Accessing Functional Aspects with Pure SQL
– Lessons Learned –

Matthias Liebisch

Database and Information Systems Group
Friedrich-Schiller-University Jena
07743 Jena, Germany
m.liebisch@uni-jena.de

Abstract. Functional aspects, known as cross-cutting concerns in software engineering, embody the extension of a conceptual schema by functionality which is beyond the intrinsic business logic. The support of functional aspects in a modularized way is the purpose of aspect-oriented data management. The present paper gives a review of this paradigm and describes the experiences of accessing the corresponding relational reference model through pure SQL. This includes an application example, the definition of a benchmark testbed with a practice-oriented workload and some key performance indicators. Ultimately the benchmark results as well as the usage of SQL to access functional aspects are discussed.

Keywords: functional aspects, aspect-oriented data management, reference model, benchmark results, access issues

1 Introduction

For storing application data, relational database management systems (RDBMS) based on the relational model[3] have become generally accepted since their introduction three decades ago. In the process of data modeling not only application specific business requirements but so called *functional aspects* must be considered, for example the ability to handle multilingual or version-controlled data. These functional aspects, also known as *cross-cutting concerns*, have an impact on most parts of the modeling units and can't be modularized in a simple way. Similar problems arise when databases are faced with multi-tenancy requirements, for example in the context of software as a service[1] respectively cloud computing. Compared to well known traditional approaches, which integrate such aspects with repetitive steps of schema evolution into an existing data model, the new paradigm of aspect-oriented data management[8] pursues an integration independent of and encapsulated with respect to the present conceptual schema. To achieve these aspirations a reference model for the aspect-oriented data management is described in [9].

The on hand paper analyzes the consequences for accessing this reference model with pure SQL, i.e. SQL:92, to query and manipulate aspect-specific data. For this purpose Sec. 2 gives an overview of aspect-oriented data management, the associated reference model and a sample application. The following main part in Sec. 3 specifies the whole testbed for a benchmark and presents some performance results. Finally, the outcome is evaluated in Sec. 4.

2 Fundamentals

This section imparts a survey of the main concepts regarding aspect-oriented data management and the related reference model. Furthermore an example of use is described, therewith all basics for the performance benchmark in Sec. 3 are established.

2.1 Aspect-Oriented Data Management

Cross-cutting concerns in software development processes, e.g. logging or exception handling, are covered for more than 10 years by the concept of aspect-oriented programming[7,2] to avoid spreading the code for such functional aspects over classic business objects. Similar challenges arise at the persistence layer, e.g. by storing multilingual data without influencing all relevant tables in a relational database. Therefor the new paradigm of aspect-oriented data management was established, including the following characteristics. Further details can be found in [8].

Modularity: All information belonging to a functional aspect should be consolidated in a logical modeling unit and not spread over the whole conceptual schema.

Orthogonality: For arbitrary functional aspects all potential orders of integration must transform the conceptual schema into semantic equivalent states. At the same time a given attribute set can be influenced by more than one functional aspect, but for two different functional aspects no interaction between their influenced attribute sets is allowed.

Locality: The integration of a functional aspect, that is the extension of the conceptual schema for representing aspect specific data, should require a minimum of transformations, at best none.

Universality: To ensure the greatest possible application area with respect to all relevant RDBMS products the realization of functional aspects should only be based on SQL:92.

Usability: The application/database administrator must be supported by semi-automatic tools in the process of functional aspect integration. Furthermore an user-friendly access to integrated functional aspects is desirable.

2.2 Reference Model

To manage functional aspects in an existing business data model, the reference model shown in Fig. 1 has to be integrated as an encapsulated schema. Thereby the tables AspectValue and AspectAssign represent central elements (aspect weaving data), whereas AspectValue implements the concept

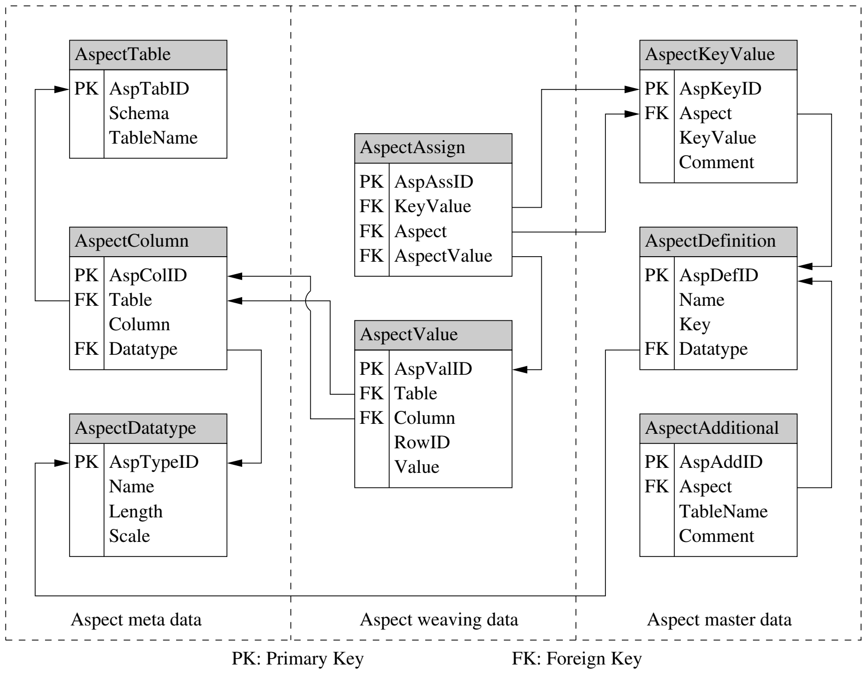

Fig. 1. Reference model to support functional aspects (based on [9])

of the entity-attribute-value (EAV) model[12]. Thus a high degree of flexibility and dynamic concerning aspect specific assignments is ensured, which works as follows. At first an aspect specific characteristic (VALUE) for an arbitrary table cell, identifiable by the triple (TABLE, COLUMN, ROWID), is stored in table ASPECTVALUE. Therefor it's necessary that all original business tables only have a single attribute primary key. Moreover a generic data type, for example VARCHAR, must be used for ASPECTVALUE.VALUE to handle values of different data types.

Consecutively in the other central table ASPECTASSIGN the assignment of aspect specific data (ASPECTVALUE) to an aspect characteristic (ASPECT, KEYVALUE) is performed, whereby a unique constraint over (ASPECT, KEYVALUE, ASPECTVALUE) must be satisfied. As depicted in Fig. 1 the foreign key attributes in ASPECTVALUE and ASPECTASSIGN references other tables, grouped into tables for a detailed description of all relevant aspects (aspect master data) and tables building a very simple but product independent database catalog (aspect meta data). In every table an internally generated primary key is used instead of the logical primary key given by the remaining attributes to provide a consistent layout of and access to the reference model. As mentioned above unique constraints must ensure consistency. A more precise description of ASPECTDEFINITION, ASPECTKEYVALUE, ASPECTADDITIONAL, ASPECTDATATYPE, ASPECTCOLUMN and ASPECTTABLE is given in [9].

2.3 Example of Use

To illustrate the approach of this paper a basic bill of material management (BOM) will be used as sample application. The incomplete schema is shown in Fig. 2 and consists of two tables. In detail, MODULE is responsible for all master data of a single item and in STRUCTURE the relationship information between items can be stored, i.e. how an item is composed[1]. Beside this pure business data model, the two functional aspects *language* and *version control* should be integrated such as distribution of multilingual products with different versions is achievable in a globalized commerce. Thereby both functional aspects have only an impact on attributes MODULE.NAME, MODULE.PRICE and STRUCTURE.AMOUNT, whereas MODULE.VAR works as indicator for variant configuration processes and should be independent from any aspect.

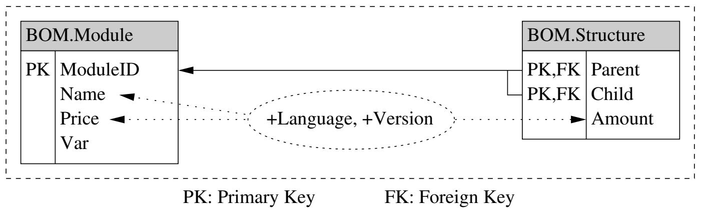

Fig. 2. Example of use (bill of material)

3 Benchmark Testbed

After the reference model for supporting functional aspects in any desired conceptual schema was explained in Sec. 2, the performance of this model is investigated in the current section. For that at first a description of the test system and the workload is given. Following this, the approach as well as the results of the benchmark are explained.

3.1 System

The benchmark was executed on a server of type *IBM RS/6000 7025 Model F80* [11] with the following features:

- hardware: 2 processors RS64-III with 450MHz clock speed and 4GB memory
- operating system: AIX 5.2
- database system: IBM DB2 8.2.4 LUW

[1] The primary key (PARENT, CHILD) will be replaced by an artifical key ROWID in the process of aspect integration, see Fig. 3 for resulting schema

The usage of any additional hardware or database system was explicitly not taken into account, because the essential motivation of this work was a qualitative performance result of the reference model instead of comparing two test systems for a given workload. The following parameters of DB2 had to be adjusted so that the complete processing of the workload described in Sec. 3.2 is ensured: `LOGFILSIZ=4000`, `STMTHEAP=16384` and `APPLHEAPSZ=SORTHEAP=1024`, each with page size of 4KB.

3.2 Workload

In general the workload of an application system is classifiable into handling individual data (OLTP), usually caused by manual interaction, and processing mass data (OLAP). For both groups typical use cases in the context of the example in Fig. 2 are listed in Table 1. Respective to the property of universality all requests will be grammatically based on SQL:92.

	Id	Operation	Use case (*sample request*)
OLTP	W1	SELECT	Display module properties (*get all internationalized data for the module with key 55*).
	W2	INSERT	Populate module translations (*insert name 'module55-spanish' and price '55.22' for module with key 55 under locale 'es-ES'*).
	W3	UPDATE	Modify module names (*change name to 'module55-new' as a new version for module with key 55*).
	W4	DELETE	Delete module translations (*remove all aspect specific data created by workload W2 and W3 for module with key 55*).
OLAP	W5	SELECT	Extraction of module properties (*get internationalized data including related locale for all modules*).
	W6	INSERT	Data transfer from other systems (*import internationalized and structured module data for locale 'pl-PL'*).
	W7	UPDATE	Data cleansing / normalization (*complete the internationalized module name for locale 'de-CH' by prefix 'I18N'*).
	W8	DELETE	Archiving / data cleansing (*delete all internationalized module records for locale 'pl-PL'*).

Table 1. Workload as verbal specification

3.3 Approach

To get expressive outcomes for the benchmark, the business tables as well as the aspect tables had to be filled with generated artificial test data. This was done by a little self-written Java program combined with a JDBC driver for DB2, which created the content following a simple pattern by incrementing numbers. The resulting tables are shown in Fig. 3, whereas some tables were truncated (marked by '. . . ') because of the required space. To stay on top of things the overall quantity structure for all tables is listed in Table 2.

MODULE			
MODULEID	NAME	PRICE	VAR
1	module1	1	1
2	module2	2	0
...	...	...	...
3000	module3000	3000	0

STRUCTURE			
ROWID	PARENT	CHILD	AMOUNT
1	1	2	2
2	1	3	1
...	...	...	...
2999	1500	3000	2

ASPECTDATATYPE			
ASPTYPEID	NAME	LENGTH	SCALE
1	Str	25	
2	Dec	7	2
3	Int	10	

ASPECTCOLUMN			
ASPCOLID	TABLE	COLUMN	DATATYPE
1	1	Name	1
2	1	Price	2
3	2	Amount	3

ASPECTTABLE		
ASPTABID	SCHEMA	TABLENAME
1	BOM	Module
2	BOM	Structure

ASPECTDEFINITION			
ASPDEFID	NAME	KEY	DATATYPE
1	I18N	Locale	1
2	Version	Revision	3

ASPECTKEYVALUE			
ASPKEYID	ASPECT	KEYVALUE	COMMENT
1	1	de-DE	German
2	1	en-GB	English
3	1	it-IT	Italian
4	1	de-CH	Swiss German
5	2	1	initial version
6	2	2	test version
7	1	es-ES	Spanish

ASPECTVALUE				
ASPVALID	TABLE	COLUMN	ROWID	VALUE
1	1	1	1	module1-english
3001	1	2	1	1.33
6001	1	1	1	module1-italian
9001	1	2	1	1.66
...	...	...	...	...

ASPECTASSIGN			
ASPASSID	ASPECT	KEYVALUE	ASPECTVALUE
1	1	2	1
3001	1	2	3001
6001	1	3	6001
6001	1	3	6001
...	...	...	...

Fig. 3. Business and aspect tables filled with test data

As next step the sample requests described as workload in Sec. 3.2 were formulated in SQL syntax with respect to the reference model. Thereby no optimization issues were considered, since this task should be fulfilled by the database management system. However, internal primary key values of all aspect master/meta tables were assumed to be well known during the query generation, e.g. AspectDefinition.AspDefID=1 for referencing the aspect *I18N* (internationalization). Normally this prerequisite can be guaranteed by saving such meta data in the invoking application. The whole SQL queries used for the performance test are listed in Fig. 5.

Table	#Rows
Module	3000
Structure	2999
AspectTable	2
AspectColumn	5
AspectDatatype	4

Table	#Rows
AspectAssign	45000
AspectValue	45000
AspectKeyValue	7
AspectDefinition	2
AspectAdditional	0

Table 2. Quantity structure for business and aspect tables

To handle all SQL queries the little Java tool mentioned above was used. By calling the method `java.sql.Statement.executeQuery` static SQL statements are processed whereby the overall execution time of a specific query is measured by invoking the method `System.currentTimeMillis` and calculating the difference as shown in Fig. 4. This approach takes account of additional costs, which aren't negligible in case of the OLAP workload for transportation overhead.

```
timeDuration = System.currentTimeMillis();
if (withResult)
    _rs = _stmt.executeQuery(query);
else
    for (int i=0; i<queryList.length; i++) _stmt.execute(queryList[i]);
timeDuration = System.currentTimeMillis() - timeDuration;
```

Fig. 4. Java code fragment to measure the SQL query execution time

For analyzing the request behavior of the workload queries under different conditions, the existence of index information (shortened as IDX) and/or distribution statistics[6] (shortened as STAT) are considered. The creation of the additional indices is restricted on all attributes of the central tables AspectValue and AspectAssign, whereas statistical information is enforced for all tables of the reference model.

```
-- [W1] --
SELECT CAST(T1.Value AS VARCHAR(50)) AS Name, CAST(T2.Value AS NUMERIC(7,2)) AS Price
  FROM AspectValue T1 INNER JOIN AspectValue T2 ON T1.RowID = T2.RowID
                      INNER JOIN AspectAssign T3 ON T1.AspValID = T3.AspectValue
                      INNER JOIN AspectAssign T4 ON T2.AspValID = T4.AspectValue
 WHERE T1.Table = 1 AND T1.Column = 1 AND T2.Table = 1 AND T2.Column = 2
   AND T3.KeyValue = T4.KeyValue AND T3.Aspect = 1 AND T1.RowID = 55;

-- [W2] --
INSERT INTO AspectValue (AspValID, Table, Column, RowID, Value)
       VALUES (-20, 1, 1, 55, 'module55-spanish'), (-21, 1, 2, 55, '55.22');
INSERT INTO AspectAssign (AspAssID, Aspect, KeyValue, AspectValue)
       VALUES (-20, 1, 7, -20), (-21, 1, 7, -21);

-- [W3] --
INSERT INTO AspectValue (AspValID, Table, Column, RowID, Value)
       SELECT -30, 1, 1, ModulID, Name FROM Modul WHERE ModulID = 55;
INSERT INTO AspectAssign (AspAssID, Aspect, KeyValue, AspectValue)
       VALUES (-30, 2, 5, -30);
UPDATE Modul SET Name = 'module55-new' WHERE ModulID = 55;

-- [W4] --
DELETE FROM AspectAssign WHERE AspAssID IN (-20, -21, -30);
DELETE FROM AspectValue WHERE AspValID IN (-20, -21, -30);

-- [W5] --
SELECT T1.RowID As ModuleID, T5.KeyValue, CAST(T1.Value AS VARCHAR(50)) AS Name,
       CAST(T2.Value AS NUMERIC(7,2)) AS Price
  FROM AspectValue T1 INNER JOIN AspectValue T2 ON T1.RowID = T2.RowID
                      INNER JOIN AspectAssign T3 ON T1.AspValID = T3.AspectValue
                      INNER JOIN AspectAssign T4 ON T2.AspValID = T4.AspectValue
                      INNER JOIN AspectKeyValue T5 ON T4.KeyValue = T5.AspKeyID
 WHERE T1.Table = 1 AND T1.Column = 1 AND T2.Table = 1 AND T2.Column = 2
   AND T3.KeyValue = T4.KeyValue AND T3.Aspect = 1
 ORDER BY T5.KeyValue, T1.RowID;

-- [W6] --
INSERT INTO AspectKeyValue (AspKeyID, Aspect, KeyValue, Comment)
       VALUES (8, 1, 'pl-PL', 'Polish');
INSERT INTO AspectValue (AspValID, Table, Column, RowID, Value)
       VALUES (45001, 1, 1, 1, 'module1-polish'), (48001, 1, 2, 1, '1.11'), ...,
              (48000, 1, 1, 3000, 'module3000-polish'), (51000, 1, 2, 3000, '3000.11');
INSERT INTO AspectAssign (AspAssID, Aspect, KeyValue, AspectValue)
       VALUES (45001, 1, 8, 45001), (48001, 1, 8, 48001), ...,
              (48000, 1, 8, 48000), (51000, 1, 8, 51000);

-- [W7] --
UPDATE AspectValue
   SET Value = CONCAT('I18N',Value)
 WHERE Table = 1 AND Column = 1
   AND AspValID IN (SELECT AspectValue FROM AspectAssign WHERE Aspect = 1 AND KeyValue = 4);

-- [W8] --
DECLARE GLOBAL TEMPORARY TABLE DelRows (AspValID int) ON COMMIT PRESERVE ROWS NOT LOGGED;
INSERT INTO SESSION.DelRows (AspValID)
       SELECT T1.AspValID
         FROM AspectValue T1 INNER JOIN AspectAssign T2 ON T1.AspValID = T2.AspectValue
        WHERE T2.Aspect = 1 AND T2.KeyValue = 8 AND T1.Table = 1;
DELETE FROM AspectAssign WHERE Aspect = 1
   AND KeyValue = 8 AND AspectValue IN (SELECT AspValID FROM SESSION.DelRows);
DELETE FROM AspectValue WHERE AspValID IN (SELECT AspValID FROM SESSION.DelRows);
DELETE FROM AspectKeyValue WHERE Aspect = 1 AND KeyValue = 'pl-PL';
```

Fig. 5. Workload as SQL requests

3.4 Results

As already recognized in the process of transforming the workload in proper SQL queries, the EAV concept used in AspectValue have a huge impact for query generation and especially their execution time. The latter one is confirmed in Table 3, which contains the averaged measurement time after three iterations for all requests described in Sec. 3.2. The fastest scenario concerning the usage of index and/or statistical information is **highlighted**.

Scenario	OLTP				OLAP			
	W1	W2	W3	W4	W5	W6	W7	W8
	4343	251	147	1220	1053401	17417	2626	168385
+IDX	**226**	**126**	180	37	3413	23121	217280	19568
+STAT	3815	249	92	105	4318	**12811**	**2260**	158186
+IDX+STAT	995	146	**73**	**24**	**1792**	20219	5078	**16201**

Table 3. Workload execution times (in ms)

4 Conclusion

The on hand paper gave a short introduction to functional aspects in data modeling processes and motivated the need to handle them. Therefor a flexible reference model using the EAV concept was presented and explained based on an application example. As clarified by the results in Sec. 3.4, pure SQL is not an effective way to access functional aspects in this reference model. Even if only the workload for OLTP is observed, the execution time of about 200ms for a simple SELECT request is much too long. This circumstances will exacerbate by increasing the quantity structure in AspectValue and AspectAssign to real values. Furthermore the necessary join operations for these tables scale with the number of selected attributes. Finally the generation of such a SQL query is very inconvenient.

The prime reason for all these problems is the missing support for pivoting tables with dedicated SQL elements[14]. The pivot operation ('rows to columns') is needed to represent or process EAV-based tables as regular tables. Leaving the SQL standard, some DBMS products have already an implementation of such a dedicated PIVOT operation, for example MS SQL Server 2005[4]. However, non-standardized methods violate the property of universality (see Sec. 2). An overview and comparison of access methods for functional aspects using the reference model can be found in [10]. Because of the evaluation results in that paper, further work will be focused on specification and implementation of an API[13], which provides an applicative interface to functional aspects as analogously suggested in [5]. Alternatively other storage concepts can be analyzed, for example ORDBMS, XML and NoSQL, whether they are more suitable for functional aspects and how they can effectively utilized.

References

1. Aulbach, S., Grust, T., Jacobs, D., Kemper, A., Rittinger, J.: Multi-tenant databases for software as a service: schema-mapping techniques. In: SIGMOD Conference. pp. 1195–1206 (2008)
2. Chitchyan, R., Sommerville, I., Rashid, A.: An Analysis of Design Approaches for Crosscutting Concerns. In: Workshop on Aspect-Oriented Design (held in conjunction with the 1st AOSD Conference) (2002)
3. Codd, E.F.: A relational model of data for large shared data banks. CACM 13(6), 377–387 (1970)
4. Cunningham, C., Graefe, G., Galindo-Legaria, C.A.: PIVOT and UNPIVOT: Optimization and Execution Strategies in an RDBMS. In: (e)Proceedings of the 30th International Conference on Very Large Data Bases. pp. 998–1009 (2004)
5. Dinu, V., Nadkarni, P., Brandt, C.: Pivoting approaches for bulk extraction of Entity-Attribute-Value data. Computer Methods And Programs in Biomedicine 82(1), 38–43 (2006)
6. Fechner, D.: Distribution statistics uses with the db2 optimizer – create efficient access plans for faster sql (2006), `http://www.ibm.com/developerworks/data/library/techarticle/dm-0606fechner/index.html`, [09.06.2011]
7. Kiczales, G., Lamping, J., Mendhekar, A., Maeda, C., Lopes, C.V., Irwin, J., Loingtier, J.M.: Aspect-Oriented Programming. In: ECOOP. pp. 220–242 (1997)
8. Liebisch, M.: Aspektorientierte Datenhaltung - ein Modellierungsparadigma. In: Proceedings of the 22nd GI Workshop Grundlagen von Datenbanken 2010. pp. 13–17. Bad Helmstedt, Germany (May 2010)
9. Liebisch, M.: Supporting functional aspects in relational databases. In: Proceedings of the 2nd International Conference on Software Technology and Engineering (ICSTE 2010). pp. 227–231. San Juan, Puerto Rico, USA (Oct 2010)
10. Liebisch, M.: Analyse und Vergleich von Zugriffstechniken für funktionale Aspekte in RDBMS. In: Proceedings zum 23. GI-Workshop Grundlagen von Datenbanken. pp. 25–30. Obergurgl, Österreich (May 2011)
11. Lutz, S., Manohar, S.: RS/6000 7025 Model F80: Technical Overview and Introduction. IBM, Austin, TX (May 2000)
12. Nadkarni, P., Marenco, L., Chen, R., Skoufos, E., Shepherd, G., Miller, P.: Organization of Heterogeneous Scientific Data Using the EAV/CR Representation. In: JAMIA. pp. 478–493. 6 (1999)
13. Pietsch, B.: Entwurf einer Zugriffsschicht für funktionale Aspekte in DBMS. Studienarbeit, Institut für Informatik, Friedrich-Schiller-Universität Jena (Mar 2011)
14. Wyss, C.M., Robertson, E.L.: A formal characterization of PIVOT/UNPIVOT. In: Proceedings of the 14th ACM CIKM. pp. 602–608 (2005)