A Powerful and SQL-Compatible Data Model and Query Language For OLAP

Dennis Pedersen **Karsten Riis** **Torben Bach Pedersen**

Department of Computer Science, Aalborg University
Fredrik Bajers Vej 7E, 9220 Aalborg Ø, Denmark
Email: {dennisp,riis,tbp}@cs.auc.dk

Abstract

In this paper we present the SQL_M OLAP data model, formal algebra, and query language that, unlike current OLAP data models and languages, are both *powerful*, meaning that they support irregular dimension hierarchies, automatic aggregation of data, and correct aggregation of data, and *SQL-compatible*, allowing seamless integration with relational technology. We also consider the requirements to the data model posed by integration of OLAP data with external XML data. The concepts are illustrated with a real-world case study from the Business-to-Business electronic commerce (B2B) domain.
Keywords: OLAP, Multidimensional Databases, Data Models, Query Languages, Data Integration

1 Introduction

OLAP systems [Codd, 1993, Shoshani, 1997] enables powerful decision support based on multidimensional analysis of large amounts of summary data commonly drawn from a number of different transactional databases. OLAP data are often organized in multidimensional *cubes* containing *measured values* that are characterized by a number of hierarchical *dimensions*. Typical operations on data cubes are *roll-up*, which aggregates data by moving up along one or more dimensions, *drill-down*, which disaggregates data by moving down dimensions, and *slice-and-dice*, which performs selection and projection on a cube. The multidimensional approach offers a number of advantages over traditional types of DBMSs, including automatic application of the pre-specified aggregation functions (automatic aggregation) [Rafanelli et al., 1990, Thomsen, 1997], visual querying [Thomsen, 1997, Thomsen, 1999], and good query performance due to the use of pre-aggregation [Gupta et al., 1995, Pedersen et al., 1999b]. Additionally, the dimensional approach is most often a natural fit for data analysis problems.

To be able to capture the complex data found in many real-world applications, the data model for the OLAP system must be able to handle *irregular* dimension hierarchies [Pedersen et al., 1999a, Pedersen et al., 1999b] that do not fit the balanced-tree hierarchies supported by current OLAP systems. More specifically, the data model must support *non-strict* hierarchies where lower-level items may have several parents in a higher dimension level, and *non-covering hierarchies*, where paths in the hierarchy may "jump" more than one level at a time. In this paper, we exemplify irregular hierarchies in the B2B domain, but they also occur in many other domains, e.g., organization hierarchies [Zurek et al., 1999], web concept hierarchies [Yahoo, 2001], and medical diagnosis hierarchies [NHS, 1999]. Having irregular dimension hierarchies violates the *summarizability* [Lenz et al., 1997] of the OLAP data, meaning that the user may get wrong result if intermediate aggregate results are re-used to compute higher-level results. Thus, it is also very important that the OLAP data model has a "safety net" that ensures that the user will always perform correct aggregations.

An ideal OLAP query language should allow OLAP queries to be posed in a compact and concise yet easily understandable way and support user-friendly functions such as automatic aggregation. However, to allow integration with legacy relational systems, the OLAP language (and data model) should also be *SQL-compatible*, meaning that the OLAP data can also be queried using standard SQL, yielding the same results as would be expected from querying a relational table.

Current OLAP data models and query languages fall in one of three categories. First, the *simple SQL-like models* [Agrawal et al., 1997, Gray et al., 1997, Gyssens et al., 1997, Jagadish et al., 1999, Kimball, 1996, Li et al., 1996] are close to the relational/SQL data model and query language, but do not support advanced features such as automatic aggregation, irregular hierarchies, and correct aggregation. Second, the *simple cube models* [Cabibbo et al., 1997, Lehner, 1998, Rafanelli et al., 1990, Thomsen, 1999, Vassiliadis, 1998] are "pure" multidimensional models, meaning that their data model is not relational-like and they cannot be queried using SQL. Also, they do not handle the full spectrum of irregular hierarchies, automatic aggregation, and support for correct aggregation. Third, the *complex cube models* [Pedersen et al., 1999a, Pedersen et al., 2001b] handle irregular hierarchies and support for correct aggregation, but the data model and query language is not compatible with relational systems. Thus, no current data model support complex, irregular data while still providing SQL-compatibility.

In this paper, we present an OLAP data model, a formal algebra over the data model, and a high-level, user-oriented query language, that are both *powerful*, meaning that they handle irregular hierarchies, automatic aggregation, and support for correct aggregation, and *SQL-compatible*, allowing seamless integration with legacy data sources. The data model, algebra, and query language are called the "Multidimensional SQL" (SQL_M) data model, algebra, and query language, respectively. Irregular dimension hierarchies are handled using powerful dimension hierarchies based on partial orders, while support for correct aggregation is based on *aggregation types* that keep track of what data can/cannot be aggregated further without giving wrong results. Special care is taken to design the model and language such that compatibility with relational systems is ensured. As XML [W3C, 2000a] is increasingly being used as a data exchange format for data available on the WWW, we consider the requirements to the data model posed by the problem of allowing integration with external XML data. The concepts are illustrated with a real-world case study from the B2B domain.

We believe this paper to be the first to present an OLAP data model, algebra, and query language that are both *powerful* and *SQL-compatible*. We also believe to be the first to consider the specific challenges that integration of OLAP and XML data poses to the data model.

The rest of the paper is organized as follows. Section 2 presents the case study used throughout the paper. Section 3.1 defines the data model and it's associated algebra. Section 4.1 defines the SQL_M query language. Section 5 presents the challenges that integration with external XML data pose to the data model. Section 6 concludes the paper and points to future work. Appendix A contains the formal syntax of the SQL_M language.

2 Case study

We now present a case study that justifies the need for capturing complex dimension hierarchies. The case study concerns B2B e-commerce in the electronics industry. It is inspired by the Electronic Component Information Exchange (ECIX)[ECIX, 2000], which is a widely adopted initiative to use XML as a means of communicating information about electronic components. The setting consists of companies producing electronic components (ECs), and of companies buying these components and integrating them to larger appliances. In the following we refer to them as suppliers and customers, respectively. Note that both suppliers and customers are companies.

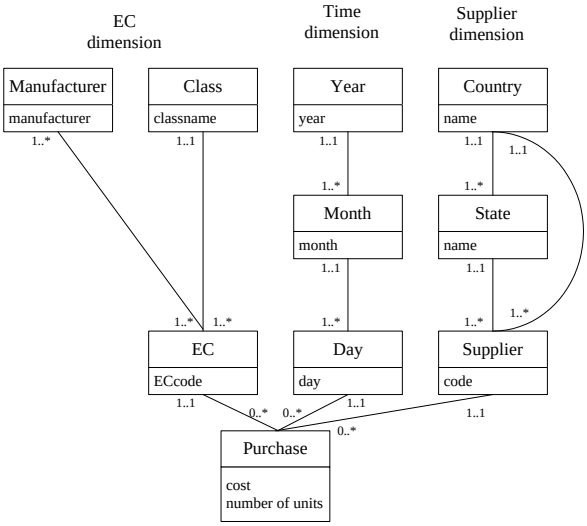

Figure 1: UML schema for the Purchases OLAP database.

Customers use an OLAP database to analyze the purchases they have made over time as shown in the UML diagram in Figure 1. The purchases are characterized by three *dimensions*, namely the EC dimension, the Supplier dimension, and the Time dimension. For each purchase we have two *measures*, the total cost and the number of units purchased. Each dimension has a number of *levels* that categorizes the lower-level items. ECs are categorized by their manufacturers and their classes, e.g. flip-flops or latches. Note that one EC may be manufactured by *several* manufacturers, i.e., the hierarchy is *non-strict*. For suppliers we have the levels Supplier, State and Country. However, the State level only applies to US supplier, so non-US suppliers are mapped directly to their respective countries, i.e., the dimension hierarchy is *non-covering*. The purchase dates are categorized according to the regular calendar, with levels Day, Month, and Year. The database allows customers to view purchases at different levels of granularity e.g. to calculate the total amount spent on ECs by class and month.

However, not all relevant information about the purchases can be included in an organization's OLAP database, e.g., because the information changes too frequently or because it is maintained by an entity of outside the organization. Thus, some relevant information is only available externally to the OLAP DB, most likely in XML format. An example of this is given below. The integrated use of OLAP and XML data, described in Section 5, is another case where the data model is required to be more powerful than traditional OLAP models.

```
<?xml version="1.0" encoding="utf-8"?>
<Components>
  <Supplier SCode="SU13"><SName>John's ECs</SName>
    <Class ClassCode="C24">
      <ClassName>Flip-flop</ClassName>
      <Component CompCode="EC1234">
        <Manufacturer MCode="M31">
          <MName>Smith Components Inc.</MName>
        </Manufacturer>
        <UnitPrice Currency="euro"
           NoOfUnits="1000">3.00
        </UnitPrice>
        <UnitPrice Currency="euro"
           NoOfUnits="10000">2.60
        </UnitPrice>
        <Description>16-bit flip-flop</Description>
      </Component>
      <Component CompCode="EC1235">
        <Manufacturer MCode="M32">
              <MName>John's ECs</MName>
        </Manufacturer>
        <UnitPrice Currency="euro"
           NoOfUnits="1000">4.25
        </UnitPrice>
        <Description>16-bit flip-flop</Description>
      </Component>
    </Class>
  </Supplier>
  <Supplier SCode="SU15"><SName>Jane's ECs</SName>
    <Class ClassCode="C27"><ClassName>Latch</ClassName>
      <Component CompCode="EC2346">
        <Manufacturer MCode="M31">
          <MName>Smith Components</MName>
        </Manufacturer>
        <UnitPrice Currency="euro"
           NoOfUnits="1000">3.31
        </UnitPrice>
        <Description>16-bit latch</Description>
      </Component>
    </Class>
    <Class ClassCode="C24">
      <ClassName>Flip-Flop</ClassName>
      <Component CompCode="EC1234">
        <Manufacturer MCode="M33">
          <MName>Johnson Components</MName>
        </Manufacturer>
        <UnitPrice Currency="euro"
           NoOfUnits="1000">2.95
        </UnitPrice>
        <Description>D-type flip-flop</Description>
      </Component>
    </Class>
  </Supplier>
</Components>
```

Figure 2: The Components document containing information about EC suppliers and their products.

Suppliers present their products on the Web at a B2B marketplace. This allows customers and others to access detailed specifications of their ECs. This information is encoded in an industry-wide markup language defined in XML, which makes it easy to limit a search to the relevant parts of specifications. An example of a document containing information from different suppliers is shown in Figure 2. The fundamental part of an XML document is the *element*. Elements are identified by a *start tag* and an *end tag*, and can contain other elements, text data, and attributes. In the example document the `Component` element has an attribute `CompCode` and contains the elements `Manufacturer`, `UnitPrice` and `Description`. For a more comprehensive explanation of XML see [W3C, 2000a].

All ECs sold by a particular supplier belong to a component class. ECs are referred to by their code. In addi-

tion to this, a document captures the manufacturer, which need not be the same as the supplier, the price per unit, and a textual description. Several aspects of ECs like textual descriptions and current prices are not included in the Purchases database. Despite this, it may sometimes be desirable e.g. to group ECs by their marketplace descriptions, or view only purchases of ECs within a specific price range. By logically integrating the Purchases database and the Components document in a federation this can be handled in an easy and flexible way.

Although we consider a case from the e-commerce domain, the use of XML data in connection with OLAP is needed in other areas, e.g., customer databases.

3 Data Model and Algebra

This section describes the SQL_M data model and algebra.

3.1 The SQL_M Data Model

The model is defined in terms of a multidimensional *cube* consisting of a *cube name*, *dimensions*, and a *fact table* with numerical *measures*. Each dimension comprises two partially ordered sets (posets) representing hierarchies of *levels* and the ordering of *dimension values*. Each level is associated with a set of dimension values. The measures represent the numerical fact properties that should be aggregated in the OLAP queries, e.g., sales price.

Definition 3.1 (Dimension) A *dimension* D_i is a two-tuple (L_{D_i}, E_{D_i}), where L_{D_i} is a poset of levels and E_{D_i} is a poset of dimension values.

L_{D_i} is the poset $(LS_i, \sqsubseteq_i, \top_i, \bot_i)$, where $LS_i = \{L_{i1}, \ldots, L_{ik_i}\}$ is a set of levels, $\sqsubseteq_i$ is a partial order on these levels, and $\top_i$ and $\bot_i$ are the top and bottom elements of the ordering. We shall use $L_{ij} \in D_i$ as a shorthand meaning that the level L_{ij} belongs to the poset of levels in dimension D_i.

A *level* L_{ij} is a name identifying a set of *dimension values*. Let E be the set of all possible dimension values and $Levels$ be the set of all levels. Then a function $\text{Values} : Levels \mapsto \mathcal{P}(E)$, returns the subset of E associated with a level in $Levels$. Thus, $\text{Values}(L_{ij}) = \{e_{ij1}, \ldots, e_{ijl_{ij}}\}$. We shall use L_{ij} as a shorthand for $\text{Values}(L_{ij})$.

E_{D_i} is a poset $\left(\bigcup_j L_{ij}, \sqsubseteq_{D_i}\right)$, consisting of the set of all dimension values in the dimension and a partial ordering defined on these. We shall use D_i as a shorthand for $\bigcup_j L_{ij}$.

For each level L we assume a function $\text{Roll-up}_L : \text{Values}(L) \times LS_i \mapsto \mathcal{P}(D_i)$, which given a dimension value in L and a level in LS_i returns the value's ancestors in the level. That is, $\text{Roll-up}_L(e, L') = \{e' \in L' | e \sqsubseteq_{D_i} e'\}$. □

The intuition behind the partial order $\sqsubseteq_i$ of levels is that given two levels $L_{i1}, L_{i2} \in D_i$ we say that $L_{i1} \sqsubseteq_i L_{i2}$ if elements in L_{i2} can be said to contain the elements in L_{i1}. For example, $Day \sqsubseteq Year$ because years contain days. Similarly, we say that $e_1 \sqsubseteq e_2$ if e_1 is logically contained in e_2 and $L_{ij} \sqsubseteq_i L_{ik}$ for $e_1 \in L_{ij}$ and $e_2 \in L_{ik}$ and $e_1 \neq e_2$. For example, the day 01/21/2000 is contained in the year 2000. Note that a lower-level value may roll up to several higher-level values, or none at all.

Example 3.1 In the case study we have a Time dimension, an EC dimension and a Supplier dimension. Letting Sup denote $Supplier$ the Supplier dimension consists of the levels $LS_{Sup} = \{\top_{Sup}, Country, State, Supplier\}$, which are ordered as follows: $Supplier < State < Country < \top_{Sup} \wedge Supplier < Country$, where the last part capture the fact that suppliers may map directly to countries, by-passing states. If we let $\sqsubseteq_{Sup}$ denote the reflexive, transitive closure of this ordering, the poset of levels is $L_{D_{Sup}} = (LS_{Sup}, \sqsubseteq_{Sup}, \top_{Sup}, Supplier)$. A graphical illustration of the levels is seen to the left in Figure 3. The poset of dimension values is $E_{D_{Sup}} = (\{\top_{D_{Sup}}, US, UK, CA, NV, S1, S2, S3\}, \sqsubseteq_{D_{Sup}})$, where $\sqsubseteq_{D_{Sup}}$ is the reflexive, transitive closure of the ordering $\{(US, \top_{D_{Sup}}), (UK, \top_{D_{Sup}}), (S1, CA), (S2, NV), (S3, UK), (CA, US), (NV, US)$. Note that the supplier "S3" maps directly to the country "UK." Hence, the Supplier dimension is given by: $D_{Sup} = (L_{D_{Sup}}, E_{D_{Sup}})$. A graphical illustration of the dimension values is seen to the right in Figure 3. □

Definition 3.2 (Cube and fact table) An *n-dimensional cube* is a three-tuple $C = (N, D, F)$ consisting of a *cube name* N which describes the type of facts contained in the cube, a non-empty set of *dimensions* $D = \{D_1, \ldots, D_n\}$ and a *fact table* $F(D_1, \ldots, D_n, M_1, \ldots, M_m)$ which is a relation containing one attribute for each dimension D_i and one attribute for each *measure* M_j. Thus, $F = \{(e_{\bot_1}, \ldots, e_{\bot_n}, v_1, \ldots, v_m) | (e_{\bot_1}, \ldots, e_{\bot_n}) \in \bot_1 \times \cdots \times \bot_n \wedge (v_1, \ldots, v_m) \subseteq T_1 \times \cdots \times T_m\}$, where $n \geq 1$, $m \geq 1$, and T_j is the domain value for the j'th measure. We will also refer to the j'th measure as $M_j = \{(e_{\bot_1}, \ldots, e_{\bot_n}, v_j)\}$. The measure domains T_j all contain the special NULL value, which denotes that no value exists for a particular combination of dimension values. A tuple in F, where at least one measure value exists, is called a *fact*.

Each measure M_j is associated with a *default aggregate function* $f_j : \mathcal{P}(T_j) \mapsto T_j$, where the input is a multi-set. Aggregate functions ignore NULL values as in SQL. □

Intuitively, a tuple in F captures the measured values associated with one combination of dimension values from the bottom levels. The number of tuples in F is equal to $\|\bot_1\| \cdot \cdots \cdot \|\bot_n\|$. That is, there is one tuple in F for each possible combination of the bottom dimension values.

Example 3.2 From the Purchases database in the case study we can construct a three-dimensional cube with the cube name $Purchases$, the dimensions, levels, and ordering of dimension values as depicted in Figure 3, and the fact table represented in Table 1. "FF" and "L" are names of classes denoting "Flip-flops" and "Latches", respectively. Note that the EC "EC2345" is manufactured by two manufacturers. Only tuples with non-NULL measure values are shown in the fact table although all combinations are logically present in the relation (to save space, we use this way of presenting fact tables throughout the paper). □

Cost	No. Of Units	Day	Supplier	EC
2940	1000	01/21/2000	S1	EC1234
6900	2000	01/21/2000	S3	EC1234
9480	3000	02/22/2000	S3	EC2345
14400	4000	02/22/2000	S2	EC1235
17650	5000	03/23/2001	S2	EC1235

Table 1: Fact Table For the Purchases Database.

Next, we define the notion of *summarizability* and discuss how it is used to ensure *safe aggregation*. Summarizability is an important cube property as it states when lower-level aggregates, which are often pre-computed, can be used to calculate higher-level aggregates, and when these must be computed from base data.

Definition 3.3 (Summarizable) Given a type T, a set $S = \{S_1, \ldots, S_k\}$ where $S_j \in \mathcal{P}(T)$, and a function $g : \mathcal{P}(T) \mapsto T$ we say that g is *summarizable* if $g(\{g(S_1), \ldots, g(S_k)\}) = g(S_1 \cup \cdots \cup S_k)$ where the argument of the left-hand side is a multi-set. □

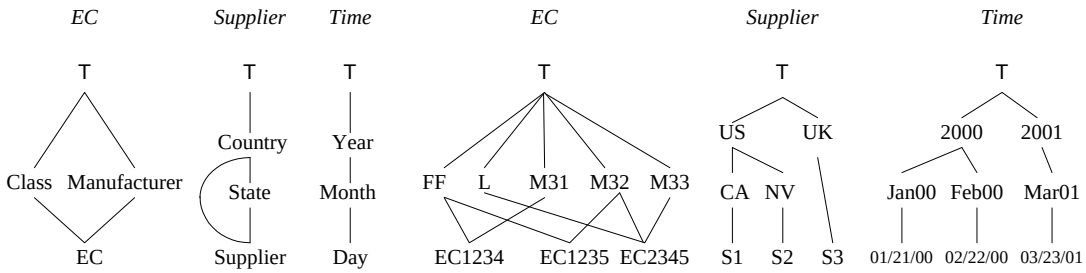

Figure 3: Schema (left) and Instances (right) of the Purchases OLAP Database.

Intuitively, an aggregate function is summarizable if aggregated results from a lower-level aggregate (left-hand side of the formula) can be combined to give the same result as when the aggregate is derived directly from base data (right-hand side of the formula). If this property is not satisfied we are generally not allowed to use the lower-level results for further aggregation. This is important because having to calculate aggregates directly from base data often leads to a considerable increase in computational cost. Also, the users may, unknowingly, get wrong results for their queries.

It has been shown that summarizability for a data model like ours is equivalent to requiring the aggregate function to be distributive, and the ordering of dimension values to be *strict* and *covering* [Lenz et al., 1997, Pedersen et al., 1999a]. Informally, a dimension hierarchy is *strict* if no dimension value has more than one parent value from the same level and *covering* if no path skips a level. Intuitively, this means that dimension hierarchies must be balanced trees. If this is not the case some lower-level values will be either double-counted or not counted at all. These concepts are formally defined below.

Definition 3.4 (Covering) Given three levels, L_1, L_2, $L_3 \in D_i$ such that $L_1 \sqsubset_i L_2 \sqsubset_i L_3$, we say that the mapping from L_1 to L_2 is *covering with respect to* L_3 iff $\forall e_1 \in L_1 \, (\forall e_3 \in L_3 \, (e_1 \sqsubseteq_{D_i} e_3 \Rightarrow \exists e_2 \in L_2 \, (e_1 \sqsubseteq_{D_i} e_2 \wedge e_2 \sqsubseteq_{D_i} e_3)))$. Otherwise, it is *non-covering with respect to* L_1. If all mappings in a dimension are covering w.r.t. any category, we say that the dimension hierarchy is *covering*. Note that we may test the non-covering property in the following way (used later) when L_1 is the bottom level of a dimension D: Given a measure M from D, if $\exists(e_{\perp_1}, \ldots, e_{\perp_n}, v) \in M(\exists e \in \{e_{\perp_1}, \ldots, e_{\perp_n}\}(\| \text{ Roll-up}_{L_1}(e, L_2)\| < 1 \wedge v \neq \text{ NULL}))$ then the mapping from L_1 to L_2 is non-covering with respect to L_3. □

Definition 3.5 (Strict) Given two levels $L_1, L_2 \in D_i$ such that $e_1, e_3 \in L_1 \wedge e_2 \in L_2 \wedge e_2 \sqsubseteq_{D_i} e_1 \wedge e_2 \sqsubseteq_{D_i} e_3 \Rightarrow e_1 = e_3)$, we say that the mapping between L_1 and L_2 is *strict*. Otherwise, it is *non-strict*. The hierarchy in dimension D_i is *strict* if all mappings in it are strict; otherwise, it is *non-strict*. Note that we may test the non-strict property in the following way (used later) when L_1 is the bottom level of a dimension D: Given a measure M from D, if $\exists(e_{\perp_1}, \ldots, e_{\perp_n}, v) \in M(\exists e \in \{e_{\perp_1}, \ldots, e_{\perp_n}\}(\| \text{ Roll-up}_{L_1}(e, L_2)\| > 1 \wedge v \neq \text{ NULL}))$ then the mapping from L_1 to L_2 is non-strict. □

Example 3.3 The hierarchy in the Supplier dimension seen in Figure 3 is non-covering as "S3" maps directly to the country "UK", while the hierarchy in the EC dimension is non-strict as the EC "EC2345" is manufactured by the two manufacturers "M32" and "M33". □

Different aggregate functions may be valid when aggregating different measures or when aggregating the same measure over different dimensions. For example, it is not meaningful to sum the number of employees over time to get the total number of employees, as most employees will be counted more than once. However, the average number of employees over time is meaningful. Calculating the sum of purchases over time is not a problem, since the same purchase will not be repeated. The two cases are examples of the *stock* and *flow* types of data [Lenz et al., 1997, Lehner, 1998], respectively.

To ensure correct aggregation of data we keep track of which aggregate functions can meaningfully be applied to measures for each dimension. We do this by associating an *aggregation type* to each combination of a measure and a dimension, thereby allowing us to prohibit or warn the user against illegal aggregations. Following previous work [Lehner, 1998, Pedersen et al., 1999a, Rafanelli et al., 1983], we distinguish between three types of data: c, data that may not be aggregated because summarizability is not preserved, ϕ, data that may be averaged but not added, and Σ, data that may also be added. Thus, we have the following ordering of these types: $c \subset \phi \subset \Sigma$. Considering only the standard SQL functions, we have that $\Sigma = \{\text{SUM, AVG, MAX, MIN, COUNT}\}$, $\phi = \{\text{AVG, MAX, MIN, COUNT}\}$, and $c = \emptyset$. A function $\text{AggType} : \{M_1, \ldots, M_m\} \times D \mapsto \{\Sigma, \phi, c\}$ returns the aggregation type of a measure M_j when aggregated in a dimension $D_i \in D$. Thus, any changes to an aggregation type apply to all levels in a dimension. Aggregation types are used both to prohibit semantically incorrect aggregation, and to prevent aggregation when irregular hierarchies may lead to incorrect results. Summing up, the use of aggregation types ensures that data will be correctly aggregated as any "misuse" of data will be prohibited.

3.2 The SQL_M Algebra

In this section we present an algebra over the SQL_M data model presented above. Two operators are defined: a selection operator and a generalized projection operator, allowing the expression of standard OLAP queries. The algebra can easily be extended with union, difference, Cartesian product (or join), and rename operators to give it the same power as relational algebra with aggregation functions [Klug, 1982].

The *selection* operator σ_{Cube} is used to slice the cube so that it contains only facts that satisfy a given predicate. The predicates we consider here are constructed from the usual SQL operators, and allows the use of roll-up functions on the form $L'(L)$ which returns the dimension values in L' that contain each dimension value in L. As described above we allow non-strict and non-covering hierarchies. This affects the semantics of the selection operator.

Example 3.4 Slice the purchase cube from Example 3.2 so that only data for ECs not supplied by 'S2' and manufactured by manufacturers starting with 'M32' are retained in the cube: $\sigma_{Cube[\text{Supplier}<>\text{'S2' AND Manufacturer(EC) LIKE 'M32\%'}]}(Purchases) = Purchases'$. □

A predicate may have more than one interpretation if a dimension value can have more than one parent in the same level, i.e. if the hierarchy is non-strict. This would be the case when selecting ECs manufactured by manufacturers that start with "M32". The predicate could be true if *all* manufacturers that manufacture an EC begins with "M32" or it could be true if *any* of them do so. We will refer to these semantics as *all semantics* and *any semantics*, respectively. Here we adopt the *any* semantics, since we consider this the more natural choice for users and since it is also the one used in the XPath standard [W3C, 1999].

The second problem concerns the covering properties of a hierarchy. In a hierarchy that is non-covering some dimension values may not have a parent value in its parent level. For example, some ECs may not be manufactured by any manufacturers and thus, the manufacturer level is skipped. Using the *all* semantics would then result in the above predicate being (trivially) satisfied, since all manufacturers in the (empty) set begins with "M32". This is most likely not what a user would expect and a special handling of the empty case will be necessary. However, adopting the *any* interpretation leads to a more natural meaning, since the predicate is not satisfied for any dimension value in the empty set.

A selection only affects the tuples in the fact table. Hence selection returns a cube with the same fact type and the same set of dimensions. All tuples for which the predicate holds are left unaffected, while those for which the predicate does not hold the measures are set to NULL.

Example 3.5 The fact table resulting from the query in Example 3.4 is:

Cost	No. Of Units	Day	Supplier	EC
9480	3000	02/22/2000	S3	EC2345
14400	4000	02/22/2000	S2	EC1235
17650	5000	03/23/2001	S2	EC1235

□

Formally, we define the selection operator as follows:

Definition 3.6 (Selection operator) Let p be a predicate over the set of levels $\{L_1, \ldots, L_k\}$ and measures $M_1, \ldots, M_m$. Selection on a cube $C = (N, D, F)$ is $\sigma_{Cube[p]}(C) = (N', D', F')$, where $N' = N$, $D' = D$ and $F' = \{t'_1, \ldots, t'_l\}$. If $t_i = (e_{\perp_1}, \ldots, e_{\perp_n}, v_1, \ldots, v_m) \in F$ then

$$t'_i = \begin{cases} t_i & \text{if } p(t_i) = tt \\ (e_{\perp_1}, \ldots, e_{\perp_n}, \text{ NULL}, \ldots, \text{ NULL}) & \text{otherwise.} \end{cases}$$

□

The *generalized projection* operator Π_{Cube} aggregates measures to a given level and removes dimensions and measures from a cube, similarly to the behavior of a SELECT statement with a GROUP BY clause in SQL.

Example 3.6 Calculate the costs per class and supplier using the default aggregate function: $Purchases' = \Pi_{Cube[Supplier,Class]<\text{DEFAULT}(Cost)>}(Purchases)$ □

Generalized projection is evaluated in three steps: First, we remove all dimensions that are not present in the arguments, and then each dimension value is rolled up to the specified level. Finally, we perform a regular grouping in the fact table removing all measures not specified in the arguments. Rolling up to a higher level may result in duplicated facts if the hierarchy is non-strict, while a non-covering hierarchy may result in losing facts.

Example 3.7 The $Purchases'$ cube resulting from the query in Example 3.6 is seen in Figure 4: □

Cost	Supplier	Class
2940	S1	FF
6900	S3	FF
9480	S3	L
32050	S2	FF

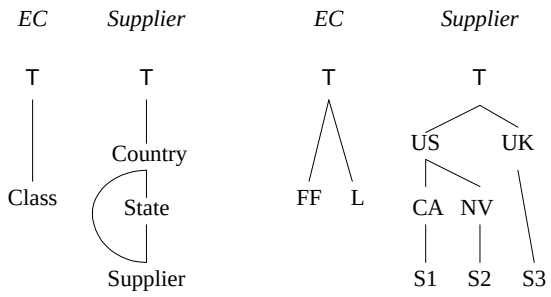

Figure 4: Facts and Dimensions For the $Purchases'$ Cube

Intuitively, the levels specified as an argument to the operator becomes the new bottom levels of their dimensions and all other dimensions are aggregated to the top level and removed. Each new measure value is calculated by applying the default aggregate function on the corresponding value for all tuples in the fact table containing old bottom values that roll up to the new bottom values. To ensure safe aggregation in case of non-covering or non-strict hierarchies, we explicitly check for this in each dimension. If a roll-up along some dimension results in missing facts (non-covering) or duplicated facts (non-strict), we disallow further aggregation along that dimension by setting the aggregation type to c.

Formally, we define:

Definition 3.7 (Generalized projection) Let $C = (N, D, F)$ be a cube as defined above. Then generalized projection is defined as: $\Pi_{Cube[\mathcal{L}_{i_1},\ldots,\mathcal{L}_{i_k}]<f_{j_1}(M_{j_1}),\ldots,f_{j_l}(M_{j_l})>}(C) = (N', D', F')$, where $\{\mathcal{L}_{i_1}, \ldots, \mathcal{L}_{i_k}\}$ is a set of levels specifying the aggregation level such that at most one level from each dimension occurs. The measures $\{M_{j_1}, \ldots, M_{j_l}\} \subseteq \{M_1, \ldots, M_m\}$ are kept in the cube and $f_{j_1}, \ldots, f_{j_l}$ are the default aggregate functions for the specified measures.

The resulting cube is given by: $N' = N$ and $D' = \{D'_{i_1}, \ldots, D'_{i_k}\}$, where $D'_{i_h} = (L'_{D_{i_h}}, E'_{D_{i_h}})$ for $h = 1, \ldots, k$. The new poset of levels in the remaining dimensions is $L'_{D_{i_h}} = (LS'_{i_h}, \sqsubseteq'_{i_h}, \top_{i_h}, \mathcal{L}_{i_h})$, where $LS'_{i_h} = \{L_{i_h p} \in LS_{i_h} | \mathcal{L}_{i_h} \sqsubseteq_{i_h} L_{i_h p}\}$, and $\sqsubseteq'_{i_h} = \sqsubseteq_{i_h|_{LS'_{i_h}}}$. Moreover, $E'_{D_{i_h}} = (\bigcup_p L_{i_h p}, \sqsubseteq_{D_{i_h}|_{\bigcup_p L_{i_h p}}})$.

The new fact table is given by: $F' = \{(e'_{\perp_{i_1}}, \ldots, e'_{\perp_{i_k}}, v'_{j_1}, \ldots, v'_{j_l}) | \ e'_{\perp_{i_g}} \in \mathcal{L}_{i_g} \wedge v_{j_h} = f_{j_h}(\{v | \ (e_{\perp_1}, \ldots, e_{\perp_n}, v) \in M_{j_h} \wedge (e'_{\perp_{i_1}}, \ldots, e'_{\perp_{i_k}}) \in \text{Roll-up}_{\perp_{i_1}}(e_{\perp_{i_1}}, \mathcal{L}_{i_1}) \times \cdots \times \text{Roll-up}_{\perp_{i_k}}(e_{\perp_{i_k}}, \mathcal{L}_{i_k})\})\}$.

Furthermore, if $\exists (e_{\perp_1}, \ldots, e_{\perp_n}, v_j) \in M_{j_h} (\exists e \in \{e_{\perp_1}, \ldots, e_{\perp_n}\} (\| \text{Roll-up}_{\perp_{i_g}}(e, \mathcal{L}_{i_g}) \| \neq 1 \wedge v_j \neq \text{NULL}))$ then $AggType(M_{j_h}, D'_{i_g}) = c$. □

To summarize this section, we have defined two algebraic operators:

- The *selection* operator.
- The *generalized projections* operator.

4 The SQL$_M$ Query Language

We now describe the SQL$_M$ query language and its relationship with standard SQL.

4.1 Query Language

The SQL$_M$ language is based on a subset of SQL, extended with several carefully chosen extension to support powerful OLAP querying. SQL is chosen as the base language for its simplicity and wide-spread use. We illustrate the considered syntax with an example. The full syntax is given in Appendix A.

Example 4.1 Calculate costs by class and supplier but only for suppliers located in UK and only when the total cost exceeds 10000:

SELECT	DEFAULT(Cost), Supplier, Class(EC)
FROM	Purchases
WHERE	Country(Supplier) = 'UK'
GROUP BY	Supplier, Class(EC)
HAVING	DEFAULT(Cost) > 10000

□

A query is constructed from SQL's SELECT-FROM-WHERE-GROUP BY-HAVING statement, with a few modifications to the standard language to capture the special OLAP concepts. Aggregation from a bottom level L to a higher level L' in the same dimension is performed using a roll-up function $L'(L)$ in the SELECT and GROUP BY clauses. Like [Agrawal et al., 1997], we assume these functions to be partial and multi-valued although this is not possible in standard SQL [Eisenberg et al., 1999]. This is necessary because we allow hierarchies to be non-covering and non-strict, e.g. ECs may have more than one manufacturer. Roll-up functions can also be used in the WHERE and HAVING clauses. Since the dimension to which the levels belong is not given in the syntax, we assume level names to be unique. This can be handled by pre-fixing level names with dimension names. As a shorthand, we allow the argument of the roll-up function to be omitted if $L' = L$, that is when no roll-up should be performed in that particular dimension.

For compatibility with SQL, we allow (but do not require) the level names being rolled up to as column aliases in the SELECT clause, i.e., "SELECT Class(EC) Class,...". Similarly, we allow (but do not require) a measure name as column alias for a column that specify an aggregate computation on that measure. Note that renaming to other names is not allowed and that the renaming has no effect. The roll-up functions allow a much more compact syntax than with the standard (star schema based) form of OLAP queries [Kimball, 1996], where the dimension tables have to be included in the FROM clause and explicitly joined to the fact table, resulting in queries that are both 2–3 times longer as well as harder to pose and understand for users. Thus, the SQL$_M$ query language is significantly more powerful for expressing OLAP queries than standard SQL.

Since we do not allow standard relational projection on cubes, the GROUP BY clause is mandatory. Each dimension must either be explicitly rolled up to some level or not mentioned at all. The latter indicates that the dimension should be rolled up to the top level and projected away as is the case in standard SQL. However, if only measures are to be removed from the cube, that is if all bottom levels are present in the SELECT clause, and no HAVING clause is present, the GROUP BY clause can be omitted.

The existing aggregate functions are complemented with a new function called DEFAULT to support automatic aggregation. When applied to a measure M_j "DEFAULT" is substituted with the default aggregate function f_j, e.g., if $f_{\text{Cost}} = \text{SUM}$ then DEFAULT(Cost) becomes SUM(Cost). In this way a user need not be concerned with the aggregate functions once they are specified in the system.

Without loss of generality we do not allow several cubes in the FROM clause and we do not consider calculated measures, as these cases may be handled by creating a standard SQL view over one or more cubes. We do allow uncorrelated nested queries in the FROM clause.

The semantics of an SQL$_M$ query can now be expressed in terms of the cube algebra defined above: First, the selection operator is applied with the predicate from the WHERE clause, then generalized projection with the levels and measures listed in the SELECT and GROUP BY clauses, and finally a new selection is performed with the HAVING predicate.

Example 4.2 The query in Example 4.1 is evaluated as follows:

$$C' = \sigma_{Cube[\text{DEFAULT(Cost)}>10.000]}(\Pi_{Cube[\text{Supplier,Class}]<\text{DEFAULT(Cost)}>} (\sigma_{Cube[\text{Country(Supplier)='UK'}]}(\text{Purchases})))$$

□

Formally, we define the semantics of an SQL$_M$ query as follows:

Definition 4.1 (Semantics of an SQL$_M$ query) Let C be a cube, $\{L_1, \ldots, L_k\}$, and $\{L'_1, \ldots, L'_k\}$ be a set of levels from a subset of dimensions in C where $L_i \sqsubseteq_i L'_i$, $\{M_1, \ldots, M_l\}$ be a set of measures from C, $pred_{where}$ be a predicate over the levels and measures in C, and $pred_{having}$ be a predicate over levels $L'_1, \ldots, L'_k$, and measures $M_1, \ldots, M_l$.

The SQL$_M$ query

SELECT	$\text{DEFAULT}(M_1), \ldots, \text{DEFAULT}(M_l)$, $L'_1(L_1), \ldots, L'_k(L_k)$
FROM	C
WHERE	$pred_{where}$
GROUP BY	$L'_1(L_1), \ldots, L'_k(L_k)$
HAVING	$pred_{having}$

is evaluated as:

$$C' = \sigma_{Cube[pred_{having}]} (\Pi_{Cube[L'_1,\ldots,L'_k]<f_1(M_1),\ldots,f_l(M_l)>} (\sigma_{Cube[pred_{where}]}(C))),$$

where f_i is the default aggregate function for measure M_i.

□

4.2 SQL Compatibility

We now proceed to define how the SQL$_M$ query language is *compatible* with SQL. Compatibility is important as it allows users to leverage their knowledge of, in this case, SQL to understand and use the new language in a better way. Intuitively, a language L_1 is compatible with another language L_2 (in this case SQL$_M$ and SQL, respectively) if a query Q gives the same result when evaluated as an L_1 and L_2 query. Of course, this property only applies to a common subset of the languages. We now proceed to formalize this property.

Theorem 4.1 Let $C = (N, D, F)$ be a cube. Let the relational representation r of C, $r = R(C)$ be defined as: $R(C) = r$, where $\text{schema}(r) = \text{schema}(F)$ and $r = \{(e_{\perp_1}, \ldots, e_{\perp_n}, v_1, \ldots, v_m) \in F | \exists v \in v_1, \ldots, v_m : v \neq \text{NULL}\}$, i.e., C is mapped to a relation with the same schema as the fact table, containing the non-empty cells of the fact table as rows. Let Q be a query of the form

SELECT	$f_1(M_1)\ M_1, \ldots, f_l(M_l)\ M_l$, $L'_1(L_1)\ L'_1, \ldots, L'_k(L_k)\ L'_k$
FROM	C
WHERE	$pred_{where}$
GROUP BY	$L'_1(L_1), \ldots, L'_k(L_k)$
HAVING	$pred_{having}$

where the f_j's are the default aggregation functions for the measures M_j (not including DEFAULT) and the L'_i roll-up functions are total and single-valued. Then the following holds:

$$Q_{[SQL]}(R(C)) = R(Q_{[\mathsf{SQL}_M]}(C)),$$

where the $[SQL]$ and $[\mathsf{SQL}_M]$ subscripts indicate the semantics used for evaluation. □

Proof sketch: By inspecting the semantics of each side of the equation, we see that the theorem holds, because the roll-up functions are total and single-valued. This means that the result of evaluating the aggregation in SQL (note that such functions can appear in the GROUP BY clause in SQL-99) on the relational representation of C is the same as when evaluating the aggregation in SQL_M and converting to relations afterwards. Note that as we have no duplicates in the starting relations and do not introduce duplicates, the result is the same whether SQL (duplicates-allowed) or relational (duplicates-disallowed) semantics are used. The statement is constructed such that the result columns have the appropriate names from the cube even when evaluated with SQL semantics. The renaming ensures that the column names of the resulting relations are the same. In SQL_M, the renaming may be omitted to make the query more compact, one example of the additional power of SQL_M for OLAP queries.

We have now seen that SQL-99 and SQL_M are compatible within a common subset, meaning that users can leverage their knowledge of SQL when posing SQL_M queries and that existing applications can query SQL_M databases using (a subset of) SQL-99, ensuring integration with legacy system. We have also seen how SQL_M extends SQL-99 by adding powerful new features for handling *irregular* dimension hierarchies, *aggregation semantics*, and *automatic aggregation* (via the DEFAULT clause). However, the extension is done *gracefully*, so that existing SQL functionality keep its meaning, ensuring compatibility.

5 Advanced Uses: Integrating XML Data

In this section we describe an advanced application of the SQL_M data model and query language, namely the integrated use of XML data in SQL_M queries. This triggers advanced requirements for the underlying data model and query language, e.g., support for non-strict hierarchies and ensuring correct aggregation. This paper focuses on the challenges posed to the data model by the integration of external XML data, the details of the actual XML integration approach can be found in another paper [Pedersen et al., 2001a].

The integration is based on the use of XPath [W3C, 1999] expressions in the SQL_M queries. XPath is a language that provides powerful navigation in XML documents using a very compact syntax. As an example, the XPath expression //Component/Manufacturer[@MCode="M31"] selects the all *components* from the XML document in the case study where the MCode attribute has the value "M31." The fundamental mechanism is the *link* that specifies a relation between a set of dimension values in a cube and a set of nodes in an XML document, e.g., the link "EC_link" links the "EC" level in the OLAP cube with "Component" elements in the XML document in the case study. Links are used together with OLAP levels and XPath expression in the so-called *level expressions* in the extended SQL_M query language. For example, the level expression "EC/EC_link/Description" gives the associated descriptions for the ECs in the EC level of the cube. Each level can have a *default link*, so that the link need not be mentioned in the level expression, meaning that the above expression can be simplified to "EC/Description" if "EC_link" was the default link for the "EC" level. A SQL_M cube and a set of links and the associated XML documents is called a *federation*.

We now proceed to describe the three extensions of the SQL_M language, namely *decoration* with XML data, *grouping by* XML data, and *performing selections* based on XML data.

It is often useful to provide supplementary information for one or more levels in the result of an OLAP query. This is commonly referred to as *decorating* the result [Gray et al., 1997]. For example, products could be decorated with a competitor's prices for the same products, employees with their addresses, or suppliers with their contact person. Such information will often be available to the relevant people as Web pages on the Internet, an intranet, or an extranet. Also, this kind of information will most likely not be stored in an OLAP database because it either changes too frequently, was not expected to be used, is owned by someone else, or for some other reason. The solution presented next is to allow OLAP queries to reference external XML data using level expressions in the SELECT clause.

Example 5.1 Consider the result "AllTimePurchases" of the query in Example 3.6. Given the federation consisting of this result, the Components document, and the links defined above, the following query decorates all suppliers with their name from the Components document.

```
SELECT  Cost, Supplier, Supplier/SName, Class(EC)
FROM    AllTimePurchases
```
□

There are two important problems with the use of level expressions for decoration which are related to the problems with non-strict and non-covering hierarchies as discussed earlier. First, a dimension value may be associated with more than one node, i.e. when the level expression has a "to-many" cardinality, introducing a non-strict hierarchy. Second, some dimension values may not be associated with any nodes at all, in which case a non-covering hierarchy will occur.

In general, three semantics are possible for decoration:

ANY: Use an arbitrary node. This is useful when summarizability should be preserved and no node is more important than another, as might be the case e.g. when decorating suppliers with a contact person.

CONCAT: Use the concatenation of string values for all different nodes. Useful when summarizability should be preserved and all nodes are needed, e.g. when decorating products with text descriptions.

ALL: Use all different nodes, possibly duplicating facts. Useful when the decoration is used for selection and, in some cases, grouping.

The user specifies the semantics of a decoration by giving a semantic modifier in the level expression as in EC[ANY]/EC_Link/Description If no semantic modifier is specified ANY semantics are assumed as the default. Notice that, if the cardinality of the level expression is "to-one" then decoration with the three semantics will produce the same result.

Intuitively, only two things are changed when decorating a cube: A new dimension is added and the fact table is updated to reflect this. The new dimension contains only the decoration level and the top level. The new dimension values in the decoration level are created from an arbitrarily chosen node found by following the link from the starting level and then applying the XPath expression. If one or more values in the starting level does not produce any decoration values the special N/A value is used instead. The new fact table is created from the Cartesian product of the dimension values from the old fact table and the

new decoration values. Measure values are replaced with NULL values such that no facts are duplicated.

Allowing level expressions in the GROUP BY clause makes it possible to group by data from XML documents, without having to physically store this data in the OLAP database. For example, product prices will often be available from a supplier's Web page or an e-marketplace. These up-to-date prices can then be used to group products in an OLAP product database without having to store the prices.

Example 5.2 The following query groups ECs after their text descriptions from the Components document.

```
SELECT    DEFAULT(Cost), EC[ALL]/Description
FROM      PurchasesFederation
GROUP BY  EC[ALL]/Description
```
□

GROUP BY queries with level expressions are evaluated in two steps. First, the cube is decorated as described in the previous section. Second, aggregation is performed by using the already defined generalized projection Π_{Cube} on the new cube.

When decorating the cube, the new decoration dimension may be non-strict if ALL semantics are used and a bottom value is decorated by more than one decoration value. This is another reason for allowing non-strictness in a cube and for handling it in the generalized projection operator as defined in Definition 3.7. Consequently, if non-strictness occurs because of the decoration and if aggregation results in duplicate facts, this is handled by setting the aggregation type to c, preventing further aggregation. Only a sufficiently advanced OLAP data model can handle this.

XML data can also be used to perform selection over cubes. This makes it possible e.g. to view only products where a certain supplier is cheaper than another supplier by referring to their Web pages. The idea adopted here is to allow level expressions in WHERE and HAVING predicates in places where levels can already be used. For example, level expressions can be compared to constants, levels, measures, or other level expressions.

Example 5.3 Show component costs by supplier and EC but only those available for less than 3.00 euro.

```
SELECT    DEFAULT(Cost), Supplier, EC
FROM      PurchasesFederation
WHERE     EC/UnitPrice[@Currency='euro'] < 3.00
GROUP BY  Supplier, EC
```
□

As discussed earlier, selection semantics are also affected by the cardinality and covering properties of level expressions. As for selection over cubes, we handle this by using *any* semantics.

Selection over federations is evaluated by first decorating with all the level expressions mentioned in the predicate. The resulting federation is then sliced using the selection operator, and finally, the new decoration dimensions are removed again. The selection operator simply applies the cube selection operator to the cube part of the federation since the link and XML parts should not be affected by selection. ALL semantics are used for the decorations to make sure that all decoration values are available. This is important since *any* selection semantics are used in predicates, and thus, a predicate may be satisfied by any of the decoration values.

This section provided another set of examples of the need for an advanced OLAP data model that can handle non-strict and non-covering hierarchies, as well as supporting correct aggregation of data by keeping track of the summarizability of the data. Thus, a practical justification of the need for a *powerful* OLAP data model was provided.

6 Conclusion

Motivated by the increasing use of OLAP systems in complex domains, and the resulting need for an OLAP data model and query language that handles advanced features such as irregular dimension hierarchies and support for correct aggregation, while still allowing integration with legacy relational data, we presented the the "Multidimensional SQL" (SQL_M) data model, formal algebra, and high-level query language.

The data model, algebra, and query language are both *powerful*, meaning that they handled irregular hierarchies, automatic aggregation, and support for correct aggregation, and *SQL-compatible*, allowing seamless integration with legacy data sources. Irregular dimension hierarchies are handled using powerful dimension hierarchies based on partial orders, while support for correct aggregation is based on *aggregation types* that keep track of what data can/cannot be aggregated further without giving wrong results. The model, algebra, and language are designed such that compatibility with relational systems is ensured. The paper also considered the requirements to the data model posed by the problem of allowing integration with external XML data. The concepts were illustrated with a real-world case study from the B2B domain.

We believe this paper to be the first to present an OLAP data model, algebra, and query language that are both *powerful* and *SQL-compatible*. We also believe to be the first to consider the data model requirements posed by the integration of OLAP and XML data.

In future work, two research directions are promising. On the theoretical side, a full set of operators should be developed, to give the algebra the same power as relational algebra. In this context, it is interesting to develop algebraic transformation rules that can be used for query optimization. Additionally, it would be desirable to obtain a notion of completeness for multidimensional algebras, similar to Codd's relational completeness. On the practical side, efficient implementation of the model must be considered, e.g., by investigating how standard OLAP performance techniques such as pre-aggregation can be applied. Pre-aggregation for irregular hierarchies is a hard problem and it should be investigated how previous work in this area [Pedersen et al., 1999b] can be used.

References

[Agrawal et al., 1997] R. Agrawal, A. Gupta, and S. Sarawagi. Modeling Multidimensional Databases. In *Proceedings of the Thirteenth International Conference on Data Engineering*, pp. 232–243, 1997.

[Cabibbo et al., 1997] L. Cabibbo and R. Torlone. Querying Multidimensional Databases. In *Proceedings of the Sixth International Conference on Database Programming Languages*, pp. 319–335, 1997.

[Codd, 1993] E. F. Codd. Providing OLAP (on-line analytical processing) to user-analysts: An IT mandate. Technical report, E.F. Codd and Associates, 1993.

[ECIX, 2000] ECIX Council. `www.si2.org/ecix/`, 2000. Current as of July 24, 2001.

[Eisenberg et al., 1999] A. Eisenberg and J. Melton. SQL: 1999, formerly known as SQL 3. *SIGMOD Record* 28(1):131–138, 1999.

[Gray et al., 1997] J. Gray, S. Chaudhuri, A. Bosworth, A. Layman, D. Reichart, M. Venkatrao, F. Pellow, and H. Pirahesh. Data Cube: A Relational Aggregation Operator Generalizing Group-By, Cross-Tab and Sub-Totals. *Data Mining and Knowledge Discovery*, 1(1):29–54, 1997.

[Gupta et al., 1995] A. Gupta, V. Harinarayan, and D. Quass. Aggregate-Query Processing in Data Warehousing Environments. In *Proceedings of 21th International Conference on Very Large Data Bases*, pp. 358–369, 1995.

[Gyssens et al., 1997] M. Gyssens and L. V. S. Lakshmanan. A Foundation for Multi-Dimensional Databases. In *Proceedings of the Twentythird International Conference on Very Large Databases*, pp. 106–115, 1997.

[Jagadish et al., 1999] H. V. Jagadish, L. V. S. Lakshmanan, and D. Srivastava. What Can Hierarchies Do for Data Warehouses? In *Proceedings of the Twentyfifth International Conference on Very Large Data Bases*, pp. 530–541, 1999.

[Kimball, 1996] R. Kimball. *The Data Warehouse Toolkit*. Wiley, 1996.

[Klug, 1982] A. Klug. Equivalence of Relational Algebra and Relational Calculus Query Languages Having Aggregate Functions. *Journal of the ACM*, 29(3):699–717, 1982.

[Lehner, 1998] W. Lehner. Modeling Large Scale OLAP Scenarios. In *Proceedings of the Sixth International Conference on Extending Database Technology*, pp. 153–167, 1998.

[Lenz et al., 1997] H.-J. Lenz and A. Shoshani. Summarizability in OLAP and Statistical Databases. In *Proceedings of the Ninth International Conference on Scientific and Statistical Databases*, pp. 39–48, 1997.

[Li et al., 1996] C. Li and X. S. Wang. A Data Model for Supporting On-Line Analytical Processing. In *Proceedings of the Fifth International Conference on Information and Knowledge Management*, pp. 81–88, 1996.

[NHS, 1999] National Health Service (NHS). *Read Codes version 3*. NHS, September 1999.

[Pedersen et al., 2001a] D. Pedersen, K. Riis, and T. B. Pedersen. XML-Extended OLAP Querying. *Submitted for publication*, 25 pages, 2001.

[Pedersen et al., 1999a] T. B. Pedersen and C. S. Jensen. Multidimensional Data Modeling for Complex Data. In *Proceedings of the Fifteenth International Conference on Data Engineering*, pp. 336–345, 1999.

[Pedersen et al., 1999b] T. B. Pedersen, C. S. Jensen, and C. E. Dyreson. Extending Practical Pre-Aggregation in On-Line Analytical Processing. In *Proceedings of 25th International Conference on Very Large Data Bases*, pp. 663–674, 1999.

[Pedersen et al., 2001b] T. B. Pedersen, C. S. Jensen, and C. E. Dyreson. A Foundation For Capturing And Querying Complex Multidimensional Data. *Information Systems* 26(5):383–423, 2001.

[Rafanelli et al., 1983] M. Rafanelli and F. Ricci. Proposal of a Logical Model for Statistical Databases. In *Proceedings of the Second International Workshop on Statistical and Scientific Database Management*, 1983.

[Rafanelli et al., 1990] M. Rafanelli and A. Shoshani. STORM: A Statistical Object Representation Model. In *Proceedings of the Fifth Conference on Statistical and Scientific Database Management*, pp. 14–29, 1990.

[Shoshani, 1997] A. Shoshani. OLAP and Statistical Databases: Similarities and Differences. In *Proceedings of Sixteenth ACM Symposium on Principles of Database Systems* , pp. 185–196, 1997.

[Thomsen, 1997] E. Thomsen. *OLAP Solutions: Building Multidimensional Information Systems*. John Wiley & Sons, 1997.

[Thomsen, 1999] E. Thomsen, G. Spofford, and D. Chase. *Microsoft OLAP Solutions*. Wiley, 1999.

[Vassiliadis, 1998] P. Vassiliadis. Modeling Multidimensional Databases, Cubes, and Cube Operations. In *Proceedings of the Tenth International Conference on Statistical and Scientific Database Management*, pp. 53–62, 1998.

[W3C, 1999] W3C. Xml path language (xpath) version 1.0. `www.w3.org/TR/xpath`, November 1999. Current as of July 24, 2001.

[W3C, 2000a] W3C. Extensible markup language (xml) 1.0 (second edition). `www.w3.org/TR/REC-xml`, October 2000. Current as of July 24, 2001.

[Yahoo, 2001] Yahoo! Corporation. <www.yahoo.com>. Current as of July 24, 2001.

[Zurek et al., 1999] T. Zurek and M. Sinnwell. Data Warehousing Has More Colours Than Just Black and White. In *Proceedings of the Twentyfifth International Conference on Very Large Data Bases*, pp. 726–729, 1999.

A Syntax

The SQL_M language is given by the following syntax:

```
<query>               ::= SELECT <select list>
                          FROM Cube | <query>
                          [WHERE <where predicate>]
                          [GROUP BY <group by list>
                          [HAVING <having predicate>]]
<select list>         ::= <select name>
                        | <select list>, <select name>
<select name>         ::= <level name>
                        | <aggregate function>(Measure) [<column alias>]
<aggregate function>  ::= DEFAULT | COUNT | SUM | MIN | MAX | AVG
<where predicate>     ::= NOT(<where predicate>)
                        | <where predicate> AND|OR <where predicate>
                        | (<where predicate>)
                        | <where expression>
<where expression>    ::= <where name> <predicate operator> <where name>
                        | <where name> <predicate operator> <value>
                        | <where name> IN (<value list>)
                        | <where name> LIKE 'String'
<where name>          ::= <level name>
                        | Measure
<group by list>       ::= <group by name>
                        | <group by name>, <group by list>
<group by name>       ::= <level name>
<having predicate>    ::= NOT(<having predicate>)
                        | <having predicate> AND|OR <having predicate>
                        | (<having predicate>)
                        | <having expression>
<having expression>   ::= <having name> <predicate operator> <having name>
                        | <having name> <predicate operator> <value>
                        | <having name> IN (<value list>)
                        | <having name> LIKE 'String'
<having name>         ::= <select name>
<level name>          ::= Level
                        | Level(Level) [<column alias>]
<column alias>        ::= String
<predicate operator>  ::= <> | = | > | >= | < | <=
<value>               ::= 'String' | Real | Integer
<value list>          ::= <value>
                        | <value>, <value list>
```