Analyse und Vergleich von Zugriffstechniken für funktionale Aspekte in RDBMS

Matthias Liebisch
Friedrich-Schiller-Universität Jena
Lehrstuhl für Datenbanken und Informationssysteme
Ernst-Abbe-Platz 2
07743 Jena
m.liebisch@uni-jena.de

KURZFASSUNG

Neben klassischen fachlichen Anforderungen existieren in Anwendungssystemen oft auch querschnittliche Belange, deren Funktionalität sich nicht einfach kapseln bzw. modularisieren lässt. Vertreter dieser sogenannten *funktionalen Aspekte* sind beispielsweise die mehrsprachige oder versionierte Darstellung und Verwaltung von Anwendungsdaten. Nachdem sich in der Software-Entwicklung seit einigen Jahren die Aspektorientierte Programmierung als Lösung etabliert hat, bietet das neuartige Paradigma der Aspektorientierten Datenhaltung ein entsprechendes Konzept zur Abbildung querschnittlicher Belange in einem relationalen Datenmodell. Dabei stehen vor allem die Unabhängigkeit vom Prozess der fachlichen Modellierung und ein hoher Wiederverwendungsgrad im Vordergrund. Basierend auf dem zu diesem Zweck entwickelten Referenzmodell untersucht der vorliegende Beitrag unterschiedliche Techniken für den Zugriff auf jene funktionalen Aspekte. Diese werden anschließend anhand wesentlicher Bewertungskriterien einer Evaluation unterzogen und miteinander verglichen.

Kategorien und Themenbeschreibung

H.4 [**Information Systems Applications**]: Miscellaneous; H.2.3 [**Database Management**]: Languages—*Query languages*

Allgemeine Bestimmungen

Design, Languages, Performance

1. EINLEITUNG

Seit der Beschreibung des relationalen Modells[4] Anfang der 1970er Jahre ist die Bedeutung auf diesem Modell basierender Datenbankmanagementsysteme (RDBMS) als Persistierungsebene stetig gewachsen. Heutzutage bilden relationale Datenbanksysteme die Grundlage für die vielfältigsten Anwendungssysteme und sind damit aus den meisten alltäglichen Prozessen nicht mehr wegzudenken. Die Entwicklung derartiger Applikationen ist deswegen auch immer mit dem Entwurf eines für den Einsatzzweck geeigneten Datenmodells verbunden. Dieses sollte unter Beachtung verschiedener Kriterien, wie beispielsweise Benutzbarkeit und Wiederverwendbarkeit, die modularisierte Speicherung der fachlichen Datenobjekte in relationalen Strukturen optimal unterstützen. Neben dieser Abbildung existieren jedoch häufig zusätzlich anwendungsweite Anforderungen wie beispielsweise die Unterstützung von Mehrsprachigkeit oder Versionierung, welche als sogenannte **funktionale Aspekte**[11] Einfluss auf das gesamte Datenmodell haben.

Dieses Problem der *cross-cutting concerns* ist bereits aus dem Umfeld der Objektorientierten Programmierung seit einigen Jahren bekannt und hat zur Entwicklung der Aspektorientierten Programmierung[3] geführt. Im übertragenen Sinne stellt die Aspektorientierte Datenhaltung[11] ein Modellierungsparadigma dar, um funktionale Aspekte in einem Datenmodell gekapselt und unabhängig von den fachlichen Datenobjekten zu integrieren. Triviale Ansätze, wie beispielsweise die Erweiterung relevanter Tabellen um eine zusätzliche Spalte zur Festlegung der Locale im Fall mehrsprachiger Datenhaltung, sind meist nur auf konkrete Anwendungsfälle zugeschnitten und versagen zudem bei der Unterstützung beliebig vieler funktionaler Aspekte unter den Anforderungen des Paradigmas der Aspektorientierten Datenhaltung [19]. Ein generischer Ansatz zur Lösung der angedeuteten Herausforderungen ist das in [12] beschriebene Referenzmodell. Darauf basierend zeigt der vorliegende Beitrag verschiedene Alternativen für den Zugriff und die Nutzung funktionaler Aspekte aus Sicht der Anwendung auf.

Nachfolgend werden in Abschnitt 2 das erwähnte Referenzmodell sowie ein kleines Anwendungsbeispiel kurz vorgestellt. Die darauf aufbauenden Zugriffstechniken stehen im Fokus von Abschnitt 3, bevor sie in Abschnitt 4 einer Bewertung unterzogen werden. Schließlich fasst Abschnitt 5 die Ergebnisse der Arbeit nochmal zusammen.

2. REFERENZMODELL

Für die vom fachlichen Datenmodell unabhängige und gekapselte Persistierung aspektspezifischer Daten wurde in [12] ein Referenzmodell vorgestellt und beschrieben, welches mit geringfügigen Anpassungen bezüglich der Fremdschlüsseldefinitionen in den Tabellen zur Aspektverknüpfung auch in diesem Beitrag zum Einsatz kommt.

23^{rd} *GI-Workshop on Foundations of Databases (Grundlagen von Datenbanken)* 31.5.-03.06.2011, Obergurgl, Austria.

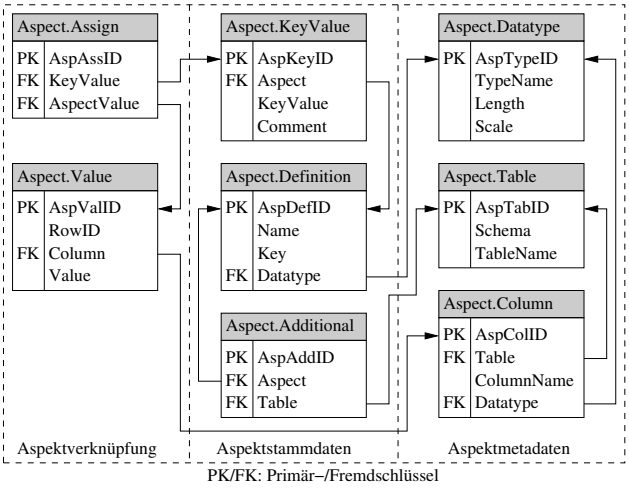
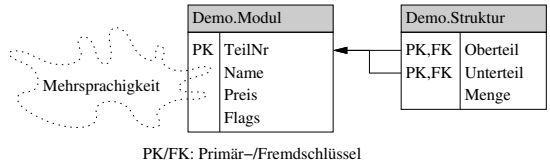


Abbildung 1: Referenzmodell

Zentraler Bestandteil dieses in Abbildung 1 skizzierten Referenzmodells sind die beiden Tabellen ASPECT.VALUE und ASPECT.ASSIGN, welche die Speicherung aspektspezifischer Attributwerte sowie deren Zuordnung zu einer konkreten Aspektausprägung (z.B. Locale 'en' im Aspekt Mehrsprachigkeit) für eine Fachtabellenzelle realisieren. Daneben existieren weitere Tabellen zur Verwaltung von Metadaten, wie beispielsweise ASPECT.KEYVALUE für die Spezifikation von Ausprägungswerten zu allen im System definierten Aspekten oder ASPECT.COLUMN zur Hinterlegung aspektrelevanter Attribute der Fachtabellen.

Die Anforderungen aus dem Paradigma der Aspektorientierten Datenhaltung werden insbesondere durch das Entity-Attribute-Value-Konzept[15] (EAV) gewährleistet, welches für ASPECT.VALUE und ASPECT.KEYVALUE zur Anwendung kommt. Die damit verbundenen Konsequenzen[7] bezüglich der Komplexität von direkten SQL-Anfragen im Referenzmodell erfordern die Analyse alternativer Zugriffsarten.



►Anfrage: *Ermittle für das Modul mit TeilNr=4711 alle mehrsprachigen Daten (*NAME *und* PREIS*) sowie den jeweiligen Aspektschlüsselwert als Locale.*

Abbildung 2: Beispiel für Datenmodell mit Anfrage

Zur Veranschaulichung der in Abschnitt 3 folgenden Techniken soll das in Abbildung 2 dargestellte Beispiel eines vereinfachten Datenmodells zur Verwaltung von Stücklisten dienen, in dem der Aspekt „Mehrsprachigkeit" für die Attribute DEMO.MODUL.NAME sowie DEMO.MODUL.PREIS aktiviert wurde. Darauf aufbauend soll jeweils die Beantwortung der zugehörigen Beispiel-Anfrage erläutert werden, welche das Prinzip für den Zugriff auf aspektspezifische Attributwerte in einem konkreten fachlichen Anwendungskontext verdeutlichen soll.

3. ZUGRIFFSTECHNIKEN

Dieser Abschnitt beschreibt verschiedene Möglichkeiten für den Zugriff auf funktionale Aspekte, welche mit Hilfe des in Abbildung 1 präsentierten Referenzmodells in ein fachliches Datenmodell integriert werden. Aufgrund der Tatsache, dass das relationale Modell als Grundlage dient, ist der direkte Zugriff mittels SQL auf die entsprechenden Strukturen die naheliegendste Möglichkeit. Allerdings stellt die zentrale Tabelle ASPECT.VALUE eine neue Herausforderung für die Anfragegenerierung dar, weil darin enthaltene Daten durch das verwendete EAV-Prinzip[15] einer sogenannten unpivotisierten („gekippten") Speicherungsform unterliegen. Dies hat zur Konsequenz, dass zu jedem Attribut (COLUMN) eines traditionellen Tupels (identifizierbar über ROWID) der jeweilige Wert (VALUE) in einer eigenen Zeile gespeichert wird. Da jedoch die klassische relationale Verarbeitung von Datensätzen mit zugehörigen Attributen als Tabellenspalten ausgeht, ist eine Pivotisierung („rows to columns") notwendig, sobald in der Anfrage die Tabelle ASPECT.VALUE involviert wird. Die anschließenden Abschnitte beschreiben drei Konstrukte in SQL sowie einen applikativen Ansatz, um die genannte Transformation zu unterstützen.

3.1 SQL mit JOIN

Bei einer Beschränkung auf normierte Sprachmittel ist die erforderliche Transformation nur mittels JOIN-Operatoren realisierbar, da weder im SQL:92-Standard[6] als Grundlage für das Paradigma der Aspektorientierten Datenhaltung[11] noch in der aktuellen SQL:2008-Norm[10] dedizierte Operatoren zur Pivotisierung einer Tabelle existieren. Das prinzipielle Vorgehen ist exemplarisch für die in Abbildung 2 formulierte Anfrage in Abbildung 3 dargestellt. Dabei wurde auf die Formatierung der Ergebnisattribute entsprechend den zugeordneten Datentypen in Tabelle ASPECT.DATATYPE verzichtet und zwecks Übersichtlichkeit die Kenntnis gewisser Metadaten wie Idents von Aspekten und Tabellenspalten als bekannt vorausgesetzt.

```
SELECT T1.Value AS Name, T2.Value AS Preis,
       T5.KeyValue AS Locale
  FROM Aspect.Value T1
       INNER JOIN Aspect.Value T2
         ON T1.RowID = T2.RowID
       INNER JOIN Aspect.Assign T3
         ON T1.AspValID = T3.AspectValue
       INNER JOIN Aspect.Assign T4
         ON T2.AspValID = T4.AspectValue
       INNER JOIN Aspect.KeyValue T5
         ON T3.KeyValue = T5.AspKeyID
 WHERE T1.Column = 1 -- /* 'Name' */
   AND T2.Column = 2 -- /* 'Preis' */
   AND T3.KeyValue = T4.KeyValue
   AND T5.Aspect = 1 -- /* 'Mehrsprachigkeit' */
   AND T1.RowID = 4711
```

Abbildung 3: Anfrage mit JOIN

Bereits für das simple in Abbildung 3 präsentierte Beispiel ist die Komplexität der Anfragegenerierung inklusive Pivotisierung der Tabelle ASPECT.VALUE erkennbar. Insbesondere skalieren die notwendigen JOIN-Operatoren linear mit den Attributen im Ergebnisschema. Dabei werden jeweils

die Tabellen ASPECT.VALUE und ASPECT.ASSIGN miteinander verbunden, welche für die Speicherung aller aspektspezifischen Ausprägungen von Attributwerten in Fachtabellen zuständig sind und dadurch mit Abstand die umfangreichsten Mengengerüste aufweisen. Erwartungsgemäß haben derartige Anweisungen eine oft inakzeptable Verarbeitungszeit wie entsprechende Analysen gezeigt haben[13].

3.2 SQL mit PIVOT

Verlässt man die SQL-Norm auf der Suche nach adäquater Unterstützung für die Pivotisierung von EAV-Tabellen, dann zeigt sich, dass DBMS-Hersteller bereits produktspezifische SQL-Erweiterungen mit den Operatoren PIVOT und UNPIVOT[20] anbieten. Unter anderem finden sich derartige Implementierungen in Datenbanksystemen wie Microsoft SQL Server 2005[18] oder Oracle 11g[14]. Ein typisches Anwendungsgebiet für diese Transformationen sind OLAP-Anfragen im Bereich Data Warehouse[17], deren Blickwinkel geändert werden soll (beispielsweise die Gruppierung nach Regionen statt Produkten in einer Umsatzübersicht). Durch Nutzung von PIVOT und UNPIVOT kann eine gezielte Optimierung auf Basis der klassischen Operatoren wie Verbund oder Projektion erfolgen[5].

```
SELECT PivotedData.[1] AS Name,
       PivotedData.[2] AS Preis,
       PivotedData.KeyValue AS Locale
  FROM
  (
    SELECT T1.RowID, T1.Column, T1.Value,
           T3.KeyValue
      FROM Aspect.Value T1
           INNER JOIN Aspect.Assign T2
             ON T1.AspValID = T2.AspectValue
           INNER JOIN Aspect.KeyValue T3
             ON T2.KeyValue = T3.AspKeyID
     WHERE T3.Aspect = 1 -- /*'Mehrsprachigkeit'*/
       AND T1.RowID = 4711
  ) AS JoinData
  PIVOT
  (
    MAX(JoinData.Value)
    FOR JoinData.Column IN ([1], [2])
    -- /* 1 und 2 = relevante Column-IDs */
  ) AS PivotedData
```

Abbildung 4: Anfrage mit PIVOT

Aber auch die Verarbeitung von EAV-Tabellen wie hier im Kontext der Aspektorientierten Datenhaltung ist mit Hilfe des PIVOT-Operators möglich. Abbildung 4 demonstriert dessen Verwendung unter Microsoft SQL Server 2005 für die Beispielanfrage in Abbildung 2. Auf den ersten Blick verwirrt dabei der Ausdruck `MAX(JoinData.Value)`, welcher die notwendige Aggregatbildung bei der Pivotisierung übernimmt. Diese Funktion lässt das Ergebnis jedoch inhaltlich korrekt, solange jede Gruppe bezüglich der gruppierten Attribute (`T1.RowID, T3.KeyValue`) und den zu Spalten umgewandelten Werten aus `T1.Column` nur einen Datensatz enthält. Um dies unter Beachtung der getrennten Speicherung der aspektspezifischen Werte (ASPECT.VALUE) und den tatsächlichen Fachtabellen-Zuordnungen (ASPECT.ASSIGN) sicherzustellen, müssen die beiden genannten Tabellen zusammen mit der Tabelle ASPECT.KEYVALUE verbunden werden. Anschließend kann über die projezierte Attributmenge sinnvoll pivotisiert werden.

Zusätzlich ist bei der Nutzung des PIVOT-Operators zu beachten, dass für die `IN`-Klausel nur eine fest definierte Spaltenmenge angegeben werden kann. Ein Ausdruck der Form `SELECT Column FROM Aspect.Value` ist beispielsweise nicht zulässig. Dies gilt sowohl für Microsoft SQL Server 2005[1] als auch für Oracle 11g[2]. Wird eine derartige Dynamik dennoch benötigt, lässt sich diese nur über eine Stored Procedure, vorgeschalteten Anwendungscode oder im Fall von Oracle unter Verwendung von XML realisieren.

3.3 SQL mit Spracherweiterung

Aufgrund der fehlenden Normierung des PIVOT-Operators einerseits und dessen Nutzungs-Einschränkungen im Kontext der Aspektorientierten Datenhaltung andererseits, verfolgt dieser Abschnitt die Idee einer SQL-Erweiterung für einen adäquaten Zugriff auf funktionale Aspekte im Referenzmodell. Die neuen Sprachelemente beeinflussen sowohl den DML-Teil als auch den DDL-Bereich, um beispielsweise für eine Tabellenspalte relevante Aspekte definieren zu können. Hier soll jedoch aus Platzgründen nur das `SELECT`-Statement im Fokus stehen.

```
<table reference> ::=
   <table name> [ <correlation specification> ]
 | <derived table> <correlation specification>
 | <joined table>
 | <aspect view>

<aspect view> ::=
  ASPECTVIEW <identifier> BASED ON <table name>
  GROUP BY <aspect key> [{, <aspect key>} ... ]

<aspect key> ::=
  ASPECTKEY(<character string literal>)
  AS <identifier>
```

Abbildung 5: SQL-Erweiterung ASPECTVIEW

Aufbauend auf dem SQL:92-Standard[6] wird das Nichtterminalsymbol `<table reference>`[3] um eine weitere Alternative `<aspect view>` ergänzt, deren Definition in Abbildung 5 dargestellt ist. Das neue Schlüsselwort `ASPECTVIEW` erzeugt dabei eine Sicht auf alle aspektspezifischen Daten der über `BASED ON` angegebenen (fachlichen) Basistabelle. Da es möglich ist, einer Tabellenspalte verschiedene Aspekte zuzuordnen, erfolgt die Aufbereitung der Daten in gruppierter Form bezüglich des einattributigen Primärschlüssels[11] und der über `<aspect key>` spezifizierten Aspekte. Diese stehen anschließend im Rahmen der Anfrageformulierung als reguläre Attribute der Sicht zur Verfügung, für eine Tabelle mit n Attributen und insgesamt k zugeordneten Aspekte ergibt sich also das in Abbildung 6 dargestellte Relationsschema. Voraussetzung hierfür ist eine im Referenzmodell ver-

[1] `http://msdn.microsoft.com/en-us/library/ms177634.aspx`

[2] `http://download.oracle.com/docs/cd/B28359_01/server.111/b28286/statements_10002.htm#CHDFAFIE`

[3] `http://savage.net.au/SQL/sql-92.bnf`

ankerte UNIQUE-Bedingung auf ASPECT.DEFINITION.KEY, dessen Werte über den Ausdruck `ASPECTKEY(<character string literal>)` referenziert werden.

$$(\overbrace{Att_1, \ldots, Att_n}^{\text{Basistabelle}}, \overbrace{Asp_1, \ldots, Asp_k}^{\text{Aspekte}})$$

Abbildung 6: Relationsschema von ASPECTVIEW

Angewendet auf das in Abbildung 2 skizzierte Beispiel ergibt sich eine gegenüber den bisherigen Zugriffstechniken sehr kompakte Anfrage, wie Abbildung 7 zeigt. Neben dem Aufwand zur Pivotisierung für die Aspektsicht `T1` ist unabhängig von der Anzahl der Attribute oder Aspekte nur noch eine JOIN-Operation erforderlich, um für die Idents in Asp_i die eigentlichen Aspektschlüsselwerte anzeigen zu können.

```
SELECT T1.Name, T1.Preis, T2.KeyValue AS Locale
  FROM Aspect.KeyValue T2 INNER JOIN
       (
         ASPECTVIEW ModulAspects
         BASED ON Demo.Modul
         GROUP BY ASPECTKEY('Locale') AS AspLoc
       ) T1
       ON T2.AspKeyID = T1.AspLoc
 WHERE T1.TeilNr = 4711
   AND T2.Aspect = 1 -- /*'Mehrsprachigkeit'*/
```

Abbildung 7: Anfrage mit ASPECTVIEW

Sollte sich eine derartige Spracherweiterung wie angedeutet für alle Bereiche (DDL und DML) als geeignetes Ausdrucksmittel im Umgang mit funktionalen Aspekten beweisen, bleibt die praktische Nutzung stark eingeschränkt, solange eine Umsetzung in SQL-Norm oder DBMS-Produkten fehlt. In solch einem Szenario kann der Einsatz sogenannter Proxys[1] oder Query Transformation Layer[2] zwischen Anwendung und Datenbank eine Lösung darstellen. Im vorliegenden Fall müssten alle neu definierten SQL-Ausdrücke, beispielsweise `ASPECTVIEW`, durch einen Parser auf effiziente Art und Weise in genormte oder produktspezifische SQL-Anweisungen transformiert werden. Damit wäre zumindest eine benutzerfreundliche Schnittstelle im Umgang mit funktionalen Aspekten unter Nutzung der trivialen Umformung (Anfrage mit JOIN, siehe Abschnitt 3.1) geschaffen. Die nachfolgend beschriebene Zugriffstechnik ist ebenfalls ein Vertreter dieser Kategorie, allerdings erfolgt die Anfrageformulierung nicht auf Basis von SQL.

3.4 Funktionsbaustein mit API

Gegenüber den bisher vorgestellten Möglichkeiten, den Zugriff auf funktionale Aspekte allein mit SQL-Mitteln auf der Persistierungsebene zu realisieren, wird mit dem vierten Ansatz eine applikationsseitige Verarbeitung verfolgt. Analog zum Konzept mit JDBC[9] eine einheitliche Schnittstelle für relationale Datenbanken zu etablieren, ermöglicht der hier vorgestellte Funktionsbaustein den Zugriff auf funktionale Aspekte, welche im Referenzmodell persistiert sind. Die zugehörige API[16] umfasst Methoden sowohl zur Verwaltung der Aspekte inklusive ihrer Metadaten als auch zur Abfrage und Änderung aspektspezifischer Attributwerte unter Beachtung sogenannter Aspektfilter und Aspektkontexte.

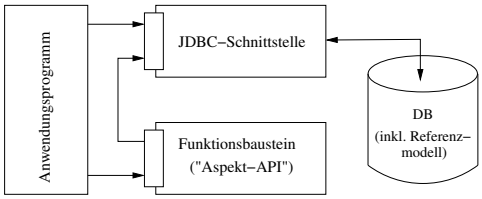


Abbildung 8: Anwendungs-Integration der API

Um einen möglichst plattformunabhängigen und universellen Funktionsbaustein bereitstellen zu können, ist dieser inklusive seiner API in Java spezifiziert. Da in den meisten Fällen bei Verwendung einer Java-Bibliothek das umgebende Anwendungsprogramm ebenfalls auf Java basiert und für Datenbankzugriffe die JDBC-Schnittstelle genutzt wird, zeigt Abbildung 8 ein typisches Integrations-Szenario. Dabei können in einer ersten Ausbaustufe über die API tatsächlich nur aspektspezifische Daten abgefragt und geändert werden, die Verarbeitung der fachlichen Daten erfolgt unverändert über die bereits existierende Datenbank-Schnittstelle. Aufgabe der Anwendung ist damit also die Zusammenführung der Ergebnisse beider Datenquellen, abhängig natürlich von den inhaltlichen Anforderungen.

```
1  // Deklarationen
   AspectManager am = /* Verweis auf Instanz holen */
3  AspectCatalogManager acm =
     am.getAspectCatalogManager();
5  int aspLang = acm.lookupAspectID("Language");
   int tID = acm.lookupTableID("Demo","Modul");
7  int cNameID = acm.lookupColumnID(tID, "Name");
   int cPriceID = acm.lookupColumnID(tID, "Preis");
9
   // Anfragespezifikation
11 QueryStatement st = am.createQueryStatement(tID);
   st.setRowIDs(new long[] {4711});
13 st.setColumnSet(am.createColumnSet(
     new int[] {cNameID, cPriceID}));
15
   // Ergebnisverarbeitung
17 ResultRows result = st.execute();
   AspectSet aSet = st.getAspectSet();
19 ColumnSet cSet = st.getColumnSet();
   int aspLangIdx = aSet.getIndex(aspLang);
21 int cNameIdx = cSet.getIndex(cNameID);
   int cPriceIdx = cSet.getIndex(cPriceID);
23 while (result.next())
   {
25   long rowID = result.getRowID();
     AspectContextElement e =
27     result.getContext().getElement();
     String name = result.getValueString(cNameIdx);
29   String price = result.getValueString(cPriceIdx);
     int localeID = e.getIndexKeyValue(aspLangIdx);
31 }
```

Abbildung 9: Anfrage mit Funktionsbaustein

Eine Abfrage aspektspezifischer Daten wie für das Referenz-Beispiel wird prinzipiell durch das in Abbildung 9 dargestellte Code-Fragment realisiert. Der Einstieg erfolgt mit einer implementierungsspezifischen Instanz der zentralen Klasse `AspectManager` (Zeile 2), welche im nächsten Schritt zur Erzeugung einer `AspectCatalogManager`-Instanz benötigt wird und zudem auch für die Verwaltung einer Datenbank-Session zuständig ist. Schließlich werden zum Aspekt Mehrsprachigkeit sowie für die Attribute NAME und PREIS der Tabelle MODUL die Idents ermittelt (Zeilen 5-8). Hierfür können entsprechende Methoden an der Aspektkatalogmanager-Klasse genutzt werden, damit sind alle für die Anfrage notwendigen Metadaten bekannt.

Für die Anfragespezifikation wird über den `AspectManager` eine Instanz der Klasse `QueryStatement` angelegt (Zeile 11). Da in der relevanten Fachtabelle DEMO.MODUL das Attribut TEILNR bereits die Anforderung eines einattributigen Primärschlüssels erfüllt, entspricht die Filterung mit der Methode `setRowIDs` in Zeile 12 genau der Einschränkung auf das Modul mit *TeilNr=4711* in einer WHERE-Klausel. Die zu projezierenden Spalten werden analog in Zeile 13 festgelegt. Weitere zusätzliche aspektspezifische Beschränkungen beispielsweise über die `AspectFilter`-Klasse entfallen, da alle mehrsprachigen Daten für die Attribute NAME und PREIS bereitzustellen sind.

Die Ausführung der Anfrage erzeugt ein `ResultRows`-Objekt (Zeile 17), welches analog zum entsprechenden Konstrukt in JDBC zeilenweise über den `next`-Iterator verarbeitet werden kann (Zeile 23). Zuvor sind für die korrekte Auswertung der Ergebnisstruktur die darin enthaltenen Indexe des Mehrsprachigkeits-Aspekts sowie der beiden angefragten Attribute zu ermitteln (Zeilen 18-22). Mit Hilfe dieser Indexe kann nun auf die entsprechenden Informationen zugegriffen werden (Zeilen 25-29). Weiterführende Details zu Datenstrukturen, Klassen und Methoden sowie zu deren Verwendung finden sich bei PIETSCH[16].

4. VERGLEICH

Nachdem der vorangegangene Abschnitt 3 die unterschiedlichen Zugriffstechniken präsentiert hat, dient dieser Abschnitt der Bewertung und dem Vergleich jener Techniken. Zuerst werden die dafür notwendigen Kriterien im Folgenden beschrieben und anschließend für jeden Ansatz evaluiert.

4.1 Kriterien

Ausgehend vom Paradigma der Aspektorientierten Datenhaltung spiegelt sich die Forderung nach Universalität[11] bezüglich SQL:92 im Kriterium der **Standardisierung** wider, welches hier jedoch auch bezüglich der Existenz einer ISO-Norm erweitert werden soll. In direktem Zusammenhang dazu stellt sich die Frage der **Praxisrelevanz** eines Ansatzes, d.h. ob mit diesem der Zugriff auf funktionale Aspekte unter den aktuellen Gegebenheiten überhaupt praktisch realisierbar ist. Eine große Bedeutung spielen natürlich auch Ausagen zur **Performance**, allerdings liegen nur für den ersten Ansatz (SQL mit JOIN) tatsächlich konkrete Messwerte[13] vor, sodass für alle anderen Varianten nur eine qualitative Abschätzung möglich ist.

Inwieweit die Strukturen des Referenzmodells zur Persistierung funktionaler Aspekte dem Nutzer bzw. der Anwendung im Rahmen der Verarbeitung bekannt sein müssen oder verborgen bleiben können, zeigt sich in der **Transparenz** einer Technik. Weiterhin wird geprüft, ob im jeweiligen Ansatz auf die bereits vorhandene Mächtigkeit eines zu Grunde gelegten RDBMS in angemessener Weise zurückgegriffen wird (**Funktionsadäquatheit**) oder ob Funktionalität, wie beispielsweise das Parsen von Sprachausdrücken und das Caching von Daten, reimplementiert werden muss. Hierbei spielen auch prinzipielle Möglichkeiten zur **Erweiterbarkeit** des Funktionsumfangs eine Rolle, erwartungsgemäß werden diese durch Standardisierung beschränkt. Schließlich soll noch mit der **Benutzerfreundlichkeit** beurteilt werden, wie sich letztlich die gesamte Auswertung funktionaler Aspekte für den Anwendungsentwickler darstellt.

4.2 Bewertung

Eine detaillierte Bewertung jeder einzelnen Zugriffstechnik bezüglich der zuvor aufgestellten Kriterien kann hier aus Platzgründen nicht beschrieben werden. Stattdessen enthält Tabelle 1 einen Überblick mit den Ergebnissen.

Zugriffstechnik / **Kriterium**	Standardisierung	Praxisrelevanz	Performance	Transparenz	Funktionsadäquatheit	Erweiterbarkeit	Benutzerfreundlichkeit
JOIN	+	+	−	−	+	−	−
PIVOT	o[1]	o[1]	o	o	+	−	o
ASPECTVIEW	−	o[3]	o	+	o[3]	+	+
API	o[2]	+	+[4]	+	−	+	o

+ : ja/hoch o : neutral/mittel − : nein/niedrig

[1] unterstützt durch Oracle 11g und MS SQL Server 2005
[2] de-facto Standard[8] aufgrund enormer Verbreitung
[3] nur bei Nutzung von Zwischenschichten[1, 2]
[4] begründet durch Ergebnisse in [7]

Tabelle 1: Bewertung der Zugriffstechniken

Dabei fällt auf, dass zwar der **JOIN-Ansatz** (Abschnitt 3.1) als einziger Vertreter auf standardisierte Sprachkonstrukte zurückgreift und damit eine große Praxisrelevanz bzw. Produktunterstützung besitzt, allerdings weder performant noch benutzerspezifisch erweiterbar ist. Zudem erzwingt die Nutzung des Verbund-Operators genaue Kenntnisse über die Aspekt-Tabellen, wodurch die Transparenz und Benutzerfreundlichkeit verloren geht.

Dagegen verspricht die **PIVOT-Methode** (Abschnitt 3.2) sowohl den transparenteren Zugang als auch eine performantere Verarbeitung der aspektspezifischen Daten in den EAV-Tabellen[5]. Dennoch fehlt hier ebenfalls die Möglichkeit zur Erweiterbarkeit, zudem ist die Anwendbarkeit aufgrund der (noch) fehlenden Normierung an Hersteller wie Oracle oder Microsoft und deren DBMS-Produkte gekoppelt.

Ähnliche Charakteristik besitzt auch der **ASPECTVIEW-Vorschlag** (Abschnitt 3.3), er soll jedoch als Erweiterung

von SQL die Konsequenzen der Aspektorientierten Datenhaltung berücksichtigen. Daher ist der praktische Einsatz momentan nur über die genannten Zwischenschichten realisierbar. Andererseits wird dem Anwendungsentwickler ein intuitives und adäquates Konstrukt zur Verfügung gestellt, um transparent auf funktionale Aspekte von Fachtabellen zugreifen zu können.

Schließlich ergibt sich für den **API-Ansatz** (Abschnitt 3.4) eine mindestens ebenso gute oder bessere Bewertung gegenüber den anderen Zugriffstechniken in vielen Kriterien, insbesondere ist ein performanter Aspekt-Zugriff möglich. Der fehlenden Standardisierung lässt sich z.B. durch Portierung auf die gängigsten Programmiersprachen begegnen. Großer Aufwand ist zudem notwendig, um der Mächtigkeit von SQL auf Seiten der API gerecht zu werden.

5. ZUSAMMENFASSUNG

Der vorliegende Beitrag hat vier verschiedene Techniken für den Zugriff auf funktionale Aspekte aufgezeigt, welche im Kontext der Aspektorientierten Datenhaltung über ein ebenfalls kurz beschriebenes Referenzmodell auf relationalen Datenbanken abgebildet werden können. Durch die Fokussierung auf RDBMS und deren hohen Verbreitungsgrad ist die Grundlage für den praktischen Einsatz gewährleistet. Die anschließende Bewertung anhand zuvor aufgestellter Kriterien sollte weitere Klarheit über die Potentiale der einzelnen Ansätze liefern. Dabei hat sich herausgestellt, dass vor allem für eine performante und transparente Auswertung funktionaler Aspekte die standardisierten Mittel von SQL nicht ausreichen.

Unter den möglichen Alternativen erscheinen der Vorschlag zur Erweiterung von SQL mit ASPECTVIEW und die applikative Verarbeitung in einem Funktionsbaustein vielversprechend. Dabei ist der Aufwand zur Erweiterung der SQL-Norm ungleich höher und nicht automatisch von Erfolg gekrönt bzw. wie bereits vorgeschlagen nur durch Implementierung einer transformierenden Zwischenschicht praktikabel. Aus diesem Grund werden sich nachfolgende Arbeiten vor allem der prototypischen Realisierung, quantitativen Performance-Vergleichen sowie funktionellen Weiterentwicklungen bezüglich aktueller Einschränkungen der vorgestellten API für funktionale Aspekte widmen.

6. LITERATUR

[1] A. Adam, S. Leuoth, and W. Benn. Nutzung von Proxys zur Ergänzung von Datenbankfunktionen. In *Proceedings of the 22nd GI Workshop Grundlagen von Datenbanken (GvDB 2010)*, pages 31–35, Bad Helmstedt, Germany, May 2010.

[2] S. Aulbach, T. Grust, D. Jacobs, A. Kemper, and J. Rittinger. Multi-tenant databases for software as a service: schema-mapping techniques. In *SIGMOD Conference*, pages 1195–1206, 2008.

[3] R. Chitchyan, I. Sommerville, and A. Rashid. An Analysis of Design Approaches for Crosscutting Concerns. In *Workshop on Aspect-Oriented Design (held in conjunction with the 1st Aspect Oriented Software Development Conference (AOSD 2002)*, 2002.

[4] E. F. Codd. A relational model of data for large shared data banks. *CACM*, 13(6):377–387, 1970.

[5] C. Cunningham, G. Graefe, and C. A. Galindo-Legaria. PIVOT and UNPIVOT: Optimization and Execution Strategies in an RDBMS. In *(e)Proceedings of the 30th International Conference on Very Large Data Bases (VLDB 2004)*, pages 998–1009, 2004.

[6] C. Date and H. Darwen. *SQL - Der Standard. SQL/92 mit den Erweiterungen CLI und PSM*. Addison-Wesley, 1998.

[7] V. Dinu, P. Nadkarni, and C. Brandt. Pivoting approaches for bulk extraction of Entity-Attribute-Value data. *Computer Methods And Programs in Biomedicine*, 82(1):38–43, 2006.

[8] T. Egyedi. Why Java Was Not Standardized Twice. *Hawaii International Conference on System Sciences*, 5(1):5015–5025, 2001.

[9] M. Fisher, J. Ellis, and J. Bruce. *JDBC API Tutorial and Reference (Java Series)*. Addison-Wesley, 2003.

[10] *ISO/IEC 9075-2:2008. Information technology – Database languages – SQL – Part 2: Foundation (SQL/Foundation)*. ISO, Geneva, Switzerland, 2008.

[11] M. Liebisch. Aspektorientierte Datenhaltung - ein Modellierungsparadigma. In *Proceedings of the 22nd GI Workshop Grundlagen von Datenbanken (GvDB 2010)*, pages 13–17, Bad Helmstedt, Germany, May 2010.

[12] M. Liebisch. Supporting functional aspects in relational databases. In *Proceedings of the 2nd International Conference on Software Technology and Engineering (ICSTE 2010)*, pages 227–231, San Juan, Puerto Rico, USA, Oct. 2010.

[13] M. Liebisch and M. Plietz. Performance-Analysen für Realisierungsansätze im Kontext der Aspektorientierten Datenhaltung. Institut für Informatik, Friedrich-Schiller-Universität Jena, Nov. 2010.

[14] D. Lorentz. *Oracle Database SQL Language Reference 11g Release 1*. Oracle, Aug. 2010.

[15] P. Nadkarni, L. Marenco, R. Chen, E. Skoufos, G. Shepherd, and P. Miller. Organization of Heterogeneous Scientific Data Using the EAV/CR Representation. In *JAMIA*, 6, pages 478–493, 1999.

[16] B. Pietsch. Entwurf einer Zugriffsschicht für funktionale Aspekte in DBMS. Studienarbeit, Institut für Informatik, Friedrich-Schiller-Universität Jena, Mar. 2011.

[17] A. B. Rashid and M. Islam. Role of Materialized View Maintenance with PIVOT and UNPIVOT Operators. In *Proceedings of the 1st IEEE International Advance Computing Conference (IACC 2009)*, pages 915–955, Mar. 2009.

[18] T. Rizzo, A. Machanic, and J. Skinner. *Pro SQL Server 2005*. Apress, 2005.

[19] T. Schilling. Realisierungskonzepte für die Aspektorientierte Datenhaltung. Studienarbeit, Institut für Informatik, Friedrich-Schiller-Universität Jena, Apr. 2011.

[20] C. M. Wyss and E. L. Robertson. A formal characterization of PIVOT/UNPIVOT. In *Proceedings of the 14th ACM international conference on Information and knowledge management*, pages 602–608, 2005.