genutzt werden. Sämtliche nun folgenden Arbeitsschritte der Load Stufe 2 finden auf dem Data Warehouse Server statt.

Die **erste Phase** der Load Stufe 2 beginnt mit dem Einladen der Datensätze aus den Load-Tabellen in die eigentlichen Data Warehouse Tabellen. Nachdem ein Datensatz übertragen wurde, wird er aus der Load-Tabelle gelöscht und gilt somit als erfolgreich verarbeitet. Jeweils eine PL/SQL Routine der Load Stufe 2 bearbeitet eine Load-Tabelle.

Beispiel-Routine:

```
CREATE OR REPLACE PROCEDURE dwh_load_aufwand_f IS
```

```
-- VARIABLEN DECLARIERUNG
v_errcode        integer;
v_errmsg         varchar2(255);
v_errdat         varchar2(30);
v_load_stufe     constant number(1) := 2;
v_start_date     date;
v_start_time     varchar2(20);
v_stop_date      date;
v_stop_time      varchar2(20);
v_lines          number := 0;
v_load_name      constant varchar2(30) := 'DWH_LOAD_AUFWAND_F';
wrong_state      exception;

-- Variablen zur Erstellung eines LOG-Files
v_file_handle    utl_file.file_type;
v_file_location constant varchar2(20) := 'e:\database\or';
v_file_name      constant varchar2(15) := 'load_log.txt';


-- Alle V_COMMIT ein COMMIT WORK STATEMENT
v_commit    constant number(6) := 100;
-- Erstellung eines Cursors mit allen noch zu übernehmenden Datensätzen
cursor c1 is select
                    azei_id,
                    tag_id,
                    akte_aktennr,
                    user_kennung,
                    sach_code,
                    sort_code,
                    KOST_CODE,
                    azei_status_kz,
                    azei_dauer,
                    azei_stundensatz,
                    azei_betrag,
                    azei_betrag_bezahlt,
                    azei_dsatz_stat                    STATUS
            from load_aufwaende_zeitaufwand;

rec_c1 c1%rowtype;

BEGIN

-- INITIALISIERUNG V_START_DATE
select sysdate into v_start_date from dual;
select to_char(sysdate, 'HH24:MI:SS') into v_start_time from dual;

open c1;
v_file_handle := UTL_FILE.FOPEN(v_file_location, v_file_name, 'a');
```

```
        loop

        fetch c1 into rec_c1;
exit when c1%notfound;


        -- Wenn Datensatzstatus = 'I', dann INSERT
        if rec_c1.status = 'I' then

            insert into aufwand_f                    (azei_id,
                                                      tag_id,
                                                      akte_aktennr_id,
                                                      user_kennung_id,
                                                      sach_code_id_v,
                                                      sort_code_id,
                                                      kost_code_id,
                                                      azei_status_kz_q,
                                                      azei_dauer_f,
                                                      azei_stundensatz_f,
                                                      azei_betrag_f,
                                                      azei_betrag_bezahlt_f)

            values                                   (rec_c1.azei_id,
                                                      rec_c1.tag_id,
                                                      rec_c1.akte_aktennr,
                                                      rec_c1.user_kennung,
                                                      rec_c1.sach_code,
                                                      rec_c1.sort_code,
                                                      rec_c1.kost_code,
                                                      rec_c1.azei_status_kz,
                                                      rec_c1.azei_dauer,
                                                      rec_c1.azei_stundensatz,
                                                      rec_c1.azei_betrag,
                                                      rec_c1.azei_betrag_bezahlt);

        -- Wenn Datensatzstatus = 'U', dann UPDATE
        elsif rec_c1.status = 'U' then

            update aufwand_f set

                tag_id              = rec_c1.tag_id,
                akte_aktennr_id     = rec_c1.akte_aktennr,
                user_kennung_id     = rec_c1.user_kennung,
                sach_code_id_v      = rec_c1.sach_code,
                sort_code_id        = rec_c1.sort_code,
                kost_code_id        = rec_c1.kost_code,
                azei_status_kz_q    = rec_c1.azei_status_kz,
```

```
                azei_dauer_f            = rec_c1.azei_dauer,
                azei_stundensatz_f      = rec_c1.azei_stundensatz,
                azei_betrag_f           = rec_c1.azei_betrag,
                azei_betrag_bezahlt_f   = rec_c1.azei_betrag_bezahlt

            where azei_id = rec_c1.azei_id;

        end if;

        v_lines := v_lines + 1;

        -- Löschen des geladenen Datensatzes aus der LOAD-Tabelle
        delete from load_aufwaende_zeitaufwand where azei_id = rec_c1.azei_id;

        -- Alle V_COMMIT ein COMMIT_WORK;
        if MOD(v_lines, v_commit) = 0 then

            commit work;

        end if;

        end loop;

select sysdate into v_stop_date from dual;
select to_char(sysdate, 'HH24:MI:SS') into v_stop_time from dual;

-- Registrierung des Load Erfolges in der Metadaten Tabelle
insert into meta$load_success values( v_load_name,
                                      v_load_stufe,
                                      v_start_date,
                                      v_start_time,
                                      v_stop_date,
                                      v_stop_time,
                                      v_lines);

commit work;

-- Registrierung des Load Erfolges in dem LOG-FILE
UTL_FILE.PUT_LINE(v_file_handle, v_stop_date||' '||v_load_name||' fehlerfrei abgeschlossen.');
UTL_FILE.PUT_LINE(v_file_handle, v_lines||' Datensätze geINSERTed');
UTL_FILE.NEW_LINE(v_file_handle);
UTL_FILE.FCLOSE(v_file_handle);

-- Ausgabe des Load Erfolges am Bildschirm
dbms_output.put_line('**** LOAD '||v_load_name||' FEHLERFREI DURCHGEFÜHRT ****');

close c1;
```

```
-- Exception Handling (= Fehlerhandling)
EXCEPTION

WHEN others THEN

        rollback work;

        close c1;

        select to_char(sysdate, 'DD.MM.YYYY HH24:MI:SS') into v_errdat from dual;
        v_errcode := SQLCODE;
        v_errmsg  := SQLERRM(v_errcode);

        -- Registrierung des Load Fehlers in der Metadaten Tabelle
        insert into meta$errors values(v_errdat,  'PROCEDURE', v_load_name, v_errcode, v_errmsg);

        commit work;

        -- Registrierung des Load Fehlers in dem LOG-FILE
        UTL_FILE.PUT_LINE(v_file_handle, v_errdat||' '||v_load_name||' durch Fehler beendet. Bitte überprüfen
        Sie die Tabelle META$ERRORS !!!!!');
        UTL_FILE.NEW_LINE(v_file_handle);
        UTL_FILE.FCLOSE(v_file_handle);
        raise_application_error(-20000,'*** Fehler beim Laden der Daten in PROCEDURE '||v_load_name||'
        ***',false);

END;
/
show errors
```

Gleichzeitig wird bei einem Insert oder Update eines Datensatzes der Tabelle ‚Aufwand_F' ein Trigger ausgelöst, der das Aufwandsdatum des Datensatzes überprüft und daraus Einträge in die Tabellen Jahr_L, Monat_L, Tag_L generiert, so daß diese Lookup-Tabellen der Dimension ‚Zeit', gemäß den Einträgen in der Tabelle Aufwand_F, mit Analysedaten gefüllt sind. Die sieben Einträge der Tabelle Wochentag_L (1 Montag, ..., 7 Sonntag) werden durch ein Initialisierungsscript vorgenommen.

Transformationen von Daten sind für diesen Protoypen nicht notwendig, würden aber in dieser Phase durchgeführt werden (siehe auch Kapitel 6.3).

In der **zweiten Phase** der Load Stufe 2 wird aus allen Aufwandsdaten der gerade aktualisierten Fakt-Tabelle ‚Aufwand_F', mittels entsprechender Summierung der Kennzahlen, die Aggregationstabelle ‚Aufwand_F_AGR1' gefüllt. Dabei wird der alte Inhalt dieser Aggregationstabelle im Vorfeld gelöscht.

Folgende PL/SQL Routine wird ausgeführt:

```
CREATE OR REPLACE PROCEDURE agr1_aufwand_f IS

-- VARIABLEN DECLARIERUNG
v_errcode   integer;
v_errmsg    varchar2(255);
v_errdat    varchar2(30);
v_load_stufe    constant number(1) := 2;
v_start_date    date;
v_start_time    varchar2(20);
v_stop_date     date;
v_stop_time     varchar2(20);
v_lines         number := 0;
v_load_name     constant varchar2(30) := 'AGR1_AUFWAND_F';
wrong_state     exception;

-- Variablen zur Erstellung eines LOG-Files
v_file_handle   utl_file.file_type;
v_file_location constant varchar2(20) := 'e:\database\or';
v_file_name     constant varchar2(15) := 'load_log.txt';


-- Alle V_COMMIT ein COMMIT WORK STATEMENT
v_commit    constant number(6) := 5000;


-- Erstellung eines Cursors mit allen aggregierten Datensätzen aus AUFWAND_F
-- Aggregiert wird von TAG zu MONAT und von AKTENNR nach AKTEN_SACHGEBIET
cursor c1 is select
                to_char(a.tag_id, 'MMYY')       MONA,
                b.sach_code_id                  SACH_AKTE,
                a.user_kennung_id               USERS,
                a.sach_code_id_v                SACH_AUFWAND,
                a.sort_code_id                  SORT,
                a.kost_code_id                  KOST,
                a.azei_status_kz_q              STATUS,
```

```
                sum(a.azei_dauer_f)                 AZEI_DAUER_F,
                sum(a.azei_betrag_f)                AZEI_BETRAG_F,
                sum(a.azei_betrag_bezahlt_f)        AZEI_BETRAG_BEZAHLT_F
                from aufwand_f a,
                     akten_l b
                where a.akte_aktennr_id = b.akte_aktennr_id
                group by
                to_char(a.tag_id, 'MMYY'),
                b.sach_code_id,
                a.user_kennung_id,
                a.sach_code_id_v,
                a.sort_code_id,
                a.kost_code_id,
                a.azei_status_kz_q;

rec_c1 c1%rowtype;

BEGIN

-- INITIALISIERUNG V_START_DATE
select sysdate into v_start_date from dual;
select to_char(sysdate, 'HH24:MI:SS') into v_start_time from dual;

-- Löschen der alten Tabelleneinträge in der Aggregationstabelle 'Aufwand_F_AGR1'
delete from aufwand_f_agr1;
commit work;

open c1;
v_file_handle := UTL_FILE.FOPEN(v_file_location, v_file_name, 'a');

            loop

            fetch c1 into rec_c1;
            exit when c1%notfound;

                insert into aufwand_f_agr1          (mona_id,
                                                     sach_code_id,
                                                     user_kennung_id,
                                                     sach_code_id_v,
                                                     sort_code_id,
                                                     kost_code_id,
                                                     agr1_azei_status_kz_q,
                                                     agr1_azei_dauer_f,
                                                     agr1_azei_betrag_f,
                                                     agr1_azei_betrag_bezahlt_f)

                values                              (rec_c1.mona,
                                                     rec_c1.sach_akte,
```

```
                                                    rec_c1.users,
                                                    rec_c1.sach_aufwand,
                                                    rec_c1.sort,
                                                    rec_c1.kost,
                                                    rec_c1.status,
                                                    rec_c1.azei_dauer_f,
                                                    rec_c1.azei_betrag_f,
                                                    rec_c1.azei_betrag_bezahlt_f);

            v_lines := v_lines + 1;

            -- Alle V_COMMIT ein COMMIT_WORK;
            if MOD(v_lines, v_commit) = 0 then

                commit work;

            end if;

            end loop;

select sysdate into v_stop_date from dual;
select to_char(sysdate, 'HH24:MI:SS') into v_stop_time from dual;

-- Registrierung des Load Erfolges in der Metadaten Tabelle
insert into meta$load_success values( v_load_name,
                                      v_load_stufe,
                                      v_start_date,
                                      v_start_time,
                                      v_stop_date,
                                      v_stop_time,
                                      v_lines);

commit work;

-- Registrierung des Load Erfolgs in dem LOG-FILE
UTL_FILE.PUT_LINE(v_file_handle, v_stop_date||' '||v_load_name||' fehlerfrei abgeschlossen.');
UTL_FILE.PUT_LINE(v_file_handle, v_lines||' Datensätze geINSERTed');
UTL_FILE.NEW_LINE(v_file_handle);
UTL_FILE.FCLOSE(v_file_handle);

-- Ausgabe des Load Erfolges am Bildschirm
dbms_output.put_line('**** AGGREGATION '||v_load_name||' FEHLERFREI DURCHGEFÜHRT ****');

close c1;
```

```
-- Exception Handling (= Fehlerhandling)
EXCEPTION

WHEN others THEN

        rollback work;

        close c1;

        select to_char(sysdate, 'DD.MM.YYYY HH24:MI:SS') into v_errdat from dual;
        v_errcode := SQLCODE;
        v_errmsg  := SQLERRM(v_errcode);

        -- Registrierung des Load Fehlers in der Metadaten Tabelle
        insert into meta$errors values(v_errdat,  'PROCEDURE',  v_load_name, v_errcode,  v_errmsg);

        commit work;

        -- Registrierung des Load Fehlers in dem LOG-FILE
        UTL_FILE.PUT_LINE(v_file_handle, v_errdat||' '||v_load_name||' durch Fehler beendet. Bitte überprüfen
        Sie die Tabelle META$ERRORS !!!!!');
        UTL_FILE.NEW_LINE(v_file_handle);
        UTL_FILE.FCLOSE(v_file_handle);
        raise_application_error(-20000,'*** Fehler beim Aggregieren der Daten in PROCEDURE
        '||v_load_name||' ***',false);



END;

/
show errors
```

Mit dem erfolgreichen Abschluß der Phase 2 wird die Datenübernahme beendet, das Data Warehouse ist **produktionsbereit**.

7.9 Die Umsetzung des multidimensionalen Modells mittels DSS Architect

Nachdem die Datenbank des Data Warehouses, und somit die technischen Metadaten (siehe Kapitel 4.2.1), erstellt wurde, kann damit begonnen werden das multidimensionale Modell (siehe Kapitel 7.5), mittels DSS Architect der Firma MicroStrategy, umzusetzen.

7.9.1 Aufgabe des DSS Architect

Mit Hilfe des DSS Architects werden die gewünschten semantischen Metadaten erstellt (siehe Kapitel 4.2.2). D. h., OLAP Objekte wie z. B. Dimensionen, Dimensionshierarchien werden hier aus den relationalen Objekten wie Tabellen und Views erstellt. Die SQL-Engine der Firma MicroStrategy ist damit in der Lage bei der Ausführung einer OLAP-Analyse, welche mit DSS Agent erstellt wurde, eine Zuordnung (= **Mapping**) von den „virtuellen“ OLAP Objekten zu relationalen Datenbank Objekten herzustellen und das notwendige SQL Statement zu erstellen (siehe auch Kapitel 4.1).

Bevor der DSS Architect benutzt werden kann, müssen Metadatentabellen, in denen die semantischen Metadaten gespeichert werden, in der Data Warehouse Datenbank erstellt werden. Dazu ist ein Metadaten-User einzurichten, der ein von MicroStrategy mitgeliefertes SQL Script ausführt, durch welches die einzelnen Metadaten Tabellen in seinem Datenbank Schema erstellt werden. Es ist auch möglich diese Metadatentabellen in einer speziellen Metadatenbank zu speichern. Dadurch ist es möglich, den eigentlichen

Data Warehouse Server von Transaktionen, die die Metadaten betreffen, zu entlasten. Durch diese Trennung der Metadaten von den Warehousedaten kann ein zusätzlicher Performancegewinn erzielt werden.

7.9.2 Der Aufbau des DSS Architects

Das Tool DSS Architect ist in fünf Mappen (= **Folder**) aufgeteilt, die bei der Neuerstellung eines Projektes nacheinander abgearbeitet werden müssen.

Folder 1: Connections

In dem ersten Folder werden grundsätzliche Informationen zum Projekt erstellt:

- Der Projektname
- Eine Projektbeschreibung
- Der Projekt-System-User (dieser User hat eine Art Administrator Funktion für das DSS Architekt Projekt, und muß natürlich auf der Data Warehouse Datenbank als User eingerichtet worden sein).
- Das Metadaten Prefix des User Datenbank Schemas, unter dem die Metadaten Tabellen für den DSS Architect angelegt wurden.
- Die Metadaten Datenbank. Unterstützt werden hier sämtliche bekannte Datenbanksysteme wie z. B. ORACLE, Informix, SQL Server, usw., auf welche per ODBC Treiber zugegriffen werden kann. In diesem Fall kommt der 16-Bit ODBC Treiber der Firma ORACLE Version 2.5.3.1.0 zum Einsatz. Die Anmeldung auf der Warehouse Datenbank erfolgt über den User, über den auch alle Warehouseobjekte (Tabellen, Views, Indizes) angelegt wurden.
- Die Data Warehouse Datenbank. Hier gilt das gleiche wie für die Metadaten Datenbank. Die Anmeldung erfolgt über den User, über den auch alle Metadaten Tabellen des DSS Architects angelegt wurden.

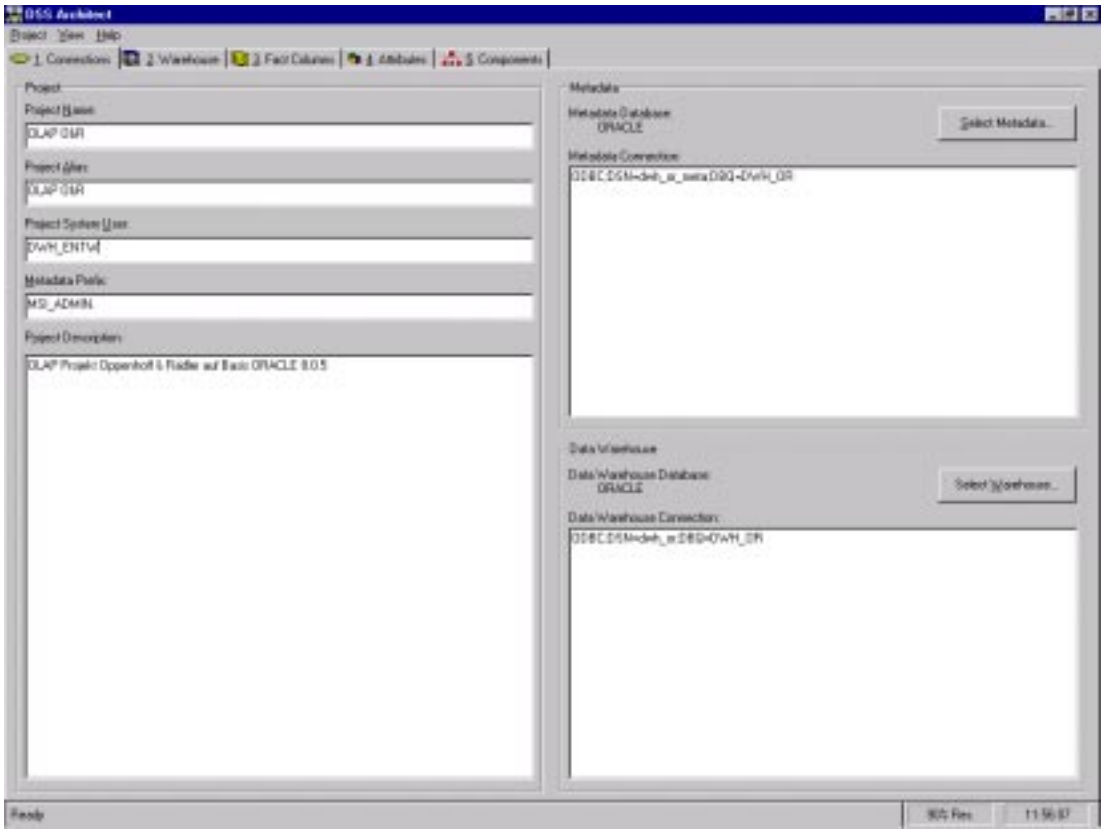


Abbildung 49: ‚Connections' Folder (DSS Architect)

Folder 2: Warehouse

In diesem Folder werden in dem linken Bereich alle in der Data Warehouse Datenbank verfügbaren Tabellen aufgeführt. Natürlich werden nur die Tabellen aufgeführt die Eigentum des Datenbank Users sind, über den eine Verbindung zu der Data Warehouse Datenbank mittels ODBC Treiber hergestellt wurde (Folder 1).

Die für die OLAP Analysen notwendigen Warehousetabellen sind jetzt nur noch auszuwählen und erscheinen dann in dem rechten Bereich dieses Folders.

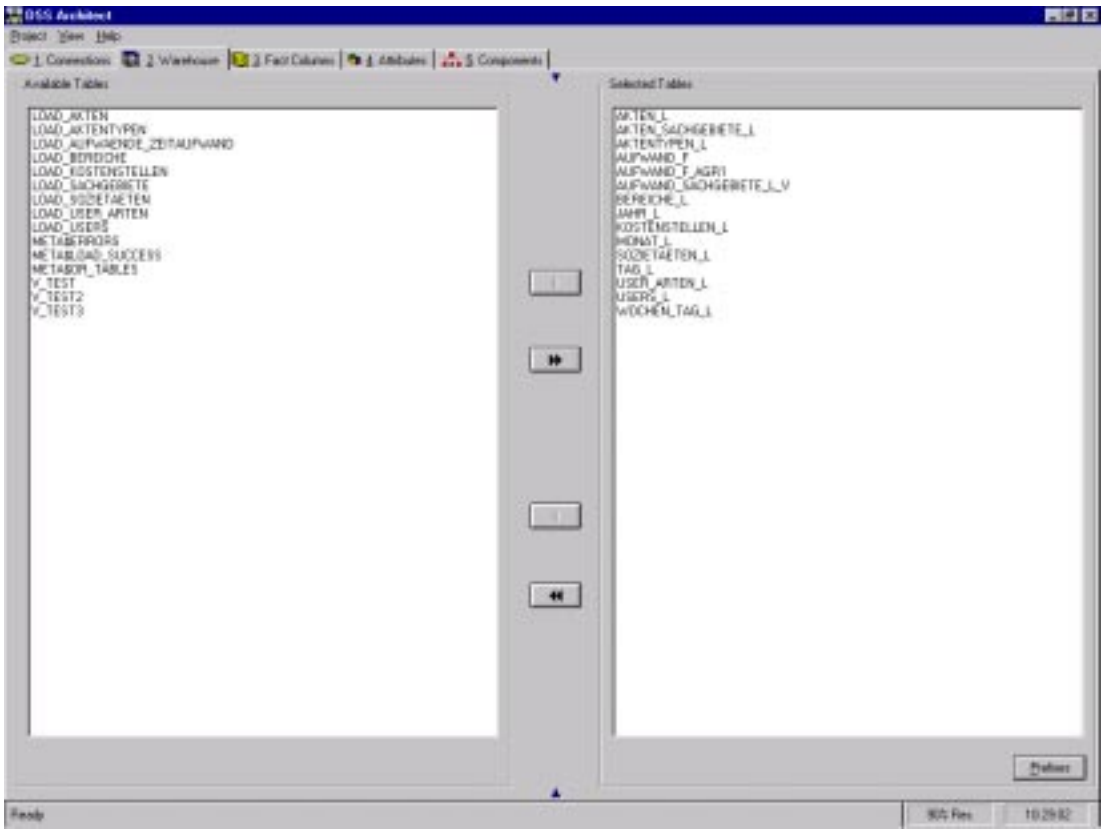


Abbildung 50: ‚Warehouse' Folder (DSS Architect)

Folder 3: Fact Columns

Dieser Folder ist genauso aufgebaut wie der Folder 2, nur daß hier anstatt der verfügbaren Tabellen alle verfügbaren Tabellenattribute der in Folder 2 ausgewählten Tabellen erscheinen. Aus diesen Tabellenattributen sind nun die auszuwählen, die die zu analysierenden Kennzahlen wie z. B. Aufwand in Stunden, Aufwand in DM, beinhalten. Diese Kennzahlen befinden sich selbstvertändlich in den Fakttabellen der Warehouse Datenbank.

Spätestens an diesem Punkt wird ersichtlich wie wichtig es ist, eine bestimmte Namenskonvention bei der Erstellung der Warehouse Tabellen einzuhalten, damit schnell erkannt werden kann, bei welchen Attributen es sich

um ID-, Bezeichnungs-, Lookup- oder Fakt-Attribute handelt. Empfehlenswert ist folgende Richtlinie:

- ID-Attributnamen werden um das Kennzeichen ‚_**ID**' erweitert. (z. B. Tag_ID).
- Bezeichnungs-Attributnamen werden um das Kennzeichen ‚_**BEZ**' erweitert (z. B. Wochentag_Bez).
- Lookup-Attributnamen werden um das Kennzeichen ‚_**L**' erweitert. (z. B. User_Vorname_L)
- Fakt-Attributnamen werden um das Kennzeichen ‚_**F**' erweitert (z. B. Azei_Dauer_F).

Der Unterschied zwischen Bezeichnungs-Attributnamen und Lookup-Attributnamen besteht darin, daß die Bezeichnungs-Attribute eine Text-Darstellung der ID-Attribute darstellen (Beispiel Wochentag: ID = 1 → Bezeichung = Montag). Lookup-Attribute dagegen stellen Zusatzinformationen über das Bezeichnungs-Attribut hinaus dar (z. B. User_Vorname).

Allerdings ist es nicht immer sinnvoll zusätzlich zu dem ID-Attribut auch ein Bezeichnungs-Attribut zu verwenden. Beispiel: JAHR_ID = 1998 → Bezeichnung ist gleich der ID, somit ist ein extra Bezeichnungs-Attribut unnötig.

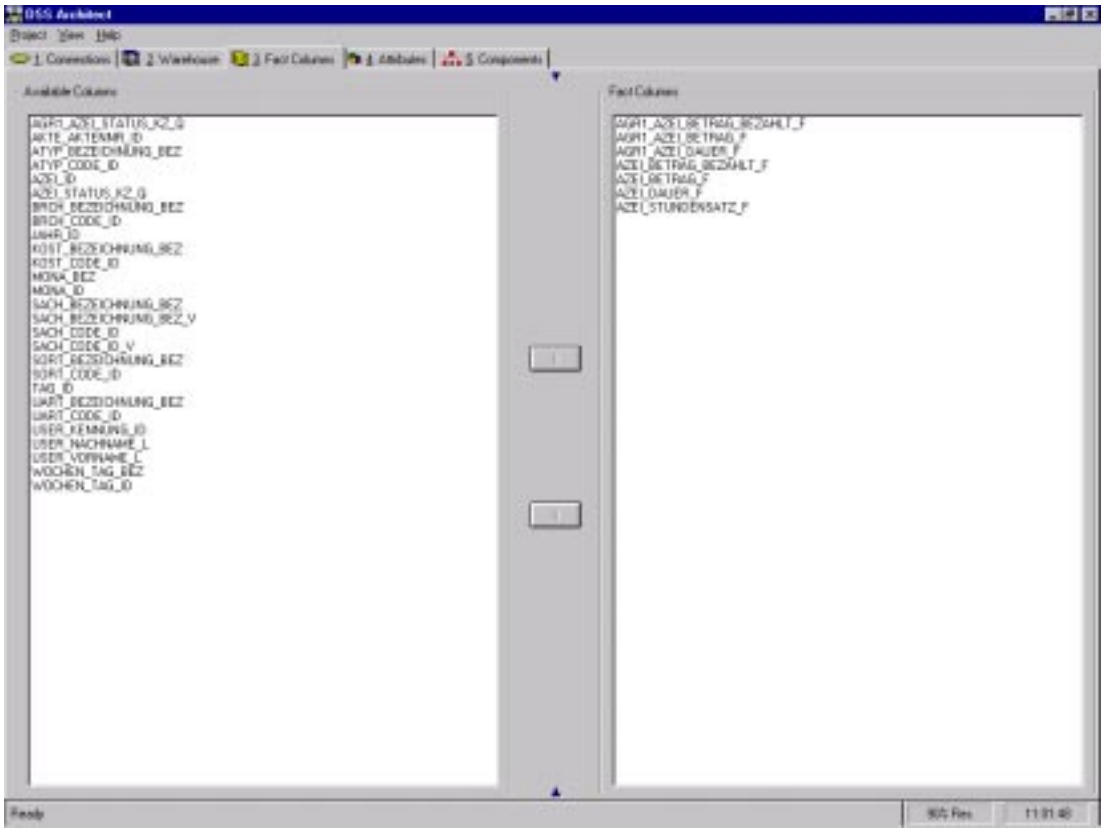


Abbildung 51: ‚Fact Columns' Folder (DSS Architect)

Folder 4: Attributes

In diesem Folder werden virtuelle OLAP Objekte angelegt und zu den relationalen Data Warehouse Tabellen gemapped.

Als erstes wird eine neue Dimension angelegt. Zu dieser Dimension werden die einzelnen Dimensionsattribute und ihre Beziehung zueinander (1:n; n:m; 1:1) festgelegt. Dabei ist zu beachten, daß sich die Attributnamen von den Dimensionsnamen unterscheiden müssen. Es ist also nicht möglich eine Dimension als ‚Kostenstelle' zu bezeichnen, und das einzige Attribut dieser Dimension ebenfalls als ‚Kostenstelle' zu bezeichnen, da dieses Objekt schon existiert.

Zusätzlich wird für jedes Attribut das Mapping durchgeführt. Jedem Attribut ist die entsprechende Lookup-Tabelle, die ID-Spalte und die Beschreibungs-Spalte der Lookup-Tabelle zuzuweisen.

Für jedes Attribut kann zudem festgelegt werden, wie dieses Attribut in den OLAP Berichten dargestellt wird (z. B. kann zusätzlicher Text zu dem eigentlichen Inhalt des Attributes hinzu konkatiniert werden: ‚Abteilung:'||abteilung_id), oder wie die Standard-Sortierung (auf- oder absteigend) für dieses Attribut in den OLAP Berichten aussehen soll.

Eine Besonderheit bei der Definition von Attributen stellt die sogenannte **Quality** dar. Eine Quality ist ein Attribut einer Fakttabelle, welches nicht aggregierbar ist. Eine Quality ist normalerweise nicht mehr als ein Flag, welches den Status eines Datensatzes der Fakttabelle kennzeichnet. Z. B. könnte eine Quality existieren, um zu kennzeichnen, daß der entsprechende Umsatz während einer Werbeaktion durchgeführt wurde. In dem Fall des Prototyps der Fa. Oppenhoff & Rädler wird eine Quality ‚Status' basierend auf das Attribut ‚Azei_Status_KZ_Q' der Fakt-Tabelle ‚Aufwand_F' für den Status des Aufwandes erstellt:

0 für ‚**nicht abrechenbar**'
1 für ‚**frei abrechenbar**'
2 für ‚**Prebill**'
3 für ‚**in Rechnung**'
4 für ‚**Teilzahlung**'
5 für ‚**bezahlt**'

Anhand der Quality ist es nun möglich einen Bericht in DSS Agent zu definieren, der die Aufwände getrennt nach Aufwands-Status analysieren läßt.

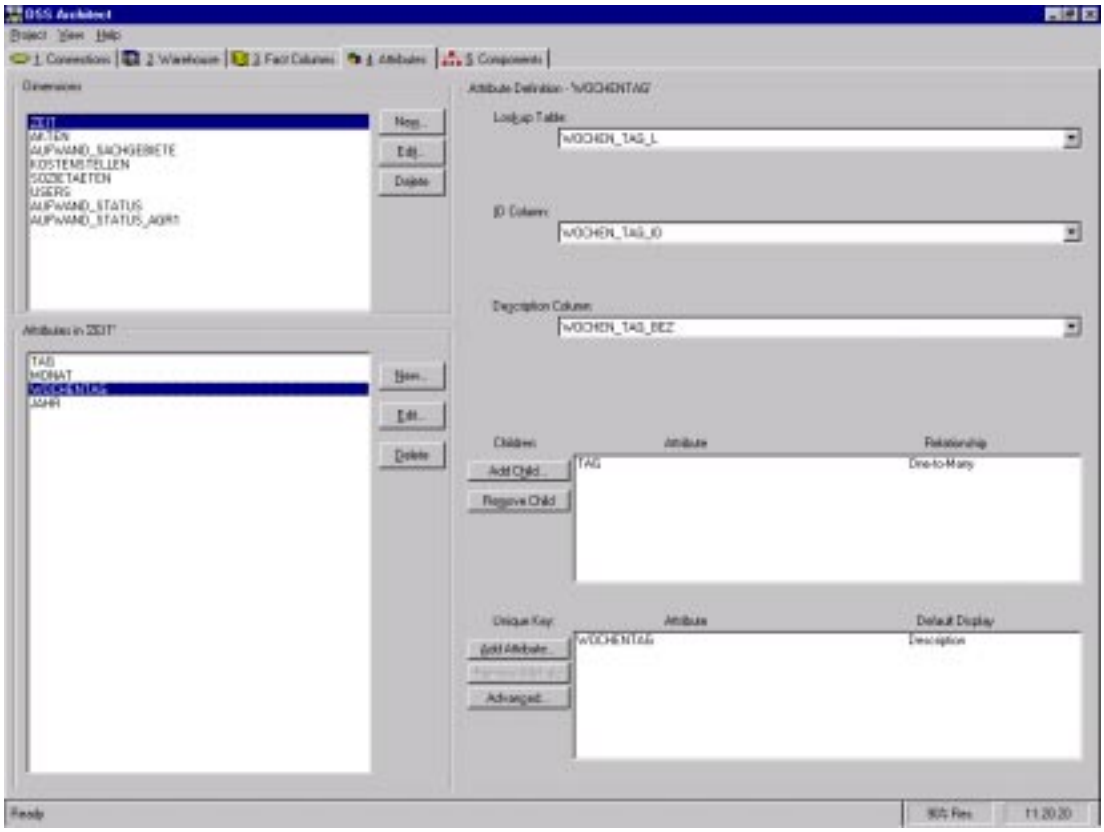


Abbildung 52: ‚Attributes' Folder (DSS Architect)

Folder 5: Components

In dem letzten Folder werden die in OLAP Berichten verfügbaren Dimensionshierarchien definiert. Diese Hierarchien entsprechen den Drill Down und Drill Up Stufen, die für einen Bericht ermöglicht werden.

Durch die Auswahl einzelner Attribute einer Dimension, wird die Hierarchie dieser Dimension festgelegt. Dabei können für jede Dimension mehrere, unterschiedliche Hierarchien entwickelt werden. So kann es z. B. gewünscht sein, daß man in einem Bericht ein Drill Down von einem Jahr, über die Monate des Jahres bis zu einem Tag des Monat ermöglicht. Für einen anderen Bericht ist es jedoch gewünscht, direkt von einem Jahr auf die entsprechenden Tage des jeweiligen Jahres zu drillen. Für beide Berichte müssen somit

zwei unterschiedliche Dimensionshierarchien der Dimension ‚Zeit' erstellt werden, damit diese unterschiedlichen Drill-Wege ausführbar sind.

Zusätzlich kann man die unterste Hierarchiestufe einer Dimensionshierarchie sperren (ersichtlich durch Schloß-Icon). Durch diese Sperrung ist es nicht möglich, bei der Erstellung eines OLAP Berichtes, versehentlich alle Elemente des Attributes der untersten Hierarchiestufe im Design-Modus angezeigt zu bekommen. Gerade bei sehr detailierten Daten können ansonsten schon mal unerwünscht einige tausend Elemente angezeigt werden. Bei der Ausführung des Berichtes ist dieses Attribut allerdings nicht gesperrt und kann dementsprechend analysiert werden.

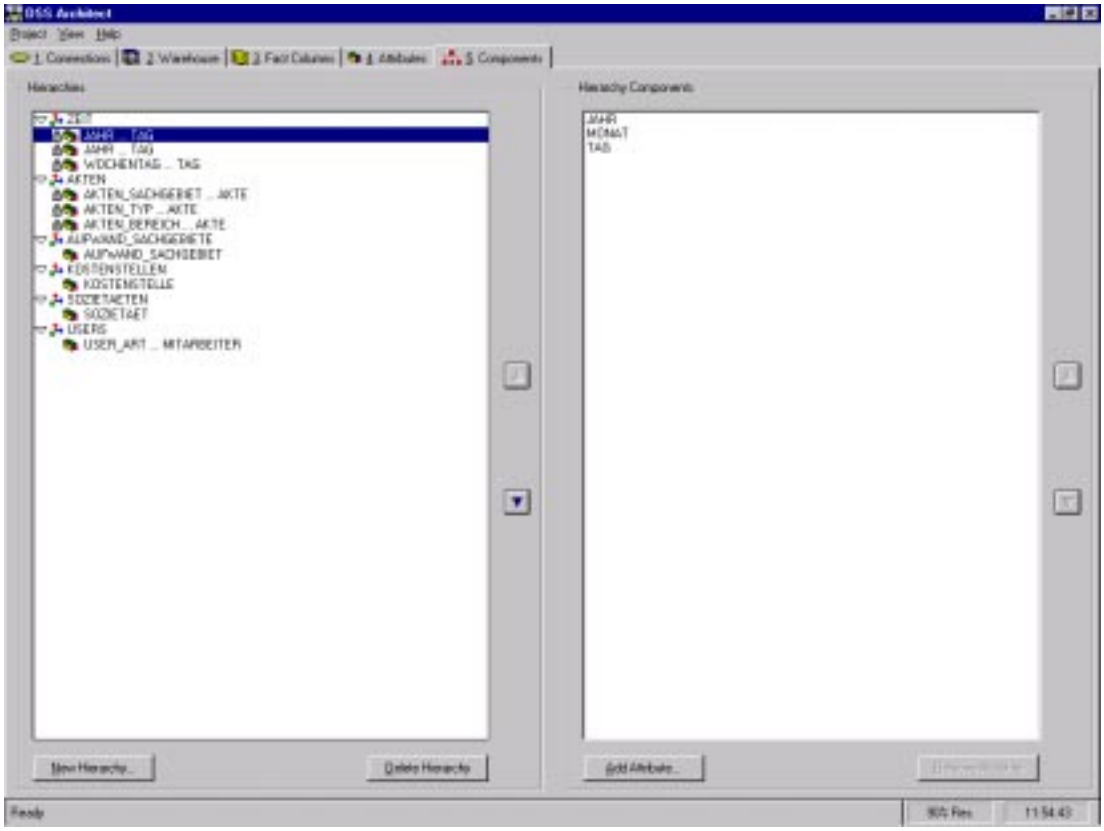


Abbildung 53: ‚Components' Folder (DSS Architect)

Nachdem auch der fünfte Folder bearbeitet wurde, kann das neu erstellte Projekt gespeichert werden. Durch die Speicherung des Projektes sind die semantischen Metadaten, aus dem DSS Agent heraus, für die Erstellung der gewünschten OLAP Berichte verfügbar.

7.10 Die Erstellung der OLAP Berichte mit DSS Agent

Bei dem Tool ‚DSS Agent' der Firma MicroStrategy handelt es zum einen um ein OLAP Berichtsentwicklungswerkzeug und gleichzeitig um ein Auswertungstool, mit dem „normale" User vorgefertigte OLAP Berichte aufrufen und analysieren können. Es gibt aber bei dem Tool keine strikte Trennung zwischen Entwicklern und Usern. Erfahrenere User können somit auch eigene OLAP Berichte entwickeln, wenn sie dies wünschen.

7.10.1 Userverwaltung mit DSS Agent

Eine Userverwaltung unter DSS Agent gibt es nicht. Die eigentliche Userverwaltung wird über das Datenbanksystem des Data Warehouses abgewikkelt. D. h. hier werden User des Data Warehouses eingerichtet, die eine Leseberechtigung auf die eigentlichen Data Warehouse Tabellen erhalten.

Unter DSS Architect wird ein User der Data Warehouse Datenbank für ein Projekt als ‚**Project System User**' ausgewählt. Besonderheit dieses Users ist, daß die OLAP Berichte und Objekte die er unter DSS Agent erstellt, auch von allen anderen Usern verwendet werden können. Eine Abänderung, oder gar eine Löschung ist allerdings durch diese „normalen" User nicht möglich. Der Project System User hat allerdings keinerlei Zugriff auf Objekte die durch andere User erstellt wurden.

Alle anderen eingerichteten User des Data Warehouses können den DSS Agent ohne weitere Einrichtungsarbeiten sofort benutzen, in dem sie sich unter ihrer Datenbankkennung in dem DSS Agent anmelden. Sie haben dadurch nicht nur die Möglichkeit die OLAP Berichte und Objekte des Project System Users zu benutzen, sondern sie können im Entwicklungsmodus auch eigene OLAP Berichte und Objekte erstellen, auf die jedoch kein anderer User Zugriff hat.

Ganz im Gegensatz zu Usern, die in einer **User-Group** zusammengefaßt wurden. Diese Einstellung wird unter DSS Agent durchgeführt, genauer gesagt wird diese Zuordnung von Usern zu einer bestimmten User-Group in einem Initialisierungsfile (PROJEKTNAME.DSS) des DSS Agents unter dem [SYSTEM] Abschnitt als Parameter eingetragen. Das Initialisierungsfile des DSS Agents wird bei jedem Start des Tools eingelesen. Durch den Eintrag:

[System]
GROUP USER = Username

wird es einem User, der diesen Eintrag in seinem DSS Agent File eingefügt hat, ermöglicht, auch die OLAP Berichte und Objekte des Users ‚USERNAME' zu verwenden, nicht jedoch zu ändern, oder zu löschen. Dadurch können ganze User Gruppen gebildet werden, wobei einer der User als Group User festgelegt wird, so daß alle andern User derselben Gruppe auf die Objekte des Group Users zugreifen können.

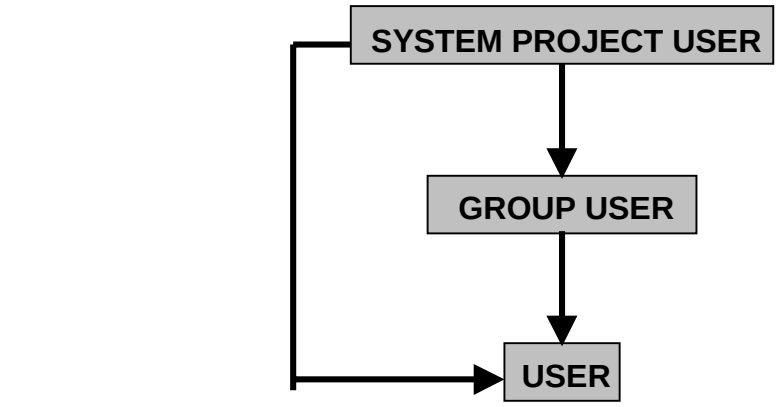


Abbildung 54: Userverwaltung in DSS Agent

7.10.2 Einstellungsmöglichkeiten des DSS Agent

Beim erstmaligen Starten des DSS Agent besteht die Oberläche nur aus einem leerem Hintergrund und einer Menüleiste. Über die Menüleiste sind sämtliche Funktionalitäten und vor allem auch Einstellungen der Oberfläche des DSS Agent steuerbar. So ist es z. B. möglich die Ausgabesprache auf Deutsch zu stellen, oder aber die verfügbaren DSS Funktionalitäten einzuschränken bzw. zu erweitern. Sämtliche Einstellungen die durchgeführt werden, werden für den angemeldeten User nach dem Schließen des DSS Agents in der Metadatenbank gespeichert, so daß beim erneuten Starten durch den gleichen User auch die **userspezifischen Einstellungen** wiederhergestellt werden. Da es keine Userverwaltung unter DSS Agent gibt kann somit auch jeder User seine eigenen Einstellungen, welche Bedienung und Funktionalität des DSS Agents betreffen, frei bestimmen.

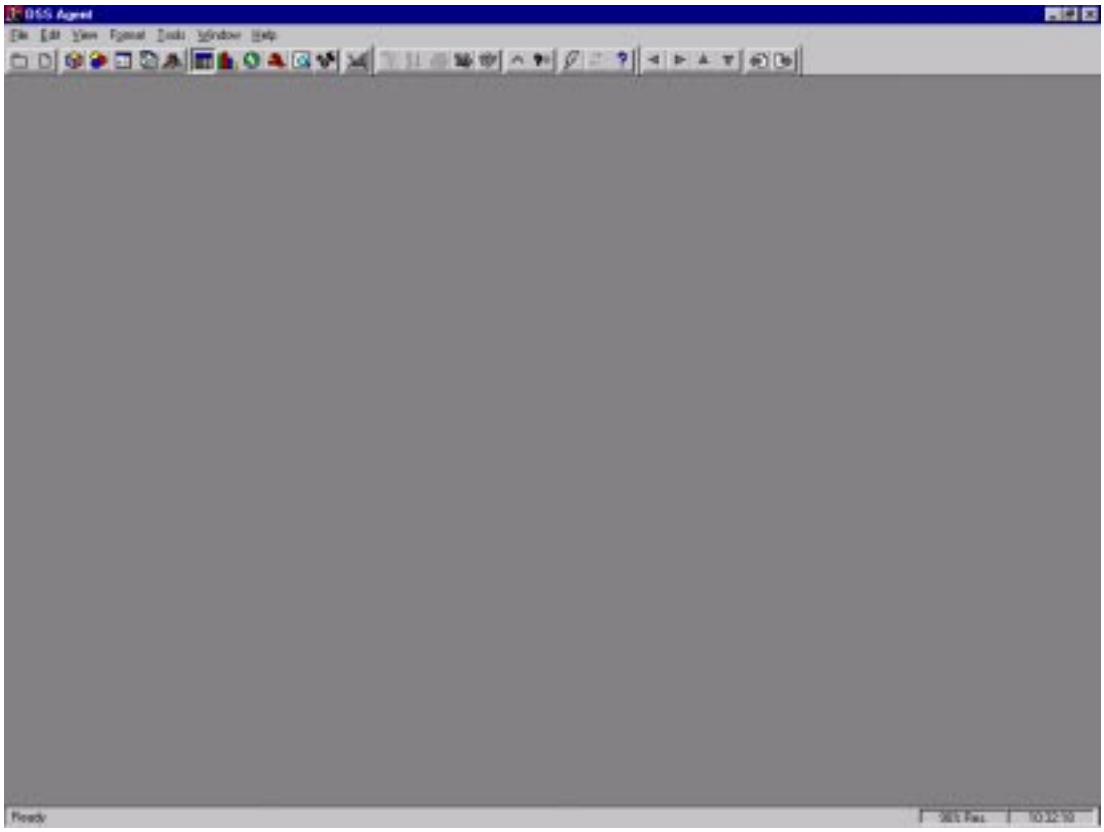


Abbildung 55: DSS Agent Basis-Bildschirm

7.10.3 Die Erstellung von DSS Agent Berichten

Bei der Erstellung eines DSS Agent Berichtes, der z. B. die Umsätze in DM pro Sozietät und Aktensachgebiet des Jahres 1998 analysieren läßt, müssen die folgenden Arbeitsschritte ausgeführt werden:

(1) Die Erstellung einer Metrik

Bei einer Metrik handelt es sich um die Berechnung einer Geschäftskennzahl wie in diesem Beispiel der Umsatz in DM. Man unterscheidet einfache und zusammengesetzte Metriken.

Eine **einfache Metrik** ist eine oder sind mehrere Faktspalten in Kombination mit Aggregationsoperatoren (SUM, AVG, MAX, MIN, COUNT), bzw. mathematischen Operatoren (+, -, (,), /, *), z. B.:

Umsatz = SUM(AZEI_BETRAG_F)

Die für eine Metrik verwendbaren Faktspalten werden im DSS Architect definiert. Dabei kann eine Faktspalte auch mehrfach in einer Metrik eingebunden werden.

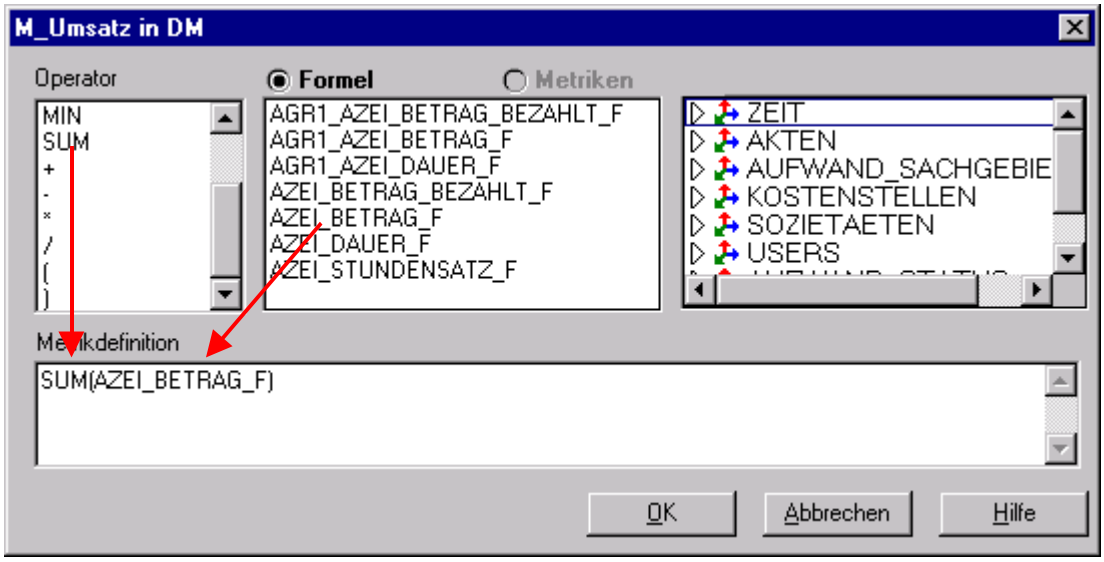


Abbildung 56: Metrikeditor für die Erstellung einer einfachen Metrik

Bei einer **zusammengesetzten Metrik** handelt es sich um eine Formel zur Berechnung einer Geschäftskennzahl, die aus mehreren einfachen Metriken über mathematischen Operatoren (+, -, (,), /, *) zusammenggesetzt wird, z. B.:

Durschnittlicher Umsatz pro Mitarbeiter = Metrik Umsatz_in_DM / Metrik Anzahl_Mitarbeiter

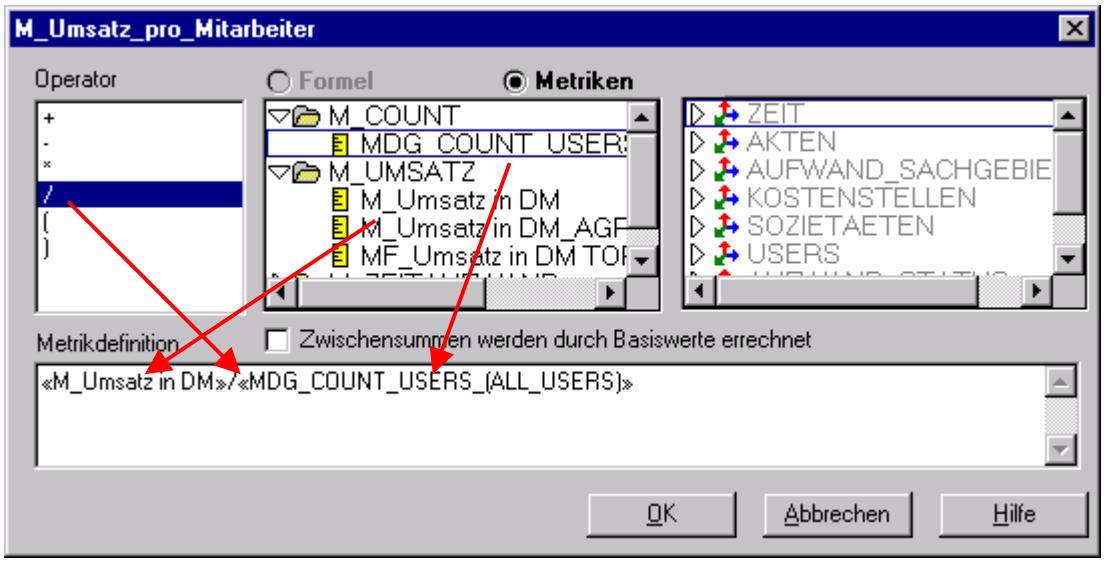


Abbildung 57: Metrikeditor für die Erstellung zusammengesetzter Metriken

Die Bestandteile der Metrikformel werden einfach durch Drag & Drop in den Abschnitt ‚Metrikdefinition' gezogen.

Für unseren Beispielbericht reicht eine einfache Metrik folgender Form aus:

M_Umsatz_in_DM = **SUM(AZEI_BETRAG_F)**

Diese Metrik wird von der SQL-Engine etwa folgendermaßen in ein SQL Statement umgewandelt:

select **sum(AZEI_BETRAG_F)**
from aufwand_f;

(2) Die Erstellung eines Filters

Da der Beispielbericht die Umsätze in DM nur für das Jahr 1998 ausgeben soll, muß ein sogenanner Filter (eine Einschränkung) definiert werden. Die Einschränkung kann einzelne Dimensionen betreffen, aber auch Metriken

und Qualities. In dem Filtereditor wird für jede Objektart ein eigenes Register angezeigt, über welches man den Filter definiert.

Beispiele:

Filter auf Metrik M_Umsatz_in_DM : Metrik > 5000

Filter auf Dimension Zeit: Jahr = 1998

Filter auf Quality Status: Status = 1

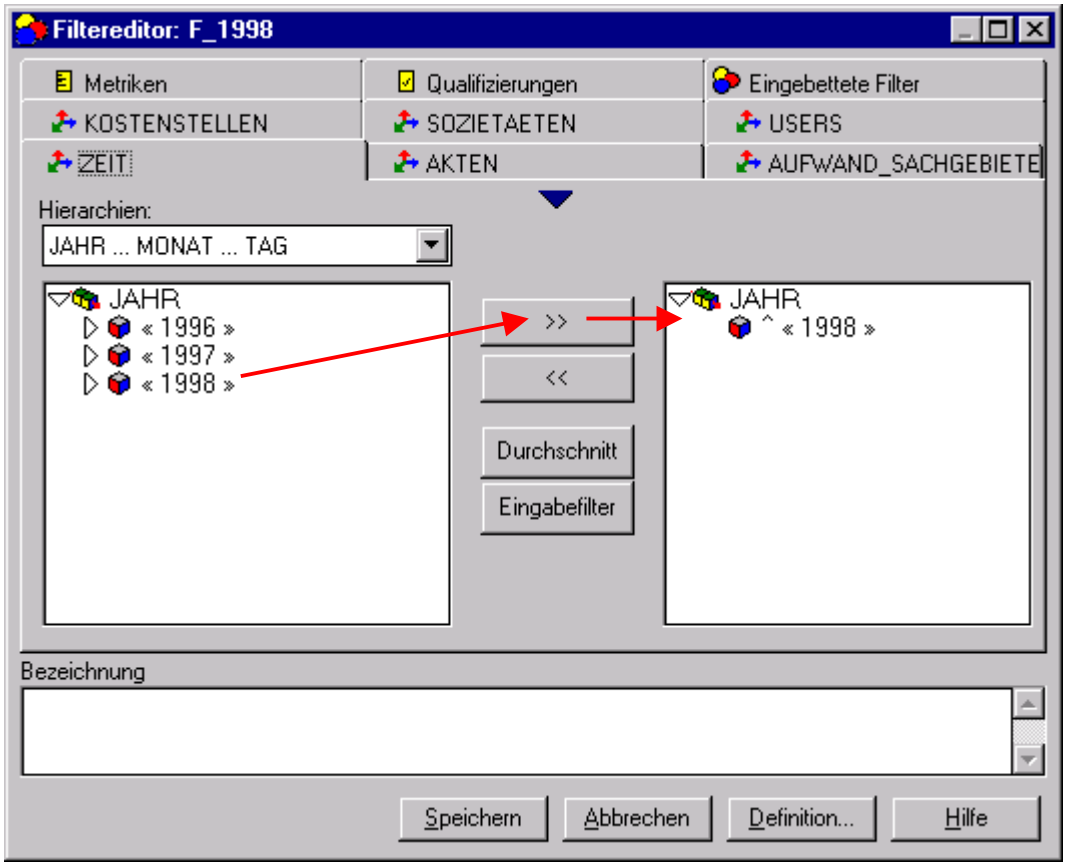


Abbildung 58: Filtereditor

In einem Filter können mehrere Filterbedingungen definiert werden. Dabei spielt es keine Rolle ob jede Filterbedingung ein anderes Objekt betrifft, oder ob mehrere Filterbedingungen einer Objektart wie z. B. der Dimension Zeit angehören.

Bei der Definition mehrerer Filterbedingungen einer Objektart wie z. B. :

Zeit = 1998
Zeit = 1997
Zeit != Januar
Zeit != Februar

müssen sogenannte Mengenoperatoren beachtet werden. Für jede Filterkomponente kann der zu verwendende Mengenoperator bestimmt werden:

Operator	**Bedeutung**	**SQL**
^	Schnittmenge	**and**
+	Vereinigungsmenge	**or**
-	Exklusion	**not**

Für das oben genannte Beispiel müßten folgende Mengenoperatoren gewählt werden:

Filter über Zeit = ^1998
+ 1997
- Januar
- Februar

Das heißt für eine Metrik würden alle Werte berechnet werden, die dem Jahr 1998 oder dem Jahr 1997 zugeordnet werden können, ausgenommen Werte der Monate Januar und Februar.

Für den Beispielbericht wird ein Filter über die Dimension Zeit definiert, durch den die berücksichtigten Umsätze in DM auf das Jahr 1998 beschränkt werden.

Die SQL-Engine würde folgendes SQL-Statement aus der schon definierten Metrik und dem eben definierten Filter generieren:

```
select sum(AZEI_BETRAG_F)
from aufwand_f a, tag_l b
where a.tag_id = b.tag_id
and b.jahr_id = ‚1998‘;
```

(3) Erstellen der Berichtsschablone

In einer Schablone wird das Format eines Berichtes vorgegeben. Dabei werden die Dimensionen, Dimensionsattribute und Metriken spezifiziert, die in den Bericht integriert werden sollen.

Eine Schablone besteht immer aus Spalten und Zeilen. Dabei werden die entsprechenden Dimensionen und Dimensionsattribute einfach per Drag and Drop aus dem Komponentenfenster des Schabloneneditors in die Spalten- und /oder Zeilendefinition der Schablone gezogen. Die Metriken werden genauso in der Schablone plaziert, mit dem Unterschied, daß mehrere Metriken einer Schablone nicht auf Spalten und Zeilen verteilt werden können, sondern vorher entschieden werden muß, ob alle Metriken in den Spalten der Schablone, oder alle Metriken in den Zeilen der Schablone plaziert werden.

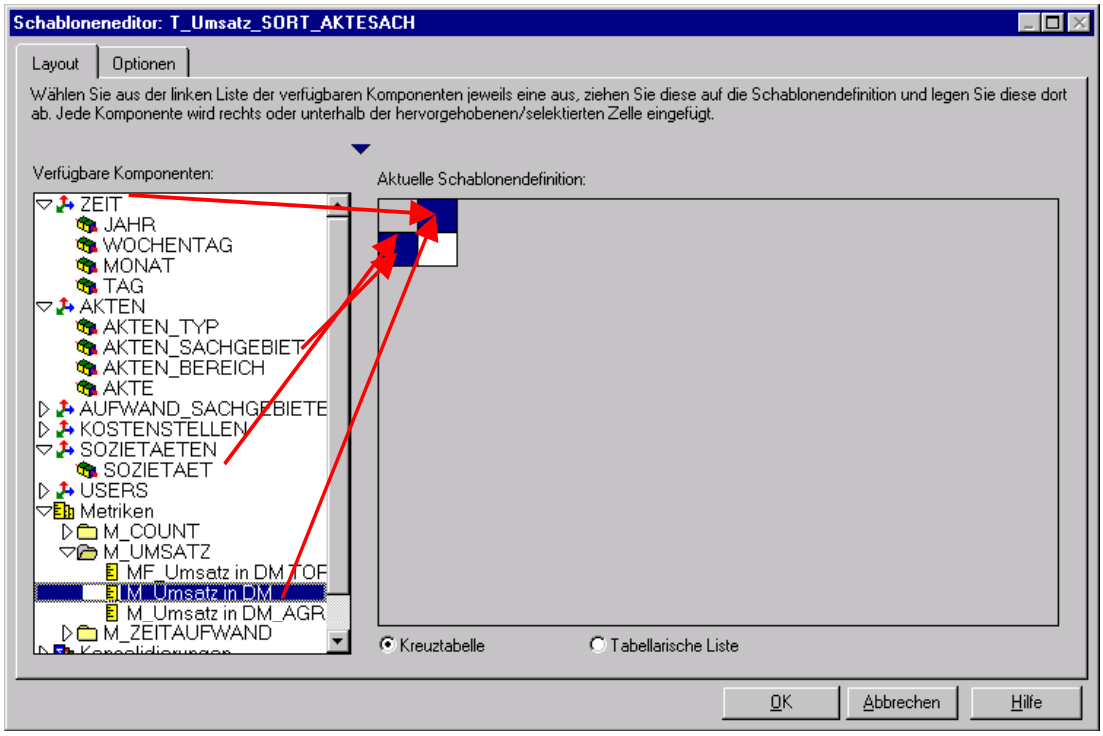


Abbildung 59: Schablonеneditor 1 (DSS Agent)

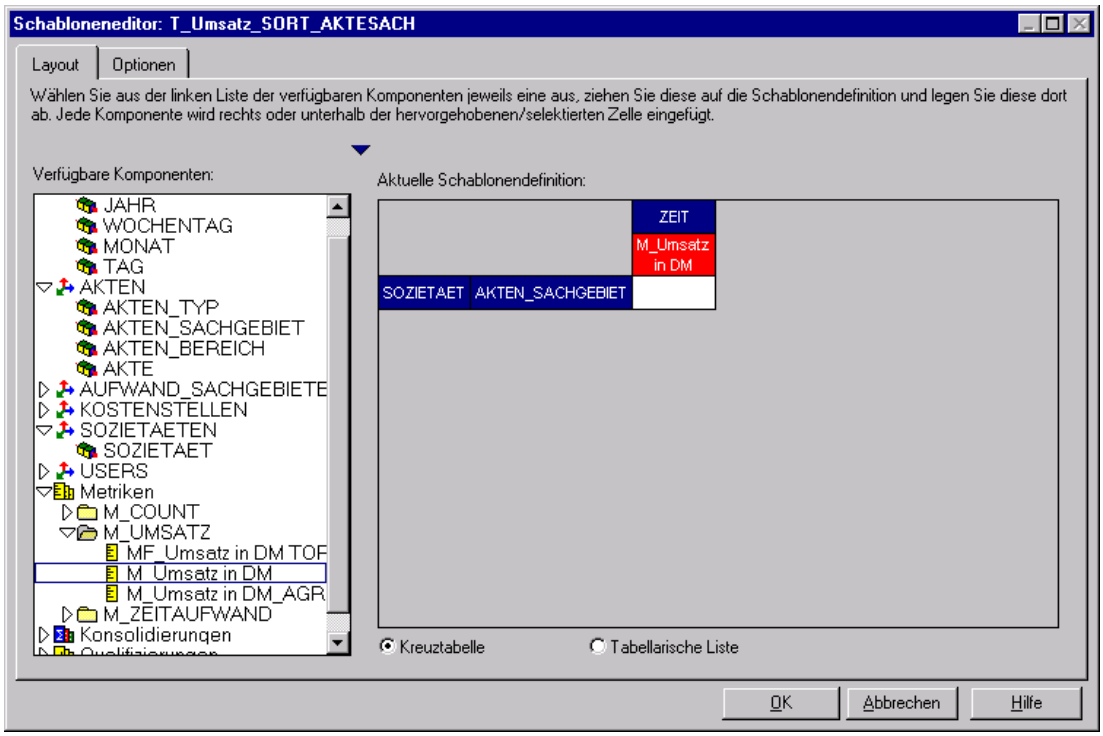


Abbildung 60: Schablonеneditor 2 (DSS Agent)

Die Erstellung des Formats des Berichtes entspricht dem SELECT einer SQL Anweisung, so daß die SQL-Engine für die eben erstellte Schablone das folgende SQL Statement erstellen würde:

```
select b.sort_bezeichnung_bez, c.sach_bezeichnung_bez,
sum(a.azei_betrag_f)
from aufwand_f a, sozietaeten_l b, akten_l c
where a.sort_id = b.sort_id
and a.akte_aktennr_id = c.akte_aktennr_id
.....
group by a.sort_id, a.akte_aktennr_id;
```

Nachdem die Berichtsschablone erstellt wurde, ist es jetzt noch möglich das Ausgabeformat der Schablone zu definieren. Dabei kann z.B. die Hintergrundfarbe der Spalten und Zeilen, das Ausgabeformat der Zahlenwerte (%, Nachkommastellen, usw.), die Darstellungsart des Berichtes (siehe Kapitel 7.10.5) usw. bestimmt werden.

(4) Die Erstellung des OLAP Berichtes

Bei der Erstellung des eigentlichen OLAP Berichtes muß jetzt nur noch die eben erstellte Berichtsschablone mit dem ebenfalls schon erstellten Filter mit Hilfe eines Berichtsassistenten kombiniert werden:

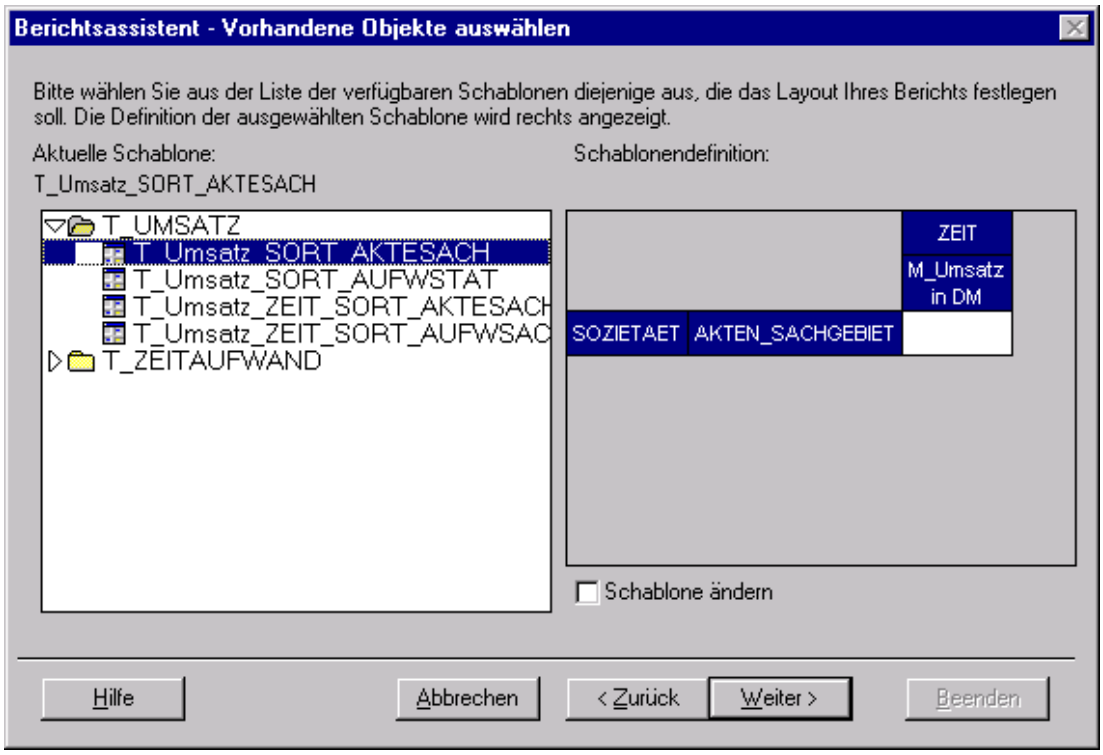


Abbildung 61: Berichtsassistent 1 (DSS Agent)

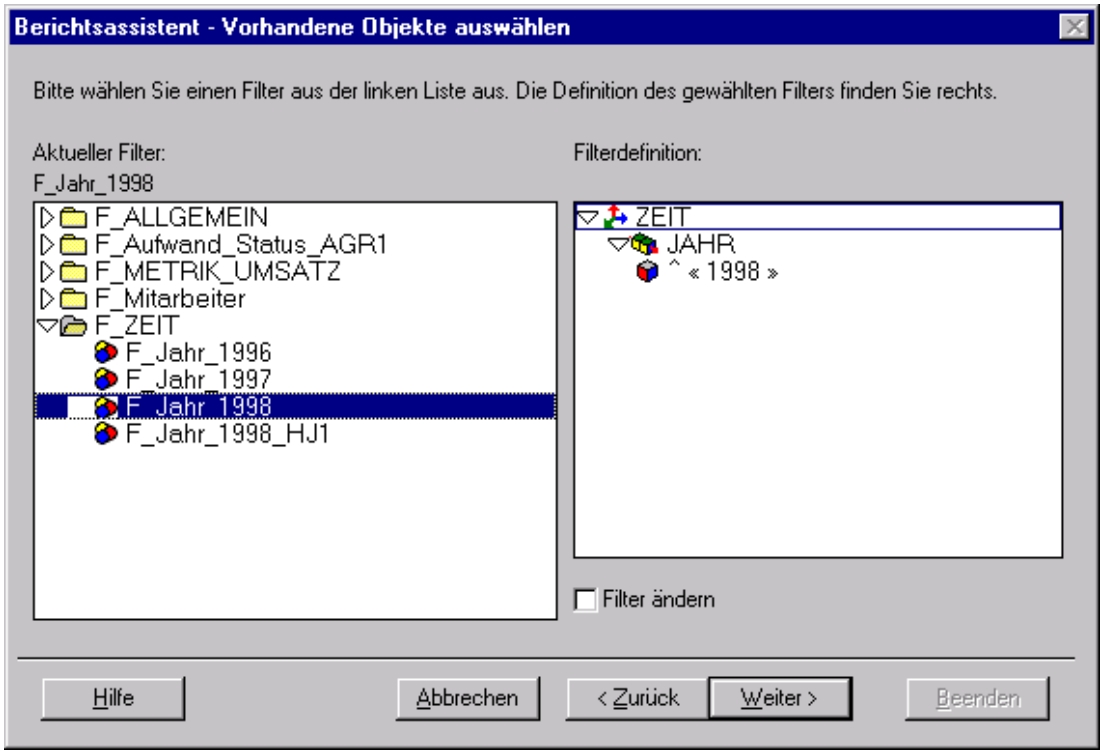


Abbildung 62: Berichtsassistent 2 (DSS Agent)

Nachdem der Bericht mit einem Namen versehen wurde wird er direkt ausgeführt:

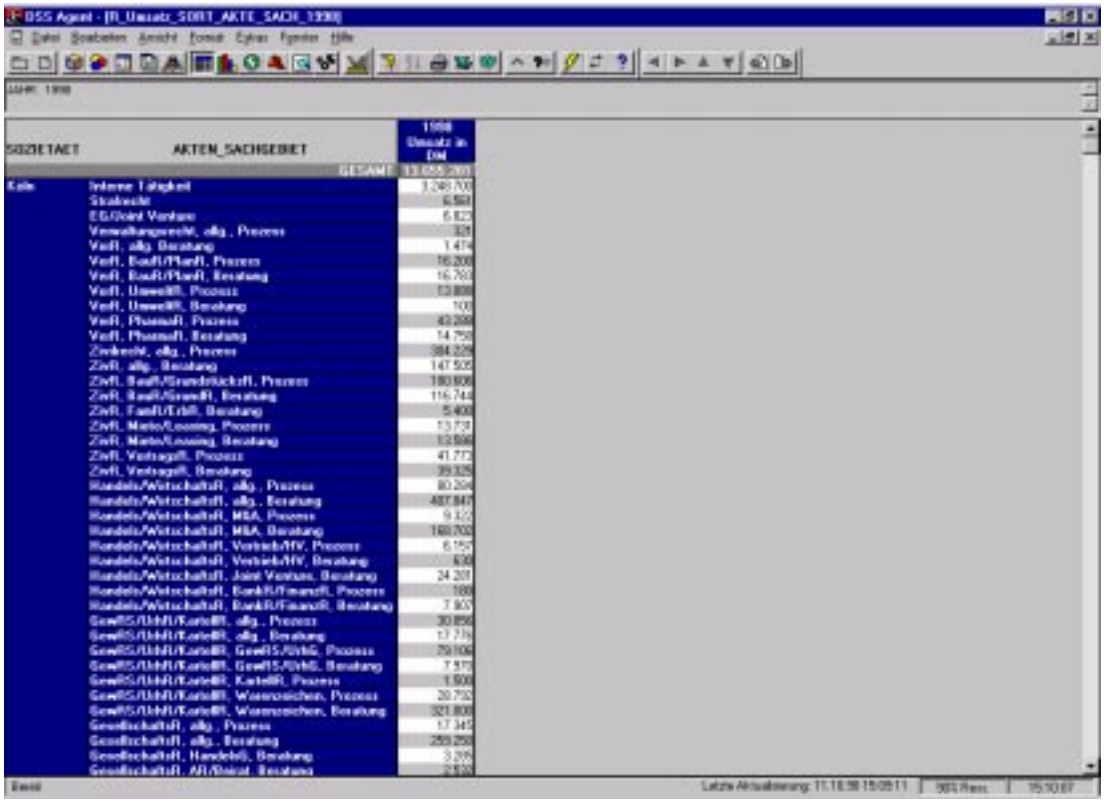


Abbildung 63: Ausgeführter Bericht ‚R_Umsatz_SORT_AKTESACH_1998‘

Jedes der eben angelegten Objekte (Filter, Metrik, Schablone, Bericht) wird in einem eigenen Objektfenster zur einfacheren Verwaltung im Designmodus des DSS Agents abgelegt:

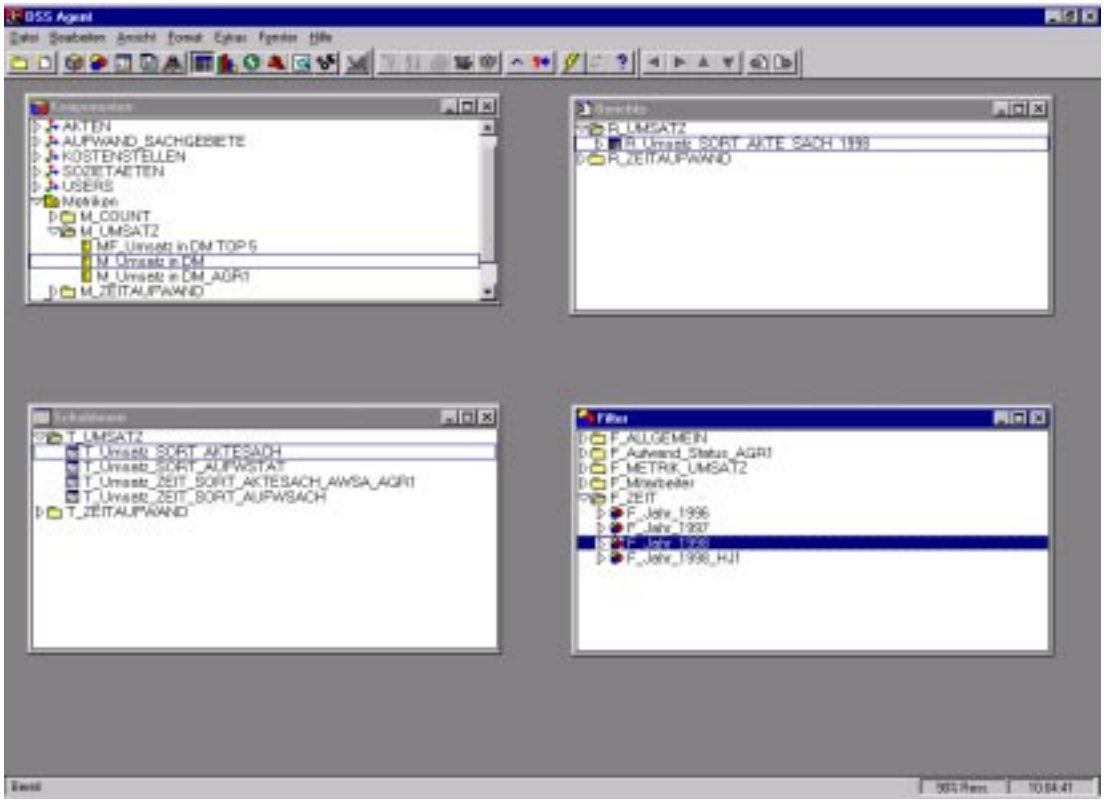


Abbildung 64: Objektfenster des Designmodus (DSS Agent)

7.10.4 Die Analysemöglichkeiten in DSS Agent

Während des **Ausführmodus** des DSS Agent ist es möglich den entsprechenden OLAP Bericht durch folgende Funktionalitäten zu analysieren, bzw. zu bearbeiten:

(1) Drill Down, Drill Up, Drill Within, Drill Across, Drill Anywhere

Ein **Drill Down** ermöglicht es den Originalbericht in eine tiefere Attributsstufe aufzugliedern. Z. B. von Aktensachgebieten auf die entsprechenden Akten.

Bei einem **Drill Up** werden die Daten wieder auf einer höheren Attributsstufe zusammengefaßt. Z. B. von einer Akte zu dem entsprechenden Aktensachgebiet.

Durch ein **Drill Within** kommt es zu einer Aufgliederung eines Attribut-Elementes nach weiteren Attributen der gleichen Dimension, die aber weder Kind- noch Eltern-Attribut sind. Z. B. von dem Attribut Monat auf das Attribut Wochentag.

Mit Hilfe eines **Drill Across** ist es mögliche ein Attribut-Element nach weiteren Attributen einer anderen Dimension aufzugliedern. Z. B. von Aktensachgebiet zu Aufwandsachgebiet.

Die Funktion **Drill Anywhere** ermöglicht es sogar, gleichzeitig ein Drill Up, Drill Down, Drill Across und Drill Within durchzuführen. D. h., hier können mehrere neue Dimensionsattribute zu dem Bericht hinzugefügt, bzw. vorhandene Dimensionsattribute des Berichtes ersetzt werden. Diese Funktion entspricht in etwa einer Neudefinition der Berichtsschablone. Wobei es allerdings nur möglich ist, Dimensionen und Dimensionsattribute zu bestimmen, keine Metriken.

Führt man einen Drill aus, so wird ein neuer Bericht in einem neuen Fenster ausgeführt. D. h., es ist möglich zwischen dem Ausgangsbericht und den danach erstellten Drill-Berichten hin- und herzuwechseln.

(2) Daten Pivotierung

Bei einer Daten Pivotierung handelt es sich um eine Umstrukturierung der Rotation einer Schablone eines offenen Berichtes (siehe auch Kapitel 3.1.1 Data Dicing). D. h., Spalten und Zeilenattribute können miteinander vertauscht werden, bzw. Attribute einer Spalte(Zeile) können vor oder hinter Atributen einer Zeile (Spalte) verschoben werden.

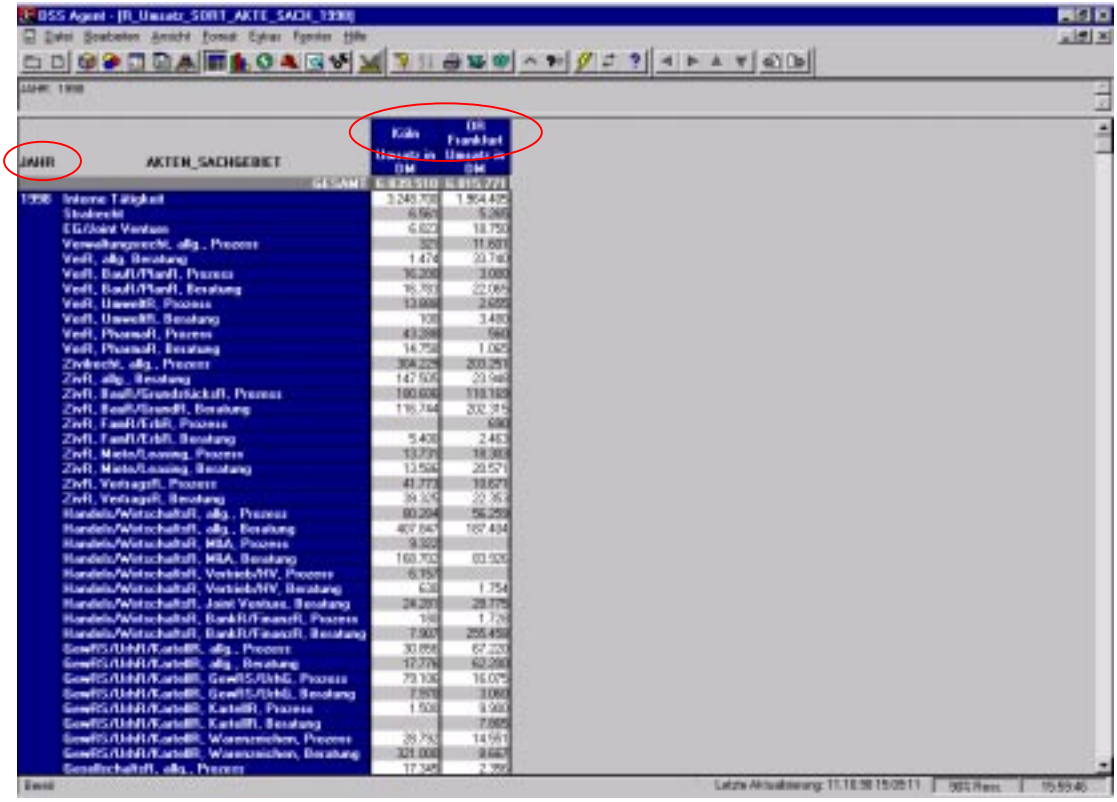


Abbildung 65: Daten-Pivotierung in DSS Agent (Tausch Sozietät mit Zeit)

(3) Aufruf des Designmodus

Über den Aufruf des Designmodus ist es möglich die dem offenen Bericht zugrunde liegende Berichtsschablone mit Hilfe des Schabloneneditors zu bearbeiten und so den Bericht in der gewünschten Form zu verändern. In erster Linie geschieht diese Änderung nur temporär, jedoch ist es möglich den veränderten Bericht unter einem neuen Namen, oder wenn der Original-bericht nicht einem Project-System-User oder Group-User gehört, unter dem alten Berichtsnamen zu speichern, so daß diese Änderungen nach dem Schließen des Berichtes nicht verloren gehen.

(4) Data Surfing

Bei dem Data Surfing wird mit drag und drop eine vorhandene Schablone oder ein Filter auf einen offenen Bericht gezogen. Der Bericht wird mit der neuen Schablone und/oder dem neuen Filter wiederausgeführt.

Dadurch wird eine vollkommen flexible Analyse von Data Warehouse Daten ermöglicht, die nicht nur von starren, vorgefertigten OLAP Berichten eingegrenzt wird, sondern die freie Verwendung von Filtern und Schablonen einbezieht.

(5) Druckoption

Wem eine Analyse des OLAP Berichtes am Bildschirm nicht ausreicht, bzw. wer die Analyseergebnisse lieber auch auf dem Papier dokumentieren möchte, für den ist es möglich den Inhalt des aktiven Berichtsfensters auszudrucken.

(6) Ad Hoc Export

Mit dieser Funktion kann das Berichtsergebnis zur Laufzeit eines OLAP Berichtes in eine andere Applikation exportiert werden. Folgende Export Zielapplikationen werden unterstützt:

- E-Mail: Lotus cc:Mail und Microsoft Mail/Exchange
- Tabellenkalkulation Lotus 1-2-3 und Microsoft Excel
- Textverarbeitung Ami Pro, Microsoft Word und Word Perfect

Um diese Funktion zu nutzen, muß vorher die entsprechende Exportapplikation geöffnet werden. Bei Ausführung des ad hoc Exports z. B. nach Excel wird der Inhalt des Berichtsfensters nach Excel in das vorher geöffnete Excel Dokument exportiert.

(7) Ad Hoc Sortierung

Durch die Ad Hoc Sortierung ist es möglich eine Spalte/Zeile eines offenen Berichtes zu sortieren. Dabei können ein oder mehrere Attribute (Dimensionen, Dimensionsattribute, Metriken) des Berichtes, nach denen die Berichtsdaten sortiert werden sollen, ausgewählt werden. Weiterhin ist anzugeben, ob die Sortierung auf- oder absteigend geschehen soll.

7.10.5 Die Darstellungsmöglichkeiten eines OLAP Berichtes

In dem Ausführungsmodus von DSS Agent ist es nicht nur möglich den OLAP Bericht in der Tabellenform anzeigen zu lassen. Insgesamt werden sechs Darstellungsformen eines OLAP Berichtes in DSS Agent unterstützt.

(1) Tabellenmodus

Dies ist die typische Darstellungsform, welche besonders gut für umfangreiche Analysen geeignet ist.

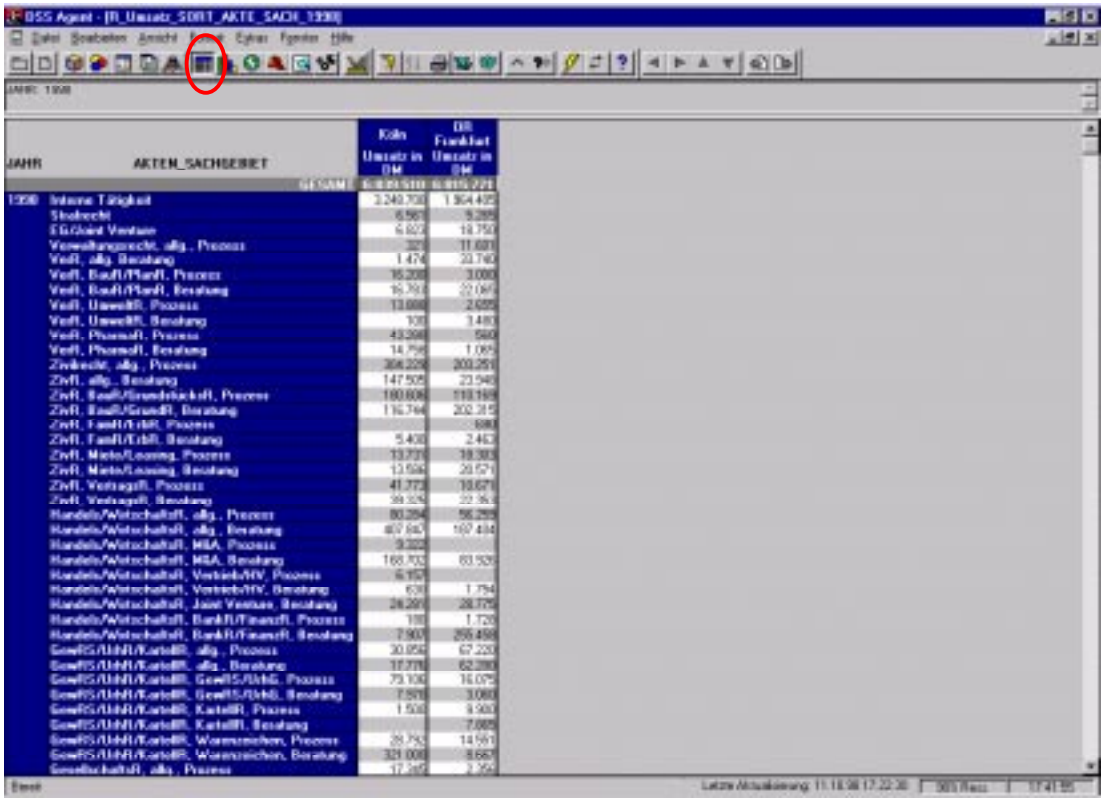


Abbildung 66: Berichtsdarstellung ‚Tabelle' (DSS Agent)

(2) Diagrammodus

Hier werden die Berichtsergebnisse grafisch dargestellt. Dabei kann die Art der grafischen Ausgabe über einen Diagrammeditor schon bei der Berichts-schablonendefinition, oder aber ad hoc zur Laufzeit des Berichtes bestimmt werden.

Insgesamt stehen 17 2D und 3D Diagrammtypen zur Darstellung von Be-richtsergebnissen zur Verfügung.

Die Ausführung der Drill-Funktionalitäten ist auch in diesem Modus möglich.

Abbildung 67: Berichtsdarstellung ‚Diagramm' (DSS Agent)

(3) Alarmmodus

Bei dem Alarmmodus handelt es sich um die Präsentation einer überschriftähnlichen Auflistung von Gegenständen, die bestimmte Voraussetzungen erfüllen. Alarme müssen voher in der dem Bericht zugrundeliegenden Schablone für den Alarmmodus definiert werden, um für diesen Bericht verfügbar zu sein.

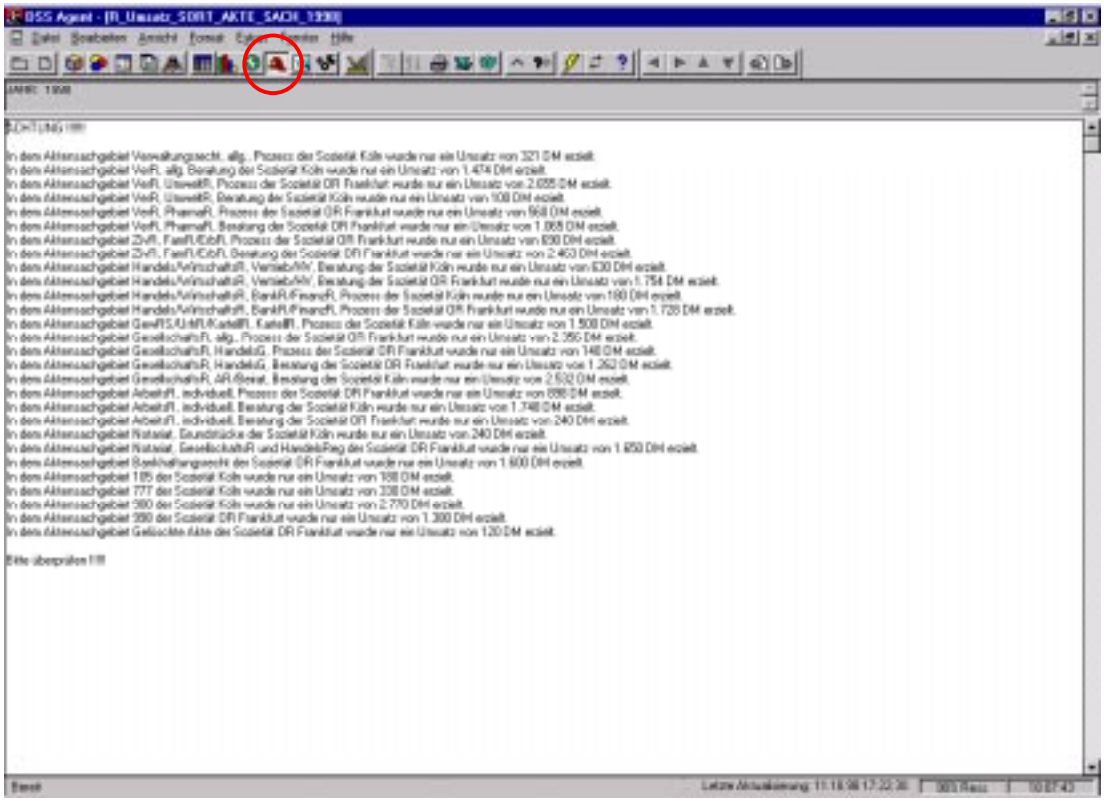


Abbildung 68: Berichtsdarstellung ‚Alarm' (DSS Agent)

(4) Kartenmodus

In dem Kartenmodus werden Berichtsdaten (z. B. Umsätze pro Region) im Kartenformat dargestellt. Dabei handelt es sich um sogenannte .WMF Files bei denen man einzelnen Bereichen der Grafik Daten des OLAP Berichtes zuweisen kann, so daß z. B. die einzelnen Umsätze der Regionen farblich in dazu passenden Bereichen der .WMF Grafik dargestellt werden können.

Die Drillfunktionalitäten stehen auch in diesem Modus zur Verfügung.

.WMF Grafiken können mit den meisten Grafiktools wie z. B. Corel Draw erstellt werden.

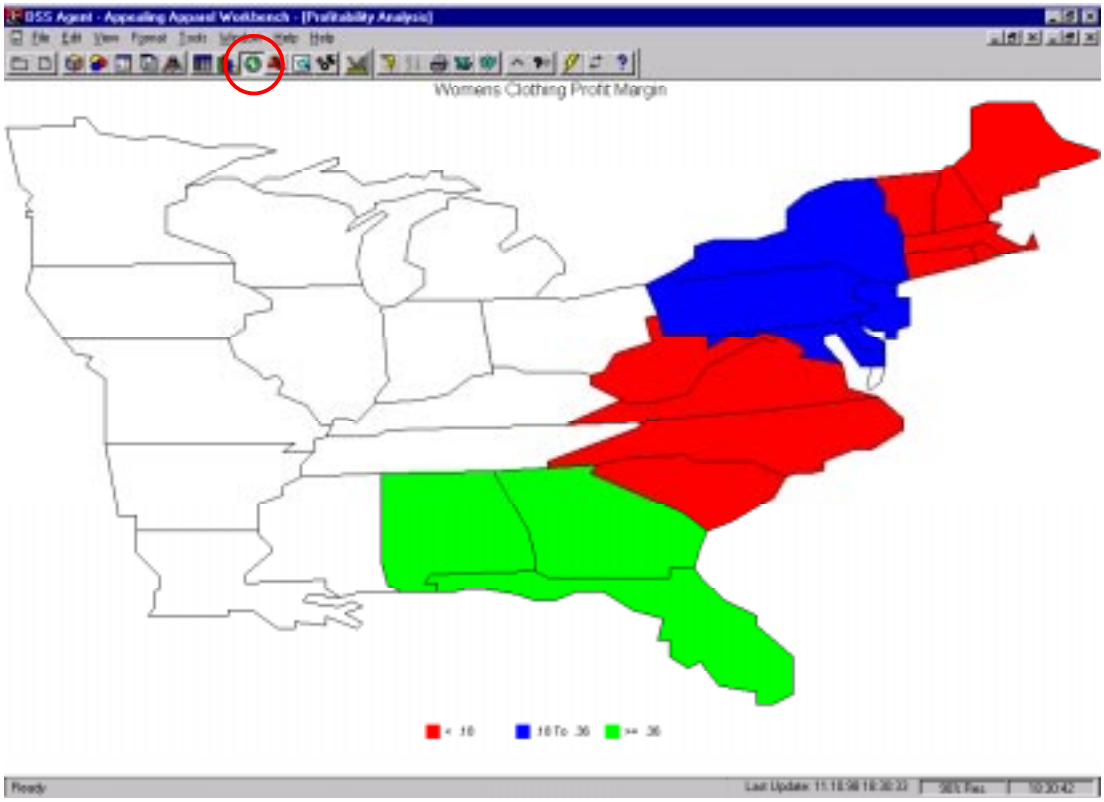


Abbildung 69: Berichtsdarstellung ‚Karte' (DSS Agent)

(5) Report Writer Modus

In diesem Modus wird ein Bericht als formatierter Text einer ‚Crystal Reports' Schablone dargestellt. Dabei muß eine Crystal Reports Schablone (.rpt Datei) mit einer dem Bericht zugrundeliegenden Schablone für den Report Writer Modus innerhalb des Schablonenditors verbunden worden sein. Erzeugt und bearbeitet wird das .rpt File über das Report-Tool ‚Crystal Reports'.

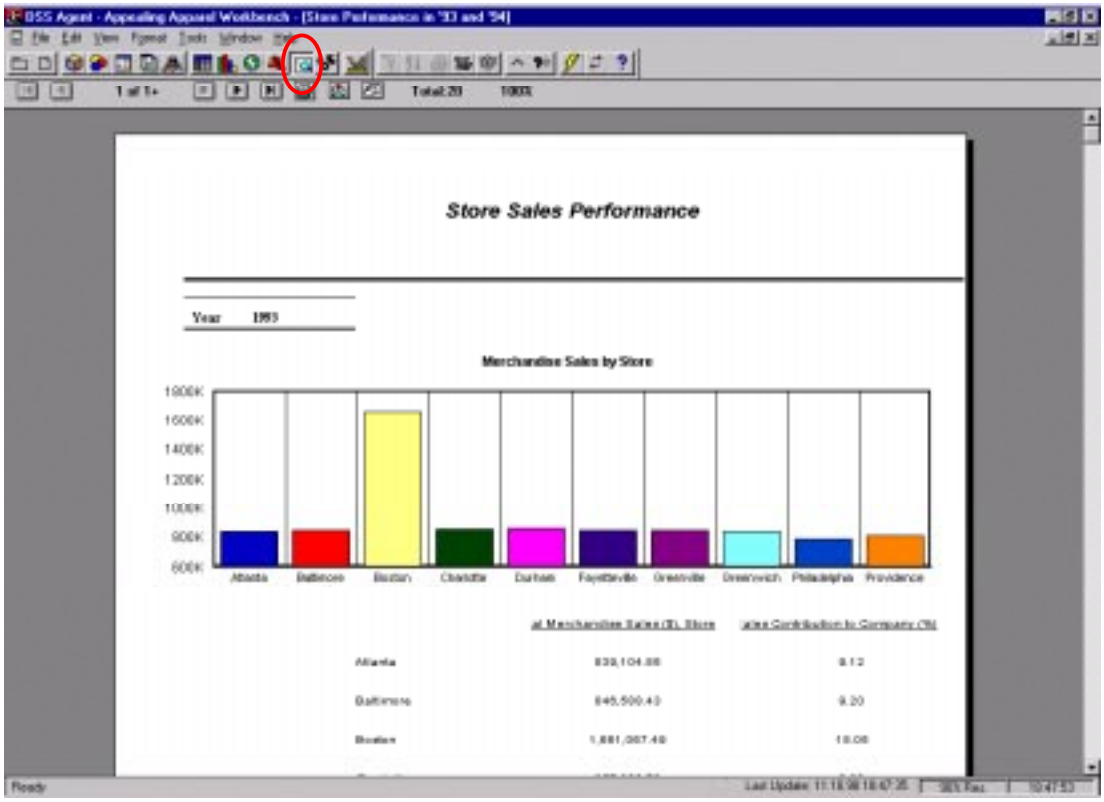


Abbildung 70: Berichtsdarstellung ‚Crystal Report Bericht' (DSS Agent)

(6) SQL-Engine Modus

In diesem Modus wird das durch die SQL-Engine erzeugte SQL Statement für den zugrundeliegenden OLAP Bericht und die Berechnungszeit dargestellt.

Dadurch ist eine Tuninganalyse der SQL-Statements möglich, um z. B. effektive Indizes auf betroffene Data Warehouse Tabellen zu erzeugen.

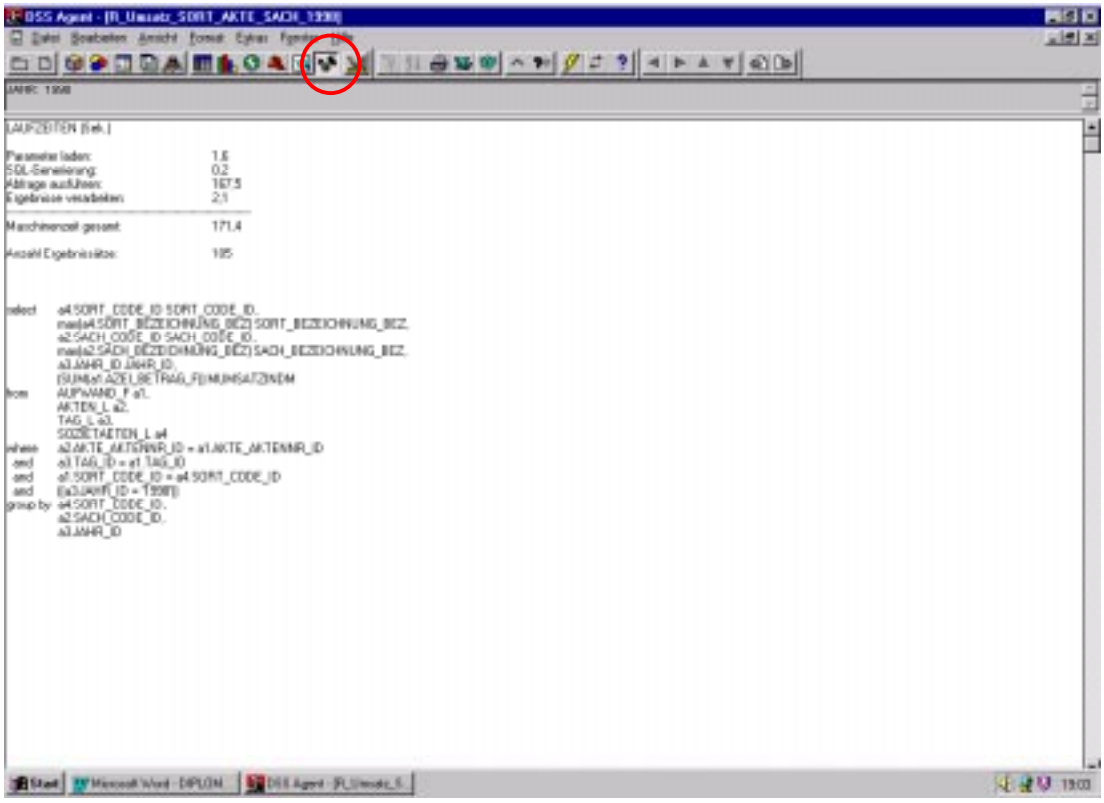


Abbildung 71: Berichtsdarstellung ‚SQL-Engine' (DSS Agent)

7.10.6 Die Erstellung komplexerer Metriken

Bei dem vorherigen Beispielbericht wurde eine sehr einfache Metrikdefinition verwendet. Jedoch können Geschäftskennzahlen für einen OLAP Bericht gewünscht sein, die wesentlich komplexere Metriken erfordern.

(1) Dimensionale Metriken

Bei einer dimensionalen Metrik handelt es sich um eine Metrik, bei der das Aggregationsniveau einer Kennzahl auf Attributs- oder Dimensionsebene festgelegt wird.

Das Aggregationsniveau normaler Metriken richtet sich immer nach dem untersten Attribut, welches auf der Berichtsschablone dargestellt wird.

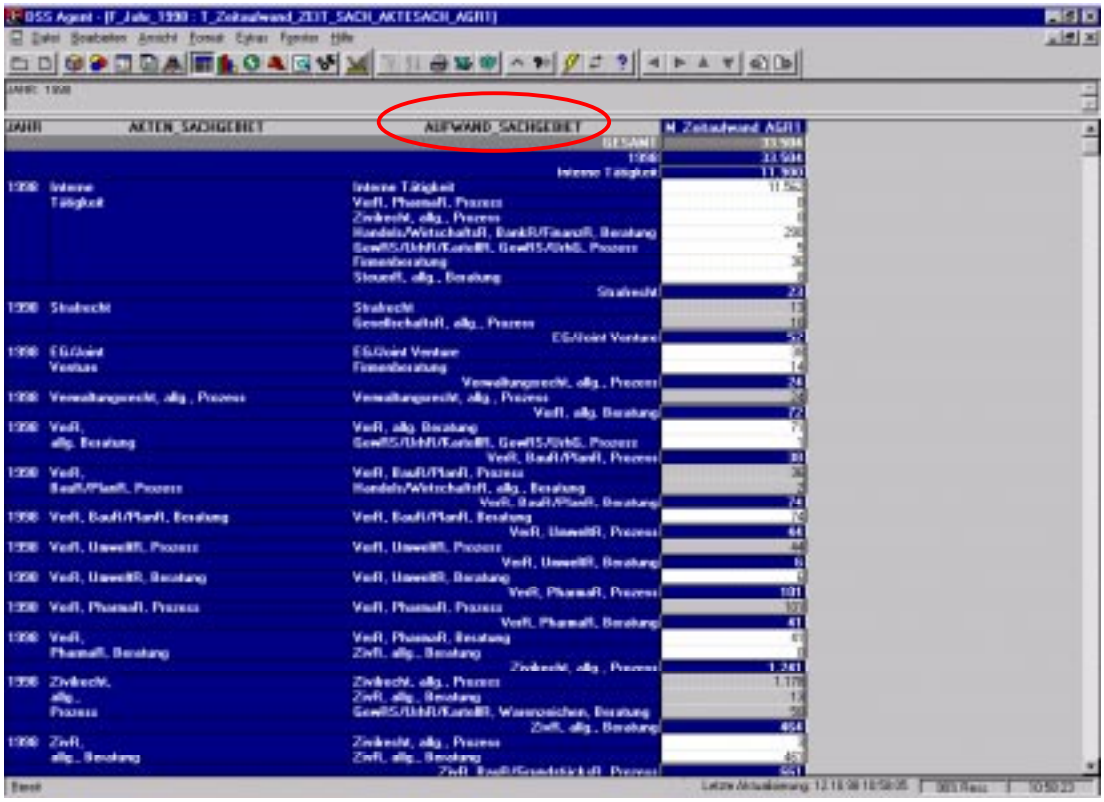


Abbildung 72: Aggregationsniveau in DSS Agent

In dem Bericht der Abbildung 72 wird die Metrik ‚M_Zeitaufwand_AGR1' auf dem Aggregationsniveau ‚Aufwand_Sachgebiet' berechnet, da für diese Metrik keine Dimensionierung gewählt wurde.

Die Berechnung einer dimensionalen Metrik richtet sich nicht nach dem untersten Attribut der Berichtsschablone, sondern nach der Dimension die dieser Metrik zugewiesen wurde.

Möchte man z. B. den prozentualen Anteil des User-Zeitaufwandes am gesamten Zeitaufwand der Sozietät, zu der der User gehört, ermitteln, so muß eine zusammengesetzte Metrik aus zwei einfachen Metriken erstellt werden.

Die **erste einfach Metrik** ermittelt den User Zeitaufwand und muß folgendermaßen definiert werden:

Zeitaufwand_User = SUM(AZEI_DAUER_AGR1)

Für diese Metrik wird keine explizite Dimensionalität festgelegt, so daß sich die Ermittlung der Kennzahl nach der Schablone und dem Filter des Berichtes richtet. Somit muß allerdings das unterste Attribut der Schablonendefinition das User-Attribut der Dimension ‚USERS' sein, damit die Metrik die Zeitaufwände pro User ermittelt.

Bei der zweiten einfachen Metrik wird eine dimensionale Metrik erstellt. Die Definition der Metrik lautet ebenfalls:

Zeitaufwand_Sozietät = SUM(AZEI_DAUER_AGR1)

Allerdings wird dieser Metrik die Dimension Sozietät zugewiesen. Dadurch orientiert sich die Berechnung der Kennzahl nicht an der Berichtsschablone, sondern an der zugewiesenen Dimension. Der Zeitaufwand dieser Metrik wird also pro Sozietät berechnet.

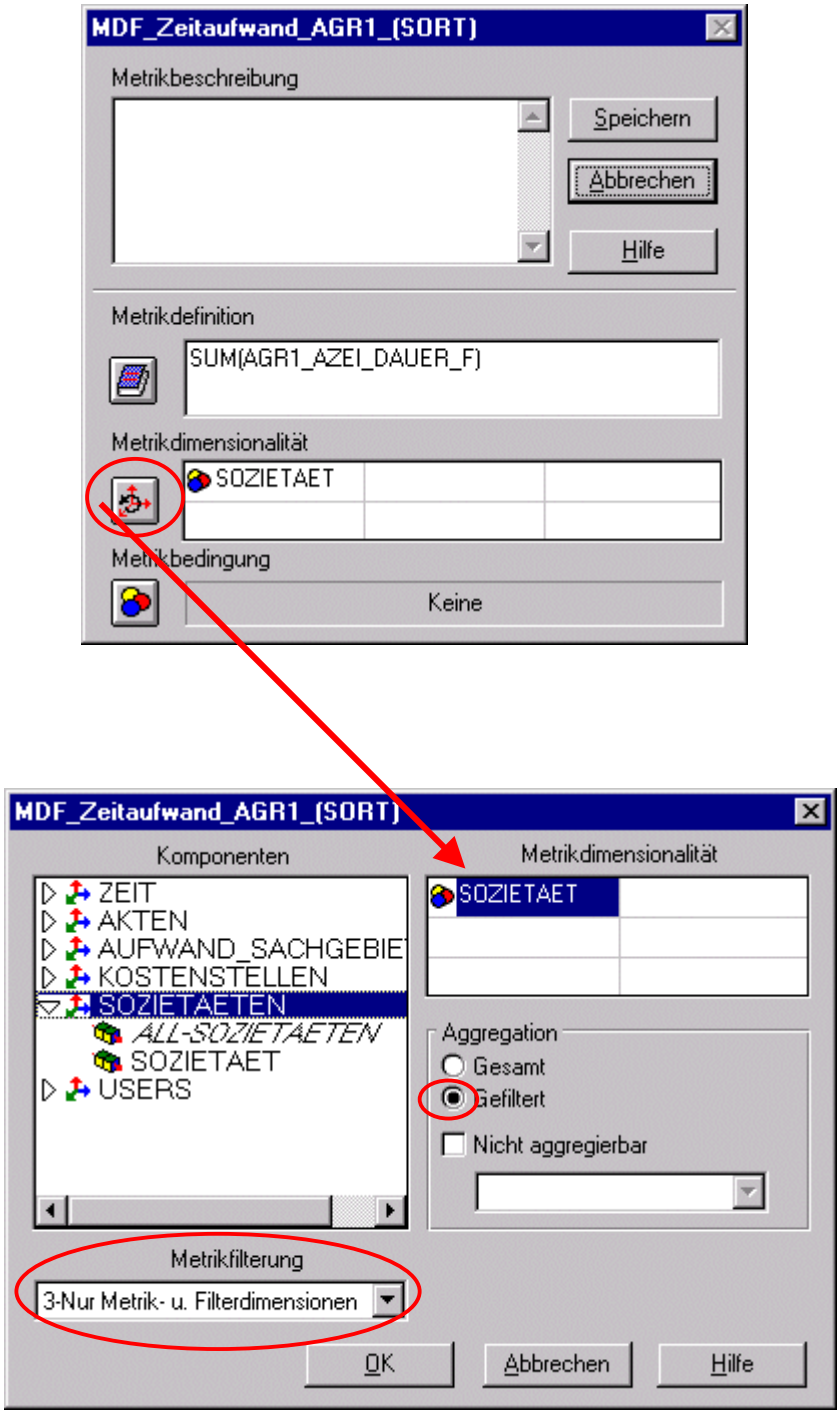


Abbildung 73: Dimensionale Metrikdefinition (DSS Agent)

Zu beachten ist hier, daß der Radio Button ‚**Gefiltert**' gewählt wird, wenn der Berichtsfilter, welcher die Dimension der dimensionalen Metrik einschränkt , bei der Berechnung der dimensionalen Metrik berücksichtigt werden soll.

Besteht z. B. ein Berichtsfilter, der den Bericht auf ausgewählte Abteilungen der Sozietät ‚Köln' einschränkt, dann wird die dimensionale Metrik auch nur für die gewählten Abteilungen der Sozietät Köln berechnet.

Wird hier ‚**Absolut**' gewählt, so hat die Einschränkung der Dimension der Metrik über einen Filter keine Auswirkung. Die dimensionale Metrik wird also komplett für jede Sozietät berechnet.

Zusätzlich muß die Art der Metrikfilterung eingestellt werden:

a) Standard

Bei der Berechnung der Metrik wird die Schablonendefinition und der Filter berücksichtigt. Die Metrikdimension wird ignoriert.

b) Nur Metrikdimensionen

Bei der Berechnung der Metrik werden die Schablonendefinition und der Filter ignoriert.

c) Metrik und Schablonendimensionen

Bei der Berechnung der Metrik wird nur der Filter ignoriert, nicht jedoch die Schablonendefinition.

d) Metrik und Filterdimensionen

Bei der Berechnung der Metrik wird die Schablonendefinition ignoriert, nicht jedoch der Filter.

Beispiel:

REGION	Standard	Nur Metrik	Metrik & Schablone	Metrik & Filter
Nord	237.224,08 DM	3.787.313,55 DM	1.925.070,77 DM	419.617,62 DM
Süd	182.393,54 DM	3.787.313,55 DM	1.862.242,77 DM	419.617,62 DM

Filter: nur Herrenbekleidung
Metrik: Umsatz (REGION)

1. **Metrikspalte:**
   Standardauflösung der Metrik, d. h. die Einträge geben die Umsätze der Herrenbekleidung in den einzelnen Regionen wieder.

2. **Metrikspalte:**
   Da sowohl die Schablone, als auch der Filter bei der Berechnung der Metrik ignoriert werden, wird in jeder Zelle der Gesamtumsatz angezeigt.

3. **Metrikspalte:**
   Hier werden die Umsätze der einzelnen Regionen dargestellt, das Filterkriterium wird ignoriert, so daß Umsätze des gesamten Sortimentes angegeben werden.

4. **Metrikspalte:**
   In diesem Fall wird der Umsatz der Herrenabteilung unabhängig von der Region dargestellt.

Nach der Erstellung der dimensionalen Metrik muß jetzt noch die zusammengesetzte Metrik erstellt werden, welche den prozentualen Anteil der

User-Zeitaufwände an den gesamten Zeitaufwänden der entsprechenden Sozietät berechnet. Die Definition dieser Metrik lautet:

%Anteil = User_Zeitaufwand * 1 / Sozietät_Zeitaufwand

Der fertige Bericht könnte folgendermaßen aussehen:

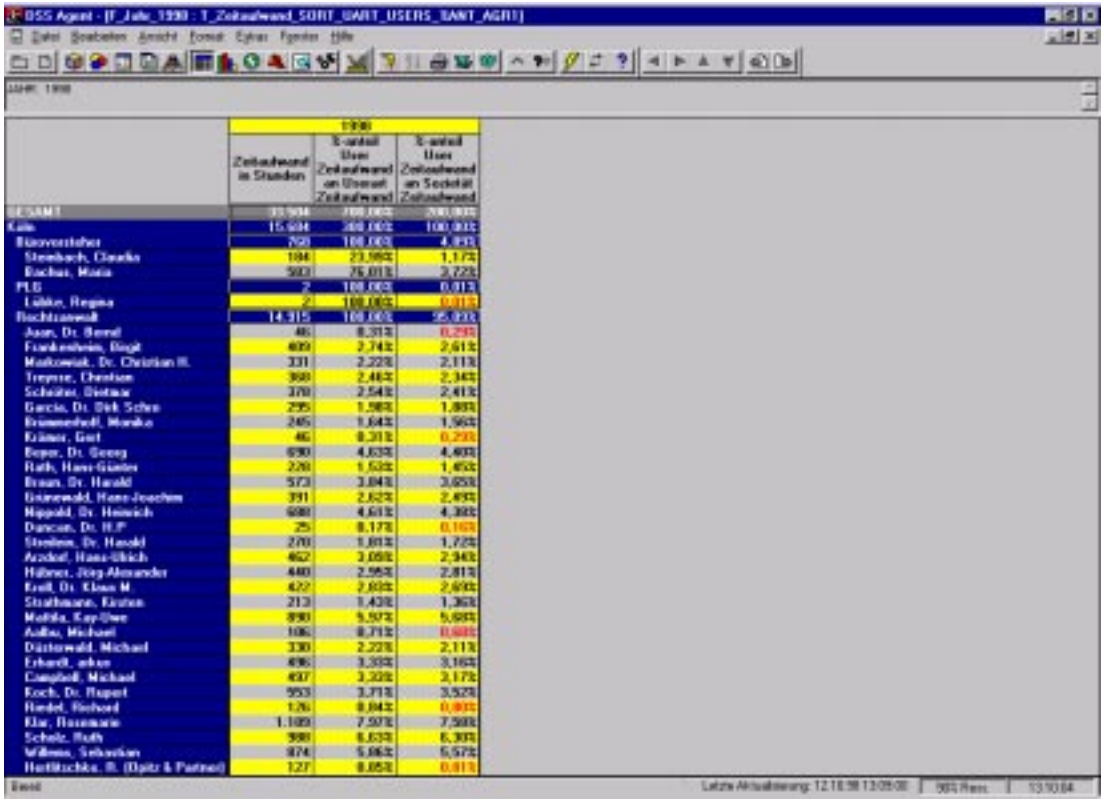


Abbildung 74: Bericht mit dimensionaler Metrik (DSS Agent)

Die Dimensionalität einer Metrik spiegelt sich in der GROUP BY Anweisung, des durch die SQL-Engine generierten SQL-Statements, wieder.

(2) Count Metriken

Count Metriken werden im wesentlichen dazu verwendet, die Attributselemente eines Attributes zu zählen, um z. B. einen Durchschnittswert zu ermitteln. Der notwendige COUNT wird dabei auf eine Dimension, bzw. auf ein Dimensionsattribut, oder auf Qualities durchgeführt.

Dabei wird eine einfache Metrik erstellt, bei der die Definition folgendermaßen aussehen müßte:

Anzahl_Mitarbeiter = count(<<Mitarbeiter>>)

Jetzt wäre es z. B. möglich, mittels einer zusammengesetzten Metrik den durchschnittlichen Zeitaufwand pro Mitarbeiter zu berechnen.

(3) Nicht-Aggregierbare Metriken

Die Option Nicht-Aggregierbar kann für die Metriken verwendet werden, die nicht über alle Dimensionen aggregiert werden dürfen.

Beispiel:

- Bestände sind über die Produkt und Geographie Dimension, aber nicht über die Zeit Dimension summierbar.

Nicht-Aggregierbare Metriken werden durch Aktivieren der ‚Nicht aggregierbar' Checkbox in der Definition der Metrikdimensionalität erstellt. Dabei wird bei der Metrikdimension jede Dimension aufgeführt über die eine Summierung nicht durchgeführt werden darf. Nach der Aktivierung der Checkbox ist auszuwählen ob stattdessen der Anfangswert oder der Endwert einer Periode als Kennzahlenwert verwendet werden soll.

Z. B. könnte man bei einer Metrik, die die Lagerbestände für die einzelnen Produkte und Lager ermittelt, das Dimensionsattribut ‚Woche' als ‚Nicht aggregierbar' definieren, wobei immer der Anfangsbestand einer Woche in der Metrik berücksichtigt wird. D. h. in dieser Metrik wird der Lagerbestand pro Produkt, pro Lager und pro Woche berechnet.

Die Definition einer Nicht-Aggregierbaren Metrik ist zugleich auch immer eine Definition einer dimensionalen Metrik.

(4) Metrische Qualifizierung

Bei einer Metrischen Qualifizierung handelt es sich um eine in einem Filter definierte Einschränkung einer Metrik. Nach Definition des Filters wird dieser in der betroffenen Metrik als Metrikbedingung ausgewählt.

Durch die Erstellung der Metrischen Qualifizierung wird die Ergebnismenge einer Metrik eingeschränkt.

Folgende Arten der Metrischen Qualifizierung können ausgewählt werden:

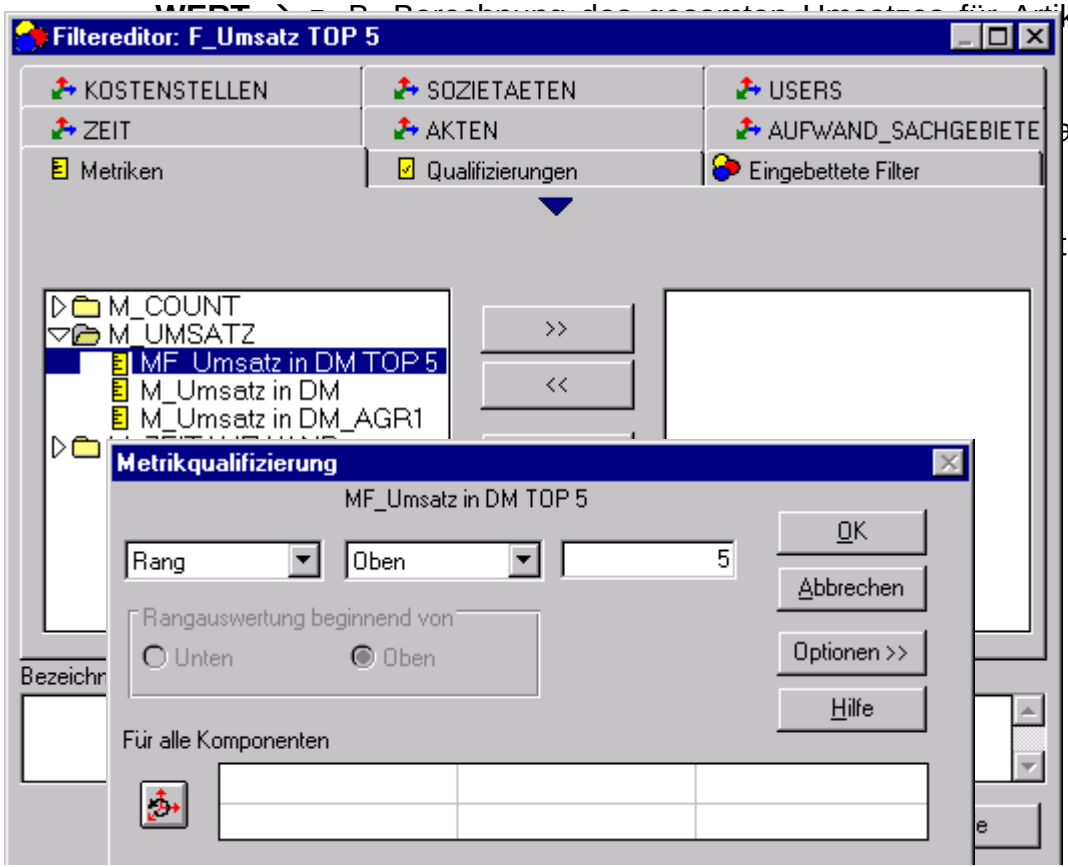


Abbildung 75: Erstellung einer Metrischen Qualifizierung (DSS Agent)

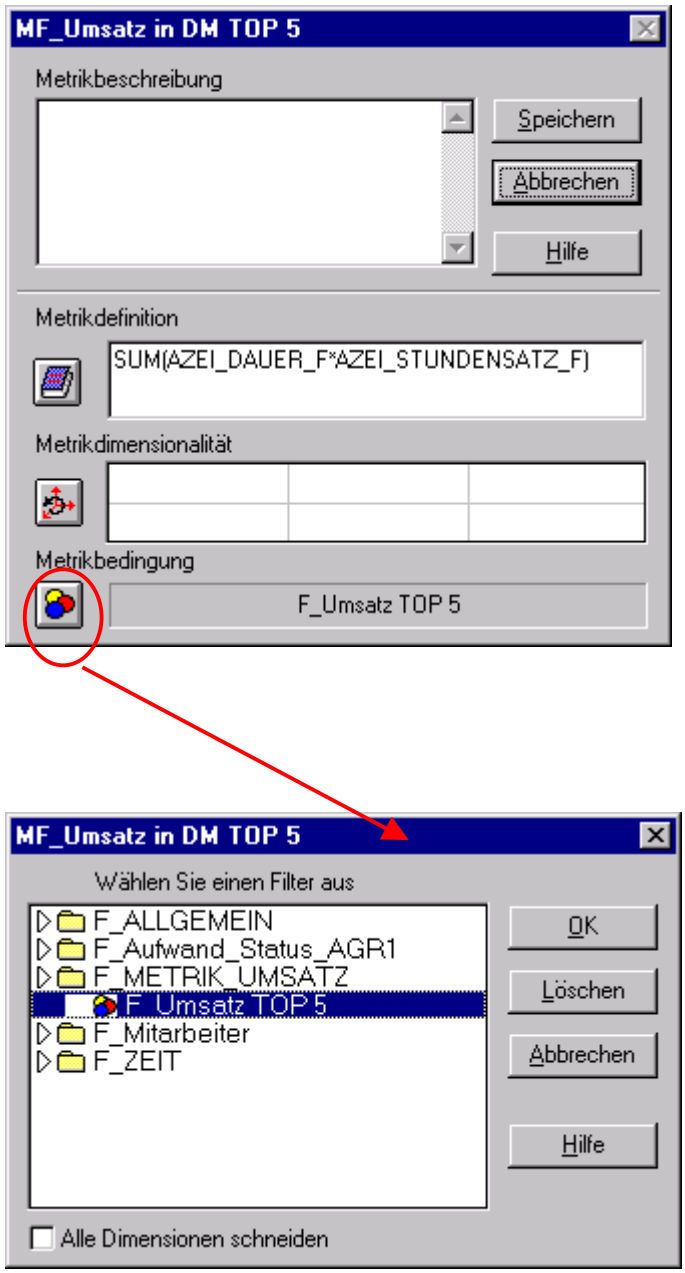


Abbildung 76: Zuweisung Metrische Qualifizierung zu Metrik (DSS Agent)

Durch die Realisierung der in Kapitel 7.1 geforderten OLAP Berichte ist die Erstellung des Prototypens für die Fa. Oppenhoff & Rädler abgeschlossen.

7.11 Schlußbemerkung

Das in diesem Kapitel beschriebene Schema zur Entwicklung eines ROLAP Data Warehouse Prototyps kann auch als **Vorgehensmodell** für die Durchführung eines kompletten ROLAP Data Warehouse Projektes verwendet werden:

- **Phase 1:** Analyse des operativen Datenmodells
- **Phase 2:** Planung des multidimensionalen Modells
- **Phase 3:** Planung des notwendigen ROLAP Datenmodells
- **Phase 4:** Aufbau der Warehouse Datenbank
- **Phase 5:** Implementierung der notwendigen Datenextraktions und -transformations Prozeduren
- **Phase 6:** Erstellung des multidimensionalen Modells
- **Phase 7:** Erstellung der OLAP Berichte
- **Phase 8:** Tuningmaßnahmen

Dabei ist schon bei diesem ‚kleinen' ROLAP Prototypen zu erkennen, daß der größte Aufwand in der Planung und Implementierung des ROLAP Modells, sowie in der Planung und Implementierung der Datenextraktions- und Datentransformationsprozeduren entsteht (Experten schätzen diesen Aufwand auf etwa 80 % des Gesamtaufwandes).

Die eigentliche Erstellung der OLAP Berichte ist aufgrund der relativ einfach zu handhabenden ROLAP Tools der Firma MicroStrategy als weniger aufwendig zu bezeichnen. Die Verwendung dieser Tools ist allerdings schon in der Planung des Data Warehouse Modells zu berücksichtigen, damit eine problemlose Erstellung der gewünschten OLAP Analysen gewährleistet werden kann.

Die Entwicklung eines Data Warehouses verläuft **iterativ**. Das vorgestellte Entwicklungsschema wird nicht nur einmal im Verlauf des Projektes linear

abgearbeitet, sondern, aufgrund der immer neu hinzukommenden Kundenwünsche, für jedes Teilprojekt ständig neu begonnen und durchlaufen (**= Iteration**). Desto weiter das Data Warehouse Projekt fortschreitet, desto mehr erkennen die Endbenutzer des Warehouses die Funktionalitäten und Möglichkeiten, die durch ein Data Warehouse zur Verfügung stehen. Aufgrund dieser Erkenntnis entstehen immer umfangreichere und komplexere Analysewünsche, die in neuen Teilprojekten umgesetzt werden sollen. Da allerdings fast jedesmal das zugrunde liegende ROLAP Datenmodell von diesen Zusatzwünschen betroffen ist, ist es notwendig das vorgestellte Entwicklungsschema auch jedesmal neu, von der ersten bis zur letzten Entwicklunsphase, durchzuführen.

Nicht zu unterschätzen ist auch der Aufwand für Tuningmaßnahmen. Damit ist nicht nur die Erstellung von geeigneten Indizes gemeint. Vielmehr muß der Tuning Aspekt schon bei der Implementierung der Data Warehouse Tabellen und der Load Prozeduren berücksichtigt werden, um zeitaufwendige Nacharbeiten zu vermeiden.

Sollte trotz umfangreicher Tuningmaßnahmen die Abfrage-Performance immer noch nicht die gewünschten Werte aufweisen, so bleibt nichts anderes übrig, als eine weniger kostengünstige, hardwaremäßige Optimierung vorzunehmen (Aufrüstung).

8 Praktische Tips für Data Warehouse Projekte

„Der Weg zum Erfolg ist mit fremden Mißerfolgen gepflastert“, Zarko Petan

Nach Gesprächen mit Projektverantwortlichen laufender Data Warehouse Projekte, die von sich zu Recht behaupten alle Höhen und Tiefen eines Data Warehouse Projektes durchlebt zu haben, entstanden folgende Empfehlungen, durch die, bei Beachtung, grundsätzliche Fehler vermieden werden können:

- 200 %ige Unterstützung durch die Führungsebene
- Festlegung der strategischen Bedeutung des Data Warehouses
- Erfahrende Partner suchen
- Beteiligte des Data Warehouse Projektes namentlich festhalten und Ziele klar formulieren
- In Geschäftsprozessen denken, nicht in IT-Strukturen
- Frühzeitig mit der Datenextraktion und –transformation beginnen (möglichst direkt nach der Informationsanalyse)
- Kleine Projektgruppen mit Beteiligten bilden, die eine möglichst hohe Entscheidungskraft besitzen
- Mit kleineren Teilprojekten die einen meßbaren Nutzen erzielen, und die schnell realisierbar sind, beginnen („Think big ,start small“)
- Die Rechenzentrum-Ressourcen rechtzeitig in das Warehouse Projekt einbeziehen
- Die notwendigen Hardwareplattformen frühzeitig klären
- Die OLAP Auswertungstools in der Anfangsphase festlegen und die Toolhersteller in die Modellentwicklung einbeziehen
- Markttrends beobachten

9 Zusammenfassung und Ausblick

Optimales Informationsmanagement ist das Schlagwort mit dem Unternehmen Wettbewerbsvorteile vor ihrer Konkurrenz erzielen wollen. Data Warehousing ist eine Methode durch die das optimale Informationsmanagement sichergestellt werden kann.

Dabei handelt es sich bei einem Data Warehouse um nichts anderes als um eine zusätzliche Datenbank die neben dem eigentlichen, operativen System, über das das Tagesgeschäft abgewickelt wird, existiert. In dieser Datenbank werden die Daten des operativen Systems zentral gesammelt und entsprechend aufbereitet, damit sie für performante Analysen zur Verfügung stehen.

Mittels sogenannter Datenextraktion und –transformation werden die Daten des operativen Systems in das Data Warehouse überführt. Dort stehen sie dann für Analysen über entsprechende Auswertungstools zur Verfügung. Bei diesen Tools handelt es sich in erster Linie um OLAP-fähige Tools, welche die typischen OLAP Funktionalitäten wie Drill Down, Drill Up, Drill Within, Drill Across, Data Dicing, Data Slicing, usw. unterstützen, so daß die Daten in betriebswirtschaftlicher Sicht und vor allem flexibel (Data Surfing) analysiert werden können. Die sonst für das operative System typische relationale Sicht (Tabellen, Views) wird im Data Warehouse durch die multidimensionale Sicht (Dimensionen, Datenwürfel) abgelöst.

Grundsätzlich sind zwei Data Warehouse Formen zu unterscheiden, das relationale Data Warehouse (ROLAP) und das multidimensionale Data Warehouse (MOLAP). Der Unterschied dieser beiden Warehouseformen besteht in der physikalischen Umsetzung. Wird bei einem ROLAP Data Warehouse die multidimensionale Sicht nur virtuell über eine relationale Datenbank realisiert, so wird diese multidimensionale Sicht in einem MOLAP Data Warehouse auch tatsächlich physikalisch implementiert. Aus diesem Grund ist es notwendig in einem ROLAP Datawarehouse ein bestimmtes, multidi-

mensionales Datenmodell einzuhalten, mit dessen Hilfe Dimensionen und Datenwürfel durch relationale Objekte wie Tabellen und Views abgebildet werden. Das Snow- und das Star-Schema haben sich hier als eine Art Standard entwickelt, wobei im Laufe der Zeit eine Vielzahl von Abwandlungen dieser beiden Modelle entwickelt wurden.

Zur Performanceoptimierung eines ROLAP Data Warehouses haben sich zusätzlich zu den datenbankspezifischen Optimierungen, die Modellierungstechniken der horizontalen und vertikalen Partitionierung, sowie die Bildung von höheren Granularitäten bewährt.

Durch die Vielzahl der schon mehr oder weniger erfolgreich abgeschlossenen Data Warehouseprojekte hat sich herausgestellt, daß der größte Projektaufwand (ca. 80 %) in der Modellierung und der Datenextraktion / –transformation einzuplanen ist.

Mittlerweile werden mehrere Entwicklungstools für die schnellere Durchführung der Datenextraktions- und –transformationsphase angeboten. Mit Hilfe dieser Tools ist es möglich, nahezu ohne Programmieraufwand, die Datenübernahme zu entwickeln. Leider sind diese Tools extrem teuer (Preise beginnen bei etwa 39.000 Dollar), so daß der erzielte Nutzen nicht in jedem Data Warehouseprojekt die hohen Kosten rechtfertigen wird.

Im Laufe der letzten Jahre haben sich besonders die OLAP Auswertungstools in ihrer Funktionalität, sowie der Handhabung weiterentwickelt. Tools wie der DSS Agent der Firma MicroStrategy sind ohne SQL Kenntnisse zu bedienen und bieten vielfältige Möglichkeiten die Daten des Data Warehouses flexibel und in unterschiedlichen Darstellungsformen zu analysieren.

Für die Zukunft ist vorauszusehen, daß sich der Data Warehouse Markt noch weiter entwickeln wird. Unternehmen, die es bisher aufgrund der Jahr

2000 Problematik versäumt haben das Thema „Data Warehouse“ anzugehen, werden spätestens nach Absolvierung der Jahr 2000 Projekte mit den ersten Data Warehouse Projekten beginnen, um keinen Wettbewerbsnachteil gegenüber der Konkurrenz, welche teilweise schon erfolgreich Data Warehouses einsetzen, zu erleiden.

Trotz den immer komfortableren Data Warehouse Auswertungs-Tools und Entwicklungsumgebungen ist „Data Warehousing“ als Konzept zu verstehen und nicht als eine Standardlösung wie z. B. SAP. Die bis jetzt abgeschlossenen Data Warehouse Projekte haben immer wieder gezeigt, daß es zu diesem Zeitpunkt noch keine Ideallösung für ein Data Warehouse gibt, auch wenn das manche Hersteller von ihren Data Warehouse Produkten vergeblich behaupten.

10 Abbildungsverzeichnis

11 Literaturverzeichnis

Schinzer, Bange, Wehner, Zeile **‚Management mit Maus und Monitor‘**
Vahlen Verlag, 1997
ISBN 3-8006-2239-4

P. Chamoni, P. Gluchowski **‚Analytische Informationssysteme‘**
Springer Verlag, 1998
ISBN 3-540-63364-2

U. Hannig **‚Data Warehouse und Managementinformationssysteme‘**
Schäffer-Poeschel Verlag, 1996
ISBN 3-7910-1097-2

M. J. Corey, M. Abbey **‚ORACLE Data Warehousing‘**
Osborne Verlag, 1997
ISBN 0471-14161-5

W. H. Inmon **‚Building the Data Warehouse, Second Edition‘**
Wiley Computer Publishing, 1996
ISBN 0471-14161-5

H. Mucksch, W. Behme **‚Das Data Warehouse-Konzept‘**
Gabler Verlag, 1996
ISBN 3-409-12216-8

Ralph Kimball **‚The Data Warehouse Toolkit‘**
John Wiley & Sons Inc., 1996
ISBN 0-471-15337-0

Gabriel, Röhrs **‚Datenbanksysteme‘**
Springer Verlag, 2. Auflage, 1995
ISBN 3-540-60079-5

Kevin Loney **‚ORACLE DBA Handbook 7.3 Edition‘**
Osborne Verlag, 1994
ISBN 0-07-882289-0

M. J. Corey, M. Abbey, D. J. Dechichio **‚Tuning ORACLE‘**
Osborne Verlag, 1995
ISBN 0-07-881181-3

P. Corrigan, M. Curry **‚ORACLE Performance Tuning‘**

O' Reilly & Associates, Inc., 1. Auflage, 1993
ISBN 1-56592-048-1

T. Portfolio **‚PL/SQL User's Guide and Reference Version 2.0'**
Oracle Corporation, 1992
Part No. 800-20-1292

J. Sweeney, M. Griese **‚The Role of a Data Mart'**
Artikel in ‚Inside Decisions', Winter 1996

P. Mertens **‚Konzeption für ein Data Warehouse'**
Artikel in ‚Datenbank Fokus', 01/98

P. Gluchowski **‚Antwortzeiten als Erfolgsfaktor'**
Artikel in ‚Datenbank Fokus', 03/98

R. Tanler, J. Frome **‚The Expanding Role of Enterprise Meta Data'**
Artikel in DMReview
http://www.datawarehouse.com/sigs/survival/article12.htm
1998

D. Marco **‚Managing Meta Data'**
Artikel in DMReview
http://www.datawarehouse.com/sigs/survival/article7.htm
1998

Ken Rudin **‚Data Warehouse Architecture'**
Artikel in DMReview
http://www.datawarehouse.com/sigs/survival/article8.htm
1998

T. Flanagan, E. Safdie **‚Getting Started with Data Warehousing'**
The Applied Technologies Group, 1997
http://www.techguide.com/index1.html

T. Flanagan, E. Safdie **‚Decision Support Architecture for Data Warehousing'**
The Applied Technologies Group, 1997
http://www.techguide.com/index1.html

ORACLE Online Dokumentation 7.3.3 **‚Oracle 7 Server Tuning: Data Access Methods'**
Part No. A48506-1
Primary Author: Rita Moran
October 31, 1996

ORACLE Online Dokumentation 7.3.3 **‚Oracle 7 Server Tuning: Parallel Query Concepts'**
Part No. A48506-1

Primary Author: Rita Moran
October 31, 1996

ORACLE Online Dokumentation 7.3.3 , Oracle 7 Server Tuning: Parallel Query Tuning‘
Part No. A48506-1
Primary Author: Rita Moran
October 31, 1996

ORACLE Online Dokumentation 7.3.3 , Oracle 7 Server Tuning: Data Access Methods‘
Part No. A48506-1
Primary Author: Rita Moran
October 31, 1996

ORACLE Online Dokumentation 7.3.3 ‚Oracle 7 Server SQL Reference Manual: Operators, Functions, Expressions, Conditions‘
Part No. A32538-1
February 1996

ORACLE Online Dokumentation 7.3.3 ‚Oracle 7 Server Concepts Manual: Schema Objects‘
Part No. A32534-1
Contributing Authors: Steven Bobrowski, Cynthia Chin-Lee, Cindy Closkey, John Frazzini, Danny Sokolsky
February 1996

ORACLE Online Dokumentation 8.0 ‚Oracle 8 Concepts: Data Blocks, Extents, and Segments‘
Part No. A58227-01
Primary Author: Lefty Leverenz
December, 1997

ORACLE Online Dokumentation 8.0 , Oracle 8 Concepts: Built-In Datatypes‘
Part No. A58227-01
Primary Author: Lefty Leverenz
December, 1997

ORACLE Online Dokumentation 8.0 , Oracle 8 Concepts: Partitioned Tables and Indexes‘
Part No. A58227-01
Primary Author: Lefty Leverenz
December, 1997

ORACLE Online Dokumentation 8.0 , Oracle 8 Concepts: The Optimizer‘
Part No. A58227-01
Primary Author: Lefty Leverenz
December, 1997

ORACLE Online Dokumentation 8.0 , Oracle 8 Concepts: Parallel Execution‘
Part No. A58227-01
Primary Author: Lefty Leverenz

December, 1997

ORACLE Online Dokumentation 8.0 **‚Oracle 8 Administrators Guide: Space Estimations for Schema Objects‘**

Part No. A58397-01
Primary Author: Joyce Fee
December, 1997

ORACLE Online Dokumentation 8.0 **‚Oracle 8 Applications Developer Guide: PL/SQL Input/Output‘**

Part No. A58241-01
Primary Author: Denis Raphaely
December, 1997

ORACLE Online Dokumentation 8.0 **‚Oracle 8 Tuning: Data Access Methods‘**
Part No. A58246-01
Primary Author: Rita Moran
December, 1997

ORACLE Online Dokumentation 8.0 **‚Oracle 8 Tuning: Tuning Parallel Execution‘**
Part No. A58246-01
Primary Author: Rita Moran
December, 1997

ORACLE Online Dokumentation 8.0 **‚Oracle 8 Tuning: Understanding Parallel Execution Performance Issues‘**

Part No. A58246-01
Primary Author: Rita Moran
December, 1997

MicroStrategy Schulungsunterlagen **‚Certified Decision Support Engineer Program‘**

12 Stichwortverzeichnis