Lutz Schlesinger, Wolfgang Lehner, Wolfgang Hümmer, Andreas Bauer

Nutzung von Datenbankdiensten in Data-Warehouse-Anwendungen

Erstveröffentlichung in / First published in:

Information Technology. 2003, 45 (4), S. 203 – 210 [Zugriff am: 05.11.2020]. De Gruyter. ISSN 2196-7032

DOI: https://doi.org/10.1524/itit.45.4.203.22727

Nutzung von Datenbankdiensten in Data-Warehouse-Anwendungen

Connecting Data Warehouse Applications with Database Services

Lutz Schlesinger, Wolfgang Lehner, Wolfgang Hümmer, Andreas Bauer, Universität Erlangen-Nürnberg

Zusammenfassung Zentral für eine effiziente Analyse der in Data-Warehouse-Systemen gespeicherten Daten ist das Zusammenspiel zwischen Anwendung und Datenbanksystem. Der vorliegende Artikel klassifiziert und diskutiert unterschiedliche Wege, Data-Warehouse-Anwendungen mit dem Datenbanksystem zu koppeln, um komplexe OLAP-Szenarien zur Berechnung dem Datenbankdienst zu überlassen. Dabei werden vier unterschiedliche Kategorien, die Spracherweiterung (SQL), die anwendungsspezifische Sprachneuentwicklung (MDX), die Nutzung spezifischer Objektmodelle (JOLAP) und schließlich der Rückgriff auf XML-basierte WebServices (XCube) im einzelnen diskutiert und vergleichend gegenübergestellt. ►►►

Summary The connection of the applications and the underlying database system is crucial for performing analyses efficiently within a data warehouse system. This paper classifies and discusses different methods to bring data warehouse applications logically close to the underlying database system so that the computation of complex OLAP scenarios may be performed within the database system and not outside at the application. In detail, four different categories ranging from language extension (SQL) over the design of a new query language (MDX) and using special object models (JOLAP) to the use of XML-based WebServices are discussed and compared in detail.

KEYWORDS Data-Warehouse-System, SQL-OLAP-Erweiterung, MDX, JOLAP, WebServices & XML

1 Einleitung

Data-Warehouse-Systeme bilden eine Umgebung, in der Daten aus unterschiedlichen Datenquellen zusammengefasst und schließlich mit Blick auf bestimmte Anwendungszwecke ausgewertet werden. Zentral aus Sicht einer effizienten Auswertung ist die Kopplung der Anwendungen – von multidimensionalen Analysewerkzeugen über umfangreiche Statistikpakete bis hin zu spezifischen Planungssystemen, wie CRM- oder SCM-Systemen – mit den entsprechenden Datenbanksystemen.

Gegenstand dieses Beitrags ist die Untersuchung unterschiedlicher Arten von Kopplungsmöglichkeiten der Anwendungen mit den Datenbanksystemen. Nach einer Klassifikation und kurzen Beschreibung werden die Kopplungsarten einem Vergleich hinsichtlich unterschiedlicher Kriterien wie Komplexität der Benutzung, Flexibilität, Produktunabhängigkeit, usw. unterzogen.

Klassifikation

Die unterschiedlichen Möglichkeiten, einen Datenbankdienst aus einer Analyseapplikation heraus zu nutzen, lassen sich im Wesentlichen in vier Klassen einteilen.

- *Spracherweiterung*: Der Ansatz, eine klassische Anfragesprache um anwendungsspezifische Sprachkonstrukte zu erweitern, wird im zweiten Abschnitt exemplarisch an den OLAP-Erweiterungen für SQL verdeutlicht.
- *Sprachneuentwicklung*: Der Ansatz, eine effiziente Kopplung durch eine völlig neu entwickelte Sprache zu realisieren, wird in Kapitel 3 am Beispiel der „MultiDimensional eXpressions“ (MDX) von Microsoft erläutert.
- *Spezifisches Objektmodell*: Für eine Kopplung auf Ebene einer Programmiersprache werden speziell entwickelte Objektmodelle verwendet, die den darunter liegenden Datenbankdienst kapseln. Für den Bereich der multidimensionalen Analyse wird das Prinzip am Beispiel von JOLAP diskutiert.
- *Lose Kopplung auf Basis von XML*: Eine noch weiter gehende Unabhängigkeit von Applika-

tionssystem und Datenbankdienst wird durch eine Kopplung auf Basis von XML erzielt. Diese Kategorie von Kopplungsarten wird in Kapitel 5 aufgearbeitet.

Neben unterschiedlichen Anfragespezifikationsmöglichkeiten bedarf es einer Untersuchung der Kopplung von Anwendung und Datenbanksystem auf technischer Ebene. Entsprechend der vorausgehenden Klassifikation wird in Kapitel 6 die Anpassung an den Kontext der Data-Warehouse-Systeme in diesem Zusammenhang diskutiert.

Ziel der Betrachtungen ist es, einen Überblick über die unterschiedlichen Ansätze der Nutzung von Data-Warehouse-bezogenen Datenbankdiensten zu geben, die Vor- und Nachteile aufzuzeigen und so eine Entscheidung für oder gegen den Einsatz einer bestimmten Kopplungsart geben zu können.

2 Spracherweiterungen

Trotz der meist multidimensionalen Gedankenwelt für die interaktive Analyse im Rahmen von Data-Warehouse-Systemen werden häufig relationale Datenbanksysteme als zentrale Datenhaltungseinheit eingesetzt. Ein Zugriff auf relational abgelegte Daten ist dabei ausschließlich durch die Formulierung einer Anfrage in SQL möglich. Selbst in SQL99 [14] ist jedoch nur eine sehr einfache Gruppierungs- und Aggregationsfunktionalität verfügbar, sodass komplexe OLAP-Anfragen in eine Vielzahl einzelner und isoliert voneinander ablaufender Anfragen aufgespaltet und die korrespondierenden Ergebnisse wieder aufwändig zusammengesetzt werden müssen.

Spezifische Erweiterungen kamen historisch in zwei Stufen hinzu, wobei die Erweiterungen dabei aus relationaler Sicht zwei unterschiedliche Konzepte adressieren: Die Familie der erweiterten Gruppierungsoperatoren und attributbasierte Sequenzoperatoren [15].

2.1 Gruppierungsoperatoren

Die erweiterten Gruppierungsoperatoren finden in der GROUP-BY-Klausel einer SQL-Anfrage Anwendung und ermöglichen das simultane Gruppieren nach unterschiedlichen Gruppierungskombinationen innerhalb einer einzelnen Anfrage. Zentral ist dabei das Konstrukt des CUBE-Operators [3], der für eine vorgegebene Menge von Gruppierungsattributen alle möglichen Gruppierungskombinationen erzeugt. Bild 1 visualisiert das Ergebnis einer Anfrage an einen dreidimensionalen Datenwürfel in Form einer statistischen Kreuztabelle.

Der CUBE-Operator ist dabei als Kurzform für die explizite Aufzählung aller möglichen Gruppierungskombinationen zu sehen. Eine Aufzählung erfolgt innerhalb einer GROUPING SETS-Klausel. Das Beispiel in Bild 2 gruppiert den Datenbestand innerhalb einer Anfrage sowohl nach der Produktfamilie als auch nach Region und den jeweiligen Kombinationen.

Relational werden Teil- und Gesamtsummen durch NULL-Werte in den Ausprägungen der für die aktuelle Teilsumme nicht benötigten Gruppierungsattribute ausgedrückt. Die GROUPING()-Funktion erlaubt darüber hinaus, derartige systemgenerierte NULL-Werte von benutzerdefinierten NULL-Werten zu unterscheiden (Bild 3).

Als weiteres Beispiel einer Spracherweiterung sei der ROLLUP-Operator angeführt, der insbesondere in hierarchischen Data-Warehouse-Systemen Anwendung findet und dessen Semantik durch folgendes Beispiel illustriert sei, wobei die als Parameter dienenden Attribute eine funktionale Abhängigkeit aufweisen müssen (z. B. a → b):

```
ROLLUP(b,a) =
    GROUPING SETS ((b,a),(b),())
```

Verkäufe		Unterhaltungselektronik				…
		Video	Audio	TV	SUMME	…
2001	Bayern	12	31	15	58	…
	Sachsen	22	51	17	…	…
	SUMME	34	…	…	…	…
2002	Bayern	48	67	55	170	…
	Sachsen	50	…	…	…	…
	SUMME	98	…	…	…	…
SUMME		257	…	…	904	…

Bild 1 Beispiel eines drei-dimensionalen Datenwürfels als statistische Kreuztabelle.

```
SELECT Produktfamilie, Region,
       GROUPING(Produktfamilie),
       GROUPING(Region), SUM(Verkäufe)
  FROM   …
  WHERE  …
GROUP BY GROUPING SETS(
  (Produktfamilie, Region),
  (Produktfamilie), (Region))
```

Bild 2 Beispiel einer GROUPING SET-Klausel.

Produkt familie	Region	GROUPING (Produkt-familie)	GROUPING (Region)	SUM (Verkäufe)
Video	Bayern	0	0	12
Video	Sachsen	0	0	22
Audio	Bayern	0	0	31
Audio	Sachsen	0	0	51
TV	Bayern	0	0	15
TV	Sachsen	0	0	17
Video	NULL	0	1	34
Audio	NULL	0	1	82
TV	NULL	0	1	32
NULL	Bayern	1	0	58
NULL	Sachsen	1	0	90

Bild 3 Relationale Darstellung komplexer Gruppierungsergebnisse.

2.2 Attributbasierte Sequenzoperatoren

Neben den skizzierten SQL-Erweiterungen, die die Problematik der Gruppierungsmöglichkeit beheben, stellt sich für Analysen ferner das Problem, dass das relationale Datenmodell keine inhärente Ordnung der Daten aufweist, obwohl Sequenzanalysen, wie z. B. TOP(n), als sehr wichtig einzustufen sind. In SQL2003 [15] ist dafür ein dreistufiges Konzept in der SELECT-Klausel als Erweiterung der Aggregationsfunktionen vorgesehen. Die erste Stufe ist durch die Einführung einer lokalen Ordnung mittels der neuen Aggregationsfunktion RANK() gegeben, die die Position bzgl. der aktuellen Sortierung angibt. Beispiel:

```
SELECT a, b,
   RANK() OVER (ORDER BY x),
   RANK() OVER (ORDER BY y)
FROM ...
```

Angemerkt sei, dass RANK keine Verdichtung der Daten vornimmt, d. h. dass für jedes eingehende Tupel ein Ausgangstupel produziert wird. Ferner ist die Sortierung nicht global, sondern nur lokal pro Attribut gültig.

Ein zweites Konzept stellt die Bildung von Partitionen mittels der PARTITION-Klausel dar, wie folgendes Beispiel zeigt:

```
SELECT a, b,
   RANK() OVER
   (PARTITION BY b ORDER BY x)
FROM ...
```

In dem angeführten Beispiel wird mit jedem Wechsel von Werten im Attribut b die Rangfolge zurückgesetzt, z. B. Rang pro Monat.

Schließlich stellt die Erweiterung durch Fensterdefinitionen ein drittes Konzept dar, das beispielsweise benötigt wird, um den gleitenden Durchschnitt zu ermitteln:

```
SELECT a,b,
   AVG(y) OVER
   (ORDER BY x ROWS
   BETWEEN 1 PRECEDING
   AND 1 FOLLOWING)
FROM ...
```

Im angeführten Beispiel werden die Tupel im Eingangsdatenstrom bei der Durchschnittsberechnung berücksichtigt, die sich direkt vor und nach dem aktuellen Tupel befinden.

Fazit

Zusammenfassend kann festgehalten werden, dass die Sprache SQL deutliche Erweiterungen zur Unterstützung von Data-Warehouse-Anwendungen erfahren hat bzw. erfahren wird. Durch Nutzung dieser Sprachkonstrukte können komplexe OLAP-Szenarien mit einer einzelnen Anfrage spezifiziert und von der Optimierungskomponente des Datenbanksystems berücksichtigt werden.

3 Anwendungsspezifische Sprachneuentwicklung

Nicht dem evolutionären, sondern dem revolutionären Ansatz folgend, steht die Entwicklung einer anwendungsspezifischen Sprache für die multidimensionale Datenanalyse. Als Beispiel wird an dieser Stelle auf das mittlerweile weit verbreitete MDX (*Multi-Dimensional eXpressions*) [12] von Microsoft eingegangen. MDX ermöglicht die Definition und Manipulation aller Objekte des multidimensionalen Datenmodells. Die Anfrage in Bild 4 erzeugt beispielhaft Teile der statistischen Kreuztabelle aus Bild 1.

Die Schlüsselwörter ON COLUMNS und ON ROWS spannen die Achsen des Ergebniswürfels auf. DESCENDANTS liefert die Nachfolger eines Knotens auf einer vorgegebenen Hierarchieebene, welche im laufenden Beispiel die Nachfolger von Unterhaltungselektronik auf Ebene der Produktfamilie sind. Die Bildung des Kreuzprodukts von Knoten verschiedener Dimensionen leitet das Schlüsselwort CROSSJOIN ein. Schließlich wird in diesem Zusammenhang MEMBERS benötigt, um alle Knoten der Hierarchiestufe zu liefern, was im Beispiel alle Jahreszahlen bzw. Regionen sind. Die WHERE-Klausel dient zur Definition von Scheiben des Datenwürfels. Die Restriktion auf der Dimension Measure entspricht dabei der Auswahl von auszuwertenden Kennzahlen.

Weitere sprachliche Möglichkeiten werden an dieser Stelle nur angedeutet; für eine vertiefende Darstellung sei auf [12] verwiesen:

- Hierarchische Navigation mittels PREVMEMBER, NEXTMEMBER, CURRENTMEMBER, PARENT, usw.
- Arithmetische Verknüpfung von Berechnungsergebnissen (*calculated members*), bspw. durch zusätzliche Kennzahlen ohne weiteren Speicherbedarf
- Generierung von Zeitserien mittels X-To-Date-Funktionen und Verwendung von PARALLELPERIOD, OPENINGPERIOD sowie CLOSINGPERIOD zur Adressierung eines Vergleichszeitraums unter Angabe einer Hierarchiestufe
- Top(n)-Anfragen mittels HEAD, TAIL, TOPCOUNT oder BOTTOMCOUNT.

Fazit

Die speziell für den Anwendungsbereich entwickelte Sprache MDX ist in ihrer Struktur (absichtlicherweise) SQL sehr ähnlich, zeichnet sich jedoch im Detail durch hohe Mächtigkeit zur Adressierung multidimensionaler Strukturen aus. Dies resultiert in der kompakten Formulierung möglicherweise sehr komplexer Anfragen.

```
SELECT {[Produkt].[Produktkategorie].
         Unterhaltungselektronik,
         DESCENDANTS(
           [Produkt].[Produktkategorie].
            Unterhaltungselektronik,
           [Produktfamilie])} ON COLUMNS,
         CROSSJOIN([Zeit].[Jahr].MEMBERS,
         [Geographie].[Region].MEMBERS) ON ROWS
 FROM Verkäufe
 WHERE (Measure.Verkaeufe, Zeit.[Alle Zeiten],
       Produkt.[Alle Produkte])
```

Bild 4 Anfrage zur Erzeugung der statistischen Kreuztabelle.

4 Anwendungsspezifische Objektmodelle

Neben der textuellen Formulierung einer Anfrage ist die Programmierung auf Basis eines anwendungsspezifischen Objektmodells möglich. Da momentan noch kein standardisiertes Objektmodell vorliegt, finden sich in den APIs (*Application Programming Interfaces*) kommerzieller Hersteller unterschiedlichste Objektmodelle wieder, wie zum Beispiel „MS Objects“ von MicroStrategy [11] oder „OLAP Services“ von Microsoft [10]. Erst in jüngster Zeit wird mit JOLAP [7] der Versuch unternommen, einen herstellerübergreifenden Standard zu definieren. So ist JOLAP eine sich in der Entwicklung befindende Java Schnittstelle zu OLAP für Java2EE-Applikationen, die im Rahmen des Java Community Process herstellerübergreifend als Standard vorgesehen, aber noch nicht verabschiedet ist. Die Zielsetzung liegt darin, den Zugriff auf die im *Common Warehouse Metamodel* (CWM) [2] definierten multidimensionalen Strukturen zu ermöglichen.

Die Architektur von JOLAP (Bild 5) enthält folgende wesentliche Komponenten:

- *JMI (Java Metadata Interface) Reflective Services*: Ansammlung von generischen Schnittstellen, die das Auffinden von Metadaten mittels Introspektion erleichtern. Über Introspektion ist es möglich, zur Laufzeit den Aufbau einer anderen Applikation zu untersuchen und sie anzusteuern.
- *JOLAP Metadata Model*: Teilmenge von CWM erweitert um für JOLAP notwendige Zusätze mit Unterscheidung in eine Client- und eine Server-Rolle.
- *JOLAP Query Model*: Schnittstelle zum Umgang mit OLAP-Anfragen. Im Gegensatz zum Metamodell basiert es nicht auf CWM oder JMI, sondern wurde direkt an die Bedürfnisse von OLAP-Anwendern angepasst.
- *JOLAP Source Model*: optionale, zweite Anfrageschnittstelle, die aus kleinen, präzisen Funktionsblöcken besteht, um den unterschiedlichen Implementierungen der Systeme Rechnung zu tragen und eher für den erfahrenen JOLAP-Anwender geeignet ist.
- *JOLAP Cursor Model*: Schnittstelle zum Umgang mit den multidimensionalen Ergebnissen einer OLAP-Anfrage. Insbesondere ermöglicht es die Navigation durch die Ergebnismenge.

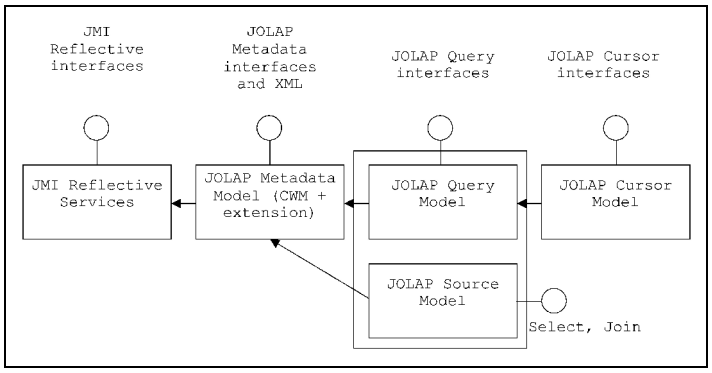


Bild 5 Architektur von JOLAP.

Das OLAP API [13] von Oracle, welches als Beispiel in diesem Vergleich dient, basiert auf JOLAP und ist ein Teil der von Oracle propagierten OLAP Services (Bild 6). Die *Oracle Business Intelligence Beans* (BI Beans) stellen zusätzlich eine Menge von Komponenten für die Entwicklung internetbasierter Anwendungen dar und greifen durch Nutzung der OLAP Services auf die Oracle Datenbank zu, wobei die OLAP Services aber auch direkt in der entsprechenden Applikation nutzbar sind.

Das OLAP API ist kompatibel mit dem JOLAP Standard ergänzt um Datenbankspezifika wie beispielsweise die Möglichkeit zur Anfrageformulierung, die Ergebnisverwaltung oder dem Transaktionsmechanismus, durch den Lesen und Schreiben auf den Daten ermöglicht wird. Das Grundgerüst im Anwendungsprogramm mit conn als JDBC-Verbindung zur gewählten Datenquelle ist im im Bild 7 gezeigten Code-Fragment angeführt.

Innerhalb der Ellipse im obigen Codefragment können die im Data-Warehouse-System vorliegenden Daten analysiert und verändert werden. Ein gekürztes Beispiel für das Auslesen der Metadaten zeigt der in Bild 8 gezeigte Abschnitt.

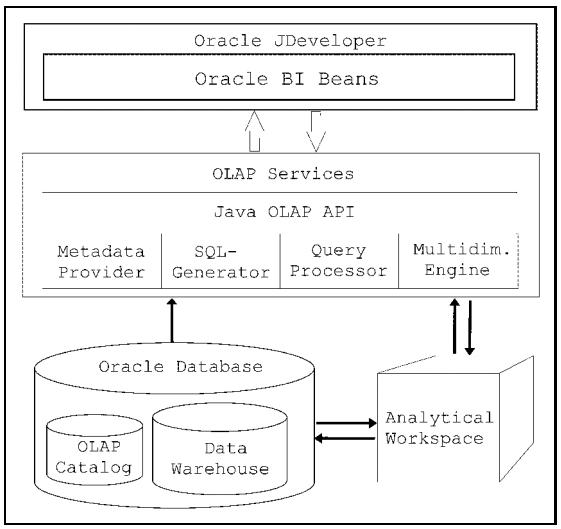


Bild 6 OLAP Services innerhalb der Oracle Architektur.

Bild 7 Grundgerüst im Anwendungsprogramm mit conn als JDBC-Verbindung zur gewählten Datenquelle.

```
// Transaktions- und Datenmanager
transProvider = new ExpressTransactionProvider();
dataProvider = new ExpressDataProvider(conn,
                   transProvider);
dataProvider.initialize();
// Stellen von Anfragen und
// Verarbeiten der Ergebnisse
…
// Schließen der Datenbankverbindung
dataProvider.close();
```

Bild 8 Abfragen der Metadaten.

```
// Beispiel: Abfrage der Metadaten
metaDataProvider =
  dataProvider.getDefaultMetadataProvider();
rootSchema = mp.getRootSchema();
MeasureDim = rs.getMeasureDimension();
```

Bild 9 Formulierung einer Abfrage.

```
// Beispiel: Formulierung einer Anfrage
prodSel = productDim.selectValue (new String[]
          {"Video", "Audio", "TV"});
timeSel = timeDim.selectValue (new String[]
          {"2001", "2002"});
geoSel  = geographyDim.selectValue(new
          String[]{"Bayern", "Sachsen"});
result  = facts.join(prodSel).join(timeSel).
          join(geoSel).total();
// Beispiel: Verarbeitung der Ergebnisdaten
transProvider.prepareCurrentTransation();
transProvider.commitCurrentTransation();
cursorMgrSpec =
  dataProvider.createCursorManagerSpecification
  (result);
CursorManager cursorMgr =
  dp.createCursorManager (cursorMgrSpec);
Cursor queryCursor = cursorMgr.createCursor();
do {
    … /// Ergebnis verarbeiten ///
} while (queryCursor.next());
cursorMgr.close();
```

Für die Formulierung von Anfragen wird auf das JOLAP Source Model zurückgegriffen, das die Spezifikation einer Anfrage erlaubt, selbst aber keine Ergebnisdaten enthält. Für die Verarbeitung der Ergebnisdaten ist die Instanziierung eines Cursors entsprechend dem JOLAP Cursor Model notwendig, wobei zuvor die laufende Transaktion geschlossen werden muss (Bild 9).

Fazit

Die Kapselung eines Datenbankdienstes durch Nutzung eines Objektmodells hat auf der einen Seite den wesentlichen Vorteil, dass eine hohe Ausdrucksmächtigkeit erzielt wird. Auf der anderen Seite steht jedoch die Komplexität des Objektmodells, welches eine aufwändige Einarbeitung für den Applikationsentwickler erfordert.

5 Lose Kopplung auf Basis von XML

Um nicht nur über Analysetools oder APIs auf Data-Warehouse-Daten zugreifen zu können, sondern um auch die multidimensional strukturierten Daten zwischen Data Warehouses effizient zwischen Systemen verschiedenster Hersteller über ein Netzwerk austauschen zu können, wird ein herstellerübergreifendes Datenformat benötigt. Das in der Entwicklung befindliche XCube [4; 5] stellt dazu einen auf XML basierenden Versuch dar. In der statischen Variante werden XCubes auf dem Web-Server bereitgestellt und können nach dem Herunterladen in ein eigenes Data Warehouse über die XCube-Schnittstelle importiert werden. Eine Dynamisierung und Flexibilisierung dieses Ansatzes wird durch Formulierung von Anfragen an das Data Warehouse erreicht, indem Anfrage und Ergebnis im XCube-Format übertragen werden.

Das XML-basierte XCube-Format beschreibt Datenwürfel und besteht aus mehreren Teilformaten. So beschreibt XCubeSchema das multidimensionale Schema eines Würfels unter Angabe der aufspannenden Dimensionen und der enthaltenen Kennzahlen (Bild 10).

XCubeDimension beschreibt die Struktur der Dimensionen mit den Klassifikationsschemata der Dimensionen und deren Alternativpfaden sowie den zugehörigen Klassifikationshierarchien auf Instanzebene (Bild 11).

XCubeFact enthält die Würfelzellen mit den Faktdaten, deren multidimensionale Anordnung durch XCubeSchema und XCubeDimension beschrieben ist (Bild 12).

Obwohl XCube ein reines Datenaustauschformat ist, sind Erweiterungen für die Abfrage einer Datenquelle angedacht. So beschreibt XCubeFunction den Leistungsumfang einer Datenquelle, ob beispielsweise nur fest vorgegebene XCube-Würfel heruntergeladen werden können oder die Quelle auch einen Anfragemechanismus mittels XCubeQuery unterstützt. XCubeQuery selbst ist eine XML-basierte multidimensionale Anfragesprache, mittels derer XCubes, beschrieben durch XCubeSchema und XCubeDimension, abgefragt werden können, Teilmengen herausgeschnitten oder weiter aggregiert werden können. Das Ergebnis ist wiederum ein XCube.

Fazit

Für das herstellerübergreifende XCube-Format lässt sich zusammenfassend festhalten, dass ein umfassendes Datenaustauschformat geschaffen ist, dem jedoch noch die Authentizitätsprüfung bzw. Signierung der XCubes zur Verhinderung von Datenmanipulationen sowie ein Zugriffsschutzmechanismus fehlt. Für letzteres ist XCubeAccess bereits in Planung. Ebenso können die

Bild 10 Beispiel einer XCube-Schema Beschreibung.

```
<cubeSchema id="Umsätze">
  <fact id="Umsatz"
        dataType="xs:positiveInteger">
    <defaultAggregate>
      <aggregation operator="sum"/>
    </defaultAggregate>
  </fact>
  <dimension id="Zeit" granularity="Jahr"/>
  <dimension id="Geographie"
   granularity="store"/>
  <dimension id="Produkt"
   granularity="Produktgruppe">
  </dimension>
</cubeSchema>

<classSchema>
  ...
  <classLevel id="Bundesland">
     <rollUp toLevel="Land"/>
  </classLevel>
  <classLevel id="Land"/>
  <classLevel id="Produktgruppe">
    <rollUp toLevel=" Produktfamilie"/>
  </classLevel>
  <classLevel id="Produktfamilie">
    <rollUp toLevel=" Produktkategorie"/>
  </classLevel>
  <classLevel id="Region">
   <rollUp toLevel="Produktkategorie"/>
  </classLevel>
</classSchema>
```

Bild 11 Beispiel einer XCube-Dimension Beschreibung.

```
...
<classification>
  <!-- dimension: Geographie -->
  <level id="Land">
    <node id="de">
    <text>Deutschland</text >
    <attribute id="inhabitants"
     value="82022000"/>
    </node>
    <node id="ch">
    <text>Schweiz</text>
    </node>
  </level>
  <level id="Bundesland">
    <node id="BY">
    <text>Bayern</text >
    <rollUp toNode="de" level="Land"/>
    </node>
    <node id="HE">
    <text>Sachsen</text >
    <rollUp toNode="de" level=" Land "/>
    </node>
        ...
  </level>
  ...
</classification>
```

Bild 12 Beispiel einer XCube-Fact Beschreibung.

```
...
<cube id="Umsätze">
  <cell>
    <dimension id="Geographie" node="Bayern"/>
    <dimension id="Zeit" node="2002"/>
    <dimension id="Produkt" node="Video"/>
    <fact id="Umsatz" value="12"/>
  </cell>
  ...
</cube>
...
```

erzeugten XML Dateien sehr groß werden, weshalb die Anwendung von Kompressionsverfahren zur Reduktion der übertragenden Daten anzuraten ist.

6 Systemtechnische Kopplung

Bei der Nutzung eines Datenbankdienstes muss konzeptionell zwischen der Formulierung der Anfrage und dem damit verbundenen Transport der Anfrage bzw. Ergebnisdaten zwischen Anwendung und Datenbanksystem unterschieden werden. Die vorangegangenen Untersuchungen beziehen sich dabei im Wesentlichen auf die Formulierung der Anfrage, sodass der Aspekt der Kopplung auf technischer Ebene zu diskutieren bleibt. Um Vergleichbarkeit realisieren zu können, ist wiederum eine Betrachtung gemäß der bereits durchgeführten Klassifikation vorzunehmen.

- *Auf Ebene von SQL*: Für den Bereich der statistischen Analyse von Data-Warehouse-Datenbanken wurde keine Erweiterung der Transportfunktionalität vorgenommen. Im Wesentlichen kommt das SQL-standardisierte *Call Level Interface* (CLI) [16] zum Einsatz, wobei zwei orthogonal zueinander stehende Besonderheiten zu vermerken sind: Bei einer Priorisierung der Portabilität kommt das als Quasi-Standard bekannte ODBC zum Einsatz, welches eine Teilmenge der Funktionalität des CLI realisiert und mittlerweile auch in der Unix-Umgebung etabliert ist. Wird dagegen der Fokus auf die Performanz gelegt, so kommen herstellerspezifische Erweiterung des CLI – z. B. das *Oracle Call Interface* (OCI) [12] – zum Einsatz, wodurch systemspezifische Optimierungen genutzt werden können.
- *Auf Ebene von MDX*: Als technische Grundlage für MDX kommt auf Ebene der programmiersprachlichen Kopplung das Microsoft spezifische und für

den Einsatz im OLAP-Bereich angepasste OLE DB [12] zum Einsatz. OLE DB kann dabei als höherwertige Schnittstelle über dem Zugriff auf Basis von ODBC gesehen werden.

- *Auf Ebene von JOLAP*: Wie bereits im Beispiel zu JOLAP gezeigt, werden analog zum SQL-Ansatz klassische Transportprotokolle wie JDBC zur Kommunikation zwischen Anwendung und Datenbankdienst verwendet.
- *Auf Ebene von XML*: Um Datenbankdienste in Form von WebServices zu realisieren, erfolgt erneut ein Rückgriff auf etablierte Basisdienste wie SOAP, wiederum basierend auf bspw. HTTP und XML. Bedingt durch die lose Kopplung ist insbesondere der Aspekt der Metadatenabfrage zu berücksichtigen. Im Fall von XCube ist dieser Aspekt in der Spezifikation selbst enthalten. Im Gegensatz dazu realisiert der von Microsoft vorgeschlagene Ansatz *XML for Analysis* (XML4A) [18] eine zusätzliche Transportschicht, um in MDX spezifizierte Anfragen durch XML zu kapseln (mdXML), um somit MDX als Zugriff und WebServices als Transportmedium etablieren zu können. Den Kern von XML4A stellen die beiden Befehle DISCOVER und EXECUTE dar. Mittels DISCOVER kann die Anwendung sich über verfügbare Datenquellen, Würfelschemata oder Aufbau einer Dimension informieren. Eine in XML gekapselte Anfrage wird über den EXECUTE-Befehl an den Datenbankdienst übermittelt.

Fazit

Während bei der Formulierung von Anfragen umfangreiche Erweiterungen für den Anwendungskontext der Data-Warehouse-Systeme realisiert worden sind, finden sich auf Ebene der Transportmechanismen nur moderate Erweiterungen bzw. Anpassungen. Alleine im Fall der losen Kopplung auf Basis von WebServices ist mit XML4A eine Ergänzung vorgeschlagen worden, die jedoch insbesondere bedingt durch die feste Verknüpfung mit MDX momentan nur geringe Akzeptanz findet.

7 Zusammenfassender Vergleich

Der vorliegende Beitrag gibt einen Überblick über Möglichkeiten, von einem Anwendungssystem auf die Daten eines Data Warehouse zuzugreifen. Nach der Skizzierung der Spracherweiterungen in SQL wird anschließend die Sprache MDX als Beispiel für eine Sprachneuentwicklung vorgestellt, die eine von Microsoft getragene Kombination aus SQL mit dem multidimensionalen Datenmodell darstellt. Beide Ansätze greifen direkt auf die Data-Warehouse-Daten zu, wodurch eine gute Performanz gegeben ist, jedoch wird dieser Vorteil durch die Abhängigkeit vom darunter liegenden Datenbanksystem erkauft. Die Unabhängigkeit sowie eine bessere Einbettung in höhere Programmiersprachen ermöglichen Objektmodelle, was am Beispiel von Oracle OLAP Services demonstriert wurde. Der anschließend vorgestellte Ansatz XCube ermöglicht eine vollständige lose Kopplung genauso wie XML for Analysis, wobei gerade dieser durch die ausschließliche Unterstützung von Microsoft durchaus von JOLAP überholt werden könnte. Sowohl JOLAP als auch MDX sind konzeptionell orthogonal zu SQL und können daher aus den SQL-Erweiterungen profitieren.

Literatur

[1] *A. Bauer* und *H. Günzel*: Data-Warehouse-Systeme: Architektur, Entwicklung und Anwendung. dpunkt.verlag, Heidelberg, 2001.

[2] *o. V.*: Common Warehouse Metamodel (CWM) Specification: Version 1.0. Object Management Group Inc., 2001. (http://www.omg.org/technology/cwm)

[3] *J. Gray, A. Bosworth, A. Layman* und *H. Pirahesh*: Data Cube: A Relational Aggregation Operator Generalizing Group-By, Cross-Tab, and Sub-Total. In: Proceedings of the 12th International Conference on Data Engineering (ICDE'96, New Orleans (LA), USA, 26.2.–1.3.), 1996, S. 152–159.

[4] *G. Harde:* XCube: Konzepte für eine XML-basierte Beschreibung von Datenwürfeln zur Realisierung eines föderativen Data-Warehouse-Netzwerkes. In: Tagungs-band 5. Workshop Föderierte Datenbanksysteme (FDBS'01, Berlin, 11.–12.10.), 2001, S. 64–77.

[5] *G. Harde:* XCube-Website. (http://www.xcube-open.org)

[6] *W. Inmon*: Building the Data Warehouse. John Wiley & Sons, Inc., New York et. al., 1996.

[7] *o. V.*: Java OLAP Interface (JOLAP), Version 0.85, Java Community Process, 2002. (http://www.jcp.org/en/jsr/detail?id=69)

[8] *R. Kimball*: The Data Warehouse Toolkit: Practical Techniques for Building Dimensional Data Warehouses. John Wiley & Sons, Inc., New York et. al., 1996.

[9] *W. Lehner*: Datenbanktechnologie für Data-Warehouse-Systeme. dpunkt.verlag, Heidelberg, 2003.

[10] *o. V.*: OLAP Services: Overview of Microsoft SQL Server 7.0 OLAP Services. In: MSDN Library, Microsoft Corp., 1999. (http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnolap/html/olapover.asp)

[11] *o. V.*: Microstrategy-Website. (http://www.microstrategy.com/)

[12] *o. V.*: Microsoft OLE DB, Version 2.7. In: MSDN Library, Microsoft Corp., 2001. (http://msdn.microsoft.com/library/default.asp?url=/library/en-us/oledb/htm/dasdkoledboverview.asp)

[13] *o. V.*: Oracle Call Interface Programmer's Guide, Release 2 (9.2), Oracle Corporation, 2002.

[14] *o. V.*: Oracle9i OLAP: Developers Guide to the OLAP API, Releas 2 (9.2), Oracle Corporation, 2002.

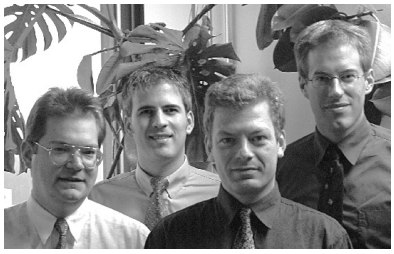
[15] *o. V.*: ISO Final Committee Draft (FCD) Database Language SQL – Part 1: SQL/Framework, 1999.

[16] *o. V.*: ISO Final Committee Draft (FCD) Database Language SQL – Part 1: SQL/Framework, 2002.

[17] *o. V.*: ISO Final Committee Draft (FCD) Database Language SQL – Part 3: Call Level Interface, 2002.

[18] *o. V.*: XML Metadata Interchange (XMI) Specification. Version 1.2. Object Management Group, 2002. (http://cgi.omg.org/docs/formal/02–01–01.pdf)

[19] *o. V.*: XML for Analysis Specification: Version 1.0. Microsoft Corp, Hyperion Solutions Corp., 2001. (http://www.xmla.org/docs_pub.asp)



(Abbildung von links nach rechts.)

Dipl.-Inf. Wolfgang Hümmer, Dipl.-Inf. Andreas Bauer, Prof. Dr.-Ing. Wolfgang Lehner, Dipl.-Inf. Lutz Schlesinger studierten an der Universität Erlangen-Nürnberg Informatik und sind Berater in nationalen und internationalen Data-Warehouse-Projekten. Wolfgang Lehner ist seit Oktober 2002 Inhaber des Lehrstuhls für Datenbanken an der TU Dresden. Die drei anderen Autoren sind wissenschaftliche Mitarbeiter am Lehrstuhl für Datenbanksysteme der Universität Erlangen-Nürnberg.
Adresse:
1) Lehrstuhl für Datenbanksysteme, Universität Erlangen-Nürnberg, D-91058 Erlangen, Tel.: +49 (91319) 85–27893, Fax: +49 (9131) 85–28854, E-Mail: bi@immd6.informatik.uni-erlangen.de
2) Arbeitsgruppe Datenbanken, Technische Universität Dresden, D-01307 Dresden, Tel.: +49 (351) 463–38383, Fax: +49 (351) 463–38259, E-Mail: lehner@inf.tu-dresden.de