Object-extended OLAP querying

Torben Bach Pedersen a,*, Junmin Gu b, Arie Shoshani b, Christian S. Jensen a

a Aalborg University, Department of Computer Science, Selma Lagerløfsgade 300, DK-9220 Aalborg Ø, Denmark
b Scientific Data Management Group, Lawrence Berkeley National Laboratory, 1 Cyclotron Road, MS 50B-3238, Berkeley, CA 94720, USA

A R T I C L E   I N F O

Article history:
Received 11 May 2007
Received in revised form 31 October 2008
Accepted 31 October 2008
Available online 19 November 2008

Keywords:
On-line analytical processing (OLAP)
Multidimensional databases
Object databases
Data integration
Summarizability

A B S T R A C T

On-line analytical processing (OLAP) systems based on a dimensional view of data have found widespread use in business applications and are being used increasingly in non-standard applications. These systems provide good performance and ease-of-use. However, the complex structures and relationships inherent in data in non-standard applications are not accommodated well by OLAP systems. In contrast, object database systems are built to handle such complexity, but do not support OLAP-type querying well.

This paper presents the concepts and techniques underlying a flexible, “multi-model” federated system that enables OLAP users to exploit simultaneously the features of OLAP and object systems. The system allows data to be handled using the most appropriate data model and technology: OLAP systems for dimensional data and object database systems for more complex, general data. This allows data analysis on the OLAP data to be significantly enriched by the use of additional object data. Additionally, physical integration of the OLAP and the object data can be avoided. As a vehicle for demonstrating the capabilities of the system, a prototypical OLAP language is defined and extended to naturally support queries that involve data in object databases. The language permits selection criteria that reference object data, queries that return combinations of OLAP and object data, and queries that group dimensional data according to object data. The system is designed to be aggregation-safe, in the sense that it exploits the aggregation semantics of the data to prevent incorrect or meaningless query results. These capabilities may also be integrated into existing languages. It is shown how to integrate relational and XML data using the technology. A prototype implementation of the system is reported, along with performance measurements that show that the approach is a viable alternative to a physically integrated data warehouse.

© 2008 Elsevier B.V. All rights reserved.

1. Introduction

On-line analytical processing (OLAP) systems have become increasingly popular in many application areas, as they considerably ease the process of analyzing large amounts of enterprise data. Designed specifically with the aim of better supporting the retrieval of higher-level summary information from detail data, these systems offer substantial additional user-friendliness over general database management systems (DBMSs). The special dimensional data models employed in OLAP systems enable visual querying, as well as contribute to enable OLAP systems to offer better performance for aggregate queries than do traditional DBMSs. As another example, most OLAP systems support automatic aggregation [34,49], which means that the system knows which aggregate functions to apply when retrieving different higher-level summaries.

* Corresponding author.
E-mail addresses: tbp@cs.aau.dk (T.B. Pedersen), jgu@lbl.gov (J. Gu), shoshani@lbl.gov (A. Shoshani), csj@cs.aau.dk (C.S. Jensen).

0169-023X/$ - see front matter © 2008 Elsevier B.V. All rights reserved.
Almost all OLAP systems are based on a dimensional view of data, in which numeric values, termed measures, are characterized by descriptive values drawn from a number of dimensions; and the values of a dimension are typically organized in a containment-type hierarchy, where an upper-level value contains several lower-level values, e.g., a diagnosis group contains several low-level diagnoses. While the dimensional view of data is particularly well suited for the aggregation queries performed in OLAP analysis, it also limits the abilities of OLAP systems to capture complex relationships in the data. As a result, an OLAP database only captures some of the structure available in the data from which it derives. Furthermore, it is often difficult or impossible to combine data from an OLAP system with data from other sources.

In contrast, object database (ODB) systems excel at capturing and querying general, complex data structures. These systems offer semantically rich data models and query languages that include constructs such as classes, inheritance, complex associations between classes, and path expressions. However, ODB systems do not support aggregate queries well. For example, the complex data structures tend to make it hard to formulate correct queries that aggregate the data in the ODB. Also, ODB systems are optimized to perform more general types of queries, mostly on the detail level, so the performance for aggregate queries is usually not satisfactory.

Federated database systems [14,25,26,52] support the logical integration of autonomous database systems, without requiring data to be physically moved and while allowing the individual autonomous database systems to function as before. Federation is a flexible solution that may leverage existing technology and adapt quickly to changing information requirements. In contrast, physical integration of data, commonly referred to as the physical (as opposed to logical) data warehousing approach [57]. This approach has its own advantages, perhaps most significantly in terms of performance when combining data from different databases, but it is very difficult to keep the warehouse data up to date. Thus, it is often impossible or impractical to use physical data warehousing, especially if the data sources belong to different organizations. The two approaches are complimentary, in that they are appropriate under different circumstances.

When integrating data from databases based on different data models, the traditional approach has been to map all data into one common data model and federate the (logically) transformed data rather than the original data [52,25,14]. In this paper, we adopt an alternative approach that combines data from multidimensional databases (MDDBs) and object databases using a federated database approach, where data is handled using the most appropriate data model and database technology: MDDB systems for multidimensional data and ODB systems for complex, general data. No attempt is made at “shoeorning” the data into one common format, which is unlikely to fit all the data.

Focus is on enabling OLAP-style queries over existing MDDBs to also include data from existing, external ODBs, without jeopardizing the benefits of OLAP queries and without having to integrate the OLAP and the object data physically. Note that this is different from a “virtual DW” approach where all data reside in the source OLTP systems, as our approach uses a physical MDDB for the core multidimensional data. This will allow the data analysis over the OLAP data to be significantly enriched by the use of the extra object data. Specifically, aggregation safety remains enforced, meaning that incorrect or meaningless extended queries are avoided. As a first step in demonstrating the capabilities of the system, a prototypical, user-oriented query language for MDDBs, termed SumQL, is defined. The concept of a link, which enables the connection of MDDBs to ODBs in a general and flexible manner, is then integrated into SumQL along with object features, yielding an extended language, termed SumQL++.

With this language as a vehicle, it is shown how the system enables using path expressions for referencing data in MDDBs in selection criteria. Queries over MDDBs may return ODB data along with the aggregate results, i.e., the result of an OLAP query may be decorated with object data. Finally, MDDB data may be grouped based on ODB data. All extensions are accompanied by formal definitions in terms of SumQL and the underlying object query language (the ODMG data model and OQL query language [9] are used for the ODBs). The paper’s contribution is presented in terms of the SumQL and SumQL++ languages, which are defined formally in the paper and concisely capture the relevant concepts, to be self-contained and ensure precision. Other languages such as SQL [36], OQL [9], and MDX [37] may take the place of SumQL++ once enriched with the constructs in SumQL++ that they do not already offer. We also describe how the approach can handle external relational and XML data using an object wrapping.

A prototype has been built [22] that supports the execution of SumQL++ queries over a federation of autonomous MDDBs and ODBs. Performance experiments show that the federated approach in most cases have faster query response time than a relational DW where the external data has been physically integrated, using only a quarter of the storage space of the relational solution.

The major contribution of this paper is to consider the integrated querying of data from existing independent multidimensional and object databases without prior physical integration of the databases, with the objective of giving OLAP users enhanced, aggregation-safe query capabilities. Surveys of OLAP data models and languages [41,54,56] indicate that this issue has not been addressed previously. To our knowledge, the paper is also novel in demonstrating a “multi-paradigm” (or “multi-model”) federation [4,24,25], where one of the data models is a dedicated multidimensional data model. Finally, the paper is the first to investigate the important issue of how OLAP concepts such as summarizability and aggregation safety are influenced by federation with external data and how they may be preserved to ensure safe query results.

The remainder of the paper is structured as follows: Section 2 presents a real-world case study and motivates the federation of multidimensional and object databases. Section 3 introduces the foundations for the MDDBs and ODBs. It describes a prototypical multidimensional data model and its high-level, user-oriented multidimensional query language, SumQL, as well as the central concept of summarizability. It also briefly presents the Object Data Management Group (ODMG) data model and its OQL query language. Section 4 describes the notion of link that connects MDDBs to ODBs, and Section 5
proceeds to describe the federated data model, which incorporates links, and its extended SumQL query language, which enables queries to access information in both MDDBs and ODBs. Section 6 describes how to integrate relational and XML data sources. Section 7 describes the prototype implementation of a system that implements the concepts and techniques presented, while Section 8 contains a set of performance experiments that demonstrate the practicality of the approach. Section 9 describes related work. The last section summarizes and offers research directions. Finally, an appendix describes the formal syntax and semantics of SumQL.

2. Motivation

In this section, we discuss why it is a good idea to federate existing multidimensional and object databases and present a real-world case study that is used for illustration throughout the paper.

2.1. Reasons for federation

Many reasons exist for preferring federating existing MDDBs and ODBs, as opposed to physically integrating these. The generic arguments for federation include leveraging existing technology, accessing the most current information, and allowing the autonomous existence of the systems being federated. These arguments also apply in this case, so we concentrate on the advantages specific to multidimensional and object databases.

In many situations, MDDBs only contain abstract summary data and do not contain the base data from which the summary data is derived, thus rendering access to external databases necessary to be able to answer certain specific queries. For example, multidimensional databases provided by the Ministry of Health do not permit access to base data, because the base data is unavailable or considered too sensitive for general disclosure, e.g., diagnosis information. The same situation arises in census databases, where only high-level information is disclosed publicly.

In many cases, an organization maintaining a specific database will not allow other organizations to maintain a copy of their database, but will however allow that certain specific queries from external systems are answered. This may be due to legislation, copyright, or license issues. For example, the Danish building register allows external systems to query the details of a specific building, but explicitly prohibits other parties to maintain a copy of the database.

Federating MDDBs and ODBs enables a simple and special-purpose MDDB system. An MDDB needs not contain all objects, attributes, and relationships in the base database, but only the elements relevant to summary querying. This is attractive, as capturing all information in the MDDB unnecessarily impedes casual use of the MDDB system. Indeed, most OLAP systems that implement MDDBs do not have the necessary facilities, e.g., category inheritance [33], to support this extra information. The federated approach allows the MDDB to remain simple, while still allowing access to relevant external data. When MDDB data resides in a special-purpose MDDB system, we cannot use existing database middleware to access it, leading to a need for technology that enables federations of MDDBs and ODBs. Using our approach, the multidimensional schema in the MDDB may be enriched with the extra object data, meaning that the extended MDDB schema becomes powerful enough to handle all analysis requirements.

It is possible to obtain better performance when performing multidimensional querying in an OLAP-type system rather than in a general-purpose DBMS. The former type of system typically employs specialized, performance enhancing techniques, such as multidimensional storage and pre-aggregation. This performance gain can often outweigh the performance loss due to the fact that the data is not physically integrated, meaning that a federated system can have comparable (or even better) performance without the limitations incurred by physical integration.\(^1\)

Next, it is easier to formulate multidimensional queries in an MDDB system than in a general (relational or object) DBMS. This is because an MDDB query language is designed exclusively for expressing multidimensional queries over categories, taking advantage of, e.g., the automatic aggregation implied by the multidimensional database semantics. Even when extending an MDDB language to access object data (as we do in Section 5), it is easier to pose multidimensional queries in the extended language than in a general database query language such as OQL or SQL.

An MDDB system may support the formulation of multidimensional queries that return correct, or meaningful, query results. When building an MDDB, the data may be shaped in order to satisfy summarizability conditions [34]. Briefly, a multidimensional query satisfies summarizability conditions if the query result is correct w.r.t. the real world. For example, summarizing the populations over cities to get summaries for states will produce incorrect results if the populations in towns and farms outside cities are not accounted for. As another example, if patients have several diseases, and we summarize over all diseases to get the total number of sick people, we will get the wrong result as some patients are counted more than once. We may enrich an MDDB system with information that enables the system to ensure correctness. For example, we may specify that inventory levels should not be added across time [34] or that patient counts for diseases should not be added. In a general-purpose DBMS, no mechanisms for ensuring correct aggregate results are available.

The federated approach offers additional flexibility when query requirements change. MDDBs may be huge, and therefore rebuilding them may be time consuming. This is true even when using optimized DW evolution techniques (please see Section 9 for a discussion). Updates to an MDDB, e.g., adding new types of information, may require a total or partial rebuild of

\(^1\) However, we are not suggesting that already integrated databases should be split up for performance reasons.
the database. Because of the rebuild time, a rebuild of the MDDB will most likely be refused by the IS department or postponed to the next scheduled rebuild, e.g., once a week or once a month. This situation occurs very frequently in practice. In contrast, a new link can be added in a matter of minutes, yielding much faster access to newly required information. This allows rapid prototyping of OLAP systems. In a relational DB setting, the ability to do this rapid prototyping is one of the key selling points for the Cohera federated DBMS [23].

An issue may be that the federated approach “disturbs” existing OLTP databases. However, the external object DBs may or may not be OLTP databases, that is an implementation choice made by the system designer, e.g., based on performance considerations. There are many examples of object databases that are read-mostly and thus not OLTP systems, so the issue of disrupting OLTP systems is orthogonal to our proposal and thus not investigated further.

Another problematic issue may be that some data integration problems are harder to handle in a federated setting than in an integrated setting. The most notable of these problems is that of data cleansing. Most often, data cleansing need not be an important issue when using our approach, due to the relatively simple queries that we perform over the object data, but in some cases, data quality may prevent the correct use of external object data. Performing data cleansing in our approach is however outside the scope of the present paper, and is left for future work.

In summary, the above reasoning suggests that in many (but not all) cases, it is advantageous to logically federate existing OLAP and object databases instead of performing physical integration of the databases.

2.2. Case study

The case study concerns data in three different databases, each managed by a separate organization. Each database serves a different purpose, but the databases contain related data. A graphical illustration of the databases is seen in Fig. 1.

The databases are modeled using the Unified Modeling Language (UML) [38]. Compound boxes denote classes. The class name is in boldface in the top part of the box, while class attributes are listed in the middle part. The bottom part is reserved for class methods, i.e., dynamic aspects of the class, but since we are only interested in the data, methods are omitted. Associations, i.e., relationships, between classes are represented by lines tagged with an association name. The cardinality of an association is shown by the numbers at the ends of the association line. Either a single cardinality or a range of cardinalities are specified. A “*” denotes any natural number.

The demographic database is maintained by the Department of the Interior and offers central access to demographic data for all cities and states in the country. Data is collected for states, for which name and area is stored, and for cities, for which name and population is recorded. The database also contains information about the current mayor of a city. There are zero or more cities in each state, and each city has exactly one current mayor.
Next, the admissions database is maintained by the Department of Health and provides an overview of the admissions patterns for all hospitals nationwide. For an admission, the date of admission and the reason for admission, e.g., accident, are recorded. Additionally, we record which hospital the patient is admitted to and the primary diagnosis that caused the admission. For hospitals, the name and the state where the hospital is located are recorded. For diagnoses, we record an alphanumeric code, determined by a standard classification of diseases, e.g., the World Health Organization’s International Classification of Diseases (ICD-10) [58]. The classification also determines how the diagnoses are grouped into diagnosis groups. Diagnosis groups consist of at least two related diagnoses and a diagnosis belongs to exactly one diagnosis group. For diagnosis groups, we record a alphanumeric code, determined by the classification.

The last database is an epidemiology database maintained by a medical school for research purposes. Data are collected from hospitals, practicing physicians, and insurance companies to obtain a rich overview of the occurrence of diseases. The database is organized around the diagnoses in the standard disease classification also used in the admissions database, but more information is recorded. In addition to the alphanumeric code and an additional descriptive text, the database also records the number of incidences per year, the number of deaths per year, and whether the disease is dependent on the lifestyle of the patient. The Diagnosis class has two subclasses, Contagious Diagnosis and Non-contagious Diagnosis. For contagious diagnoses, we additionally record the mode of transfer of the disease, e.g., by air. The symptoms of diseases are also recorded. For symptoms, we record a name and a description of the symptom.

The three databases were built and are used separately, which explains the differences in their information contents. But, we want to use them together, to include information from the demographic and epidemiology databases in queries against the admissions database. Thus, we need to provide at least a logical integration of the databases.

In this case study, the organizations maintaining the separate databases will not allow other organizations to maintain a copy of their database. However, they do allow for certain, specific queries from external systems to be answered. Thus, physical integration of the three databases is not possible, but a federated solution, based on our approach, is. The federated solution will possess all the advantages described in Section 2.1, both the advantages related to the MDDB system, and in particular the flexibility with respect to new query requirements.

To obtain some example data, we assume a standard mapping of the UML schemas to relational schemas, i.e., one table per class, and relationships expressed using foreign keys. We also assume the use of surrogate keys, named ID, with globally unique values. Subclasses are supported by sharing of IDs with the superclass. For example, the Contagious Diagnosis subclass is represented by a separate table with the ID shared with the Diagnosis table. The tables for the demographic, admissions, and epidemiology databases are shown in Tables 1–3, respectively.

### 3. Federation data models and query languages

This section defines a prototypical multidimensional data model and query language used for the MDDB component in the federation; and it briefly presents the data model and query language of the federation’s ODB component.

The multidimensional model precisely and concisely captures core multidimensional concepts such as categories, dimensions, and automatic aggregation. As part of this, the notion of summarizability is defined. The multidimensional data model and query language are equivalent in expressive power to previous approaches such as the ones proposed by Cabbibo et al. [8] and Jagadish et al. [30]. The ODB data model and query language is the ODMG data model and OQL query language.

### Table 1
Data for the demographic database.

<table>
<thead>
<tr>
<th>State table</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
</tr>
<tr>
<td>----</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Mayor table</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
</tr>
<tr>
<td>----</td>
</tr>
<tr>
<td>20</td>
</tr>
<tr>
<td>21</td>
</tr>
<tr>
<td>22</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>City table</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
</tr>
<tr>
<td>----</td>
</tr>
<tr>
<td>10</td>
</tr>
<tr>
<td>11</td>
</tr>
<tr>
<td>12</td>
</tr>
</tbody>
</table>
3.1. Multidimensional data model

The model has constructs for defining the schema, the instances, and the aggregation properties.

An n-dimensional fact schema is a two-tuple $\mathcal{F} = (\mathcal{F}, \mathcal{D})$, where $\mathcal{F}$ is a fact type and $\mathcal{D} = \{ \mathcal{D}_i, i = 1, \ldots, n \}$ is its corresponding dimension types. A fact type is a name describing the type of the facts considered.

### Table 2
Data for the admissions database.

<table>
<thead>
<tr>
<th>ID</th>
<th>Day</th>
<th>Reason</th>
<th>HospitalID</th>
<th>DiagnosisID</th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
<td>05/23/99</td>
<td>Accident</td>
<td>40</td>
<td>50</td>
</tr>
<tr>
<td>31</td>
<td>04/12/99</td>
<td>F.P. referral</td>
<td>41</td>
<td>51</td>
</tr>
<tr>
<td>32</td>
<td>05/01/98</td>
<td>Specialist referral</td>
<td>41</td>
<td>52</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>StateID</th>
</tr>
</thead>
<tbody>
<tr>
<td>40</td>
<td>Alta Bates</td>
<td>70</td>
</tr>
<tr>
<td>41</td>
<td>Portland General Hospital</td>
<td>71</td>
</tr>
<tr>
<td>42</td>
<td>Portland Kaiser</td>
<td>71</td>
</tr>
</tbody>
</table>

### Table 3
Data for the epidemiology database.

<table>
<thead>
<tr>
<th>ID</th>
<th>Code</th>
<th>Text</th>
<th>Deaths</th>
<th>Incidences</th>
<th>Lifestyle</th>
</tr>
</thead>
<tbody>
<tr>
<td>80</td>
<td>E10</td>
<td>Insulin dependent diabetes</td>
<td>50,000</td>
<td>900,000</td>
<td>Yes</td>
</tr>
<tr>
<td>81</td>
<td>E11</td>
<td>Non insulin dependent diabetes</td>
<td>20,000</td>
<td>1,500,000</td>
<td>Yes</td>
</tr>
<tr>
<td>82</td>
<td>N12</td>
<td>Pneumonia</td>
<td>100,000</td>
<td>1,000,000</td>
<td>No</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ID</th>
<th>TransferMode</th>
</tr>
</thead>
<tbody>
<tr>
<td>82</td>
<td>Air</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>90</td>
<td>Cough</td>
<td>The lungs of the patient …</td>
</tr>
<tr>
<td>91</td>
<td>Acetone Breath</td>
<td>The breath of the patient …</td>
</tr>
<tr>
<td>92</td>
<td>Fever</td>
<td>The temperature of the patient…</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>DiagnosisID</th>
<th>SymptomID</th>
</tr>
</thead>
<tbody>
<tr>
<td>80</td>
<td>91</td>
</tr>
<tr>
<td>81</td>
<td>91</td>
</tr>
<tr>
<td>82</td>
<td>90</td>
</tr>
<tr>
<td>82</td>
<td>92</td>
</tr>
</tbody>
</table>
Example 1. In the case study we will have Admissions as the fact type, and Diagnosis, Place, Reason, and Time as the dimension types.

A dimension type $\mathcal{F}$ is a four-tuple $\langle \mathcal{E}, \leq_\mathcal{F}, \top_\mathcal{F}, \bot_\mathcal{F} \rangle$, where $\mathcal{E} = \{ \mathcal{E}_j | j = 1, \ldots, k \}$ are the category types of $\mathcal{F}$, $\leq_\mathcal{F}$ is a partial order on the $\mathcal{E}_j$’s, with $\top_\mathcal{F} \in \mathcal{E}$ and $\bot_\mathcal{F} \in \mathcal{E}$ being the top and bottom element of the ordering, respectively. The intuition is that one category type is “greater” than another category type if each member of the former’s extension logically contains several members of the latter’s extension, i.e., they have a larger element size. The top element of the ordering corresponds to the largest possible element size, that is, there is only one element in its extension, logically containing all other elements. We say that $\mathcal{E}_j$ is a category type of $\mathcal{F}$, written $\mathcal{E}_j \in \mathcal{F}$, if $\mathcal{E}_j \in \mathcal{E}$. We assume a function $\text{Anc} : \mathcal{E} \rightarrow 2^\mathcal{E}$ that gives the set of immediate ancestors of a category type $\mathcal{E}_j$. The immediate ancestor(s) of a category type lie immediately above the category type in the hierarchy.

Example 2. Diagnoses are contained in Diagnosis Groups. Thus, the Diagnosis dimension type has the following order on its category types: $\bot_{\text{Diagnosis}} = \text{Diagnosis} < \text{DiagnosisGroup} < \top_{\text{Diagnosis}}$. Thus, $\text{Anc} (\text{Diagnosis}) = \{ \text{Diagnosis Group} \}$. Other examples of category types are Day, Month, and Year. Fig. 2, to be discussed in detail in Example 6, illustrates the dimension types of the case study.

A category $C_j$ of type $\mathcal{E}_j$ is a set of dimension values $e$. A dimension $D$ of type $\mathcal{F} = \langle \{ \mathcal{E}_j \}, \leq_\mathcal{F}, \top_\mathcal{F}, \bot_\mathcal{F} \rangle$ is a two-tuple $D = (\mathcal{C}, \leq)$, where $\mathcal{C} = \{ C_j \}$ is a set of categories $C_j$ such that $\text{Type}(C_j) = \mathcal{E}_j$, and $\leq$ is a partial order on $\bigcup C_j$, the union of all dimension values in the individual categories.

The partial order is defined as follows. Given two values $e_1, e_2$ then $e_1 \leq e_2$ if $e_1$ is logically contained in $e_2$, i.e., $e_2$ can be considered as a set containing $e_1$. We say that $C_j$ is a category of $D$, written $C_j \in D$, if $C_j \in C$. For a dimension value $e$, we say that $e$ is a dimensional value of $D$, written $e \in D$, if $e \in \bigcup C_j$.

We assume a partial order $\leq_D$ on the categories in a dimension, as given by the partial order $\leq_\mathcal{F}$ on the corresponding category types. The category $\bot_D$ in dimension $D$ contains the values with the smallest value size. The category with the largest value size, $\top_D$, contains exactly one value, denoted $\top$. For all values $e$ of the categories of $D$, $e \leq \top$. Value $\top$ is similar to the ALL construct of Gray et al. [21]. We assume that the partial order on category types and the function $\text{Anc}$ work directly on categories, with the order given by the corresponding category types.

Example 3. The Diagnosis dimension has the following categories, named by their type. Diagnosis = (50,51,52), Diagnosis Group = (60,61), and $\top_{\text{Diagnosis}} = \{ \top \}$. The values in the sets refer to the ID fields in the Diagnosis and Diagnosis Group tables in Table 2. The partial order $\leq$ is given by the GroupID field in the Diagnosis table. Additionally, the top value $\top$ is greater than, i.e., logically contains, all the other diagnosis values.

Let $C_1, \ldots, C_n$ be categories and $T$ a domain that includes the special value null. A measure for these categories and this domain is a function $M : C_1 \times \cdots \times C_n \rightarrow T$. We say that $M$ is a measure for the set of dimensions $D = \{ D_1, \ldots, D_n \}$, if $M$ is a measure for the categories $\bot_{D_1}, \ldots, \bot_{D_n}$. Every measure $M$ has associated with it a default aggregate function $f_M : T \times T \rightarrow T$. The default aggregate function must be distributive. The null value is used to indicate that no data exists for a particular combination of category values. As is the case for SQL, the aggregate functions ignore null values.

Example 4. In the case study we have one measure, TotalAdmissions, which is the total number of admissions by Diagnosis, Place, Time, and Reason. The default aggregation function is SUM.

![Fig. 2. Multidimensional model for the admissions database.](image-url)
The measures associated with each dimension may have different aggregation properties. For different kinds of measures, different aggregate functions are meaningful. For example, it is meaningful to sum up the number of admissions; and because this data is ordered, it is also meaningful to compute the average, minimum, and maximum values. In contrast, in at least some situations, it may not be meaningful to compute the sum (over time) of measures such as the number of patients hospitalized, but it remains meaningful to compute the average, minimum, and maximum values. Next, it makes little sense to compute these aggregate values on data such as diagnoses, which do not have any ordering defined on them. Here, the only meaningful aggregation is the count of occurrences. Whether or not an aggregate function is meaningful also depends on the dimensions being aggregated over. For example, patient counts may be summed over the Place dimension, but not over the Time dimension. For additional discussion of these issues, we refer to [34].

By recording what aggregate functions may be meaningfully applied to what data, it is possible to support correct aggregation of data. With such information available, it is possible to either completely reject “illegal” aggregation or to warn the users that the results may not be meaningful.

Following previous research [33,41,48], we distinguish between three distinct sets of aggregate functions: $\Sigma$, applicable to data that may be added together, $\phi$, applicable to data that can be used in average calculations, and $c$, applicable to data that may only be counted.

Considering only the standard SQL aggregate functions, we have that $\Sigma = \{\text{SUM, COUNT, AVG, MIN, MAX}\}$, $\phi = \{\text{COUNT, AVG, MIN, MAX}\}$, and $c = \{\text{COUNT}\}$. The aggregation types are ordered, $c \subseteq \phi \subseteq \Sigma$. If a set of aggregate functions is meaningful for some data, so are the functions in lower sets.

For each measure $M$ for a set of dimensions $D = \{D_1, \ldots, D_n\}$, we assume a function $\omega(M): D \mapsto \{\Sigma, \phi, c\}$ that gives the aggregation type for each dimension. In Section 3.2 we further discuss issues related to correct aggregation of data.

**Example 5.** In the case study, $\omega_{\text{TotalAdmissions}}(\text{Diagnosis}) = \Sigma$.

An $n$-dimensional multidimensional database (MDDB) is a 3-tuple $S = (\mathcal{S}, D, M)$, where $\mathcal{S}$ is the schema, $D = \{D_1, \ldots, D_n\}$ is a set of dimensions, and $M = \{M_1, \ldots, M_k\}$ is a set of measures for the categories $\perp D_1, \ldots, \perp D_n$.

**Example 6.** The case study has a four-dimensional multidimensional database with Diagnosis, Place, Reason, and Time as dimensions. There is one measure, the TotalAdmissions, as described above. A graphical illustration of the MDDB is seen in Fig. 2.

### 3.2. Summarizability

This section defines summarizability, an important property of MDDBs related to the use of pre-computed aggregates. Intuitively, summarizability captures when higher-level aggregates may be obtained directly from lower-level aggregates.

**Definition 1.** Given a type $T$, a set $S = \{S_j, j = 1, \ldots, k\}$, and a function $g: 2^T \mapsto T$, we say that $g$ is summarizable for $S$ if $g(g(S_1), \ldots, g(S_k)) = g(S_1 \cup \cdots \cup S_k)$. The argument on the left-hand side of the equation is a multiset, i.e., the same value may occur multiple times.

Summarizability is important since it is a condition for the flexible re-use of computed aggregates. Without summarizability, (pre-computed) lower-level results generally cannot be correctly combined into higher-level results. In such situations, we have to compute the higher-level results from base data, which may be computationally expensive.

It has been shown that summarizability is equivalent to the aggregate function ($g$) being distributive and the mappings between dimension values in the hierarchies being, strict, covering, and onto [34]. These properties are defined formally elsewhere [34,41,42]. Informally, summarizability requires that the dimension hierarchies take the form of balanced trees, i.e., all paths from the root have the same length (onto), links between values do not “skip” levels (covering), and all values below the root have exactly one parent (strictness). If hierarchies do not have this form, some lower-level values will either be double-counted or ignored.

Summarizability is closely related to the aggregation types defined in the previous section. We use the aggregation types to capture when it is safe to aggregate a measure over a given dimension. If we have aggregated over a non-summarizable hierarchy, e.g., a diagnosis hierarchy where one diagnosis is part of several diagnosis groups, it is not permissible to use the aggregate results for the diagnosis groups to compute the result for the entire dimension, as the same admissions will then be counted more than once. We use the aggregation types to prevent this. Problems related to summarizability also occur when we extend the queries over MDDBs to include data from external ODBs, see Section 5 for details.

### 3.3. The summary query language

The query language of the MDDB component is termed SumQL and is meant to be language that makes it easy for the user to pose aggregate queries over MDDBs. We have chosen to define a separate summary language rather than attempting to augment the object query language, OQL, for querying MDDBs because we wish to refer explicitly to the special data structures in MDDBs.

Using OQL, or some variant thereof, for querying MDDBs would mean that we would have to overload some of the language constructs, re-using them with a different meaning. This is undesirable, as it confuses the meaning of statements in the language. Also, we would have to introduce OLAP constructs such as measures, dimensions, and hierarchies, which would conflict
with the generality of the object model. Finally, the focus of this paper is to allow integrated querying of existing MDDBs which do not use ODB technology and existing ODBs, rather than providing OLAP-style querying over object databases only.

SumQL is reminiscent of SQL but includes constructs that reflect MDDB concepts such as measures, dimensions with hierarchically organized categories, and automatic aggregation, thus supporting naturally the expression of aggregate queries over multidimensional databases. Using SumQL enables us to concisely and precisely define the extensions for referencing object data.

The general format of a SumQL query is displayed below and explained in the following. Symbol “+” indicates one or more occurrences and square brackets denote optional parts. The formal syntax and semantics of SumQL are given in Appendix A.

\[
\text{SumQL query ::= } \text{SELECT measure}^* \\
\text{[INTO MDDB]} \\
\text{BYCATEGORY category}^* \\
\text{FROM \{MDDB\}[SumQL query]} \\
\text{[WHERE predicate}\_\text{clause]}
\]

The SELECT clause contains a list of measures for which a result is to be computed. Unlike in SQL, aggregate functions such as SUM need not be specified; rather, the default aggregation function specified in the schema is automatically applied to aggregate the data. An optional INTO clause follows that specifies the MDDB into which the result of the query is stored. Thus, a SumQL query takes an MDDB as argument and return an MDDB as result.

The BYCATEGORY clause specifies the aggregation level at which the measures are to be computed. For each dimension not mentioned in this clause, the measures are aggregated over the whole dimension, i.e., the same behaviour as SQL GROUP BY clauses. Effectively, all dimensions and measures not mentioned in the BYCATEGORY and SELECT clauses are ignored.

The FROM clause specifies either the MDDB from which to aggregate or a subquery that returns an MDDB for further aggregation. For simplicity, we only consider queries over one MDDB, and no "drill-across" or "union" functionality is provided. However, the data model and query language can easily be extended to handle this (see [41] for an example). The optional WHERE clause specifies predicates that are applied to the MDDB before aggregation occurs. The predicates can include standard constructs such as comparison operators, set operators, and string operators. These constructs are equivalent or similar to those found in SQL and OQL [9].

Example 7. The following SumQL statement computes the “Total Admissions” measure from the “admissions” MDDB, aggregated to the level of Year and State, for the years after 1997. The resulting MDDB is called "testdb."

\[
\text{SELECT TotalAdmissions INTO testdb BYCATEGORY year, state} \\
\text{FROM admissions WHERE year > 1997}
\]

3.4. The object model and query language

This section briefly reviews the object data model and query language used by the ODB component of the federation. We use the Object Data Management Group’s object data model, ODMG 2.0 [9], and its associated query-language, OQL. The ODMG data model includes constructs such as object class definitions, attributes, object identifiers, set-valued attributes, reference attributes, tuple attributes, inverse attributes, inheritance structures, and object class unions. An in-depth coverage of the ODMG data model and the OQL language may be found in the literature [9].

Example 8. Data definitions for the demographic (left column) and epidemiology (right column) databases from the case study are shown in Fig. 3. Keyword “INVERSE” indicates that the contents are the inverse of a relationship in another class. The "::" denotes a sub-class relationship, while “Set (X)” specifies a set-valued relationship.

The OQL query language has constructs such as path expressions and class selectors. Path expressions are used to navigate through reference attributes to other classes using dot-notation, while class selectors restrict queries to operate only on a certain subclass.

Example 9. The following query uses a path expression to select the city name only for cities where the current mayor is more than 40 years old. The path expressions navigates from cities to mayors via reference attribute “current_mayor”.

\[
\text{SELECT C.name FROM C IN City WHERE C.current_mayor.age > 40}
\]

Example 10. The next query navigates from symptoms to the diagnoses that exhibit those symptoms using a path expression and then applies a class selector (the square brackets) to select the attribute “transfer_mode” of the Diagnosis sub-class “Contagious Diagnosis.” Thus, only transfer modes for contagious diagnoses with the symptom “Cough” are returned:

\[
\text{SELECT S.diagnoses[ContagiousDiagnosis].transfer_mode} \\
\text{FROM S IN Symptom WHERE S.name="Cough"}
\]
4. Linking databases

This section defines the links that are used to connect MDDBs and ODBs. As mentioned in the introduction, we use explicit links to connect the databases, rather than relying solely on implicit knowledge of relationships among the databases when formulating queries.

Explicit links are preferable for several reasons. First, even if the data in the MDDB is derived from source data in an ODB, the complete mapping may be unknown because of substitutions for missing data and other types of data cleansing, interpolation, etc. Second, explicit links are needed when linking an MDDB to an unrelated ODB, i.e., an ODB other than the base data from which the MDDB was extracted. So, we propose to explicitly link even multidimensional data to the base data from which it was derived.

Links are considered separately from the federated databases to better capture their special semantics and to aid the optimization of queries involving links. However, links can be physically implemented as part of these databases.

Formally, a link \( L \) from a category \( C \) to an object class \( O \) is a relation \( L = \{ (c, o) \} \), where \( c \in C \) and \( o \in O \). All links have a name to distinguish them. This is because each category and even pair of category and object class may have several links.

Links may be specified in several ways. An equivalence link is specified by a predicate \( C = O.a \), where \( C \) is a category, \( O \) is an object class, and \( a \) is an attribute of \( O \) that uniquely identifies instances of \( O \), i.e., \( a \) is a candidate key for \( O \) in relational database terms. Equivalence links occur when a category in the MDDB represents the same real-world entities as does some object class in an ODB. An attribute link is specified by the same type of predicate, the only exception being that \( a \) does not uniquely identify instances of \( O \). An enumerated link is given by a link relation \( L = \{ ((c, o)) \} \), where pairs of dimension values in \( C \) and object ids from class \( O \) are explicitly enumerated. Therefore multiple dimension values may be assigned to the same object. Enumerated links are typically used for linking a category in an MDDB and an object class that do not represent the same real-world entities. The actual mapping from dimension values to object ids in an enumerated link may be obtained from (a combination of) existing data sources (if such sources exist) or it may be “hand crafted” by domain experts solely for the purpose of serving in the federation.

**Example 11.** In our case study, we can specify an equivalence link between the Diagnosis category in the Admissions MDDB and the Diagnosis Class in the Epidemiology ODB by the predicate “Diagnosis = Diagnosis.Code,” as the values of the diagnosis category are the codes of the diagnoses. In subsequent examples, we term this link “diag_link”.
Example 12. An enumerated link from the Hospital category in the MDDB to the City class in the Demographic ODB may be specified by explicitly assigning hospitals to cities based on where the hospitals are located. The contents of the link relation is \( L = \{ \text{"Alta Bates", "Berkeley"}, \text{"Portand General Hospital", "Portland"}, \text{"Portand Kaiser", "Portland"} \} \). We will use the name "city_link" for this link.

The cardinality of a link is an important property, as the cardinality may affect summarizability. The cardinality of a link \( L = \{ (c, o) \} \) between category \( C \) and object class \( O \) is \( [1 - 1] \) if \( |L| = |\pi_c(L)| = |\pi_O(L)| \), where \( \pi \) denotes relational projection and \( \cdot \) denotes relation cardinality; the cardinality of \( L \) is \( [n - 1] \) if \( |L| = |\pi_c(L)| > |\pi_O(L)| \); and the cardinality is \( [1 - n] \) if \( |L| = |\pi_c(L)| < |\pi_O(L)| \). Finally, if the cardinality of \( L \) is not \( [1 - 1] \), \( [1 - n] \), or \( [n - 1] \), its cardinality is \( [n - n] \). For some link properties, only the cardinality of the object side of a link is interesting. As a short-hand notation, we say that the cardinality of a link is \( [1 - 1] \) if the cardinality is \( [1 - 1] \) or \( [n - 1] \). Similarly, the cardinality of a link is \( [1 - n] \) if the cardinality is \( [1 - n] \) or \( [n - n] \).

Example 13. The cardinality of link "diag_link" is \( [1 - 1] \) and the cardinality of "city_link" is \( [n - n] \).

It is also necessary to capture whether some dimensions values or objects do not participate in a link. For that purpose, we define that a link \( L = \{ (c, o) \} \) from category \( C \) to object class \( O \) covers \( C \) if \( C = \pi_c(L) \). Similarly, \( L \) covers \( O \) if \( O_i = \pi_O(L) \), where \( O_i \) is the set of object ids for \( O \). If \( L \) covers both \( C \) and \( O \), \( L \) is complete; otherwise, \( L \) is incomplete.

Example 14. The "diag_link" link is complete, while the "city_link" link covers the Hospital category, but not the City class. For example, the city of Oakland is not present in the link.

In Section 5 we will explore the effect of these link properties on the semantics of queries. Specifically, we shall see that incomplete links and \( [1 - n] \) links, which are analogous to non-summarizable hierarchies, require special attention. Interestingly, an attribute link always has a link cardinality that is \( [1 - n] \), while an equivalence link always has a \( [1 - 1] \) cardinality.

In some situations, it is desirable to have links that are more powerful than enumerated links. For example, the database designer may want to annotate links with what may be termed metadata, e.g., the reason why the link was added, who added the link, or the time interval when the link is valid.

Such annotated links do offer additional modeling capabilities, but are nevertheless excluded. The reason is that offering a general solution along these lines—which allows general annotations, including complex object structures with set-valued attributes, references to other classes, embedded objects, etc.—would amount to the reinvention of a complete object model, an unnecessary complication.

Instead, we propose that annotations be stored in a separate ODB, and we propose to store the potentially complex link information in a separate ODB using a link class that represents the instances of the link. We may then create a normal link from the desired category to this link class. The link class would also be linked to the desired object class that we wanted to link to originally.

We do not consider links between ODBs, as this is supported by object database federation systems, e.g., the "OPM*QS" multidatabase system [13].

5. The federated data model and query language

Having described the data models and query languages of the MDDB and ODB components to be federated, as well as a minimal mechanism for linking MDDBs and ODBs, the next step is to provide language facilities that enable OLAP-type queries across the entire federation. Specifically, we extend SumQL.

The federation approach presented here has the distinguishing feature that it uses the aggregation semantics of the data to provide aggregation-safe queries, i.e., queries that do not return results that are incorrect or meaningless to the user. This section describes how the previously defined concepts of aggregation types, summarizability, link cardinality, and link coverage combine to provide aggregation-safety for queries.

5.1. The federated data model

The federation consists of a collection of independent components, supplemented with additional information and components that enable functioning of the federation. Specifically, the federation consists of an MDDB, a number of ODBs, and links that interrelate information in the different databases. Formally, a federation \( F \) of an MDDB \( S \) and a set of ODBs \( O = \{ O_1, \ldots, O_n \} \) is a three-tuple \( F = (S, O, L) \), where \( L = \{ L_1, \ldots, L_m \} \) is a set of links from categories in the dimensions of \( S \) to classes in \( O_1, \ldots, O_n \).

We assume only a single MDDB. Permitting multiple MDDBs introduces additional challenges, e.g., the matching of categories and dimensions, which are not covered here. The case of a single MDDB is very useful, as typical queries to a federation naturally centers around one MDDB: Typical queries concern MDDB measures, grouped by MDDB categories, and involving selection criteria relating to data from the ODBs; or queries retrieve ODB data along MDDB data; and in some cases, it is desirable to actually group MDDB data by categorical ODB data.

Rather than requiring that the MDDB and ODB data comply with one common data model, the federation adopts a multi-paradigm approach [24,4], where the data remain in their original data models. This approach has previously been advocated...
in programming languages, where research has been done on how to allow programs to be written that exploit imperative, object-oriented, functional, and logical programming paradigms in a single program [7].

Allowing multiple data models (or paradigms) to co-exist in the federation enables us to exploit the strengths of the different data models and query languages when managing and querying the data. In particular, the availability of multiple paradigms allows a problem solution to take advantage of the fact that certain subsets of a problem are often well suited for one solution paradigm, while other problem subsets are better suited for other paradigms.

Like the arguments to queries are federated databases, the results are also federated databases, i.e., query results may have MDDB, ODB, and link components. This closure property mirrors those of the well-known relational, object, and multidimensional data models and query languages, and permits the result of one query to be used in a subsequent query. We allow the sets of ODBs and links, $O$ and $L$, to be empty. Thus, an MDDB in itself is a federation.

5.2. The SumQL++ language

As our objective is to allow more powerful OLAP queries over MDDBs by allowing the queries to include data from ODBs, we take SumQL as the outset and extend this language. The new, extended language is termed “SumQL++” as it introduces object-oriented concepts into its predecessor, akin to the C++ successor to the C programming language.

The queries we are interested in are the typical OLAP queries that select a set of measures from an MDDB, grouped by a set of categories. Three extensions of SumQL are useful in this respect. First, we introduce path expressions in selection predicates, in order to integrate ODB data. Second, we introduce so-called decorations [21] of SumQL results, which enable ODB data to be returned along with the SumQL result. Third, SumQL is extended to enable MDDB data to be grouped by data belonging to ODBs, i.e., attributes of object classes, rather than just the built-in MDDB categories.

5.2.1. Extended selection predicates

The first extension of SumQL is to allow selection predicates that reference ODB data. The most basic idea is to allow the use of standard OQL path expressions, as described in Section 3.4, in the category expressions in the selection predicates, using the well-known dot-notation for path expressions.

The link that is used to get to the ODB is included in the category expression. A category expression always starts with an MDDB category, and is followed by an optional part consisting of the link name and a path expression. Inside the path expressions, class selectors may occur that restrict predicates to work on selected (sub)classes. The syntax is shown below. The square brackets in single quotes in the “class_connector” rule denote (sub)class selection and are part of the language being defined. Nonterminals not defined below are strings.

\[
\begin{align*}
\text{category} \rightarrow & \text{category} \cdot \text{link} \cdot \text{object_path} \cdot \text{attribute} \\
\text{object_path} \rightarrow & \text{class} \cdot \text{connector} \cdot \text{path_list} \\
\text{class_connector} \rightarrow & \cdot [\text{class}] \\
\text{path_list} \rightarrow & \text{class} \cdot \text{connector} \cdot \text{path_element} \cdot \text{path_list} \cdot \text{path_element} \\
\text{path_element} \rightarrow & \text{reference} \cdot \text{attribute} \cdot \text{class} \cdot \text{connector}
\end{align*}
\]

Example 15. We want to use the Epidemiology ODB to get the total admissions by year for only the diagnoses for which cough is a symptom. We use the “diag_link” link to do so in the following the SumQL++ statement.

```
SELECT TotalAdmissions INTO testdb BY_CATEGORY Year FROM Admissions
WHERE Diagnosis.diag_link.symptoms.name="Cough"
```

Example 16. We use a class selector in the Epidemiology ODB to get the total admissions by year for only contagious diagnoses with the transfer mode “Air,” with the following SumQL++ statement.

```
SELECT TotalAdmissions INTO testdb BY_CATEGORY Year FROM Admissions
WHERE Diagnosis.diag_link[ContagiousDiagnosis].transfer_mode="Air"
```

To describe the semantics of this extension to SumQL, we first need some additional definitions. Given a category expression $E$ of the form $E = C \cdot L \cdot OP \cdot a$, where $C$ is a category, $L$ is a link, $OP$ is an object path (as defined in the syntax above), and $a$ is an attribute of an object class, the cardinality of $E$ is defined next.

Let $R$ be the set of attribute values resulting from the OQL query “SELECT $X \cdot k, X \cdot OP \cdot a$ FROM $X \cdot Y$”, where $Y$ is the class that $L$ links to, $k$ is the attribute that $L$ links to in $Y$, and $OP$ and $a$ are as above. Let $L'$ be the link relation obtained by performing a natural join of $L$ with $R$, i.e., $L' = L \bowtie R$, where $\bowtie$ denotes natural join. We say that $L'$ is the link specified by $E$. The cardinality of $E$ is defined as the link cardinality of $L'$.

Informally, the cardinality of a category expression is the combination of the cardinalities that we encounter as we go through the link and the subsequent (possibly set- or bag-valued) reference-attributes, i.e., going through a $[-1]$ relation.
ship in a link or a reference attribute does not change the running cardinality, but a \([-n]\) relationship causes the total cardinality to be \([-n]\). The type of the result depends on the declarations in the ODB. If the object path includes only set-(or single-)-valued attributes, the result type is a set, if the path includes at least one attribute declared as a bag (including list and array types), the result type is a bag.

Using the definitions above, and following the definitions given for links, we say that \(E\) covers \(O\), does not cover \(O\), covers \(C\), does not cover \(C\), is complete, and is incomplete, if \(L\) covers \(O\), does not cover \(O\), covers \(C\), does not cover \(C\), is complete, or is incomplete, respectively. Above, \(O\) is the object class that \(a\) is an attribute of, i.e., the last object class reached in the category expression. We say that \(O\) is the final class of \(E\). \(C\) is the category in the beginning of \(E\). We say that \(C\) is the starting category of \(E\).

**Example 17.** The cardinality of the category expression “Hospital.city_link.locatedin.name” is \([n-1]\) as we only go through \([n-1]\) relationships and the state name is a key attribute. The cardinality of the category expression “Diagnosis.diag_link.symptoms.name” is \([n-1]\) because the “symptoms” reference attribute is set-valued.

The cardinality and covering properties of a category expression affect the meaning of a SumQL++ statement. If the cardinality is \([-1]\), the predicate will only reference one attribute value and the meaning is clear. However, if the cardinality is \([-n]\), the predicate will reference more than one attribute value, leading to several possible semantics for the query.

For example, the category predicate “Diagnosis.diag_link.symptoms.name” in **Example 15** has a \([-n]\) cardinality. One possible interpretation of this is that all the referenced attribute values must match the predicate, e.g., that all symptoms must have name “Cough”. Another interpretation is that at least one attribute value must satisfy the predicate, e.g., that at least one symptom has name “Cough”. This is the interpretation chosen in the OQL language, and as we also think it is the most sensible to end users, we will also adopt this interpretation.

Similar problems may arise when a category expression \(E\) does not cover its starting category \(C\), because \(L\) then will be undefined for the uncovered dimension values of \(C\). However, if we adopt our previous interpretation, that at least one attribute value must match the predicate, the meaning is well-defined. The values in \(C\) not covered by \(E\) will then be excluded from the selection. There are no problems if \(E\) does not cover its final class \(O\), as \(L\) will be defined for all the instances of \(O\) referenced by \(E\).

Formally, the semantics of the extended SumQL++ predicates are as follows. We are given a SumQL++ query \(Q\) with a number of category predicates \(P_1, \ldots, P_n\) of the form \(P_i = E \in \text{OP}_i \text{V}_i\). The \(E_1, \ldots, E_n\) are category expressions of the form \(E_i = C \cdot L_i \cdot \text{OP}_i \cdot a_i\), \(i = 1, \ldots, n\), where \(C_i\) is a category, \(L_i\) is a link, \(\text{OP}_i\) is an object path, and \(a_i\) is an attribute of the final class of \(E_i\). The \(\text{OP}_i\) are the predicate operator parts of \(P_i\), i.e., comparison and BETWEEN, IN, and MATCH operators. The \(V_i\) are the value parts of the predicates.

For each \(E_i\), let \(R_i\) be the set of attribute values resulting from the OQL query “\(\text{SELECT } X_i \cdot k_i \text{ FROM } X_i \text{ IN } Y_i \text{ WHERE } \text{OP}_i \cdot a_i \text{OP}_i\)” where \(Y_i\) is the class that \(L_i\) links to, and \(k_i\) is the attribute that \(L_i\) links to. For each predicate \(P_i\), we now form a modified predicate \(P_i' = C_i \text{IN}(e_1, \ldots, e_k)\), where \(\{e_1, \ldots, e_k\} = \pi_{C_i}(L_i \bowtie R_i)\) (\(\bowtie\) denotes natural join). Informally, we obtain the attribute values for the link class for which the predicate holds, then obtain the corresponding dimensions values by joining with the link, and finally form a (pure) SumQL predicate with the resulting dimension values using the “IN” notation.

With \(Q\) being the query obtained from \(Q\) by substituting all the \(P_i\)s with the \(P_i'\)s, the result of evaluating \(Q\) on a federation \(F = (S, O, L)\) is the federation \(F' = (S', \emptyset, \emptyset)\), where \(S'\) is the MDBB resulting from evaluating \(Q'\) on \(S\). This federation has no ODB or links components, which makes sense as the ODB data was only used to select a subset of the MDBB for evaluation.

**Example 18.** We evaluate the query from **Example 15**. First we get the result of the query “\(\text{SELECT } D\text{.code }\text{FROM } D \text{ IN Diagnosis }\text{WHERE } D\text{.symptoms.name} = \text{"Cough"}\)” The result of this is the set \(R = \{\text{"N12"}\}\) (the code for pneumonia). We then join \(R\) with the link relation “\(\text{diag_link}\)”, which is the identity relation, and project over the Diagnosis category, obtaining the dimension value “\(\text{N12}\)”. We then form the pure SumQL query: “\(\text{SELECT } \text{TotalAdmissions} \text{ INTO } \text{testdb BY}_\text{CATEGORY} \text{ Year FROM Admissions }\text{WHERE Diagnosis IN }\{\text{"N12"}\}\)” evaluating it over the Admissions MDBB.

### 5.2.2. Decorating the query result

It is often desirable to display additional descriptive information along with the result of an MDBB query. This is commonly referred to as decorating the result of the query [21]. For example, when asking for the number of admissions by hospital, it may be desirable to display the name of the city and the name of the city’s mayor along with the hospital name.

This can be achieved by extending the SumQL with features for decorating the result. One possibility would be to allow category expressions with path expressions in the SELECT clause, but we advise against this as it would then be unclear which parts of the SELECT clause referred to measures and which parts referred to decorations. Instead, we extend SumQL with an optional “WITH” clause. The extended syntax is shown below.

\[
\text{SumQL query} ::= \text{SELECT measure}^* \\
\quad \text{[INTO MDBB]} \\
\quad \text{BY}_\text{CATEGORY} \text{ category}^* \\
\quad \text{[WITH expression]*} \\
\quad \text{FROM (MDBB | (SumQL query))} \\
\quad \text{[WHERE predicate_clause]}
\]
Example 19. Using this extension, we select the number of admissions by hospital, decorated with the names of the city and its mayor.

```
SELECT TotalAdmissions INTO testdb
    BY_CATEGORY Hospital
    WITH Hospital.city_link.name, Hospital.city_link.current_mayor.name
    FROM Admissions
```

It only makes sense to decorate the result with data that is correlated to the original query result, so the categories referenced in the WITH clause MUST be part of the BY_CATEGORY clause.

Formally, assume a SumSQL++ query $Q$ with category expressions $E_1, \ldots, E_n$ in the WITH clause of the form $E_i = C_i \cdot L_i \cdot OP_i \cdot a_i$ $i = 1, \ldots, n$, where $C_i$ is a category, $L_i$ is a link, $OP_i$ is an object path, and $a_i$ is an attribute of the final class of $E_i$, the semantics is as follows. For each $E_i$, let $R$ be the result of the OQL query “SELECT $X_i \cdot k_i \cdot X' \cdot OP_i \cdot a$ FROM $X_i$ IN $Y'$”, where $Y'$ and $k_i$ are the class that $L_i$ links to and the attribute that $L_i$ links to, respectively. Then form a new object class $Z_i$ from the set of tuples $L_2 \cdot R$, using the concatenation of the category $C_i$ and the attribute $a_i$ as its object identifier. Let $Q'$ denote $Q$, but without the WITH clause. The result of evaluating $Q$ over a federation $F = (S, O, L)$ is the federation $F' = (S', O', L')$, where $S'$ is the result of evaluating $Q'$ over $F$, $O' = \{Z_i\}$, and $L' = \{L'_i\}$, where $L'_i$ are attribute links specified by $C_i = Z_i \cdot C_i$.

Thus, the decoration data is returned in the ODB and link parts of the federation and is not integrated into the result MDDB. This loose coupling of decoration data and MDDB data is essential in avoiding semantic problems, which might otherwise occur if the category expressions $E_i$ do not cover the categories $C_i$. In this case, we just return decoration data matching a subset of the $C_i$, i.e., we perform an operation equivalent to an outer join. Similarly, no cardinalities for the $E_i$s cause problems. If the cardinality of $E_i$ is $[n]$, e.g., for the expression “Diagnosis.diag_link.symptoms.name,” the object class simply contains several objects for each $C_i$ value, e.g., there will be two objects, with the symptom names “Cough” and “Fever,” with Diagnosis value “N12” (pneumonia).

Example 20. For the query in Example 19, we get two object classes in the result, CityName with the attributes “hospital,” “name,” and “city=hospital,” with the latter as the object identifier, and MayorName with the attributes “hospital,” “name,” and “mayor=hospital,” again with the latter as the object identifier. The links have the specifications “Hospital=CityName.Hospital” and “Hospital=MayorName.Hospital.”

5.2.3. Grouping by object class attributes

The last extension is to allow the measures of an MDDB to be grouped by attribute values in ODBs, enabling aggregation over hierarchies outside the MDDB. This feature will be used when aggregation requirements change suddenly.

To achieve this, we allow category expressions instead of just categories in the BYCATEGORY clause. The syntax of the extension is given below. The only difference from the previous syntax is that the BYCATEGORY clause now is a list of category expressions rather than just a list of categories. Remember that a category expression is either a category or a category followed by a link, an object path, and an attribute.

```
SumSQL query::=SELECT measure* 
    [INTO MDDB] 
    BYCATEGORY expression* 
    [WITH expression*] 
    FROM (MDDB | (SumSQL query)) 
    [WHERE predicate_clause]
```

Example 21. The number of admissions grouped by symptoms may be retrieved as follows.

```
SELECT TotalAdmissions INTO testdb 
    BYCATEGORY Diagnosis.diag_link.symptoms.name FROM Admissions
```

This type of SumSQL++ queries will return MDDBs where one new dimension is added for each category expression in the BYCATEGORY clause, thereby reflecting the hierarchy specified by the category expression, and aggregation will occur over these new dimensions.

Formally, given a SumSQL++ query,

```
Q = SELECT M_1, \ldots, M_k INTO db BYCATEGORY E_1, \ldots, E_n FROM S WHERE P,
```

with the category expressions in the BYCATEGORY clause being of the form $E_i = C_i \cdot L_i \cdot OP_i \cdot a_i, i = 1, \ldots, n$, where $C_i$ is a category, $L_i$ is a link, $OP_i$ is an object path, and $a_i$ is an attribute of the final class of $E_i$, the result of $Q$ on federation $F = (S, O, L)$ may be specified as follows.
First, let \( S = (\mathcal{S}', D', M') \) be the MDDB obtained from \( S \) as follows. For each \( E_i \), add a new dimension type to \( S \) with the category types \( \mathcal{T}_i, A_i \), and \( \bot_i \). Category type \( A_i \) represents the attribute values of \( a_i \), while category type \( \bot_i \) represents the dimension values of the bottom category in \( S \). The ordering of the types is \( \mathcal{T}_i > A_i > \bot_i \). Thus, \( \mathcal{S}' \) is specified.

For each dimension type, new dimensions \( D' \) are added to \( D' \). The categories of \( D' \) correspond to the category types. The \( \mathcal{T}_i \) category has just the \( \top \) value. If \( L'_i \) is the resulting link of \( E_i \), category \( A'_i \) has the values given by \( \pi_{a_i}(L'_i) \). Let \( R_i \) be the relation specified by \( (e_1, e_2) \in R_i \iff e_1 \in \bot_i \land e_2 \in A_i \land e_1 \leq e_2 \), i.e., the relation specified by the partial order between \( \bot_i \) values and \( A_i \) values. Let \( B_i = R_i \bowtie L'_i \) (\( \bowtie \) is the natural join). Then the values of the category \( \bot_i \) is the set \( \pi_{\bot_i}(B_i) \). The partial order on dimension \( D'_i \subseteq A'_i \), is specified as follows: \( e_1 \leq e_2 \iff e_2 = \top \lor e_2 = e_\top (e_1, e_2) \in B_i \). This completes the specification of \( D' \).

The set of measures \( M' \) is identical to the original set of measures \( M \) as the measures operate on the same categories. If \( S \) is a SumQL query itself, we define the semantic meaning of \( S \) to be the MDDB resulting from the evaluation of the subquery.

The result of evaluating the SumQL query “\( \text{SELECT } M_1, \ldots, M_k \text{ INTO } S' \text{ BY_CATEGORY } A'_1, \ldots, A'_n \text{ FROM } S' \text{ WHERE } P' \)” is the federation \( P' = (S', \emptyset, \emptyset) \). The ODB and links components are empty, as the ODB data has been turned into dimensions in this result.

**Example 22.** For the query in Example 21 we get one new dimension type “\( \text{SymptomName} \)” with the category types “\( \top \text{SymptomName} \)” “\( \text{SymptomName} \)” and “\( \text{Diagnosis} \)” The new “\( \text{SymptomName} \)” dimension has the categories specified by the category types. The partial order on the new dimension is given by joining the “\( \text{Diagnosis} \)” “\( \text{Diagnosis_Symptom} \)” and “\( \text{Symptom} \)” tables from Table 3 and then projecting on the “\( \text{Code} \)” and “\( \text{Name} \)” attributes. We note that the resulting hierarchy is non-strict, as the “\( \text{Acetone Breath} \)” symptom occurs for both “\( \text{Insulin Dependent Diabetes} \)” and “\( \text{Non Insulin Dependent Diabetes} \)”.

Depending on the properties of the \( E_i \)S, problems may occur in the aggregation process. If \( E_i \) does not cover \( C_i \), some of the data in the MDDB (the data characterized by the non-covered subset of \( C_i \)) will not be considered in the aggregate result. Reversely, if \( E_i \) does not cover its final class \( O_i \), there will not be any measure data associated with the non-covered objects in \( O_i \). This means that the result of the aggregation function will be undefined for multidimensional tuples containing the non-covered objects. To remedy these problems, we require that the \( E_i \)s be **complete**.

Even when the category expressions are complete, special attention is needed to ensure summarizability. Problems may occur when the cardinality of an \( E_i \) is \([-n] \), in which case the same measure data, e.g., the same admissions, will be accounted for more than once in the overall result, e.g., for different symptoms.

This result is meaningful and correct in itself because the data belongs to several groups. However, the result should not be used for further aggregation as the same data may then be accounted for more than once for the same group, e.g., we may not aggregate over all symptoms to get the total number of admissions. To avoid this, we set the aggregation type for all measures to \( c \), i.e., we disallow further aggregation on the data, if the cardinality of \( E_i \) is \([-n] \). If the cardinality of \( E_i \) is \([-1] \) the aggregation types are not changed.

Finally, we note that it does not make sense to introduce a “\( \text{WITH-like} \)” construct for the \( \text{BY_CATEGORY} \) clause. The reason is that it does not make sense to “decorate” the grouping attributes, as the query either groups by a given attribute (and thus includes it in the \( \text{BY_CATEGORY} \) clause) or not (and thus leaves it out of the \( \text{BY_CATEGORY} \) clause).

5.3. **Summary**

Although the extensions to SumQL were described separately above, they can be used together in one SumQL++ statement. Assuming an SumQL++ statement that contains all three extensions, query evaluation proceeds as follows. First, the rules for handling grouping by object attributes are used, producing a statement without object attribute grouping. This statement is then processed using the rules for the \( \text{WITH} \) clause described in Section 5.2.2, resulting in a statement without a \( \text{WITH} \) clause, which can then be evaluated using the rules for extended selection predicates as described in Section 5.2.1. The statement produced by the extended predicate rules is a pure SumQL statement which may be evaluated following standard SumQL semantics.

6. **Integrating relational and XML data**

We have now demonstrated the capability of our system to logically integrate external object data with OLAP data. However, a lot of structured data is stored in relational databases and, increasingly, in XML formats. Thus, it is highly desirable to be able to access relational and XML data as well. This section describes how to integrate such data sources using our approach.

The extension with relational and XML data can be achieved in several ways. One way would follow the “multi-paradigm” approach of the paper, and extend the current data model and query language with two new sets of constructs for integrating relational and XML data, respectively. For example, for the XML integration, these constructs would be similar to those proposed for OLAP-XML federations [43,61]. However, this would be a very significant, and thus infeasible, extension of the scope (and size) of the present paper. Instead, we will explain a more pragmatic approach that wraps relational and XML data as object data, and thus allows their integration using the current approach.
Relational data. For relational systems, the mapping to object classes is rather simple. For each table, one object class with the same name is constructed. Non-key attributes in the table are simply transferred to the object class as attributes. The primary key for the table is designated as the object id for the corresponding class. Foreign keys in the table are mapped to object reference attributes, i.e., single-valued relationships, and corresponding reverse relationships are added in the target classes. This is the default mapping performed by the OPM toolset [10,11]. Thus, integration with relational data can easily be performed using our approach. In fact, the object database used in the prototype system described in the next section is a relational DBMS with an OPM object mapping on top of the relational schema. The OPM system already takes care of mapping all OQL data manipulation queries into their SQL equivalents, and we thus need not specify this mapping ourselves.

XML data. The solution for handling XML data is rather similar. We first note that we are only interested in integrating with XML data that is highly structured and complying with a strict schema. If the XML data is highly irregular and does not comply with a strict schema, we will have very little knowledge of the possible data values and thus it will be either very risky or entirely meaningless to perform OLAP-style analyses based on such data. In the XML database community, it is common to distinguish between these two forms of XML data, where the structured form is called “XML data” and the unstructured form is called “XML documents” [50]. Object databases are well suited for storing the regular “XML data” variety [50]. For regular XML data, the following mapping to object classes is performed. We assume that at least a Document Type Description (DTD) [59] is available, more advanced schema information such as XML Schema is handled similarly. For each element type, an object class with the same name (if necessary prefixing parent element names for uniqueness) is constructed. Simple (non-key and non-nested) attributes are transferred to the object class. If an attribute declared as ID is found this is used as the object id, otherwise an artificial object id is constructed. IDREF(S) attributes are transformed into single-valued (set-valued) object reference attributes, and the corresponding reverse relationship attributes are added to the target classes. Nested elements are handled by adding a set-valued reference attribute with the name of the nested element in the “parent” class, pointing to the “child” class, and adding a single-valued “parent” reference attribute to the child class, pointing to the parent class. If the document order should be preserved, an “order” attribute is added to the child class. The order attribute is initialized based on the child’s order relative to the parent and can be queried like any other class attribute. As for data manipulation queries, we have to map OQL queries to XQuery/XPath queries. This can be done using the approach of the STyx system [3].

7. Prototype implementation

This section describes the architecture and a proof-of-concept prototype implementation of the OLAP++ federated system capable of answering SumQL++ queries, the query evaluation strategy, and presents performance results obtained with the prototype.

7.1. System architecture

The overall architecture of the federated system is seen in Fig. 4. The system has six major components: the GUI, the ODB systems, the link DB system, the MDDB system, the federation coordinator, and the metadata database. The ODB, link DB, and MDDB components are treated as independent units by the federation system; only their published interfaces are used, and

Fig. 4. Architecture of the federated system.
The link component stores enumerated links and is placed in an independent “link” DB, as it cannot generally be assumed that these links may be stored in some ODB component. Should this be possible, we can choose to do so, e.g., to obtain better performance. The operation of the prototype is entirely based on federation metadata specified in the metadata database. This allows for a very flexible system that may adapt quickly to changes. For example, if a new connection to an outside ODB is desired, appropriate links just needs to be specified and stored as metadata, after which queries can start using the new ODB. Queries are generated by the GUI and sent to the federation coordinator which then parses the query. Based on the content of the query, the system looks up the relevant metadata (link specifications, ODB names, etc.) in the metadata database and processes the query according to the metadata by issuing queries to the DB components.

The parts of the system handling object and link data are based on the commercially available OPM tools [10,35,18] that implement the Object Data Management Group’s (ODMG) object data model [9] and the Object Query Language (OQL) [9] on top of a relational DBMS, in this case the ORACLE RDBMS. The use of the OPM tools means that it is not important which particular version of the RDBMS is used, e.g., it makes no difference whether the RDBMS has object-relational features or not. In-depth descriptions of the OPM toolset exist in the literature [10–12]. The OLAP part of the system is based on Microsoft’s SQL Server OLAP Services using the Multi-Dimensional eXpressions (MDX) [37] query language. The graphical user interface (GUI) is implemented as Java classes running in a standard Web browser for optimal flexibility. The prototype is a proof-of-concept prototype aiming to demonstrate the feasibility of our approach. It handles only extended predicates, not decoration or grouping by object data, as we think that extended selection predicates is by far the most useful of the operations. Since grouping by object data is the only operation that may have an impact on summarizability, it is thus not necessary for the prototype to perform checks for summarizability.

7.2. Query evaluation

The query evaluation strategy is explained using an example. Example 23 below explains how a particular query is processed.

Fig. 5 shows the specification of query conditions. Initially, each dimension is shown with its categories and links to the object database. If a category is selected, a category condition can be entered. In the figure, Region=‘‘ASIA’’ was selected. If a link is clicked on, then the attributes of the object linked to are shown. The user can select an attribute to specify a condition. In the figure, the condition “population >100 Million” was selected through the “nationlink”. The result of the above selections is a concise SumQL++ query, as shown next.

```
SELECT TotalAmount INTO testdb
BY CATEGORY Manufacturer, Nation
FROM OrderSummary
WHERE (Region=‘‘ASIA’’) AND
    (Nation.nationlink.population > 10000000)
```

Example 23. The query below selects the total admissions by diagnosis, state and year, restricted to diagnoses with “Cough” as a symptom and years later than 1997.

```
SELECT TotalAdmissions INTO testdb BYCATEGORY Diagnosis, State, Year
FROM Admissions
WHERE (Diagnosis.diag_link.symptoms.name="Cough") AND (Year > 1997)
```

This query is processed as follows. The Federation Coordinator (FC) parses the query and identifies the link and ODB parts of the query. Based on the link name (diag_link), the FC looks up in the metadata which ODB, object class, and attribute the link is to and the type of the link, i.e., equivalence, attribute, or enumerated. For this query, the ODB is the “Epidemiology” DB, the class is “Diagnosis,” the attribute is “code,” and the link type is “equivalence.” The object path to be followed is “.symptoms,” and the final attribute is “name.”. Based on this information, the FC forms the OQL query seen below.

```
SELECT code=@n001 FROM @n000 IN SUMDB:Diagnosis, @n001
IN @n000.code
WHERE @n000.symptoms.name="Cough"
```

The OQL query is then executed against the Demographic ODB, giving as result the single diagnosis code “N12”. Based on the result of the OQL query, the FC now forms the SumQL query seen below, which is executed against the MDDB component of the federation to obtain the final result. We see that the original predicate "Diagnosis.diag_link.symptoms.name="Cough"") has been transformed into the predicate “Diagnosis IN (‘N12’)” by substituting literal values from the Diagnosis dimension.
into the new predicate. The new predicate can then be evaluated entirely in the OLAP component. We refer to this process as inlining of the predicate.

```sql
SELECT TotalAdmissions INTO testdb BY_CATEGORY Diagnosis, State, Year
FROM Admissions
WHERE (Diagnosis IN ('N12') AND Year > 1997)
```

The SumQL query is now translated into the MDX statement seen below and executed against the MDDB managed by MS SQL Server OLAP Services. The reason for using the intermediate SumQL statement is to isolate the implementation of the OLAP data from the FC. As another alternative, we have also implemented a translator into SQL statements against a relational “star schema” design.

```mdx
SELECT [Measures].[TotalAdmissions] ON COLUMNS, INTERSECT
    (CROSSJOIN(CROSSJOIN([Diagnosis],[N12]), [Place],[State],MEMBERS),
     [Time],[Year],MEMBERS),CROSSJOIN([Diagnosis],
     [Diagnosis],MEMBERS, [Place],[State],MEMBERS),
     FILTER([Year],MEMBERS,
     [Time].CURRENTMEMBER.NAME > "1997")))
ON ROWS FROM Admissions
```

We have now illustrated the amount of work that users will ultimately have to perform on their own, if they are without the aid of the user interface and the federated translation tools. In particular, we wish to emphasize the usefulness of the OLAP-object database links to generate the combined result. Also, the users are spared the verbosity of MDX, which is hidden from them.
8. Performance evaluation

We now describe the performance experiments that we have carried out in order to validate the practical applicability of our approach.

Fig. 6 shows the federation schema used for the performance experiments, in UML notation. The schema is based on the TPC-R benchmark [55]. We have chosen to use TPC-R data for our experiments for two reasons. First, the TPC-R benchmark and its data are well-known to the database community. Second, we did not have access to databases containing the medical information used as examples previously in the paper that were large enough to make interesting performance studies. The schema has been divided into an OLAP part and an object part. The measured facts in the OLAP schema are the total number of orders and the total cost amount for the orders. The facts are characterized by a Supplier dimension and a Manufacturer dimension. The Supplier dimension has Supplier, Nation, and Region categories that allow the facts to be summarized to the required level of detail. The Manufacturer dimension has the categories Part and Manufacturer. The object part of the schema has Region, Nation, Supplier, and Part classes and relationships between them. Link nationlink connects the Nation category in the OLAP part to the Nation class in the object part as indicated by the dotted lines. Links supplierlink and partlink connect the Supplier category and class, and the Part category and class, respectively. We used a TPC database with “scale factor” 1, which means that the database contains around 6,000,000 line items, 200,000 parts, 10,000 suppliers, 25 nations and five regions. The TPC-R based case has only equivalence links, thus the Link Data component is not used in the query evaluation.

The idea of the experiments is to compare the performance of the OLAP++ system with an alternative implementation where all data is physically integrated in the same data warehouse. Dedicated multidimensional OLAP (MOLAP) systems such as MS OLAP Services cannot handle complex data such as irregular hierarchies. Specifically, MOLAP systems cannot handle non-strict hierarchies and concept inheritance [33], in contrast to our federated approach. This is true even for the latest versions of commercial OLAP software such as MS Analysis Services. This means that a comparison of our federated approach to an integrated DW based on a MOLAP system would not be fair, as the MOLAP system cannot handle data with the same complexity as our federated approach. However, existing RDBMS technology can handle data with this complexity and is also the natural choice for integrated data warehouses for most enterprises. The most logical RDBMS alternative to a given MOLAP system is the RDBMS offering from the same vendor. Thus, we have chosen to compare OLAP++ to an integrated relational data warehouse using MS SQL Server 7.0 as the RDBMS.

As a side note, experiments comparing the performance of a federation of OLAP and XML data with the same data physically integrated in a physical (MOLAP) MDDB actually showed that the OLAP-XML federation approach (which has a query processing strategy very similar to the one in our OLAP-object federation) still only entailed a modest overhead (less than 50% for most queries, and in many cases almost no overhead) [43,61–63]. This is true both for queries with XML-extended selection predicates and queries that use external XML data for grouping.
The hardware/software configuration is as follows. The OLAP component of OLAP++ uses MS OLAP Services running under Windows 2000 on a 400 MHz Pentium machine with 164 MB RAM and one disk. The OPM tools runs on top of an Oracle DBMS under Solaris on a Sun Sparc machine with 1 disk, i.e., quite a modest hardware setup. The federation coordinator also runs on the Sun machine. The relational system uses MS SQL Server 7.0 running on the Pentium machine mentioned above. Note that OLAP++ runs on two machines, introducing additional communication overhead, whereas the integrated DW runs on a single machine. Thus, OLAP++ would have been faster on a single machine, but we considered the multiple-machine scenario to be more realistic.

We compare OLAP++ to three different configurations of the relational data warehouse: “relational with no indexing” (RNI), “relational with partial indexing” (RPI), and finally “relational with optimal indexing” (ROI). For the RPI configuration, single-column B-tree indices have been built on all columns. For the ROI configuration, a set of optimal indices has been built for each query using the SQL Server Indexing Wizard. Thus, RPI corresponds to a general indexing strategy that does not favor particular queries, while ROI corresponds to an indexing strategy that fine-tunes the system for a very particular set of queries. As it is generally not possible to do such a complete fine-tuning for a large data warehouse with many users and queries, we believe the RPI configuration to be the closest to a real-world setting. The RNI configuration is included to show the effect of indexing. We note that the TPC schema used in the experiments is very similar to typical relational DW schemas (star or snowflake schemas) as all the keys (primary and foreign) are auto-generated numeric keys and thus optimized for the comparison operations in large joins.

The first comparison concerns the storage space used by the four different alternatives. The storage use is illustrated in Fig. 7.

The exact figures are: OLAP++ (414 MB), RNI (1340 MB), RPI (1700 MB), ROI (2620 MB). The figure for OLAP++ consists of 359 MB for the OLAP component, including 10 pre-computed aggregations, and 55 MB for the object data. We see that RNI takes over three times the space of OLAP++, whereas RPI and ROI used more than four and six times the space of OLAP++, respectively. Thus, OLAP++ is far more efficient with respect to storage use. This is due to the fact that the OLAP data in OLAP++ can be stored using dedicated multidimensional database technology. The storage use of 359 MB for OLAP data should be compared to the approximately 1285 MB (1340 minus the 55 used for object data in OLAP++) used in the relational system. This difference is even more remarkable as the 359 MB even includes multidimensional indices and 10 pre-computed aggregates.

Next, we consider the query performance of the four alternatives. As our focus is on OLAP applications, we evaluate a set of TPC-R style aggregate queries that aggregate OrderSummary data to the different levels in the cube, namely Region (Reg), Nation (Nat), and Supplier (Sup) from the Supplier dimension and Part and Manufacturer (Man) from the Part dimension. All queries use at least one link + path expression in the conditions as this is the main functionality offered by OLAP++. Fourteen different queries were run to investigate different types of queries. The main characteristics of the queries, and the query performance for the four different alternatives are seen in Table 4. The actual queries are given in Appendix B.

The “Query” column indicates the query number. The “Dims” column indicates the number of dimensions used in the query, in the BY_CATEGORY and/or WHERE clauses. The “Diff” column indicates the level difference between a level used in the BY_CATEGORY clause and the lowest level from the same dimension used in the WHERE clause (0 means same level, NR means that there are no related levels, i.e., levels from the same dimension, between the BY_CATEGORY and the WHERE clauses). For example, the query “SELECT TotalAmount INTO testdb BY_CATEGORY Region FROM OrderSummary WHERE Nation.nationlink.[Nation].population > 100,000,000” uses one dimension and has a level difference of one. Note that this query requires that the OLAP++ system first selects the nations with population > 100,000,000 from the Object database. Then it provides the nation IDs to the OLAP system as a restriction on the nations to be considered for the aggregate (sum) operation. We would assume that the Dim and Diff columns are a measure of the complexity of the query, the higher their values, the more complex the query. Thus, as the queries are sorted by Dims and Diff, the queries should gradually become more complex from top to bottom in the table. The next two columns show what levels used in the BYCATEGORY and WHERE clauses, respectively. The “Cond. size” column shows the total number of dimension values that match the predicates in the WHERE clause. For conjunctive/disjunctive predicates, the number of dimension values matching each part of
the predicate are added together. The importance of the condition size while be seen shortly. Finally, the last four columns give the query response time (in seconds) for the four alternatives. The query response time is also illustrated graphically in Fig. 8.

We first note that OLAP++ is always much faster (4.6 times faster on the average) than the RNI configuration. Most of the queries ran 15–100 times faster in OLAP++. This is hardly surprising given the lack of indices for RNI. However, it is worth remembering that RNI uses more than three times the space of OLAP++. More interesting, OLAP++ is faster than the RPI configuration for 13 out of 14 queries (4.1 times faster on the average). Again, most of the queries ran 12–100 times faster in OLAP++. For the one query where OLAP++ is slower than RPI, the difference is only around 19%. Thus, we can conclude that OLAP++ is clearly faster than the RPI configuration, using less than a fourth of the storage space. For the ROI configuration, representing the best possible situation for the relational system, results are more equal. ROI is faster than OLAP++ for 9 out of 14 queries, but the average response time for ROI is 44% higher than for OLAP++, meaning that some queries will run slowly for ROI even with the optimized indices. We note that the storage space for ROI is more than six times the space for OLAP++. Thus, we can conclude that OLAP++ performs as well as even the best indexed relational system, using only a small fraction of the storage space.

The main reason for the good performance of OLAP++ is that MS OLAP Services can use optimized MOLAP storage, including pre-aggregated data, while SQL Server 7.0 cannot use pre-aggregated data (this is also true for SQL Server 2000). Also, as the database size is four times smaller for OLAP++, the cache performance will invariably be better than for the relational system. Additionally, MS OLAP Services caches both aggregate results as well as detailed data. One could ask whether this is unfair to the relational system (SQL Server). Indeed, SQL Server could have performed much better if materialized aggregates were available, but materialized aggregates would have take even more space than the ROI configuration. Thus, top performance for the relational system can only be achieved at a very high cost. Summing up, it is safe to say that OLAP++ will almost always perform better than a relational system (SQL Server or other RDBMSes) for the same storage use, due to the optimized MOLAP storage.

As a side remark, our measurements show that most of the response time for OLAP++ is spent in the OLAP component. Apart from this time around 2–3 s is spent in the OPM/Oracle system plus the FC to get the object data and form new “inlined” predicates.
If we look into how the performance varies over the queries, we see that neither the number of dimensions, nor the level differences, i.e., the assumed parameters of query complexity, determine the query performance for OLAP++ or any of the relational configurations. However, we note that queries 4–8 are particularly slow in OLAP++, but not in ROI. For these queries, we see that the condition sizes (query selectivities) are high. To investigate this further, we have plotted the response times as a function of the condition size. The result is seen in Fig. 9.

Fig. 9 shows that the condition size has no visible effect on the query performance for the three relational configurations. However, for OLAP++, the response time is an almost linear function of the condition size. We see that OLAP++ is faster than ROI for small condition sizes, but slower for larger condition sizes. This makes sense when we think about the query processing strategy of OLAP++: conditions are “inlined” into the OLAP predicates which in turn will contain many literal dimension values, i.e., become very long, as the condition size grows. Thus, it seems that the OLAP system is not efficient in processing such queries, i.e., even well designed OLAP systems may not be optimized for queries with large predicates in them.

As a side note, a similar behavior is seen in the above-mentioned OLAP-XML federation system, where queries with high selectivities (large condition sizes) are not performing as well as queries with a lower selectivity [62,63].

In summary, the performance experiments showed that in comparison to relational systems, OLAP++ performed approximately as well as the (unrealistic) optimally tuned relational configuration (ROI), and much better than a more realistic relational configuration (RPI), using much less storage space (from four to six times less). In comparison to OLAP systems, OLAP++ provides the performance level of specialized OLAP systems, while still being able to handle the complex data that OLAP systems cannot.

9. Related work

Federated database systems [52,25,26,14] support the logical integration of autonomous database systems, without requiring data to be physically moved and while allowing the individual autonomous database systems to function as before. Federation is a flexible solution that may leverage existing technology and adapt quickly to changing information requirements. In contrast, physical integration of data, commonly referred to as the physical (as opposed to logical) data warehousing approach [57]. This approach has its own advantages, perhaps most significantly in terms of performance when combining data from different databases, but it is very difficult to keep the warehouse data up to date. Thus, it is often impossible or impractical to use physical data warehousing, especially if the data sources belong to different organizations. The two approaches are complimentary, in that they are appropriate under different circumstances. This paper tries to combine the advantages of both approaches, i.e., providing the flexibility and data “freshness” of the federated approach while offering the aggregation safety and a level of performance that is comparable to that of a physically integrated data warehouse.

The issue of providing very “fresh” data is also considered by the recent trend of “(near) real-time data warehousing” [6,46,53] where the goal is to update the DW as quickly as possible when data in the source systems change. However, always keeping the DW updated is only feasible in some situations. In comparison, our approach is useful in other situations where it is more advantageous not to physically integrate all the data.

There has been a great deal of previous work on data integration, e.g., on integrating relational data [23,28,39], object-oriented data [51], semi-structured data [17], and a combination of relational and semi-structured data [20,32,60]. In this line of work, the arguably most related previous work concerned the system based on the nD-SQL language [19]. This system enables the querying of a federation of solely relational data sources, which are treated symmetrically, using nD-SQL. However, none of these approaches handle the advanced issues related to OLAP systems, e.g., dimensions with (possibly irregular) hierarchies and the problems related to correct aggregation. This renders the formulation of distributed OLAP-style queries cumbersome and error prone in comparison to this paper’s proposal. In contrast, this paper fully supports general
dimension hierarchies and aggregation semantics, enabling safe aggregation and shielding the users from incorrect results, while at the same time providing much easier formulation of distributed OLAP-style queries.

When integrating data from databases based on different data models, the traditional approach has been to map all data into one common data model (often called the “canonical” data model) and federate the (logically) transformed data rather than the original data [14,25,52]. This is also true for the federated data warehousing approach presented in [1] where the BLOOM data model is used as the canonical data model. In this paper, we adopt an alternative approach that combines data from multidimensional databases (MDDBs) and object databases using a federated database approach, where data is handled using the most appropriate data model and database technology: OLAP systems for multidimensional data and ODB systems for complex, general data. No attempt is made at “shoe-horning” the data into one common format, which is unlikely to fit all the data.

Several papers propose extensions of the Unified Modeling language (UML) for the conceptual multidimensional modeling of DWs [2,40] and one paper uses a conceptual multidimensional model to model OLAP security [47]. Common to these papers is that they consider multidimensional modeling at the conceptual level, whereas the present paper considers the logical level (and of course the associated federation mechanisms). This work can thus be considered orthogonal to ours. Indeed, the mentioned models could be used to model a DW at the conceptual level that was then implemented as a federation using our approach.

The work in this paper builds on previous work on multidimensional data models supporting irregular hierarchies [41,49] and work on ensuring the summarizability of OLAP data [34,41,49]. This previous work is here leveraged in a federated scenario where there is much less control over the contents of the data, and thus many more potential problems related to incorrect aggregation, than in a traditional physically integrated DW where such problems will often have been handled during the warehouse construction phase. The primary tool for supporting correct aggregation is the use of aggregation types [33,41,48] for blocking further aggregation on “unsafe” data. The SQL:1999 standard has some support for OLAP operations [16,29], however, the explicit support for dimension hierarchies is still not strong. Unlike SQL:1999, our SumQL language offers built-in support for irregular dimension hierarchies and associated support for summarizability checking.

An orthogonal line of work considers the correct and efficient handling of updates of dimensions [15,27] or complete cubes [5,15,31]. In contrast, we do not need to perform any updates on the underlying cube structure and data, and thus save the (still) large cost of updating the physical structures. This line of work can be considered as orthogonal to ours, and one can even imagine a combined approach where our federation approach is used when external data is first queried, followed by a physical integration of the most commonly used external data using the techniques mentioned above. However, in some situations (source data or data requirements changing too frequently, IT department refusing a change of the DW, etc.), it will still not be feasible to physically integrate the data, leaving a federated approach as the only choice.

A related line of work concerns the logical federation of OLAP and XML data sources [43,61,62], please see [45] for a survey. This work can be considered to be parallel to our contribution, in the sense that they apply to different scenarios: OLAP-XML integration is more applicable for the more loosely-structured data on the web, while the OLAP-object integration of our approach is more applicable for integrating the complex, but highly structured, data found in many enterprise databases, due to the rich and complex semantics that can be captured (and exploited) using the object data model.

The present paper significantly extends two previous conference/demo papers [44,22]. In comparison, the present paper has a much more detailed case study (Section 2.2), gives many more details on the data model, summarizability issues, and the object model and language (Section 3), provides more details on linking (Section 4), provides more details on federation aspects (Section 5), provides a new Section 6 on integrating relational and XML sources, provides a new Section 8 with performance experiments, gives a more thorough treatment of related work, and gives the formal syntax and semantics of the SumQL language (appendices). In total, the present paper is approximately twice as long as the conference papers combined.

10. Conclusion and future work

Motivated by the increasingly widespread use of OLAP technology, we have presented the concepts and techniques underlying a prototype system that logically integrates data in OLAP databases with data from outside object databases, without requiring physical integration of the data.

Multidimensional data is best handled using OLAP technology, while complex detail-level data structures are best handled with object database technology. The enables the handling of the data using the most appropriate data model and technology, while still allowing queries to reference data across the different databases and data models. No attempt is made to map data into one common data model, which would be sub-optimal for some of the data. To our knowledge, this is the first example of a “multi-model” federation that includes a dedicated multidimensional data model. We also believe this study to be the first that considers the impact on core OLAP concepts, e.g., summarizability, when federating with external data. In contrast to earlier works, the approach presented here uses the aggregation semantics of data to guard against meaningless or incorrect queries.

More specifically, as a vehicle for presenting the paper’s contributions, a high-level language for multidimensional databases, SumQL, has been introduced. This has then been extended to support queries that reference data in separate object
The resulting language, SumQL++ embodies the concept of links that connect an MDDB to ODBs in a general and flexible way, in addition to object-oriented concepts. SumQL++ permits selection criteria that reference data in the ODBs using path expressions, facilities for decorating the aggregate results of MDDB queries with external object data, and the ability to group data in the MDDB according to object data. We have focused on the extension of aggregate queries over MDDBs to also include data from ODBs. The formal semantics of SumQL++ is given in terms of a formal multidimensional data model and the ODMG data model and OQL query language. It is possible to use other languages such as SQL, QQL, and MDX in the place of SumQL++ once these are enriched with the necessary SumQL++ constructs that they do not already offer.

Interesting research directions include extending the approach to handle federations with several MDDBs, as well as the federation with XML databases, which offer less structure than object databases and thus may benefit even more from the enforcement of aggregation semantics by the federation. Next, it would be of interest to investigate the dynamic restructuring of the OLAP schema, enabling the use of measures as dimensions and vice versa. Yet another interesting direction would be to consider the optimization of queries over the federation. For example, it may in some situations be advantageous to perform aggregation before selection, to take advantage of OLAP techniques such as pre-aggregation. Finally, it should be investigated how to perform data cleansing in our approach.

Appendix A. Formal definition of SumQL

This section formally defines the syntax and semantics of the SumQL language.

A.1. Syntax of SumQL

We now list the syntax for SumQL. The following notation is used in the syntax below: lower case letters are used for variable names; upper case letters are used for keywords; | denotes ‘or’; [] is used to designate optional expressions. To save space, we have not included definitions of strings, reals, and integers, as their definitions are obvious.

| select_query | ::= SELECT measure_list |
|             | [INTO MDDB] |
|             | BY CATEGORY category_list |
|             | FROM {MDDB|{select_query}} |
|             | [WHERE predicate_clause] |
| measure_list | ::= measure | measure_list measure |
| measure      | ::= string |
| MDDB         | ::= string |
| category_list | ::= category | category_list category |
| category     | ::= string |
| predicate_clause | ::= predicate_factor | |
|             | predicate_clause boolean_op predicate_element |
| predicate_factor | ::= predicate_element | (predicate_clause) |
| boolean_op   | ::= AND | OR |
| predicate_element | ::= category_predicate | NOT |
|             | category_predicate |
| category_predicate | ::= category_exp predicate_op value |
|             | category_exp BETWEEN (value, value) |
|             | category_exp IN value_list |
|             | category_exp MATCH ‘string’ |
| category_exp | ::= category |
| predicate_op  | ::= | = | > | < |
| value         | ::= integer | real | ‘string’ |
| value_list    | ::= value | value_list value |

A.2. Semantics of SumQL

To describe the formal semantics of SumQL, we first specify a formal algebraic query language on the multidimensional data model. The algebraic query language is rather low-level and not for end-users, but is convenient for describing semantics. Next, we specify the semantics of SumQL by translation to the algebraic language. The algebraic language presented here is not meant to be computationally complete. We only include the operators that correspond to standard OLAP functions, such as aggregation and selection, while other operators such as union are left out. This is done purposefully, to make sure that the computational power of the language will not surpass that of any commercial OLAP tool, rendering the results presented here widely applicable to commercial OLAP tools.
A.3. Selection

Given an MDDB $S = (\mathcal{F}, D, M)$ and a predicate $p$ on the dimension types $\mathcal{D} = \{ \mathcal{F}_i \}$, we define the selection $\sigma$ as:

$$\sigma[p](S) = (\mathcal{F}', D', M')$$

where $\mathcal{F}' = \mathcal{F}, D' = D, M' = \{ M'_i | \sigma[p](S) \}$ if $p(e_1, \ldots, e_n)$ then $M'_i(e_1, \ldots, e_n)$ else null. The aggregation types are not changed by the selection operator.

Thus, the schema and the dimensions are retained, while the measures are restricted to the part of the multidimensional space where predicate $p$ holds.

Example 24. If selection is applied to the sample MDDB with the predicate $Year = 1998$, the resulting MDDB has the same schema and dimensions, but the Total Admissions measure is restricted to only return non-null values for the multidimensional combinations where the days $d \leq Time 1998$ and where the original measure returned non-null values for those combinations. All other combinations return the null value.

A.4. Projection

Given an MDDB $S = (\mathcal{F}, D, M)$, where $\mathcal{F} = (\mathcal{F}, \mathcal{D})$, a set of measures $M_1, \ldots, M_n \in M$, and a set of dimension types $\{ \mathcal{F}_1, \ldots, \mathcal{F}_n \}$, such that $\mathcal{F}_i = (\{ T_i \}, \text{identity}, \mathcal{T}_i, \mathcal{T}_i)$ for $i \notin \{j_1, \ldots, j_m\}$, we define the projection $\pi$ as:

$$\pi[\mathcal{F}_1, \ldots, \mathcal{F}_n](S) = (\mathcal{F}', \mathcal{D}', M')$$

where $\mathcal{F}' = (\mathcal{F}', \mathcal{D}'), \mathcal{F} = (\mathcal{F}_1, \ldots, \mathcal{F}_n), \mathcal{D}' = \{ D_i | \text{Type}(D_i) \in \mathcal{D}' \}, M' = \{ M'_i(i \in \{q_1, \ldots, q_p\}), \text{and } M'_i(e_1, \ldots, e_n) = M_i(e'_1, \ldots, e'_n), \text{where } e'_j = \text{if } j \in \{j_1, \ldots, j_m\} \text{ then } e_j \text{ else } T. \text{ The aggregation types are not changed by the projection operator.} $

Thus, we require that the dimensions left out in the projection are "simple," having only the $T$ categories. We then keep only the dimension types specified in the projection and their corresponding dimensions. The measures are modified to take only the remaining dimensions as arguments. Only the measures specified in the projection are kept. Note that we do not have to perform any other modifications (such as aggregation) on the measures, as the requirement on the dimensions left out makes sure that the measures have well-defined results, even when the number of dimensions is reduced.

Example 25. Imagine having a version of the example MDDB, $S'$, where the Reason and Time dimensions have only the $T$ category. This could for instance be the result of aggregating along these dimensions (see the aggregation operator below). The result of the projection $\pi[\text{Diagnosis, Place, Total Admissions}] (S')$ is the MDDB where Reason and Time are removed from the set of dimension types and dimensions, making the MDDB 2-dimensional, and the new "Total Admissions" measure gives the same values for the combination $(d, p)$ as the old measure gave for the combination $(d, p, T, T)$.

A.5. Aggregation

Given an MDDB $S = (\mathcal{F}, D, M)$ and a set of categories $C_1, \ldots, C_n$ such that $C_i \in D, i = 1, \ldots, n$, we define aggregation $\alpha$ as:

$$\alpha[C_1, \ldots, C_n](S) = (\mathcal{F}', D', M')$$

where $\mathcal{F}' = (\mathcal{F}', \mathcal{D}'), \mathcal{F} = (\mathcal{F}_1, i = 1, \ldots, n), \mathcal{F}_i = (\{ T_i \}, \text{identity}, \mathcal{T}_i, \mathcal{T}_i), \mathcal{D}' = \{ D_i | \text{Type}(D_i) \in \mathcal{D}' \}, M' = \{ M'_i(i \in \{q_1, \ldots, q_p\}), \text{and } M'_i(e_1, \ldots, e_n) = M_i(e'_1, \ldots, e'_n), \text{where } e'_j = \text{if } j \in \{j_1, \ldots, j_m\} \text{ then } e_j \text{ else } T. \text{ The set on the right-hand side of the last equation is a multi-set, or bag.}$

If the hierarchies up to the grouping categories are summarizable, the aggregation types for the new dimensions are the same as for the original. If one or more of the hierarchies in the dimensions being aggregated over are not summarizable, then the aggregation types for all dimensions are set to $c$, as no further aggregation should be based on the data.

Example 26. On the example MDDB, $S$, we apply the operation $\alpha[\text{Diagnosis, Hospital, Time Reason, Time Time}](S)$, i.e, we aggregate over all of the Reason and Time dimensions, but not over the Diagnosis and Place dimensions. This gives us the MDDB described in the previous example. To make the new MDDB, for each (diagnosis,hospital) combination $(dh), \text{ we find the group of (diagnosis,hospital,reason,day) combinations (dh,rd)}$ for each (diagnosis,hospital,reason,day) combination $(dh,rd)$ such that $r \leq \text{ Reason Reason and } 0 \leq \text{ Time Time, i.e., all the 4-dimensional combinations that } di \text{ and } h \text{ are part of.}$

For each $(dh,rd)$, we apply the "Total Admissions" measure, $M$, to the combination to get the corresponding measure value. We store the measure values for each $(dh,rd)$ combination in their own multi-set, to which we apply the default aggregation operator, SUM. The measure values for the new "Total Admissions" measure, $M'$ for a combination $(dh)$ is thus $M(dh) = \text{SUM}_{r \leq \text{ Reason Reason and } 0 \leq \text{ Time Time}} (\text{(diagnosis,hospital,reason,day)})$, i.e., the sum over all the $(dh,rd)$ combinations that $(dh)$ is a part of. Note that the set on the right-hand side of the equation is a multi-set, or bag.

A.6. We can now give the formal semantics of a SumQL statement in terms of the algebraic query language. The semantics are as follows. Given an MDDB $S = (\mathcal{F}, D, M)$, categories $C_{j_1}, \ldots, C_{j_m}$ in dimensions $D_{j_1}, \ldots, D_{j_m}$ with dimension types $\mathcal{F}_{j_1}, \ldots, \mathcal{F}_{j_m}$ and measures $M_1, \ldots, M_p$ the result of the SumQL statement:

$$\text{SELECT } M_1, \ldots, M_p \text{ INTO S' BY CATEGORY } C_{j_1}, \ldots, C_{j_m} \text{ FROM S WHERE } p$$

is:

$$S = \pi[\mathcal{F}_{j_1}, \ldots, \mathcal{F}_{j_m}, M_1, \ldots, M_p][\alpha[C_{j_1}, \ldots, C_{j_m}](\sigma[p](S))], \text{ where } C_i \text{ if } i \in \{j_1, \ldots, j_m\} \text{ then } C_i \text{ else } T_i.$$
Appendix B. Experiment queries

The actual queries used for the experiments are seen in Table B.1.

Table B.1
Experiment queries.

<table>
<thead>
<tr>
<th>Query no.</th>
<th>Query</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>SELECT TotalOrder, TotalAmount FROM OrderSummary BY_CATEGORY Nation WHERE Nation.nationlink.population &gt; 100000000</td>
</tr>
<tr>
<td>2</td>
<td>SELECT TotalOrder FROM OrderSummary BY_CATEGORY Nation WHERE Nation.nationlink.population &gt; 100000000 AND Supplier.supplinkacctbal &gt; 9950</td>
</tr>
<tr>
<td>3</td>
<td>SELECT TotalOrder, TotalAmount FROM OrderSummary BY_CATEGORY Region WHERE Nation.nationlink.population &gt; 100000000</td>
</tr>
<tr>
<td>4</td>
<td>SELECT TotalAmount FROM OrderSummary BY_CATEGORY Region WHERE Nation.nationlink.population &gt; 100000000 AND Supplier.supplinkacctbal &gt; 9900</td>
</tr>
<tr>
<td>5</td>
<td>SELECT TotalOrder, TotalAmount FROM OrderSummary BY_CATEGORY Region WHERE Supplier.supplinknationkey &lt; ‘2’ AND Supplier.supplinknationkey &lt; ‘1’</td>
</tr>
<tr>
<td>6</td>
<td>SELECT TotalAmount FROM OrderSummary BY_CATEGORY Region WHERE Supplier.supplinknationkey = ‘1’</td>
</tr>
<tr>
<td>7</td>
<td>SELECT TotalAmount FROM OrderSummary BY_CATEGORY Region WHERE Supplier.supplinknationkey &lt; ‘2’</td>
</tr>
<tr>
<td>8</td>
<td>SELECT TotalOrder FROM OrderSummary BY_CATEGORY Nation, Manufacturer WHERE Nation.nationlink.population &gt; 100000000 AND Supplier.supplinkacctbal &gt; 9500</td>
</tr>
<tr>
<td>9</td>
<td>SELECT TotalOrder FROM OrderSummary BY_CATEGORY Nation, Manufacturer WHERE Nation.nationlink.population &gt; 100000000 AND Supplier.supplinkacctbal &gt; 9900</td>
</tr>
<tr>
<td>10</td>
<td>SELECT TotalOrder, TotalAmount FROM OrderSummary BY_CATEGORY Region WHERE Nation.nationlink.population &gt; 100000000 AND Part.partlink.P_Mfgr &gt; ‘Manufacturer#2’</td>
</tr>
<tr>
<td>11</td>
<td>SELECT TotalOrder FROM OrderSummary BY_CATEGORY Nation, Manufacturer WHERE Supplier.supplinknationkey &lt; ‘2’</td>
</tr>
<tr>
<td>12</td>
<td>SELECT TotalOrder FROM OrderSummary BY_CATEGORY Region, Manufacturer WHERE Nation.nationlink.population &gt; 100000000 AND Supplier.supplinkacctbal &gt; 9500 AND Part.partlink.P_Mfgr &gt; ‘Manufacturer#2’</td>
</tr>
<tr>
<td>13</td>
<td>SELECT TotalOrder FROM OrderSummary BY_CATEGORY Region WHERE Nation.nationlink.population &gt; 500000000 AND Supplier.supplinkacctbal &gt; 9950 AND Part.partlink.P_Mfgr &gt; ‘Manufacturer#2’</td>
</tr>
<tr>
<td>14</td>
<td>SELECT TotalOrder, TotalAmount FROM OrderSummary BY_CATEGORY Region WHERE Part.partlink.P_Mfgr &gt; ‘Manufacturer#2’</td>
</tr>
</tbody>
</table>

References

Torben Bach Pedersen is a full professor of computer science at Aalborg University, Denmark. He received MS degree in computer science from Aarhus University, Denmark, and the PhD degree in computer science from Aalborg University. Before joining Aalborg University, he worked in the software industry for more than 6 years. His research interest includes multi-dimensional databases, OLAP, data warehousing, federated databases, data streams, and location-based services. He has published more than 70 scientific papers on these issues in journals such as The VLDB Journal, Information Systems, and IEEE Computer and in conferences such as VLDB, ICDE, SSDBM, SSTD, IDEAS, ACM-GIS, ECIR, Hypertext, DOLAP, and DaWaK. He is a member of the editorial board of the International Journal on Data Warehousing and Mining and has served on more than 40 program committees including VLDB, ICDE, EDBT, SSDBM, and DaWaK. He is a member of the IEEE, the IEEE Computer Society, and the ACM.

Junmin Gu currently works for the Scientific Data Management Research Group at Lawrence Berkeley National Laboratory (LBNL). She develops software for accessing and manipulating scientific data. In the past few years she has been working on the Storage Resource Management (SRM) project for the Grid. This project provides technology needed to manage large distributed data volumes.

Dr. Arie Shoshani is the head of the Scientific Data Management Research Group at Lawrence Berkeley National Laboratory (LBNL). The group’s research activities focus on the development of algorithms and software for the organization, access and manipulation of scientific data. The group is involved in practical projects (such as Climate modeling, combustion modeling, High Energy Physics, and others), and has been applying their research results by providing prototype software. Dr. Shoshani’s own technical work is in query languages, modeling of statistical data, temporal data, multi-dimensional data, and data compression. More recently, he is involved with Grid projects, and initiated the concept of Storage Resource Management (SRM) for the Grid. He is co-developer of the FastBit indexing method for very large high-dimensional data, which uses a patented bitmap compression method, and received the 2008 R&D100 award. Currently, Dr. Shoshani is the director of a Scientific Data Management (SDM) Integrated Software Infrastructure Center (budget $3.3M/year) selected by the SciDAC program at DOE. In this capacity, he is coordinating the work of collaborators from five DOE laboratories and five universities. Dr. Shoshani published over 80 papers in refereed journals and conferences, served as a vice president of the VLDB foundation, and organized many database conferences.

Christian S. Jensen, Ph.D., Dr. Techn., is a Professor of Computer Science at Aalborg University, Denmark, where he directs Center for Data-Intensive Systems. His research concerns data management and spans issues of semantics, modeling, and performance.

He is an IEEE Fellow and a member of the Danish Academy of Technical Sciences, the EDBT Endowment, and the VLDB Endowment’s Board of Trustees. He received Ib Henriksen’s Research Award for his research in mainly temporal data management and Telenor’s Nordic Research Award for his research in mobile services and data management.


He serves on the boards of directors and advisors for a small number of technology companies. He serves regularly as a consultant and delivers lectures to industrial and academic audiences.