Anfragesprachen für On-Line Analytical Processing (OLAP)

Seminar: *Datenbanken und Informationssysteme* im Sommersemester 2003

René Rondot
Betreuer: Jernej Kovse

27. Juni 2003

Inhaltsverzeichnis

1 Einleitung

Datenbanken dienen heutzutage längst nicht mehr nur dazu, Daten zu speichern und wieder abrufbar zu machen. Vielmehr werden im Rahmen des On-Line Analytical Processing (OLAP) die vorhandenen Daten dazu genutzt, betriebswirtschaftliche Entscheidungsprozesse zu unterstützen. Dazu müssen häufig aus den vorhandenen Daten analytisch neue Informationen gewonnen werden. Dies geschieht durch Zusammenfassung von Daten und Darstellung in mehrdimensionalen Strukturen. Um mit diesen Strukturen arbeiten zu können, benötigt man jedoch Möglichkeiten, die über die traditionelle Relationenalgebra und deren standardisierte Anfragesprache SQL hinausgegehen. Zu diesem Zweck sind eine Reihe von neuen Anfragesprachen entwickelt worden, die den Umgang mit mehrdimensionalen Strukturen unterstützen und zudem häufig viele weitere für das OLAP nützliche Operationen bieten. Eine gute Einführung in OLAP und die dort verwendeten Operationen findet sich in der Arbeit von Burkhard Schäfer [Sch03].

In den folgenden Abschnitten werden vier dieser Konzepte vorgestellt, an Beispielen erläutert und hinsichtlich ihrer Mächtigkeit untersucht. Den Anfang macht in Abschnitt 2 der von Gray et al. [GCB+97] entwickelte Data-Cube-Operator, der Grundlage vieler neuerer Konzepte ist. Einen sehr mächtigen und komplexen Ansatz entwickelten Hacid et al. [HMR97] mit einer Regel-basierten Sprache, die in Abschnitt 3 beschrieben wird. Diese ermöglicht Operationen auf mehrdimensionalen Strukturen ebenso wie Aggregationen in verschiedenen Aggregationsebenen. Die Firma Microsoft entwickelte einen weiteren Ansatz, die Multidimensional Expressions (MDX), die vor allem auf mehrdimensionale Strukturen ausgelegt und bereits in vielen kommerziell verfügbaren System verwendbar sind. Gingras und Lakshmanan [GL98] entwickelten eine Sprache, nD-SQL, die ebenfalls auf mehrdimensionalen Strukturen arbeitet, aber zusätzlich Möglichkeiten bereitstellt, mehrere Datenbanken mit heterogenen Schemata miteinander zu verknüpfen und OLAP-Operationen darauf auszuführen.

2 Der Data Cube Operator

2.1 Notwendigkeit des Data Cube Operator

Beim OLAP fallen für gewöhnlich große Datenmengen an, die zu analysieren sind. Diese müssen in der Regel gefiltert, aggregiert und anschließend analysiert werden. Dies sind gewöhnliche Vorgänge, deren Formulierung bereits im SQL-Standard[1] enthalten ist und in vielen Datenbankanwendungen genutzt wird. Der SQL-Standard enthält die Aggregations-Funktionen `COUNT`, `SUM`, `MIN`, `MAX` und `AVG` und den `GROUP BY`-Operator zur Gruppierung der Ergebnisse. Zu den Standard-Operationen im OLAP gehören jedoch Histogramme, Kreuztabellen und Roll-Ups, welche sich alle im allgemeinen Fall nur sehr schwer mit den Standard-SQL-Operatoren beschreiben lassen. Wie sich diese Schwierigkeiten vermeiden lassen, wurde 1997 von Gray et al. [GCB+97] vorgestellt. Dieser Artikel ist die Grundlage der folgenden Erläuterungen.

Bei Histogrammen (Aggregationen über berechneten Kategorien) ist die Problematik darin begründet, dass es im SQL-Standard nicht vorgesehen ist, über berechnete Attribute zu gruppieren. Hat man beispielsweise ein Attribut *Datum* und eine Funktion *Jahr(Datum)*, die ein Datum auf das zugehörige Jahr abbildet und möchte seine Daten nach Jahren gruppieren, so erlaubt der SQL-Standard einen Ausdruck wie `GROUP BY Jahr(Datum)` nicht.

[1] Ist in den folgenden Abschnitten vom *SQL-Standard* die Rede, so ist bis Abschnitt 2.7 stets der SQL:1992-Standard (auch als SQL-2 bekannt) gemeint.

Dieses Problem lässt sich jedoch durch eine relativ einfache Erweiterung des SQL-Standards beheben, indem man eine solche Gruppierung zulässt. Tatsächlich bieten viele verfügbare Datenbanksysteme diese erweiterte Funktionalität bereits an. Die dazu notwendige Syntax-Erweiterung wird in Abschnitt 2.5 erläutert.

Schwerwiegender ist das Problem, dass sich mit den Operatoren des SQL-Standards nur sehr umständlich eine Kreuztabelle oder ein Roll-Up realisieren lässt. Diese wichtigen Operationen, mit denen sich Unregelmäßigkeiten und Strukturen in großen Datenmengen erkennen und darstellen lassen, können zwar mittels normaler SQL-Statements (inbesondere Aggregatfunktionen und der `GROUP BY`-Operator) ebenfalls durchgeführt werden, dies ist jedoch überaus umständlich und kann in vielen Fällen zu exponentiell wachsenden Statementlängen führen. Bei automatisch erzeugten Statements ist dies zwar möglicherweise akzeptabel, allerdings führen solche komplexen Statements oft auch zu sehr hohen Laufzeiten, da der Anfragen-Optimierer diese nicht mehr ideal optimieren kann. Es ist daher sehr nützlich, einen Operator zur Verfügung zu haben, der diese Funktionen in einfacher und kompakter Weise ermöglicht – für den Mensch wie die Maschine lesbar.

Beim *Roll-Up* der Daten einer Relation werden Aggregats-Werte mit verschiedenen Detailiertheitsgraden berechnet und den ursprünglichen Daten hinzugefügt. Hat man beispielsweise eine Relation, in der Autoverkäufe gespeichert sind, könnte es interessant sein, zu ermitteln, wie oft bestimmte Fahrzeuge in einem bestimmten Zeitraum verkauft worden sind. Man stelle sich eine Relation mit den Attributen *Modell*, *Jahr*, *Farbe* und *Verkaeufe* vor. Möchte man nun einen Roll-Up über diese Attribute machen, so benötigt man folgendes SQL-Statement:

```
SELECT 'ALL', 'ALL', 'ALL', SUM(Verkaeufe)
FROM VerkaufTabelle

UNION
SELECT Modell, 'ALL', 'ALL', SUM(Verkaeufe)
FROM VerkaufTabelle
GROUP BY Modell

UNION
SELECT Modell, Jahr, 'ALL', SUM(Verkaeufe)
FROM VerkaufTabelle
GROUP BY Modell, Jahr

UNION
SELECT Modell, Jahr, Farbe, SUM(Verkaeufe)
FROM VerkaufTabelle
GROUP BY Modell, Jahr, Farbe;
```

Hier wird also auf der gröbsten Ebene zunächst eine Gesamtsumme aller Verkäufe berechnet. Als zweite Ebene wird diese dann aufgeschlüsselt nach den verschiedenen Modellen. Diese Aufschlüsselung wird erneut verfeinert, indem auch die Summen der einzelnen Jahre gebildet werden und zuletzt noch zusätzlich die Summen der einzelnen Farben.

Wie man erkennen kann, wächst die Zahl der benötigten `SELECT`-Statements hier noch linear mit der Anzahl der Ebenen, auf denen man aggregieren möchte (n Ebenen ergeben $n+1$ `SELECT`-Statements). Oft reichen diese Informationen jedoch nicht aus, da der Roll-Up asymmetrisch ist und zwar eine Aufschlüsselung nach Jahren, jedoch keine Informationen über die Verkaufsraten der Farben enthält. Möchte man auch noch diese fehlenden Kombinationen hinzunehmen, erhält man die symmetrische Aggregation, *Kreuztabelle* genannt.

Eine Kreuztabelle wird n-dimensional genannt, wenn n Attribute aggregiert werden. Um eine Kreuztabelle zu erhalten, müsste oben stehendes SQL-Statement um folgenden Teil erweitert werden:

```
UNION
SELECT 'ALL', Jahr, Farbe, SUM(Verkaeufe)
FROM VerkaufTabelle
GROUP BY Jahr, Farbe

UNION
SELECT Modell, 'ALL', Farbe, SUM(Verkaeufe)
FROM VerkaufTabelle
GROUP BY Modell, Farbe

UNION
SELECT 'ALL', 'ALL', Farbe, SUM(Verkaeufe)
FROM VerkaufTabelle
GROUP BY Farbe

UNION
SELECT 'ALL', Jahr, 'ALL', SUM(Verkaeufe)
FROM VerkaufTabelle
GROUP BY Jahr;
```

Die relationale Form, die das Ergebnis dieser Anfrage hat, ist nicht besonders übersichtlich, da sie sehr viele Zeilen mit den einzelnen Summen enthält. Daher wird oft eine andere Darstellung gewählt, die für den Menschen besser lesbar, allerdings nicht mehr in Relations-Form ist. Man fügt im zwei-dimensionalen Fall eine zusätzliche Zeile und Spalte hinzu, in der jeweils die Aggregate der restlichen Zeilen und Spalten stehen (Tabellen 1 und 2). Für sich genommen ist jede dieser beiden Tabellen zweidimensional. Nimmt man sie beide zusammen und erweitert diese dann noch um die Tabelle 3, so erhält man eine dreidimensionale Kreuztabelle (die sich zweidimensional nur durch Zerlegung in mehrere zweidimensionale Tabellen darstellen lässt).

A	2002	2003	Total (ALL)
Schwarz	50	85	135
Weiß	40	115	155
Total (ALL)	90	200	290

Tabelle 1: Kreuztabelle für Verkäufe Modell A

B	2002	2003	Total (ALL)
Schwarz	50	85	135
Weiß	10	75	85
Total (ALL)	60	160	220

Tabelle 2: Kreuztabelle für Verkäufe Modell B

Total (ALL)	2002	2003	Total (ALL)
Schwarz	100	170	270
Weiß	50	190	240
Total (ALL)	150	360	510

Tabelle 3: Kreuztabelle für Verkäufe aller Modelle

Wie man sieht, wächst die Zahl der benötigten `SELECT`-Statements jetzt exponentiell mit der Zahl der gewünschten Ebenen N und beträgt 2^N. Dies bedeutet, dass bereits eine sechs-dimensionale Kreuztabelle aus 64 verschiedenen `SELECT`-Statements besteht, die jeweils mit `UNION` verknüpft sind. In vielen existierenden Datenbanksystemen würde dies zu 64 Durchläufen durch die Daten, 64 Sortiervorgängen und einer langen Wartezeit führen.

2.2 Der `CUBE`-Operator

Die im vorigen Abschnitt beschriebene Möglichkeit, eine Kreuztabelle zu berechnen, beruht darauf, die Potenzmenge aller zu aggregierenden Attribute zu bestimmen und dann für jede der darin enthaltenen Attributmengen ein `SELECT`-Statement mit `GROUP BY`-Operator zu erzeugen. Diesen Vorgang kann man in das Datenbanksystem selbst verlagern, indem man die SQL-Syntax erweitert. Dies bietet zum einen die Möglichkeit einer sehr kompakten Formulierung und gibt zum anderen dem Datenbanksystem mehr Möglichkeiten zur Anfrageoptimierung. Die von Gray et al. [GCB+97] vorgeschlagene Syntax sieht einen `GROUP BY CUBE`-Operator als Erweiterung des normalen `GROUP BY`-Operators vor. Mit Hilfe dieses Operators lässt sich das in Abschnitt 2.1 entwickelte Statement zur Erzeugung der Kreuztabelle für die Fahrzeugverkäufe folgendermaßen formulieren:

```
SELECT Modell, Jahr, Farbe, SUM(Verkaeufe) AS Verkaeufe
FROM VerkaufTabelle
GROUP BY CUBE Modell, Jahr, Farbe;
```

Die Semantik des `CUBE`-Operators ist wie folgt definiert: zunächst werden die bezeichneten Attribute wie bei einem normalen `GROUP BY` aggregiert. Dann werden – wie in Abschnitt 2.1 dargestellt – alle Super-Aggregate (das sind Aggregate, in denen einzelne Attribute wiederum aggregiert werden) mittels `UNION` hinzugenommen, wobei die aggregierten Attribute den Wert `ALL` erhalten. Soll ein Cube über N Attribute erzeugt werden, gibt es $2^N - 1$ Super-Aggregat-Ausdrücke. Auch die Zahl der zusätzlichen Tupel in der Ergebnisrelation lässt sich angeben, sie hängt jedoch von der Zahl der möglichen Ausprägungen der Attribute ab. Haben die N Attribute $C_1, C_2, \cdots, C_N$ mögliche Ausprägungen und sind diese Ausprägungen in allen Kombinationen in der Ursprungsrelation enthalten, so enthält die Ergebnisrelation $\Pi(C_i + 1)$ Tupel. Dies resultiert daraus, dass jeder Wertebereich um den Wert `ALL` erweitert wird.

2.3 Der `ROLLUP`-Operator

Möchte man einen Roll-Up-Bericht erzeugen, könnte man dazu auch den `CUBE`-Operator verwenden, es werden dann jedoch zu viele Tupel erzeugt. Teile des vollen Cube haben keine Bedeutung für einen Roll-Up-Bericht und müssen daher auch nicht berechnet werden. Deshalb ist es sinnvoll, zusätzlich zum `CUBE`-Operator noch einen Operator für Roll-Ups anzubieten, der von Gray et al. [GCB+97] mit `ROLLUP` bezeichnet wird. Dieser erzeugt, ähnlich

wie das erste SQL-Statement in Abschnitt 2.1, lediglich die Super-Aggregate, bei denen kein ALL-Wert links von einem Wert aus dem ursprünglichen Wertebereich steht (bezogen auf die Reihenfolge der Attribute in der GROUP-BY ROLLUP-Klausel).

2.4 Die Algebra der Operatoren GROUP BY, CUBE und ROLLUP

Die Operatoren stehen in einer interessanten algebraischen Beziehung zueinander. Da ein CUBE alle Tupel eines ROLLUP und ein ROLLUP alle Tupel eines GROUP BY enthält, ergibt sich folgender Zusammenhang:

```
CUBE(ROLLUP) = CUBE
ROLLUP(GROUP BY) = ROLLUP
```

Daher ist es auch sinnvoll, die Operatoren in aufsteigender Reihenfolge nach ihrer Mächtigkeit zu verwenden. Dies spiegelt sich in der im nächsten Abschnitt vorgestellten Syntax der Operatoren wider.

2.5 Syntax

Von Gray et al. [GCB+97] wurde 1997 eine Syntax für einen um Cube- und Roll-Up-Funktionalität erweiterten GROUP BY-Operator vorgeschlagen. Mittlerweile ist dieser Vorschlag in leicht abgewandelter Form in den SQL:1999-Standard aufgenommen worden (siehe Abschnitt 2.7). Zunächst soll hier jedoch der ursprüngliche Vorschlag von Gray et al. [GCB+97] vorgestellt werden.

Die Syntax des GROUP BY-Operators ist in SQL:1992 folgendermaßen definiert:

```
GROUP BY {<column name> [collate clause], ...}
```

Um die im Abschnitt 2.1 geforderte Verallgemeinerung des GROUP BY-Operators zur Erstellung von Histogrammen mittels Aggregationen über Funktionswerte zu realisieren, muss diese Syntax folgendermaßen erweitert werden:

```
GROUP BY <aggregation list>
<aggregation list> ::=
  { ( <column name> | <expression> )
    [ AS <correlation name> ]
    [ <collate clause>      ]
    ,...}
```

Erweitert man diese Syntax noch um die oben vorgeschlagenen Operatoren CUBE und ROLLUP, ergibt sich folgende Syntax:

```
GROUP BY [ <aggregation list> ]
  [ ROLLUP <aggregation list> ]
    [ CUBE <aggregation list> ]
```

2.6 Vermeiden des ALL-Wertes

Die Darstellung des Cube mittels des zusätzlichen ALL-Wertes in den gruppierten Spalten ist ungünstig, da dieser Wert mit besonderer Semantik wie der NULL-Wert zu Problemen und vielen Ausnahmefällen führt. Es ist oft günstiger, wenn man den zusätzlichen Spezialwert vermeidet. Dazu lässt sich der bereits definierten NULL-Wert verwenden, welcher jedoch

das Problem aufwirft, dass er auch regulär im Wertebereich eines Attributes auftauchen kann. Um diese beiden Bedeutungen des NULL-Wertes zu unterscheiden, führt man einen zusätzlichen Operator ein, mit dessen Hilfe sich feststellen lässt, welche Attribute gruppiert wurden, also in welchen Spalten ein NULL-Wert die ALL-Bedeutung hat. Dieser Operator heißt GROUPING und erhält den Attributnamen als Parameter. Sein Wert ist true, wenn in diesem Tupel ein NULL-Wert des Attributes die ALL-Bedeutung hat, ansonsten ist es false. Dies kann man dann beispielsweise folgendermaßen anwenden:

```
SELECT Modell, Jahr, SUM(Verkaeufe) AS Verkaeufe,
  GROUPING(Modell), GROUPING(Jahr)
FROM VerkaufTabelle
GROUP BY CUBE Modell, Jahr;
```

Modell	Jahr	Verkaeufe	GROUPING(Modell)	GROUPING(Jahr)
A	2002	90	FALSE	FALSE
A	2003	200	FALSE	FALSE
A	NULL	290	FALSE	TRUE
B	2002	60	FALSE	FALSE
B	2003	160	FALSE	FALSE
B	NULL	220	FALSE	TRUE
NULL	NULL	510	TRUE	TRUE

Tabelle 4: Ergebnis der Anfrage mit GROUPING

Als Ergebnis auf diese Anfrage auf denselben Beispieldaten wie in den Tabellen 1 bis 3 würde man dann die in Tabelle 4 dargestellte Relation erhalten. Hier tauchen jetzt keine ALL-Werte mit besonderer Bedeutung mehr auf. Der Nachteil dieser Darstellung ist, dass man für jedes gruppierte Attribut eine zusätzliche Spalte benötigt, was die Ergebnisrelation unnötig vergrößert.

2.7 Der Data-Cube-Operator im SQL:1999-Standard

Die Vorschläge von Gray et al. [GCB+97] wurden zu großen Teilen in den SQL:1999-Standard übernommen. Die Syntax wurde geringfügig geändert, so dass die Attribute, auf die ein CUBE oder ROLLUP angewendet werden soll, in runden Klammern stehen müssen. Statt der ursprünglichen Variante mit dem zusätzlichen ALL-Wert wurde die in Abschnitt 2.6 vorgestellte Variante mit NULL-Werten und GROUPING-Operator verwendet. Weiterhin definiert der SQL:1999-Standard sogenannte GROUPING SETS, mit deren Hilfe beliebige Super-Aggregate erzeugt werden können, so dass auch Zwischenformen zwischen CUBE und ROLLUP möglich sind.

2.8 Zugriff auf den Data-Cube

Um mit einem Data-Cube rechnen zu können, muss man in Anfragen auf einzelne Elemente des Data-Cube zugreifen können. Will man beispielsweise Prozentanteile als Aggregatfunktion realisieren, musste man bisher eine Anfrage mit geschachtelten SELECT-Statements wie folgt formulieren:

```
SELECT Modell, Jahr, Farbe, SUM(Verkaeufe),
       SUM(Verkaeufe) / ( SELECT SUM(Verkaeufe) FROM Verkaeufe )
FROM Verkaeufe
GROUP BY CUBE Modell, Jahr, Farbe;
```

Da im Cube der Wert für `SELECT SUM(Verkaeufe) FROM Verkaeufe` allerdings schon berechnet wurde, wäre es sinnvoll, diesen zu verwenden, da dann die Anfrage zum einen effizienter ausgewertet und zum anderen einfacher formuliert werden kann. Dies könnte dann wie folgt aussehen:

```
SELECT Modell, Jahr, Farbe, SUM(Verkaeufe),
       SUM(Verkaeufe) / total(ALL,ALL,ALL)
FROM Verkaeufe
GROUP BY CUBE Modell, Jahr, Farbe;
```

2.9 Berechnung des Data-Cube

Da `CUBE` und `ROLLUP` den `GROUP BY`-Operator generalisieren, sind hier alle Technologien für die Berechnung von `GROUP BY`s auch auf den Kern des Data Cube anwendbar. Jedoch werden bei `CUBE` und `ROLLUP` zusätzliche Tupel erzeugt, die Super-Aggregate. Die darin enthaltenen `ALL`-Werte führen zu einem zusätzlichen Wert pro Attribut, was, wie bereits in Abschnitt 2.2 erläutert, zu $\Pi(C_i + 1)$ Ergebnistupeln führt. Ein vierdimensionaler Cube mit $C_i = 4, i = 1, \ldots, 4$ ist dann 2,4 mal größer als das Ergebnis eines normalen `GROUP BY`. Dieses Verhältnis verringert sich jedoch rasch mit wachsenden C_i. Im Vergleich enthält ein Roll-Up lediglich N zusätzliche Tupel.

Der `CUBE`-Operator erlaubt die Berechnung mehrerer Aggregate auf einmal. Im Folgenden wird jedoch nur eine Aggregatfunktion F behandelt, die Verallgemeinerung auf mehrere Funktionen ist einfach.

In den meisten Datenbank-Systemen werden Aggregatfunktionen bei der Berechnung zunächst initialisiert, indem ein Handle für diese Instanz der Aggregatfunktion erzeugt wird. Danach wird sie für jeden neuen Wert einmal aufgerufen und dann am Ende ein weiteres Mal aufgerufen um den Aggregatwert zu erhalten. Auf diese Weise werden in der Regel sowohl systeminterne als auch benutzerdefiniert Funktionen realisiert. Konkret muss zu diesem Zweck beispielsweise beim Informix Illustra System eine Aggregatfunktion die Methoden `Init`, `Iter` und `Final` anbieten, die genau diese Funktionalität realisieren.

Am einfachsten lässt sich nun der Data-Cube berechnen, indem für jede Zelle des Cube ein neues Handle der Aggregatfunktion erzeugt wird. Jede dieser Instanzen wird dann für jedes Tupel der Ausgangsrelation mit dem entsprechenden Wert aufgerufen. Da der Cube 2^N Zellen enthält bedeutet dies, dass für jedes Tupel der Ausgangsrelation die Aggregatfunktion 2^N mal aufgerufen wird. Möchte man nur einen Roll-Up berechnen, genügen N Instanzen der Aggregatfunktion. Enthält die Ausgangsrelation T Tupel, wird die Aggregatfunktion $T \times 2^N$ mal aufgerufen. Im Folgenden wird dieser Algorithmus daher 2^N-Algorithmus genannt. Diese Zahl lässt sich oft relativ einfach reduzieren, indem man die Super-Aggregate direkt aus den Kerndaten des `GROUP BY` berechnet. Wie diese Berechnung möglich ist, hängt davon ab, welche Aggregatfunktion berechnet werden soll. Betrachtet man die Aggregation einer zweidimensionalen Wertemenge $\{X_{ij} \mid i = 1, \ldots, I; j = 1, \ldots, J\}$, lassen sich die Aggregatfunktionen in drei Kategorien einteilen (nach [GCB+97]):

Distributiv: Eine Aggregatfunktion F heißt distributiv, wenn es eine Funktion G gibt, so dass $F(\{X_{ij}\}) = G(\{F(\{X_{ij} \mid i = 1, \ldots, I\}) \mid j = 1, \ldots, J\})$. Die üblichen Aggregat-

funktionen `COUNT`, `MIN`, `MAX` und `SUM` sind distributiv. Außer für `COUNT` gilt hier sogar stets $F = G$, für `COUNT` gilt $G = SUM$.

Algebraisch: Eine Aggregatfunktion F heißt algebraisch, wenn es eine M-wertige Funktion G und eine Funktion H gibt, so dass $F(\{X_{ij}\}) = H(\{G(\{X_{ij} \mid i = 1, \ldots, I\}) \mid j = 1, \ldots, J\})$. Durchschnitt und Standardabweichung sind Beispiele für algebraische Funktionen. Beispielsweise berechnet beim Durchschnitt die Funktion G Summe und Anzahl der Teilmenge. Die Funktion H addiert diese Werte und erzeugt dann den globalen Durchschnitt. Die Idee bei algebraischen Funktionen ist, dass ein Ergebnis mit fester Größe (ein M-Tupel) die Sub-Aggregate beschreibt.

Holistisch: Eine Aggregatfunktion F ist holistisch, falls es keine konstante Grenze für den notwendigen Platz zur Beschreibung eines Sub-Aggregates gibt. Das heißt es gibt keine Konstante M, so dass ein M-Tupel die Berechnung von $F(\{X_{ij} \mid i = 1, \ldots, I\})$ charakterisiert. Ein Beispiel für eine holistische Funktion ist der Median.

Für holistische Funktionen war Gray et al. [GCB+97] kein effizienterer Weg zur Berechnung von Super-Aggregaten als der oben beschriebene 2^N-Algorithmus bekannt, weshalb diese Funktionen hier nicht weiter behandelt werden.

Am einfachsten ist die Berechnung von Super-Aggregaten für distributive Funktionen. Hier lassen sich die Super-Aggregate des Data-Cube schrittweise aus den Sub-Aggregaten in höheren Dimensionen berechnen. Dies kommt daher, dass die Aggregatfunktion aufgrund ihrer Distributivität die Aggregation von Aggregaten erlaubt. Man kann also einfach die Ergebnisse der höheren Dimensionen wieder mittels einer Funktion zusammenfassen.

Auch bei algebraischen Funktionen existiert eine bessere Möglichkeit als der 2^N-Algorithmus, allerdings reicht es hierbei nicht aus, die reinen Ergebniswerte der höheren Dimensionen zur weiteren Berechnung zu verwenden. Vielmehr ist es notwendig, die Zwischenwerte, aus denen diese entstanden sind, weiter zu reichen. Möchte man beispielsweise den Durchschnitt berechnen, so reicht es nicht den reinen Durchschnittswert aus der höheren Dimension für die Berechnung zu verwenden, sondern man benötigt auch noch die Zwischendaten, die bei der Berechnung des Durchschnittes verwendet werden, nämlich die Summe und den Zähler.

3 Regel-basierte Sprachen (Rule-Based Languages)

Eine Möglichkeit, OLAP-Anfragen zu formulieren sind die sogenannten *Regel-basierten Sprachen* (*Rule-Based Languages*). Eine solche Sprache, beschrieben 1997 von Hacid et al. [HMR97] soll im Folgenden vorgestellt werden. Es handelt sich hierbei um eine Erweiterung der logischen Anfragesprache Datalog. Daten werden beim OLAP in einer n-dimensionalen Matrix, dem sogenannten *Cube*, organisiert. Basis dieser Spracherweiterung ist nun, dass ein Datalog-Fakt als ein Eintrag im Cube (*Zellenreferenz* genannt) angesehen werden kann. Die daraus entstandene Sprache erlaubt dann auf intuitive Weise die Beziehung zwischen Zellen zu definieren und damit die Beschreibung von:

1. allen Basis-Cube-Operationen (z. B. Push, Pull, Slicing-Dicing, Roll-Up, ...), die in OLAP-Systemen genutzt werden (siehe [Sch03])

2. komplexen Aggregationen von Daten in verschiedenen Ad-Hoc-Aggregationsebenen.

3.1 Das Datenmodell

Im Folgenden soll zunächst das der Sprache zugrunde liegende Datenmodell erläutert werden. Dieses besteht zunächst aus *Namen*, die sowohl für *Zellenreferenzen* als auch zur Beschreibung der *Zelleninhalte* verwendet werden. Eine Zellenreferenz zusammen mit einem Zelleninhalt wird *Zelle* genannt. Eine Menge von Zellen bildet einen *Cube*, die grundlegende Datenstruktur des OLAP. Mehrere Cubes werden dann zu einer *Datenbank* zusammengefasst. Dieses Datenmodell wird nun formal definiert:

Definition (Namen) Konstanten des Datenmodells werden *atomare Namen* genannt. Außerdem können mittels des Konstruktors „·“ *strukturierte* bzw. *geschachtelte Namen* gebildet werden. Im Folgenden werden sowohl atomare als auch geschachtelte Namen als *Namen* bezeichnet.

Beispiel In einer Prüfungsergebnis-Relation könnten Attribute mit dem Namen „Note“ oder den zusammengesetzten Namen „Note · Mathe“ und „Note · Physik“ vorkommen.

Definition (Zellen) In diesem mehrdimensionalen Datenmodell werden die Daten in *Zellen* organisiert. Eine Zelle wird über eine *Zellenreferenz* identifiziert und enthält einen eindeutigen *Zelleninhalt*. Eine Zellenreferenz hat die Form $N(N_1, N_2, \ldots, N_p)$, wobei N, N_1, N_2, ..., N_p Namen sind. N wird der *Cube-Name* genannt und $N_1, N_2, \ldots, N_p$ werden *Attribut-Namen* (im OLAP-Zusammenhang häufig auch als *Mitglieder* bezeichnet) genannt. Eine Zellenreferenz $N(N_1, N_2, \ldots, N_p)$ kann auch als Koordinate $(N_1, N_2, \ldots, N_p)$ in einem p-dimensionalen Raum N angesehen werden. Ein *Zelleninhalt* ist ein q-Tupel von Namen. Verknüpfungen von Zelleninhalten mit Zellenreferenzen werden durch Grundatome der Form $N(N_1, N_2, \ldots, N_p) : \langle N_{p+1}, \ldots, N_{p+q} \rangle$ dargestellt, wobei das Tupel $\langle N_{p+1}, \ldots, N_{p+q} \rangle$ den Zelleninhalt (im OLAP-Zusammenhang häufig auch als *Maße* bezeichnet) darstellt. Diese Form von Atomen wird *Zellenatome* genannt.

Definition (Cube) Ein *Cube* ist eine Menge von Grund-Zellenatomen, die denselben Cube-Namen haben und in denen dieselbe Referenz höchstens einmal enthalten ist, so dass die Zellen-Monovalution sichergestellt wird, das heißt jede Zelle einen eindeutigen Inhalt zugeordnet bekommt.

Definition (Datenbank) Eine *mehrdimensionale Datenbank* ist eine Menge von Grund-Zellenatomen in der dieselbe Referenz nicht mehrfach vorkommt. Wichtig hierbei ist die Unterscheidung zwischen einer Zelle, die nicht existiert (für die also keine Zellenreferenz in der Menge enthalten ist) und einer leeren Zelle, die durch ein Grund-Zellenatom der Form $N(N_1, N_2, \ldots, N_p) : \langle\rangle$ dargestellt wird.

3.2 Die Sprache

3.2.1 Syntax und intuitive Bedeutung

Zur Definition der Syntax werden zunächst die Mengen der Konstanten, Variablen und Aggregatsoperatoren definiert.

Definition (Konstanten und Variablen) Sei $\mathcal{D}$ eine abzählbar unendliche Menge von Konstanten, atomare Namen genannt, und $\mathcal{V}$ eine abzählbar unendliche Menge von Variablen, die mit $\mathcal{D}$ disjunkt ist.

Definition (Aggregatsoperator) Ein *Aggregatsoperator* f ist eine partielle Abbildung von Multimengen von Tupeln über $\mathcal{D}$ auf einen einzelnen Wert. Sei $\mathcal{AGG}$ eine Menge von Aggregatsoperatoren.

Definition (Regel-basierte Sprache [HMR97]) Syntaktisch erlaubte Ausdrücke einer Regel-basierten Sprache sind:

$$
\begin{aligned}
\mathit{atomicName} &:= c \mid v \\
\mathit{name} &:= \mathit{atomicName} \mid \mathit{name} \cdot \mathit{name} \\
\mathit{contents} &:= \langle \mathit{name}, \ldots, \mathit{name} \rangle \\
\mathit{reference} &:= \mathit{name}(\mathit{name}, \ldots, \mathit{name}) \\
\mathit{cell-atom} &:= \mathit{reference} : \mathit{contents} \\
\mathit{groupingAtom} &:= \mathit{in}(\mathit{atomicName}, \mathit{atomicName}) \\
\mathit{atom} &:= \mathit{cell-atom} \mid \mathit{groupingAtom} \\
\mathit{aggregateSubgoal} &:= \mathit{atomicName} = f(\mathit{reference}) \\
\mathit{literal} &:= \mathit{atom} \mid \mathit{aggregateSubgoal} \\
\mathit{body} &:= \mathit{literal}, \ldots, \mathit{literal} \\
\mathit{head} &:= \mathit{atom} \\
\mathit{rule} &:= \mathit{head} \longleftarrow \mathit{body}
\end{aligned}
$$

wobei $c \in \mathcal{D}, v \in \mathcal{V}$ und $f \in \mathcal{AGG}$.

Regeln (*rules*) werden verwendet, um neue Zellenreferenzen und deren Inhalte durch bereits existierende Zellen zu definieren. Variablen haben dabei als Wertebereich alle atomaren Namen, die in Zellenreferenzen oder Zelleninhalten verwendet werden. Wichtig ist hierbei, dass Variablen nur atomare und keine geschachtelten Namen als Wert annehmen können.

Die intuitive Bedeutung der Regeln ist dabei wie folgt zu sehen: Die Standard-Datalog-Bedeutung der Regel $p(X) \longleftarrow q(X,Y), r(Y)$ ist „*wenn* $q(X,Y)$ *wahr ist und* $r(Y)$ *wahr ist, dann ist auch* $p(X)$ *wahr*“. Im Sinne der hier beschriebenen Spracherweiterung ändert sich diese intuitive Bedeutung zu „*wenn es zwei Zellen mit Referenz* $q(X,Y)$ *und* $r(Y)$ *gibt, dann gibt es eine Zelle mit der Referenz* $p(X)$“. Da hier auch die Zelleninhalte zu einer Regel gehören sollen, wird eine typische Regel so aussehen: $p(X) : \langle W \rangle \longleftarrow q(X,Y) : \langle W \rangle, r(Y) : \langle X \rangle$. Diese Regel würde informell gelesen als „*wenn es eine Zelle mit Referenz* $q(X,Y)$ *und Inhalt* W *sowie eine Zelle mit Referenz* $r(Y)$ *und Inhalt* X *gibt, dann gibt es auch eine Zelle mit Referenz* $p(X)$ *und Inhalt* W“.

Um ein Gruppierungs-Atom (*groupingAtom*) auswerten zu können, benötigt man eine Gruppierungs-Beziehung zwischen den Attributnamen. Diese Beziehung repräsentiert die beim OLAP häufig verwendeten *Hierarchien*. Die Gruppierungs-Beziehung wird durch eine Menge von Literalen in Form von Gruppierungs-Atomen auf atomaren Namen spezifiziert. Enthält die Datenbank beispielsweise Produktdaten mit den atomaren Namen *fahrrad*, *auto*, *milch* und *brot*, so würde deren Zugehörigkeit zu den Produktgruppen fahrzeuge und nahrungsmittel und wiederum deren Zugehörigkeit zu den gesamten Produkten über folgende Menge von Gruppierungs-Atomen spezifiziert:
{*in(fahrrad, fahrzeuge), in(auto, fahrzeuge), in(milch, nahrungsmittel), in(brot, nahrungsmittel), in(nahrungsmittel, produkte), in(fahrzeuge, produkte)*}

Aggregat-Teilziele (*aggregateSubgoal*) werden verwendet, um Daten mittels Aggregats-Operatoren wie *max, min, sum* zusammenzufassen. Dazu formuliert man ein Aggregat-Teilziel, das eine (möglicherweise Nicht-Grund-)Referenz enthält. Es werden dann alle atomaren

Namen dieser Referenz, die nicht auf der untersten (detailliertesten) Ebene einer Hierarchie bzw. Gruppierungs-Beziehung stehen, folgendermaßen behandelt: man definiert eine Menge *detailRef*(*aggregateSubgoal*) so, dass alle atomaren Namen A, die nicht auf der untersten Ebene der Hierarchie stehen in die ihnen entsprechenden atomaren Namen der untersten Hierarchie-Ebene expandiert werden, sofern diese existieren. Dann wird der Aggregats-Operator auf alle Inhalte der so gewonnenen Zellen angewendet. Zu beachten ist hierbei, dass für *detailRef* stets *alle* Koordinaten expandiert werden; das heißt wenn mehrere der atomaren Namen nicht auf der untersten Ebene der Hierarchie stehen, werden alle Kombinationen der entsprechenden Detailnamen erzeugt.

Für die formale Festlegung der Semantik werden noch einige weitere Begriffe benötigt.

Definition (Referenz-Anteil) Mit $ref(A)$ wird der Referenz-Anteil eines Zellenatomes oder Aggregat-Teilziels A bezeichnet.

Definition (Variablenmenge) Sei *var* eine berechenbare Funktion, die jedem syntaktischen Ausdruck die Teilmenge von $\mathcal{V}$ zuordnet, die der Menge der in diesem Ausdruck vorkommenden Variablen entspricht.

Definition (Grund-Name) Ein Grund-Name ist ein Name n für den $var(n) = \emptyset$ gilt. Diese Bezeichnung wird analog auch für Referenzen und Literale verwendet.

Definition (Bereichsbeschränkte Regel) Eine bereichsbeschränkte Regel ist eine Regel $r = A \longleftarrow B_1, \ldots, B_n$ wobei:

- $var(A) \subseteq var(\{B_1, \ldots, B_n\})$ und
- sei $\mathcal{A}g$ die Menge von Referenzen, die in Aggregat-Teilzielen von r vorkommen und $\mathcal{B}$ die Menge von Atomen, die im Body von r vorkommen, dann gilt $var(\mathcal{A}g) \subseteq var(\mathcal{B})$.

Definition (Programm) Ein Programm ist eine endliche Menge bereichsbeschränkter Regeln.

Definition (Restrukturierungs-Programm) Ein Restrukturierungs-Programm ist ein Programm, dessen Regel kein Aggregat-Teilziel in ihrem Body haben.

Definition (Zusammenfassungs-Programm) Ein Zusammenfassungs-Programm ist ein Programm, dessen Regeln kein Gruppierungs-Atom in ihrem Head haben.

Definition (OLAP-Programm) Ein OLAP-Programm ist ein Paar $\langle R, S \rangle$, wobei R ein Restrukturierungs-Programm und S ein Zusammenfassungs-Programm ist. Informell bedeutet dies, dass zunächst das Programm R und dann S benutzt wird.

3.2.2 Semantik

In diesem Abschnitt wird formal eine modelltheoretische Semantik der im vorigen Abschnitt definierten und motivierten Syntax angegeben.

Definition (Eingabe) Die Semantik eines Programmes wird relativ zu einer Menge von Grundatomen, der sogenannten Eingabe beschrieben. Diese stellt den *extensionalen* Teil der Datenbank dar (also die schon vorhandenen Daten in der Datenbank).

Definition (Gruppierungs-Beziehung) Sei J eine Menge von Grundatomen. Dann ist in_J die Relation zu den in J enthaltenen Gruppierungs-Atomen. Formal wird die Relation in_J folgendermaßen definiert: $\forall x, y \in \mathcal{D}.\ in_J(x, y) \Longleftrightarrow in(x, y) \in J$.

Definition (Erweiterte Gruppierungs-Beziehung) Sei J eine Menge von Grundatomen. Dann ist die Relation $<_J$ über Referenzen folgendermaßen definiert: für alle Referenzen $rf = n(n_1, \ldots, n_p)$ und $rf' = n(n'_1, \ldots, n'_p)$ gilt $rf < rf' \Longleftrightarrow rf \neq rf'$ und $\forall i \in [1, \ldots, p]$ entweder $in_J(n_i, n'_i)$ oder $n_i = n'_i$.

Das heißt $rf <_J rf'$ wenn rf eine Zelle auf einem strikt kleineren Detaillevel referenziert als rf' hinsichtlich der durch J definierten Gruppierungs-Relation.

Definition (Interpretation bezüglich einer Eingabe) Sei I eine Eingabe. Eine Menge J von Grundatomen ist eine *Interpretation* bezüglich I, falls die folgenden Bedingungen erfüllt sind:

- für alle Zellenatome $A_1, A_2 \in I$ gilt $ref(A_1) = ref(A_2) \Longrightarrow A_1 = A_2$, wobei „=“ die syntaktische Gleichheit bezeichnet. Diese Kriterium garantiert, dass eine Zelle stets einen eindeutigen Zelleninhalt hat;
- der transitive Abschluss von in_J ist irreflexiv. Dadurch wird garantiert, dass die Gruppierungs-Beziehung keine gerichteten Zykel enthält;
- $I \subseteq J$, das heißt, wenn etwas in der Eingabe wahr ist, so ist es auch in der Interpretation wahr.

Definition (Detaillierteste Information) Die Erfüllung eines Grund-Aggregat-Teilziels der Form $k = f(n(n_1, \ldots, n_p))$ hängt von der detailliertesten Information über $n(n_1, ..., n_p)$ in der Eingabe ab. Dies wird durch die bereits in Abschnitt 3.2.1 motivierte Funktion *detailRef* sowie die Funktion *detailCont* formalisiert. Die Menge $detailRef(n(n_1, \ldots, n_p))$ ist die Menge der Zellenreferenzen in der Eingabe mit der detailliertesten Information über $n(n_1, \ldots, n_p)$. Die Menge $detailCont(n(n_1, \ldots, n_p))$ ist die Multimenge der Inhalte dieser Zellen. Formal lässt sich dies für ein Grund-Aggregat-Teilziel B folgendermaßen formulieren:

$$\begin{aligned} detailRef_i^J(B) &= \{A \in I \mid ref(A) <_J ref(B)\} \\ detailCont_i^J(B) &= \{k \mid k = val(A), A \in detailRef_i^J(B)\} \end{aligned}$$

Definition (Evaluierungsfunktion) Eine Evaluierungsfunktion v ist eine totale Funktion von $\mathcal{V}$ nach $\mathcal{D}$. v wird erweitert zur Identität auf $\mathcal{D}$. Außerdem wird v kanonisch auf Namen, Literale und Regeln erweitert.

Definition (Erfüllung) Sei J eine Interpretation bezüglich einer Eingabe I. J erfüllt das Grundliteral B bezüglich I (Schreibweise: $J \models_I B$), genau dann, wenn

- B ist ein Grundatom und $B \in J$, oder
- B ist ein Grund-Aggregat-Teilziel der Form $k = f(n(n_1, \ldots, n_p))$, so dass $detailRef_I^J(B) \neq \emptyset, f(detailCont_I^J)$ definiert ist und $f(detailCont_I^J(B)) = k$.

Eine Interpretation J erfüllt eine Regel $r = A \longleftarrow B_1, \ldots, B_n$ bezüglich I (Schreibweise: $J \models_I r$) genau dann, wenn für jede Evaluierungsfunktion v gilt:

- $J \models_I v(A)$ oder
- $\exists B_i, i \in [1, \ldots, n].J \not\models_I (B_i)$.

Definition (Modell eines Programms) Eine Interpretation J ist ein Modell eines Restrukturierungs- oder Zusammenfassungsprogramms P bezüglich einer Eingabe I (Schreibweise: $J \models_I P$) genau dann, wenn $\forall r \in P.\ J \models_I r$.

Definition (Semantik eines Programms) Für ein Restrukturierungs- oder Zusammenfassungsprogramm P und eine Eingabe I ist die Semantik von P für I das eindeutige, minimale Modell von P bezüglich I, falls dieses existiert. Es wird mit $P(I)$ bezeichnet.

Es lässt sich beweisen, dass folgender Zusammenhang gilt:

Satz 1 Sei P ein Restrukturierungs- oder Zusammenfassungsprogramm und I eine Eingabe. Falls P ein Modell hat, existiert $P(I)$ und $P(I)$ ist endlich.

Definition (Semantik eines OLAP-Programms) Sei $Q = \langle R, S \rangle$ ein OLAP-Programm und I eine Eingabe. Die Semantik von Q für I ist $S(R(I))$, falls es existiert. Es wird mit $Q(I)$ bezeichnet.

3.3 Anwendung der Sprache bei OLAP-Datenmanipulationen

In diesem Abschnitt soll beispielhaft dargestellt werden, wie sich die zuvor entwickelte Sprache zur Formulierung von typischen, komplexen OLAP-Datenmanipulationen wie push – pull, Roll-Up und Drill-Down sowie den in Abschnitt 2 vorgestellten Cube-Operator verwenden lässt. Die Operatoren werden dabei in Form von OLAP-Programmen angegeben. Die Regeln des Restrukturierungs-Anteils werden durch ein Pfeil der Form $\xleftarrow{r}$, die des Zusammenfassungs-Anteils durch einen Pfeil der Form $\xleftarrow{z}$ gekennzeichnet.

In den folgenden Beispielen gilt die Konvention, dass Namen stets mit einem großen Anfangsbuchstaben und Variablen stets komplett klein geschrieben werden.

3.3.1 Beschreibung der Beispiel-Datenbank

Als Beispiel-Datenbank soll hier eine vereinfachte Version der im OLAP Benchmark APB-1 [The97] definierten Datenbank verwendet werden. Diese enthält einen Cube $c1$, der Informationen über Verkäufe eines Herstellers an seine Kunden (wobei der Hersteller zunächst an Zwischenhändler verkauft) im Laufe der Zeit enthält. Die Verkaufsinformationen beinhalten die verkauften Einheiten und die Einnahmen in Euro. Im Datenmodell werden diese Informationen durch Zellenatome der Form $c1(monat, produkt, kunde) : \langle einheiten, einnahmen \rangle$ dargestellt. Diese repräsentieren die verkauften Einheiten (*einheiten*) und die Einnahmen in Euro (*einnahmen*) für ein Produkt *produkt*, das an den Kunden *kunde* im Monat *monat* verkauft worden ist. Die möglichen Gruppierungen, beschrieben durch die *in*-Relation, sind:

- Endkunden werden nach Zwischenhändlern gruppiert, von denen sie beliefert werden. Diese wiederum werden zur gesamten Kundschaft des Herstellers, *Kundschaft* genannt, gruppiert.
- Die verschiedenen Produkte werden zur gesamten Produktpalette des Herstellers, *Produktpalette* genannt, gruppiert.
- Die Monate können zum ganzen Jahr gruppiert werden, genannt *Zeitraum*.

3.3.2 OLAP-Datenmanipulationen

Push – Pull: Die Operatoren Push und Pull erlauben die Gleichbehandlung von Metadaten (Zellenreferenzen) und Daten (Zelleninhalten).
Zum Beispiel:

- Pushen des Monats in die Zelleninhalte:

$$c2(monat, produkt, kunde) : \langle einheiten, einnahmen, monat\rangle$$
$$\stackrel{r}{\longleftarrow} c1(monat, produkt, kunde) : \langle einheiten, einnahmen\rangle$$

- Pullen der verkauften Einheiten in die Zellenreferenz:

$$c3(monat, produkt, kunde, einheiten) : \langle einnahmen\rangle$$
$$\stackrel{r}{\longleftarrow} c1(monat, produkt, kunde) : \langle einheiten, einnahmen\rangle$$

Der Einfachheit halber wird im Folgenden nur noch der Cube $c4$ benutzt, der nur die monatlichen Einnahmen enthält und wie folgt definiert ist:

$$c4(monat, produkt, kunde) : \langle einnahmen\rangle$$
$$\stackrel{r}{\longleftarrow} c1(monat, produkt, kunde) : \langle einheiten, einnahmen\rangle$$

Roll-Up auf alle möglichen Ebenen: Eine einzige Regel genügt, um eine Summierung der Verkäufe aus dem Cube $c4$ auf allen Gruppierungs-Ebenen zu spezifizieren. Die resultierenden Daten können sogar in den Cube selbst integriert werden. Möchte man beispielsweise die Zusammenfassung der Verkäufe jedes Produktes P im Monat Januar auf den zwei möglichen Ebenen *Zwischenhändler* und *Kundschaft* zum Cube $c4$ hinzufügen, so kann dies mit der folgenden Regel spezifiert werden:

$$c4(Januar, produkt, kunde) : \langle s\rangle \quad \stackrel{z}{\longleftarrow} \quad s = sum(c4(Januar, produkt, kunde)),$$
$$in(produkt, Produktpalette),$$
$$in(x, kunde)$$

Jede Instanziierung von *kunde* in $in(x, kunde)$ ergibt eine unterschiedliche Gruppierung. Da $in(x, kunde)$ erfüllt sein muss, werden alle die Instanzen für *kunde* ausgewählt, für die ein x existiert, das in der *in*-Relation mit *kunde* steht. Es wird also auf allen Gruppierungs-Ebenen der Kunden, außer der untersten, gruppiert, das heißt auf der Ebene der Zwischenhändler und auf der Ebene der gesamten Kundschaft, nicht jedoch auf der Ebene der Endkunden.

Drill-Down Auch ein Drill-Down, also die Hinzunahme von Details, kann mittels Regeln spezifiziert werden. Möchte man aus dem Cube $c4$, mit den Zellen der Roll-Up aus dem vorherigen Beispiel erweitert, aus dem man die Einträge der Zwischenhändler mit Einnahmen von mehr als 100 filtert, die Verkäufe in einer detaillierteren Ebene (d.h. der Geschäftsebene) erhalten, kann man dies mit folgender Regel erreichen ($\geq$ ist ein eingebautes Prädikat mit der üblichen Bedeutung):

$$zwischenhaendlerDetails \cdot januar(kunde, produkt) : \langle einheiten\rangle$$
$$\stackrel{z}{\longleftarrow} \quad c4(Januar, produkt, zwischenhaendler) : \langle summe\rangle,$$

$$
\begin{aligned}
& summe \geq 100, \\
& in(zwischenhaendler, Kundschaft), \\
& in(produkt, Produktpalette), \\
& c4(Januar, produkt, kunde) : \langle einheiten \rangle, \\
& in(kunde, Zwischenhaendler)
\end{aligned}
$$

Cube-Operator Ein vollständigen Data-Cube wie Abschnitt 2 beschrieben lässt sich mit der Regel

$$
\begin{aligned}
& cubeOperatorErgebnis(monat, produkt, kunde) : \langle S \rangle \\
\xleftarrow{z} \quad & S = sum(c4(monat, produkt, kunde)), \\
& ebene(monat) = \langle\rangle, \\
& ebene(produkt) = \langle\rangle, \\
& ebene(kunde) = \langle\rangle
\end{aligned}
$$

erzeugen, wobei *ebene* durch die folgenden zwei Regeln spezifiziert wird:

$$
\begin{aligned}
ebene(x) : \langle\rangle \quad & \xleftarrow{r} \quad in(x, y), \\
ebene(y) : \langle\rangle \quad & \xleftarrow{r} \quad in(x, y).
\end{aligned}
$$

4 Multidimensional-Expressions (MDX)

Eine Entwicklung der Firma Microsoft auf dem Gebiet der OLAP-Anfragesprachen sind die Multidimensional-Expressions (MDX). Diese wurden in Zusammenhang mit Microsofts Datenzugriffs-Spezifikation OLE DB MD, der mehrdimensionalen Erweiterung des Datenzugriffs-Standards OLE DB, definiert und zuerst im Microsoft SQL-Server eingesetzt, später dann auch in Produkten anderer Hersteller [MW97].

4.1 Vergleich von MDX und SQL

Die Struktur von MDX-Anfragen ist der Struktur von klassischen SQL-Anfragen sehr ähnlich. Auch hier gibt es einen `FROM`-Ausdruck, der die Datenquelle bezeichnet, einen `WHERE`-Ausdruck, mit dem die Daten gefiltert werden sowie einen `SELECT`-Ausdruck, um die Daten in Zeilen und Spalten abzubilden. In MDX existieren jedoch noch weitere Schlüsselwörter, um Cubes abzufragen und analysierbare Daten zurückzuliefern. Weiterhin existieren in MDX viele Funktionen, mit denen die Daten manipuliert werden können und es besteht auch die Möglichkeit, benutzerdefinierte Funktionen einzusetzen.

Neben diesen Möglichkeiten einer Daten-Manipulations-Sprache (Data Manipulation Language, DML) bietet MDX auch die Möglichkeit als Daten-Definitions-Sprache (Data Definition Language, DDL) eingesetzt zu werden. Dabei gibt es Befehle, mit denen Cubes, Dimensionen, Maße und andere OLAP Strukturen erzeugt, verändert und gelöscht werden können. Im Folgenden wird dieser Teil der Sprache jedoch nicht weiter betrachtet werden.

4.2 Terminologie von MDX

MDX-Ausdrücke arbeiten auf mehrdimensionalen Daten, die in mehrdimensionalen Strukturen, den *Cubes*, angeordnet sind [Pea02].

Eine *Dimension* ist eine *Hierarchie* von *Kategorien* (oder Ebenen). Beispielsweise könnte eine Dimension für Geschäfte eine Hierarchie mit den Ebenen Land, Bundesland, Stadt und Geschäft haben. Es ist auch möglich, dass mehrere Hierarchien für eine Ebene existieren. Beispielsweise kann eine Zeit-Dimension sowohl eine Hierarchie für das Kalenderjahr als auch eine für das Geschäftsjahr haben. Die Elemente der untersten Hierarchieebene werden *Mitglieder* (members) der Dimension genannt.

Die Datenelemente, *Maße* (measures) genannt, sind numerische Werte und werden an den Schnittpunkten der Dimensionen, den Zellen, gespeichert. Es ist möglich, mehr als ein Maß in einer Zelle zu speichern. Jeder Cube kann maximal 64 Dimensionen haben, wobei eine dieser Dimension fest als die Menge der Maße definiert ist und daher „`Measures`“ genannt wird.

Um Daten zu identifizieren und extrahieren wird in MDX ein Referenzsystem, das auf *Tupeln* basiert, verwendet. Tupel bezeichnen Dimensionen und deren Mitglieder um individuelle Zellen sowie Gruppen von Zellen im Cube zu referenzieren. Da jede Zelle ein Schnittpunkt der Cube-Dimensionen ist, können Tupel eindeutig jede Zelle im Cube identifizieren. Außerdem können Tupel auch Teile des Cube identifizieren, die *Scheiben* (slices) genannt werden, indem Elemente höherer Hierarchiebenen angegeben werden.

4.3 Beispiele für MDX-Anfragen

Die folgenden Beispiele sind größtenteils aus dem Artikel von Carl Nolan im MSDN (Microsoft Developer Network) vom August 1999 [Nol99] entnommen. Den Beispielen liegt das mit den OLAP-Services des Microsoft SQL-Server ausgelieferte FoodMart-Beispiel zugrunde. Dieses Beispiel enthält einen Cube (`Sales`), in dem Kunden, Produkte und Promotion-Aktionen zu den Verkäufen eines Lebensmittelhandels gespeichert sind.

Eine einfache Form eines MDX-Ausdruckes, der zwei der Cube-Dimensionen in Form einer Tabelle liefert, hat folgendes allgemeines Aussehen:

```
SELECT axis_specification ON COLUMNS,
       axis_specification ON ROWS,
  FROM cube_name
 WHERE slicer_specification
```

Die Schnitt-Spezifikation `slicer_specification` im `WHERE`-Teil definiert einen Ausschnitt des Cube, aus dem die Ergebnisdaten der Anfrage entnommen werden sollen (vgl. die OLAP-Operation *Slicing* und *Dicing* [Sch03]). Dies geschieht durch Angabe einer Menge von Tupeln, die den Cube jeweils in Scheiben zerschneiden. Hier kann zum einen eine Scheibe aus einer der Dimensionen gewählt werden (beispielsweise ein bestimmtes Jahr der Zeit-Dimension) oder das Maß, das in den Zellen des Ergebnisses eingetragen wird. Dieser Teil ist optional – wird er ganz weg gelassen, wird das zuvor definierte Standard-Maß des Cube für die Zellen des Ergebnisses ausgewählt. Solange nicht explizit die Measures-Dimension abgefragt wird, sollte jedoch immer eine `slicer_specification` angegeben werden.

Die Achsen-Spezifikation `axis_specification` erlaubt die Auswahl der Mitglieder, die für die Achsen des Ergebnisses verwendet werden sollen. Wenn das Ergebnis eindimensional ist, muss das Schlüsselwort `COLUMNS` verwendet werden, die zweite Achse wird mit `ROWS` bezeichnet und für weitere Achsen stehen `PAGES`, `CHAPTERS` und `SECTIONS` zur Verfügung. Allgemeiner lassen sich die Achsen auch mittels `AXIS(index)` benennen, wobei `index` eine bei 0 beginnende Achsen-Referenz ist. In der Regel wird man nicht alle Dimensionen eines Cube auf die Achsen des Ergebnisses abbilden. Die Maße werden dann über die nicht verwendeten Dimensionen aufsummiert. Eine Möglichkeit einer Achsen-Spezifikation ist die Auswahl der `MEMBERS` einer Dimension (einschließlich der speziellen `Measures`-Dimension):

```
SELECT Measures.MEMBERS ON COLUMNS,
       [Store].MEMBERS  ON ROWS,
  FROM [Sales]
```

Dieser Ausdruck hat als Ergebnis eine zweidimensionale Tabelle. Die erste Achse (die Spalten) besteht aus den Mitgliedern der `Measures`-Dimension (also den Bezeichnungen von allen in den Zellen gespeicherten Maßen) und die zweite Achse (die Zeilen) aus den Mitgliedern der `Store`-Dimension, also den einzelnen Geschäften. Die Zellen enthalten dann das entsprechenden Maß für ein Geschäft, wobei dieses als die Summe des Maßes über die nicht spezifizierten Dimensionen (wie z. B. Zeit) berechnet wird. Zusätzlich werden die Maße auf allen definierten Hierachie-Ebenen der `Store`-Dimension in einer gesonderten Ergebniszeile zusammengefasst.

Statt mittels `MEMBERS` alle Mitglieder einer Dimension auszuwählen, können auch gezielt einzelne Mitglieder bestimmt werden. Dies würde in der Beispielanfrage dann so aussehen:

```
SELECT Measures.MEMBERS ON COLUMNS,
       {[Store].[Store State].[CA],[Store].[Store State].[WA]} ON ROWS,
  FROM [Sales]
```

Dieser Ausdruck liefert dieselben Daten wie der vorherige, aufsummiert für die Staaten Kalifornien und Washington. Möchte man die Daten für die Mitglieder dieser beiden Staaten erhalten (also die Städte, die in den Staaten liegen), die bei MDX als `CHILDREN` bezeichnet werden, würde man folgenden Ausdruck formulieren:

```
SELECT Measures.MEMBERS ON COLUMNS,
       {[Store].[Store State].[CA].CHILDREN,
        [Store].[Store State].[WA].CHILDREN } ON ROWS,
  FROM [Sales]
```

Bei den bisherigen Beispielen wurden stets die Mitglieder der `Measures`-Dimension für eine Achse des Anfrageergebnisses verwendet. Möchte man dies nicht, muss man entweder ein konkretes Mitglied dieser Dimension mittels einer `WHERE`-Klausel auswählen oder erhält das bei der Definition der Dimension festgelegte Standard-Maß in den Zellen des Ergebnisses, da die Ergebniszellen stets höchstens ein Maß enthalten können. Möchte man also ein zweidimensionales Ergebnis, bei dem man beide Dimensionen mit einer der ursprünglichen Cube-Dimensionen belegt, ist es sinnvoll, über eine `WHERE`-Klausel das gewünschte Maß auszuwählen. Eine Anfrage, die beispielsweise die durchschnittlichen Verkaufszahlen der Geschäfte auf der Bundesstaat-Ebene zusammengefasst und verknüpft mit dem Geschäftstyp ausgibt, könnte folgendermaßen aussehen:

```
SELECT {[Store Type].[Store Type].MEMBERS} ON COLUMNS,
       {[Store].[Store State].MEMBERS) ON ROWS,
  FROM [Sales]
 WHERE (Measures.[Sales Average])
```

Wie bereits erläutert wurde, erzeugt die `WHERE`-Klausel einen Ausschnitt aus einer Dimension des Cube. Im Beispiel wurde dies zunächst auf die `Measures`-Dimension angewandt. Durch das Tupel `(Measures.[Sales Average])` wird die Scheibe aus der `Measures`-Dimension herausgeschnitten, die die Daten zu den durchschnittlichen Verkaufszahlen enthält. Natürlich ist dies auch mit allen anderen Cube-Dimensionen möglich. Ist man beispielsweise nur an den Verkaufszahlen für das Jahr 2003 interessiert, müsste man die `WHERE`-Klausel der Beispielanfrage folgendermaßen erweitern:

```
WHERE (Measures.[Sales Average], [Time].[Year].[2003])
```

Durch diese Anfrage würde dann zusätzlich aus der Zeit-Dimension (`TIME`) ein Schnitt auf der Hierarchie-Ebene der Jahre (`YEAR`) gemacht, so dass nur die Scheibe für das Jahr 2003 in die Summation auf der Zeit-Dimension einbezogen wird. Der Unterschied zwischen dem Filtern (mittels `SELECT`) und dem Schneiden (mittels `WHERE`) liegt darin, dass beim Schneiden nicht die Achsen-Mitglieder ausgewählt werden, sondern die Daten, die in diese eingebracht werden. Im Gegensatz dazu wird beim Filtern die Zahl der Mitglieder der Achsen reduziert.

5 *n*D-SQL

Mit der SQL-Spracherweiterung nD-SQL verfolgten Gingras und Lakshmanan [GL98] zwei Ziele:

1. Die Möglichkeit, Anfragen auf einer föderierten relationalen Datenbank (also einem evtl. auf mehrere Orte verteilten Zusammenschluss von Datenbanken mit heterogenen Schemata) durchzuführen, indem die Konflikte zwischen den unterschiedlichen Schemata aufgelöst werden.

2. Unterstützung zur Formulierung von OLAP-Anfragen mit Aggregationen auf mehreren Granularitätsebenen.

Eine Sprache zur Verfügung zu haben, die diese beiden Eigenschaften vereint, ist wichtig, da in großen Unternehmen oft die Daten aus verschiedenen Abteilungen oder Niederlassungen in getrennten Datenbanken gespeichert werden. Es ist dann für die Datenanalyse im Rahmen des OLAP notwendig, eine Anfragesprache zur Verfügung zu haben, die die Daten aus den einzelnen Quellen zusammenführt und darauf OLAP-Operationen ermöglicht. Natürlich ist es in einem solchen Fall oft sinnvoll, ein zentrales Data-Warehouse mit den benötigten Daten zu erstellen. Dieser Vorgang kann jedoch sehr lange dauern und ist manchmal gar nicht möglich oder erwünscht. In diesem Fall kann mit einer Sprache, die die Zusammenarbeit zwischen verschiedenen Datenquellen direkt ermöglicht, eventuell ein Weg gefunden werden, ohne ein zentrales Data-Warehouse dennoch die Daten aus mehreren Quellen miteinander in Verbindung zu bringen und zu analysieren.

5.1 Das Modell

Es soll nun das Modell von Gingras und Lakshmanan [GL98] für föderierte relationale Datenbanken vorgestellt werden. Dieses Modell hat folgende Eigenschaften:

1. Es behandelt heterogene Schemata von relationalen Datenbanken, die in der Praxis auftreten. Dabei werden Daten und Schemata symmetrisch behandelt.

2. Es legt eine starke Betonung auf die drei physikalischen Dimensionen, die im traditionellen relationale Modell nur implizit enthalten sind: Zeile, Spalte und Relation.

3. Unter Ausnutzung von 2. wird eine präzise Bedeutung für die Repräsentation n-dimensionaler Daten in drei physikalischen Dimensionen festgelegt.

Seien $\mathcal{N}$ eine Menge von Namen und $\mathcal{V}$ eine Menge von Werten. Seien diese Mengen disjunkt sowie unendlich. Im Folgenden werden Namen mit Schreibmaschinenschrift gekennzeichnet (z. B. `Kurs`) und Werte mit normaler Serifenschrift gekennzeichnet (z. B. Eröffnung).

Definition (föderiertes Schema) Ein *föderierter Name* ist ein Paar (N, X), wobei $N \in \mathcal{N}$ ein Name und $X \subset \mathcal{N}$ eine endliche Teilmenge von Namen ist, so dass $N \notin X$. In einem föderierten Namen wird N als *Konzept* und die Menge X als *assoziierte Merkmalsmenge* bezeichnet. Ein föderierter Name $N(X)$ ist *einfach*, falls $X = \emptyset$ und *komplex*, falls $X \neq \emptyset$. Im Folgenden werden einfache föderierte Namen $N(\emptyset)$ meist nur mit N bezeichnet. Ein *föderiertes Attribut-* oder *föderierter Relationsname* ist ein beliebiger föderierter Name. Ein *föderiertes Relationsschema* hat die Form $R(C_1, \ldots, C_n)$ wobei R ein föderierter Name ist und die C_i föderierte Attributnamen sind. Ein *föderiertes Datenbankschema* ist eine Menge von föderierten Relationsschemata und ein *föderiertes Schema* ist eine Menge von mit Namen bezeichneten föderierten Datenbankschemata.

Das föderierte Schema der in Tabelle 5 dargestellten Instanz beispielsweise sieht wie folgt aus:

$\mathcal{S}_1 = \{$ `nyse::preise(Ticker, Datum, Kurs, Preis)`,
`tse::kurse(Ticker, Datum, (Preis, {Kurs}))`,
`bse::preise(Datum, (Preis, {Kurs, Ticker}))`,
`mse::(preise, {Ticker})(Datum, (Preis, {Kurs}))` $\}$

Die Intuition hinter dieser Definition ist, dass ein komplexer Attribut-Name (bzw. Relations-Name) in einer Instanz des Schemas in eine Menge von komplexen Spalten-Beschriftungen (bzw. Relationen-Beschriftungen) umgesetzt wird. Beispielsweise könnte der komplexe Attributname `(Preis, {Kurs, Ticker})` im Schema durch die Menge {`Preis FOR Kurs` = hoch `AND Ticker` = ibm, ..., `Preis FOR Kurs` = tief `AND Ticker` = ms} von Spalten-Beschriftungen realisiert werden (in Tabelle 5 wurden diese Namen abgekürzt zu hoch bzw. tief). Das Konzept ist in diesem Fall also `Preis` und die Merkmalsmenge sind der `Kurs` und der `Ticker`. Damit dem Datenbanksystem diese Information zugänglich ist, wird eine Katalog-Datenbank genutzt, in der Konzepte und Merkmale der Attribute und Relationen gespeichert werden.

5.2 Syntax und Semantik

Im Folgenden wird ein Teil der Syntax von nD-SQL beschrieben, indem einige Erweiterungen zum Standard-SQL aufgezeigt werden. Die Semantik wird in erster Linie anhand eines Beispiels erläutert werden, das von Gingras und Lakshmanan [GL98] übernommen ist. Die Relationen in den Beispieldatenbanken sind in Tabelle 5 dargestellt. Wie man sieht, sind in diesem Beispiel eine klassische Relation mit ausschließlich einfachen Attribut- und Relationsnamen (`nyse::preise`) sowie einige Relationen enthalten, die auch komplexe Namen enthalten und bei denen daher Werte im Schema auftauchen (z.B. `tse::kurse`).

5.2.1 Mehrdimensionalität und Restrukturierung

In nD-SQL werden die Schlüsselwörter `SELECT`, `FROM`, `WHERE`, `GROUP BY` und `HAVING` vom Standard-SQL übernommen und erweitert. Eine Anfrage, die die Daten aus den Relationen der Datenbank `mse` in eine Relation mit der gleichen Struktur wie die Relation `preise` aus der Datenbank `nyse` restrukturiert, könnte beispielsweise so aussehen:

```
SELECT R.Ticker, T.Datum, C.Kurs, T.C AS Preis
  FROM mse -> R, mse::R T, mse::R -> C
 WHERE R HASA Ticker AND C ISA Preis
```

Ticker	Datum	Kurs	Preis
ibm	27.10.2003	Eröffnung	63,67
...	...	...	...
ibm	27.10.2003	Schluss	62,56
...	...	...	...
ms	01.11.2003	Tief	44,60
...	...	...	...

nyse::preise

Ticker	Datum	Tief	Hoch	...
ibm	27.10.2003	62,00	64,00	...
...	...	...	...	...
ms	01.11.2003	46,00	48,72	...
...	...	...	...	...

tse::kurse

Datum	Eröffnung, ibm	Eröffnung, ms	...	Schluss, ibm	Schluss, ms	...
27.10.2003	59,89	45,00	...	62,05	46,17	...
...	...	...	...	...	...	...
01.11.2003	60,89	43,98	...	62,05	46,17	...
...	...	...	...	...	...	...

bse::preise

Datum	Tief	Hoch	...
27.10.2003	58,21	59,05	...
...	...	...	...
01.11.2003	55,75	63,00	...
...	...	...	...

mse::ibm

Datum	Tief	Hoch	...
27.10.2003	48,21	49,05	...
...	...	...	...
01.11.2003	65,75	67,00	...
...	...	...	...

mse::ms

Tabelle 5: Eine föderierte Datenbank mit heterogenen Schemata, in der Börsenkurse gespeichert werden. Die Notation `db::rel` bedeutet dass die Datenbank `db` die Relation `rel` enthält.

Wie bereits zuvor beschrieben, ist das Schema für die Datenbank `mse` definiert als: `mse`::(`preise`, {`Ticker`}), (`Datum`, (`Preis`, {`Kurs`})). Diese Anfrage arbeitet also auf den Relationen `R` in `mse`, die das Merkmal `Ticker` besitzen (spezifiziert durch die Bedingung `R HASA Ticker`). Der Name des Tickers (`R.Ticker`) wird dann in die erste Spalte des Ergebnisses eingetragen. Die Tupel aus diesen Relationen erhalten den Alias `T` und steuern die zweite Spalte (`T.Datum`) zum Ergebnis bei. Für die Spalten `C` der Relation `mse::R`, die `Preis` als Konzept haben (spezifiziert durch `C ISA Preis`), wird dann jeweils ein Ergebnis-Tupel für jeden Wert des Merkmals `Kurs` erzeugt. Dieser Wert des Merkmals `Kurs` wird in die dritte Spalte der Ergebnisrelation eingetragen. Der eigentliche Wert, der in dieser Spalte steht (also der Preis), bildet die letzte Spalte der Ergebnisrelation, die dann im Ergebnis als `Preis` erscheint.

Bei dieser ersten Beispielanfrage wurden komplexe Spalten und Relationen in einfache Spalten und Relationen restrukturiert. Natürlich ist auch eine Restrukturierung in die umgekehrte Richtung möglich, nämlich die Erzeugung von komplexen Spalten und Relationen.

Beispielsweise lassen sich die Daten aus der Tabelle `nyse::preise` in das Format der Tabelle `tse::kurse` restrukturieren:

```
SELECT T.Ticker, T.Datum, T.Preis AS T.Kurs FOR T.Kurs,
  FROM nyse::preise T
```

Hierbei wird mittels `T.Preis AS T.Kurs FOR T.Kurs` angegeben, dass der Preis für einen Kurs unter der Spalten-Beschriftung des Kurses in die Ergebnisrelation eingehen soll. Das heißt, es wird für jeden Wert, den `Kurs` in der Ursprungsrelation hat, eine Spalte erzeugt, in die dann der entsprechende Preis eingetragen wird. Es entsteht also die komplexe Spalte `(Preis, {Kurs})` wie im Schema von `tse::preise`. Möchte man komplexe Relationen erzeugen, schreibt man die Argumente der `SELECT`-Klausel in Klammern und dahinter eine weitere `FOR`-Subklausel.

5.2.2 OLAP-Erweiterungen: Mehrfach-Visualisierungen und Subaggregate

Der in Abschnitt 2 vorgestellte Data-Cube-Operator realisiert eine Aggregation von Daten auf exponentiell vielen Gruppierungs-Hierarchieebenen. In der Praxis werden jedoch oft nicht alle dieser Aggregats-Werte benötigt, weshalb in Abschnitt 2 neben dem `CUBE`-Operator zusätzlich der `ROLLUP`-Operator eingeführt wurde. Das Konzept von nD-SQL geht hier noch einen Schritt weiter und erlaubt beliebige Teilmengen dieser Gruppierungen. Zu diesem Zweck wird eine neue Variable `DIM` definiert, die alle Namen der logischen Dimensionen der Anfrage (das sind die Namen der in der Abfrage vorkommenden Konzepte und Merkmale) abdeckt, außer denen, die aggregiert werden. Eine nD-SQL -Anfrage A mit Dimensionsvariablen ist äquivalent zu der Vereinigung einer Menge von Anfragen ohne Dimensionsvariablen, die der Anfrage A entsprechen, wobei alle Dimensionsvariablen zu allen möglichen Dimensionsnamen instanziiert wurden, die den Bedingungen der `WHERE`-Klausel entsprechen.

Um beispielsweise einen Data-Cube wie in Abschnitt 2 beschrieben zu erzeugen, könnte man folgende Anfrage verwenden:

```
  SELECT X, Y, Z, SUM(T.Price)
    FROM nyse::preise T, DIM X, Y, Z
   WHERE X < Y < Z AND DIMS CAN BE NONE
GROUP BY X, Y, Z
```

In dieser Anfrage können die Dimensionsvariablen Werte aus der Menge {`T.Ticker`, `T.Datum`, `T.Kurs`, `NONE`} annehmen, wobei das Schlüsselwort `NONE` eine ähnliche Funktion wie der im Anschnitt 2 verwendete `ALL`-Wert hat. Die Relation `<`, die auf den Dimensionsvariablen verwendet wird, wird dabei bezüglich der alphabetischen Ordnung der Dimensionsnamen interpretiert.

Dass sich Restrukturierungs-Möglichkeiten und Aggregate auf mehreren Hierarchieebenen auch kombinieren lassen, ist eine der wesentlichen Stärken von nD-SQL . Dazu kann man Dimensionsvariablen in der Restrukturierung verwenden und dadurch mehrere Visualisierungen derselben Daten erhalten, indem die Relations- und Zeilen-Dimensionen mit Variablen belegt werden, wie das folgende Beispiel zeigt:

```
  SELECT (AVG(T.Preis) AS Y FOR Y) AS X FOR X
    FROM nyse::preise T, DIM X, Y
   WHERE DIMS IN {T.Date, T.Measure, T.Ticker}
GROUP BY X, Y
```

6 Zusammenfassung

Wie in den vorherigen Abschnitten deutlich geworden ist, sind die Ansätze für OLAP-Anfragesprachen sehr vielfältig. Einige sind bereits realisiert worden, wie der Data Cube, der bereits Teil des SQL:1999-Standards geworden ist oder die Multidimensional Expressions, die von Microsoft und auch anderen Firmen bereits in kommerzielle Produkte integriert werden. Andere Ansätze wie die Regel-basierten Sprachen und auch nD-SQL sind vor allem in wissenschaftlichen Arbeiten zu finden. Drei der Ansätze (Data Cube Operator, MDX, nD-SQL) haben den bequemen Weg gewählt, auf die bereits existierende und weit verbreitete Sprache SQL zurückzugreifen und deren syntaktische Elemente, die vielen bereits vertraut sind, übernommen. Im Falle von MDX und nD-SQL wurde jedoch auch deren Semantik stark verändert, so dass es hier zu Verwirrungen kommen kann.

Das Spektrum der Sprachen für diesen Zweck ist mit den hier vorgestellten Ansätzen jedoch längst nicht vollständig dargestellt. Es gab und gibt eine ganze Reihe weiterer Entwicklungen, die zu Teil auf den hier vorgestellten aufbauen oder auch ganz andere Wege gehen. Vor allem in den 90er Jahren des 20. Jahrhunderts wurden viele Ansätze entwickelt und teilweise wieder verworfen. Durch neue SQL-Standards setzt nun langsam eine Bewegung ein, die die Erfahrungen mit den neuen Konzepten nutzt und auch hier einen einheitlichen Standard schaffen möchte.

Literatur

[GCB+97] Gray, Jim ; Chaudhuri, Surajit ; Bosworth, Adam ; Layman, Andrew ; Reichart, Don ; Venkatrao, Murali ; Pellow, Frank ; Pirahesh, Hamid: Data Cube: A Relational Aggregation Operator Generalizing Group-By, Cross-Tab, and Sub-Totals. In: *J. Data Mining and Knowledge Discovery* 1 (1997), Nr. 1, S. 29–53

[GL98] Gingras, Frédéric ; Lakshmanan, Laks V. S.: nD-SQL: A Multi-Dimensional Language for Interoperability and OLAP. In: Gupta, Ashish (Hrsg.) ; Shmueli, Oded (Hrsg.) ; Widom, Jennifer (Hrsg.): *VLDB'98, Proceedings of 24th International Conference on Very Large Data Bases, August 24-27, 1998, New York City, New York, USA*, Morgan Kaufmann, 1998. – ISBN 1–55860–566–5, S. 134–145

[HMR97] Hacid, Mohand-Said ; Marcel, Patrick ; Rigotti, Christophe: A rule based data manipulation language for OLAP systems / LuFg Theoretical Computer Science, RWTH Aachen. Springer, 1997 (LTCS-97-05). – LTCS-Report

[MW97] Moran, Brian ; Whitney, Russ: Getting to know OLAP and MDX. In: *SQL Server Magazine* (April 1997). – online, verfügbar unter: `http://www.sqlmag.com/Articles/Print.cfm?ArticleID=5112`

[Nol99] Nolan, Carl: Introduction to Multidimensional Expressions (MDX). In: *Microsoft Developer Network* (1999). – online, verfügbar unter: `http://msdn.microsoft.com/library/en-us/dnolap/intromdx.asp`

[Pea02] Pearson, William E.: Introducing the SQL Server 'MDX in Analysis Services' Series. In: *Database Journal* (2002). – online, verfügbar unter: `http://www.databasejournal.com/news/article.php/1550061`

[Sch03] Schaefer, Burkhard: *Logische Modelle für On-Line Analytical Processing*. 2003. – Seminararbeit Universität Kaiserslautern

[The97] The OLAP Council: APB-1 OLAP Benchmark. (1997). – online, verfügbar unter: `http://www.olapcouncil.org/research/spec1.htm`