Extending OLAP Querying To External Object Databases

Authors: Torben Bach Pedersen
Arie Shoshani
Junmin Gu
Christian S. Jensen

Technical Report 00-5002
Department of Computer Science
Aalborg University
Created on October 17, 2000
Extending OLAP Querying To External Object Databases

Torben Bach Pedersen† Arie Shoshani‡ JunMin Gu‡ Christian S. Jensen†

† Department of Computer Science, Aalborg University
Fredrik Bajers Vej 7E, DK–9220 Aalborg Ø, Denmark
{tbp,csj}@cs.auc.dk

‡ Scientific Data Management Group, Lawrence Berkeley National Laboratory
1 Cyclotron Road, MS 50B-3238, Berkeley, CA 94720, USA
{shoshani,jgu}@lbl.gov

Abstract

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. Additionally, physical data integration 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. A prototype implementation of the system is reported.

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 [29, 21], which means that the system knows which aggregate functions to apply when retrieving different higher-level summaries.

Almost all OLAP systems are based on a dimensional view of data, in which measured values, termed facts, 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. 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 [31, 15, 16, 9] 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 [35]. 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 [31, 15, 9]. In this paper, we adopt an alternative approach that combines data from summary databases (SDBs) and object databases using a federated database approach, where data is handled using the most appropriate data model and database technology: SDB systems for summary data and ODB systems for complex, general data. No attempt is made at “shoehorning” the data into one common format, which is unlikely to fit all the data.

Focus is on enabling OLAP-style queries over existing SDBs to also include data from existing, external ODBs, without jeopardizing the benefits of OLAP queries and without having to integrate the data physically. 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 SDBs, termed SumQL, is defined. The concept of a link, which enables the connection of SDBs 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 SDBs in selection criteria. Queries over SDBs may return ODB data along with the aggregate results, i.e., the result of an OLAP query may be decorated with object data. Finally, SDB 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 [4] 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 [23], OQL [4], and MDX [24] may take the place of SumQL++ once enriched with the constructs in SumQL++ that they do not already offer. Additionally, the approach can easily be extended to allow queries over external relational databases that allow path expressions in queries, e.g., as proposed in the SQL:1999 standard [18].

A prototype has been built [13] that supports the execution of SumQL++ queries over a federation of autonomous SDBs and ODBs.

The arguably most related previous work concerned the system based on the nD−SQL language [11]. This system enables the querying of a federation of solely relational data sources, which are treated symmetrically, using nD−SQL. In contrast, we extend OLAP-style queries on an identified SDB to object databases with related data. Further, nD−SQL supports neither dimension hierarchies nor the aggregation semantics that enable safe aggregation and shield the users from incorrect results. Other existing middleware offerings such as DataJoiner [17], Cohera [8], and Oracle Gateways [25] exhibit the same limitations, which renders the formulation of distributed OLAP queries cumbersome and error prone in comparison to this paper’s proposal.

More specifically, we believe this paper to be the first to consider the integrated querying of data from independent summary and object databases without prior physical integration, with the objective of giving OLAP users enhanced, aggregation-safe query capabilities. Surveys of OLAP data models and languages [26, 33, 34] indicate

1Although the paper’s contributions are applicable to almost all current OLAP systems, we use the term SDB instead of OLAP DB to emphasize the focus on aggregate queries over summary data.
that this issue has not been addressed previously. To our knowledge, the paper is also the first to demonstrate a “multi-paradigm” (or “multi-model”) federation [1, 14, 15], where one of the data models is a dedicated summary 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 considers the arguments for why federating summary and object databases is a good idea. Section 3 introduces the foundations for the SDBs and ODBs. It describes a prototypical summary data model and its high-level, user-oriented summary 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 SDBs 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 SDBs and ODBs. Section 6 describes the prototype implementation of a system that implements the concepts and techniques presented. 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 summary 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 SDBs 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 summary and object databases.

In many situations, SDBs 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 queries. For example, summary 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.

Federating SDBs and ODBs enables a simple and special-purpose SDB system. An SDB 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 SDB unnecessarily impedes casual use of the SDB system. Indeed, most OLAP systems that implement SDBs do not have the necessary facilities, e.g., category inheritance [20], to support this extra information. The federated approach allows the SDB to remain simple, while still allowing access to relevant external data. When SDB data resides in a special-purpose SDB system, we cannot use existing database middleware to access it, leading to a need for technology that enables federations of SDBs and ODBs.

It is possible to obtain better performance when performing summary 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.

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

However, we are not suggesting that already integrated databases should be split up for performance reasons.
An SDB system may support the formulation of summary queries that return correct, or meaningful, query results. When building an SDB, the data may be shaped in order to satisfy summarizability conditions [21]. Briefly, a summary 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 SDB system with information that enables the system to ensure correctness. For example, we may specify that inventory levels should not be added across time [21] or that patient counts for diseases should not be added. In a general-purpose DBMS, no mechanisms for ensuring correct summary results are available.

The federated approach offers additional flexibility when query requirements change. SDBs may be huge, and therefore rebuilding them may be time consuming. Updates to an SDB, e.g., adding new types of information, may require a total or partial rebuild of the database. Because of the rebuild time, a rebuild of the SDB 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. 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 [8].

The above reasoning suggests that in many cases, it is advantageous to logically federate existing OLAP and object databases instead of performing physical integration.

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 Figure 1.

Figure 1: UML Schema of Case Study
The databases are modeled using the Unified Modeling Language (UML) [30]. 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) [36]. The classification also determines how the diagnoses are grouped into diagnosis groups. Diagnosis groups consist of at least 2 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 a logical integration of the databases.

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, 2, and 3, respectively.

<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>Area</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>California</td>
<td>100000</td>
</tr>
<tr>
<td>1</td>
<td>Oregon</td>
<td>40000</td>
</tr>
</tbody>
</table>

State Table

<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>Population</th>
<th>StateID</th>
<th>MayorID</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>Berkeley</td>
<td>140000</td>
<td>0</td>
<td>21</td>
</tr>
<tr>
<td>11</td>
<td>Portland</td>
<td>500000</td>
<td>1</td>
<td>22</td>
</tr>
<tr>
<td>12</td>
<td>Oakland</td>
<td>400000</td>
<td>0</td>
<td>20</td>
</tr>
</tbody>
</table>

City Table

<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>Age</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>Mr. X</td>
<td>45</td>
</tr>
<tr>
<td>21</td>
<td>Ms. Y</td>
<td>57</td>
</tr>
<tr>
<td>22</td>
<td>Ms. Z</td>
<td>33</td>
</tr>
</tbody>
</table>

Mayor Table

Table 1: Data for the Demographic Database

3 Federation Data Models and Query Languages

This section defines a prototypical multidimensional data model and query language used for the SDB 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
Table 2: Data for the Admissions Database

<table>
<thead>
<tr>
<th>ID</th>
<th>Code</th>
<th>GroupID</th>
<th>Text</th>
<th>Deaths</th>
<th>Incidences</th>
<th>Lifestyle</th>
</tr>
</thead>
<tbody>
<tr>
<td>60</td>
<td>E1</td>
<td>60</td>
<td>Diabetes</td>
<td>50000</td>
<td>900000</td>
<td>Yes</td>
</tr>
<tr>
<td>61</td>
<td>N1</td>
<td>61</td>
<td>Infections</td>
<td>200000</td>
<td>15000000</td>
<td>Yes</td>
</tr>
<tr>
<td>62</td>
<td></td>
<td></td>
<td>Pneumonia</td>
<td>100000</td>
<td>10000000</td>
<td>No</td>
</tr>
</tbody>
</table>

Table 3: Data for the Epidemiology Database

Example 1 In the case study we will have Admissions as the fact type, and Diagnosis, Place, Reason, and Time as the dimension types.

Example 2 Diagnoses are contained in Diagnosis Groups. Thus, the Diagnosis dimension type has the following order on its category types: $\perp_{\text{Diagnosis}} = \text{Diagnosis} < \text{Diagnosis Group} < \top_{\text{Diagnosis}}$. Thus, $\text{Pred} (\text{Diagnosis}) = \{ \text{Diagnosis Group} \}$. Other examples of category types are Day, Month, and Year. Figure 2, to be discussed in detail in Example 6, illustrates the dimension types of the case study.
A category \( C_j \) of type \( C \) is a set of dimension values \( e \). A dimension \( D \) of type \( T = (\{C_j\}, \preceq_T, \top_T, \bot_T) \) is a two-tuple \( D = (C, \preceq) \), where \( C = \{C_j\} \) is a set of categories \( C_j \) such that \( \text{Type}(C_j) = C_j \) and \( \preceq \) is a partial order on \( \bigcup_j C_j \), 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 \( e \in \bigcup_j C_j \).

We assume a partial order \( \preceq_C \) on the categories in a dimension, as given by the partial order \( \preceq_T \) 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 \( \text{ALL} \) construct of Gray et al. [12]. We assume that the partial order on category types and the function \( \text{Pred} \) work directly on categories, with the order given by the corresponding category types.

**Example 3** The \( \text{Diagnosis} \) dimension has the following categories, named by their type. \( \text{Diagnosis} = \{50, 51, 52\} \), \( \text{Diagnosis Group} = \{60, 61\} \), and \( \top_{\text{Diagnosis}} = \{\top\} \). The values in the sets refer to the \( \text{ID} \) fields in the Diagnosis and Diagnosis Group tables in Table 2. The partial order \( \preceq \) is given by the \( \text{GroupID} \) 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 \( M : C_1 \times \ldots \times C_n \mapsto T \). 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 \( f_M : T \times T \mapsto T \). 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, \( \text{TotalAdmissions} \), which is the total number of admissions by \( \text{Diagnosis} \), \( \text{Place} \), \( \text{Time} \), and \( \text{Reason} \). The default aggregation function is \( \text{SUM} \).

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 \( \text{Place} \) dimension, but not over the \( \text{Time} \) dimension. For additional discussion of these issues, we refer to reference [21].

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 [20, 28], 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 \( \Sigma = \{\text{SUM}, \text{COUNT}, \text{AVG}, \text{MIN}, \text{MAX}\} \), \( \phi = \{\text{COUNT}, \text{AVG}, \text{MIN}, \text{MAX}\} \), and \( c = \{\text{COUNT}\} \). The aggregation types are ordered, \( c \subset \phi \subset \Sigma \). 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 \( a_M : D \mapsto \{\Sigma, \phi, c\} \) 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, \( a_{\text{TotalAdmissions}}(\text{Diagnosis}) = \Sigma \).
An \( n \)-dimensional summary database (SDB) is a 3-tuple \( S = (S, D, M) \), where \( S \) 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 \( \bot D_1, \ldots, \bot D_n \).

**Example 6** The case study has a 4-dimensional summary database with Diagnosis, Place, Reason, and Time as dimensions. There is one measure, the *TotalAdmissions*, as described above. A graphical illustration of the SDB is seen in Figure 2.

![Summary Model for the Admissions Database](image)

Figure 2: Summary Model for the Admissions Database

### 3.2 Summarizability

This section defines *summarizability*, an important property of SDBs 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 \( S = \{S_j, j = 1, \ldots, k\} \), where \( S_j \in 2^T \), and a function \( g : 2^T \rightarrow T \), we say that \( g \) is *summarizable* for \( S \) if \( g(\{g(S_1), \ldots, g(S_k)\}) = g(S_1 \cup \ldots \cup S_k) \). 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 [21]. These properties are defined formally elsewhere [26, 27, 21]. 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 SDBs to include data from external ODBs, see Section 5 for details.
3.3 The Summary Query Language

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

Using OQL, or some variant thereof, for querying SDBs 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 SDBs 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 SDB concepts such as measures, dimensions with hierarchically organized categories, and automatic aggregation, thus supporting naturally the expression of aggregate queries over summary 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.

\[
\text{SumQL query ::= } \text{SELECT measure}^+ \\
\quad \text{INTO summary\_database} \\
\quad \text{BY\_CATEGORY category}^+ \\
\quad \text{FROM summary\_database} \\
\quad \text{[ WHERE predicate\_clause ]}
\]

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 INTO clause follows that specifies the SDB into which the result of the query is stored. Thus, SumQL queries take SDBs as arguments and return an SDB.

The BY\_CATEGORY 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 BY\_CATEGORY and SELECT clauses are ignored.

The FROM clause specifies the SDB from which to aggregate. For simplicity, we only consider queries over one SDB, and no “drill-across” or “union” functionality is provided. However, the data model and query language can easily be extended to handle this (see [26] for an example). The optional WHERE clause specifies predicates that are applied to the SDB 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 [4].

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

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

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 [4], 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 [4].
Example 8  Data definitions for the demographic (left column) and epidemiology (right column) databases from the case study are shown in Figure 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.

```
INTERFACE State
  EXTENT states
  KEY name
  ATTRIBUTE STRING(30) name
  ATTRIBUTE UNSIGNED LONG area
  RELATIONSHIP Set<City> cities
       INVERSE City::in_state

INTERFACE City
  EXTENT cities
  KEY name
  ATTRIBUTE STRING(30) name
  ATTRIBUTE UNSIGNED LONG population
  RELATIONSHIP State in_state
       INVERSE State::cities
  RELATIONSHIP Mayor current_mayor
       INVERSE Mayor::city

INTERFACE Mayor
  EXTENT mayors
  KEY name
  ATTRIBUTE STRING(30) name
  ATTRIBUTE UNSIGNED LONG age
  RELATIONSHIP City city
       INVERSE City::current_mayor
```

```
INTERFACE Symptom
  EXTENT symptoms
  KEY name
  ATTRIBUTE STRING(50) name
  ATTRIBUTE STRING(255) description
  RELATIONSHIP Set<Diagnosis> diagnoses
       INVERSE Diagnosis::symptoms

INTERFACE Diagnosis
  EXTENT diagnoses
  KEY code
  ATTRIBUTE STRING(10) code
  ATTRIBUTE STRING(100) text
  ATTRIBUTE UNSIGNED LONG deaths_per_year
  ATTRIBUTE UNSIGNED LONG incidences_per_year
  ATTRIBUTE STRING(1) lifestyle_disease
  RELATIONSHIP Set<Symptom> symptoms
       INVERSE Symptom::diagnoses

INTERFACE ContagiousDiagnosis:Diagnosis
  EXTENT contagious_diagnoses
  ATTRIBUTE STRING(30) transfer_mode

INTERFACE NonContagiousDiagnosis:Diagnosis
  EXTENT non-contagious_diagnoses
```

Figure 3: Data Definitions for the Demographic and Epidemiology Databases

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.”

```
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 the symptom “Cough” are returned:

```
SELECT S.diagnoses[ContagiousDiagnosis]transfer_mode FROM S IN Symptom
WHERE S.name = “Cough”
```
4 Linking Databases

This section defines the links that are used to connect SDBs 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 SDB 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 SDB to an unrelated ODB, i.e., an ODB other than the base data from which the SDB was extracted. Third, the source data may be sensitive and thus unavailable to the SDB user. So, we propose to explicitly link even summary 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 SDB 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 SDB and an object class that do not represent the same real-world entities.

Example 11 In our case study, we can specify an equivalence link between the Diagnosis category in the Admissions SDB 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 “\( \text{diag}_{\text{link}} \).”

Example 12 An enumerated link from the Hospital category in the SDB 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 \( L = \{ ("Alta Bates","Berkeley"), ("Portland General Hospital","Portland"), ("Portland Kaiser","Portland") \} \). We will use the name “\( \text{city}_{\text{link}} \)” 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 \(|.\| \) 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 \(|L| = \|\pi_O(L)\| > \|\pi_C(L)\|\). 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 \([-1]\) if the cardinality is \([1-1]\) or \([n-1]\). Similarly, the cardinality of a link is \([-n]\) if the cardinality is \([1-n]\) or \([n-n]\).

Example 13 The cardinality of link “\( \text{diag}_{\text{link}} \)” is \([1-1]\) and the cardinality of “\( \text{city}_{\text{link}} \)” is \([n-1]\).

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 \( O_l = \pi_O(L) \), where \( O_l \) 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 “\( \text{diag}_{\text{link}} \)” link is complete, while the “\( \text{city}_{\text{link}} \)” 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 \([-\infty]\) links, which are analogous to non-summarizable hierarchies, require special attention. Interestingly, an attribute link always has a link cardinality that is \([-\infty]\), while an equivalence link always has a \(1 \rightarrow 1\) 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 [7].

5 The Federated Data Model and Query Language

Having described the data models and query languages of the SDB and ODB components to be federated, as well as a minimal mechanism for linking SDBs 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 SDB, a number of ODBs, and links that interrelate information in the different databases. Formally, a federation \(F\) of an SDB \(S\) 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 SDB. Permitting multiple SDBs introduces additional challenges, e.g., the matching of categories and dimensions, which are not covered here. The case of a single SDB is very useful, as typical queries to a federation naturally centers around one SDB: Typical queries concern SDB measures, grouped by SDB categories, and involving selection criteria relating to data from the ODBs; or queries retrieve ODB data along SDB data; and in some cases, it is desirable to actually group SDB data by categorical ODB data.

Rather than requiring that the SDB and ODB data comply with one common data model, the federation adopts a multi-paradigm approach [14, 1], 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 [2].

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 SDB, 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 SDB in itself is a federation.

5.2 The SumQL++ Language

As our objective is to allow more powerful OLAP queries over SDBs 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 SDB, 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 [12] of SumQL results, which enable ODB data to be returned along with the SumQL result. Third, SumQL is extended to enable SDB data to be grouped by data belonging to ODBs, i.e., attributes of object classes, rather than just the built-in SDB categories.

5.2.1 Extended Selection Predicates

The first extension of SumQL is to allow selection predicates that reference ODB data. The 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 SDB 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.

\[
\begin{align*}
\text{category_exp} & ::= \text{category} [ . \text{link} \text{object}\text{path}\text{attribute} ] \\
\text{object_path} & ::= \text{class}\text{connector} | \text{path}\text{list} \\
\text{class_connector} & ::= . | '['\text{class'}']' \\
\text{path_list} & ::= \text{class}\text{connector}\text{path}\text{element} | \text{path}\text{list}\text{path}\text{element} \\
\text{path_element} & ::= \text{reference}\text{attribute}\text{class}\text{connector}
\end{align*}
\]

**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 $E = C.L.OP.a$, 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 $X.k, X.OP.a$ FROM $X$ IN $Y$,” where $Y$ is the class that $L$ links to, $k$ is the attribute that $L$ links to in $Y$, and $OP$ and $a$ as are above. Let $L'$ be the link relation obtained by performing a natural join of $L$ with $R$, i.e., $L' = L \Join R$, where $\Join$ denotes natural join. We say that $L'$ is the link specified by $E$. The cardinality of $E$ is defined as the link cardinality of $L'$. 

14
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-valued) reference-attributes, i.e., going through a $[-1]$ relationship in a link or a reference attribute does not change the running cardinality, but a $[-n]$ relationship causes the total cardinality to be $[-n]$.

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 $[n - n]$ 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 = “Cough” 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 $P_i = E_i \ POP_i \ V_i$. The $E_1, \ldots, E_n$ are category expressions of the form $E_i = C_i.L_i.OP_i.a_i$, $i = 1, \ldots, n$, 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 $POP_i$ 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 “SELECT $X_i,k_i$ FROM $X_i$ IN $Y_i$ WHERE $OP_i.a_i$ POP$_i$,” 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 $P_i' = C_i \ IN \ \{e_1, \ldots, e_k\}$, where $\{e_1, \ldots, e_k\} = \pi_{C_i}(L_i \ CROSS \ R_i)$ ($\times$ 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 SDB 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 SDB for evaluation.

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

It is often desirable to display additional descriptive information along with the result of an SDB query. This is commonly referred to as *decorating* the result of the query [12]. 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.

SumQL query ::= SELECT measure
INTO summary_database
BY CATEGORY category
[ WITH expression ]
FROM summary_database
[ WHERE predicate ]

**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 SumQL++ 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 a 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 \( R_i \) be the result of the OQL query “SELECT \( X_i \cdot k_i \cdot X_i \cdot OP_i \cdot a_i \) FROM \( X_i \) IN \( Y_i \)”, where \( Y_i \) 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 \( L_i \Join R_i \) 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 \( C_i \Join Z_i \).

Thus, the decoration data is returned in the ODB and link parts of the federation and is not integrated into the result SDB. This loose coupling of decoration data and SDB 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 \([-\infty]\), 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 SDB to be grouped by attribute values in ODBs, enabling aggregation over hierarchies outside the SDB. This feature will be used when aggregation requirements change suddenly.
To achieve this, we allow category expressions instead of just categories in the BY CATEGORY clause. The syntax of the extension is given below. The only difference from the previous syntax is that the BY CATEGORY 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.

\[
\text{SumQL query ::= } \text{SELECT measure}^+ \\
\text{INTO summary_database} \\
\text{BY CATEGORY expression}^+ \\
[ \text{WITH expression}^+ ] \\
\text{FROM summary_database} \\
[ \text{WHERE predicate}_\text{clause} ]
\]

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

```
SELECT TotalAdmissions INTO testdb
BY CATEGORY Diagnosis.diag
link.symptoms.name FROM Admissions
```

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

Formally, given a SumQL++ query,

\[ Q = \text{"SELECT } M_1, \ldots, M_k \text{ INTO } db \text{ BY CATEGORY } E_1, \ldots, E_n \text{ FROM } S \text{ WHERE } P, \]

with the category expressions in the BY CATEGORY clause being of the form \( E_i = C_i \cdot L_i \cdot OP_i \cdot a_i \cdot i = 1, \ldots, n \), where \( C_i \) is a category, \( L_i \) is a link, \( OP_i \) is a 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 \( S' = (S', D', M') \) be the SDB obtained from \( S \) as follows. For each \( E_i \), add a new dimension type to \( S \) with the category types \( \top_i \), \( A'_i \), 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 \( \top_i > A'_i > \bot_i \). Thus, \( S' \) is specified.

For each dimension type, new dimensions \( D'_i \) are added to \( D' \). The categories of \( D'_i \) correspond to the category types. The \( \top_i \) 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 \( (e_1, e_2) \in R_i \iff e_1 \in \bot_i \land e_2 \in C_i \land e_1 \leq_i e_2 \), i.e., the relation specified by the partial order between \( \bot_i \) values and \( C_i \) values. Let \( B_i = R_i \times L'_i \) (\( \times \) 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 \( D'_i \), \( \leq'_i \), is specified as follows: \( e_1 \leq'_i e_2 \iff e_2 = \top \lor e_1 = e_2 \lor (e_1, e_2) \in B_i \). 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.

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

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

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 WITH clause described in Section 5.2.2, resulting in a statement without a 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 Implementation Overview

This section briefly describes the prototype implementation of the federated system capable of answering SumQL++ queries. The overall architecture of the federated system is seen in Figure 4. The parts of the system handling object and link data are based on the commercially available OPM tools [22, 10] that implement the Object Data Management Group’s (ODMG) object data model [4] and the Object Query Language (OQL) [4] on top of a relational DBMS, in this case the ORACLE8 RDBMS. In-depth descriptions of the OPM toolset exist in the literature [5, 6]. The OLAP part of the system is based on Microsoft’s SQL Server OLAP Services using the Multi-Dimensional eXpressions (MDX) [24] query language. The graphical user interface (GUI) is implemented as Java classes running in a standard Web browser for optimal flexibility. A description of the user interface may be found elsewhere [13].

The system has six major components: the GUI, the ODB systems, the link DB system, the SDB system, the federation coordinator, and the metadata database. The ODB, link DB, and SDB components are treated as independent units by the federation system; only their published interfaces are used, and no assumptions about their internal workings are made. 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. Example 23 below explains how a particular query is processed.

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 BY CATEGORY 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 SDB component of the federation to obtain the final result. The reason for using the intermediate SumQL statements 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.

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 SDB managed by MS SQL Server OLAP Services.

SELECT [Measures].[TotalAdmissions] ON COLUMNS, INTERSECT (CROSSJOIN (CROSSJOIN([Diagnosis],[N12], [Place],[State].MEMBERS), [Time].[Year].MEMBERS), CROSSJOIN(CROSSJOIN([Diagnosis],[Diagnosis].MEMBERS, [Place],[State].MEMBERS), FILTER([Year].MEMBERS, [Time].CURRENTMEMBER.NAME > "1997"))) ON ROWS FROM Admissions
This example was intended to illustrate the amount of work that a user will have to go through 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).

7 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.

Summary 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 summary 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 summary databases, SumQL, has been introduced. This has then been extended to support queries that reference data in separate object databases. The resulting language, SumQL++ embodies the concept of links that connect an SDB 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 SDB queries with external object data, and the ability to group data in the SDB according to object data. We have focused on the extension of aggregate queries over SDBs 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, OQL, 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 SDBs, 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.

References


21
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.

\[
\text{select\_query ::= SELECT measure\_list INTO summary\_database BY\_CATEGORY category\_list FROM summary\_database [ WHERE predicate\_clause ]}
\]

\[
\text{measure\_list ::= measure | measure\_list measure}
\]

\[
\text{summary\_database ::= string}
\]

\[
\text{category\_list ::= category | category\_list category}
\]

\[
\text{category ::= string}
\]

\[
\text{predicate\_clause ::= predicate\_factor | predicate\_clause boolean\_op predicate\_element}
\]

\[
\text{predicate\_factor ::= predicate\_element | ( predicate\_clause )}
\]

\[
\text{boolean\_op ::= AND | OR}
\]

\[
\text{predicate\_element ::= category\_predicate | NOT category\_predicate}
\]

\[
\text{category\_predicate ::= category\_exp predicate\_op value | category\_exp BETWEEN (value, value) | category\_exp IN value\_list | category\_exp MATCH \ 'string' }
\]

\[
\text{category\_exp ::= category}
\]

\[
\text{predicate\_op ::= = | != | > | >= | < | <=}
\]

\[
\text{value ::= integer | real | \ 'string'}
\]

\[
\text{value\_list ::= value | value\_list value}
\]

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 purposely, 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.

**selection:** Given an SDB \( S = ( S, D, M ) \) and a predicate \( p \) on the dimension types \( D = \{ T_i \} \), we define the selection \( \sigma \) as: \( \sigma[p](S) = ( S', D', M' ) \), where \( S' = S \), \( D' = D \), \( M' = \{ M'_i, i = 1, .., k \} \), \( M'_i(e_1, .., e_n) = \text{if } ( p(e_1, .., e_n) ) \text{ then } M_i(e_1, .., e_n) \text{ else } null \). 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 SDB with the predicate \( \text{Year} \geq 1998 \), the resulting SDB 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 \( d \leq \text{T\_date} \) 1998 and where the original measure returned non-null values for those combinations. All other combinations return the null value.
Example 25

Imagine having a version of the example SDB, \(S'\), where the Reason and Time dimensions have only the \(\tau\) category. This could for instance be the result of aggregating along these dimensions (see the aggregation operator below). The result of the projection \(\pi[\text{Diagnosis}, \text{Place}, \text{Total Admissions}]\) of \(S'\) is the SDB where Reason and Time are removed from the set of dimension types and dimensions, making the SDB 2-dimensional, and the new “Total Admissions” measure gives the same values for the combination \((d_p, p)\) as the old measure gave for the combination \((d, p, \tau, \tau)\).

aggregation: Given an SDB \(S = (S, D, M)\) and a set of categories \(C_1, ..., C_n\) such that \(C_i \in D_i, i = 1, ..., n\), we define aggregation \(\alpha\) as: \(\alpha[C_1, ..., C_n](S) = (S', D', M')\), where \(S' = (F', D'), \ F' = F, \ D' = \{T'_i, i = 1, ..., n\}\), \(T'_i = (C'_i, \tau, \tau)\), \(C'_i = \{C_{ij} \in T_i \mid \text{Type}(C_i) \leq T_i \}\), \(\leq T_i = \{T_i' \leq T_i \mid \text{Type}(C_i) \leq T_i \}\), \(D' = \{D_i', i = 1, ..., n\}\), \(D_i' = \{C_{ij}' \leq \leq T_i' \}\), \(M' = \{M_i'(e_1, ..., e_n) = f_{M_i}(\{M_i(e_1, ..., e_n) \mid e_1 \in D_1 \land \ldots \land e_n \in D_n \land e_1 \leq e_1 \land \ldots \land e_n \leq e_n\})\}\) (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 SDB, \(S\), we apply the operation \(\alpha[\text{Diagnosis}, \text{Hospital}, \text{Total Admissions}]\) of \(S\), i.e., we aggregate over all of the Reason and Time dimensions, but not over the Diagnosis and Place dimensions. This gives us the SDB described in the previous example. To make the new SDB, for each (diagnosis,hospital) combination \((d, h)\), we find the group of (diagnosis,hospital,reason,day) combinations \((d, h, r, d)\) such that \(r \leq \text{Reason} \land r \leq \text{Time}\), and \(d \leq \text{Time}\), i.e., all the 4-dimensional combinations that \(d\) and \(h\) are part of. For each \((d, h, r, d)\), we apply the “Total Admissions” measure, \(M\), to the combination to get the corresponding measure value. We store the measure values for each \((d, h)\) 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 \((d, h)\) is thus \(M'(d, h) = \sum_{r \leq \text{Reason} \land \text{Time} \land r \leq \text{Time}} \sum_{d \leq \text{Time}} \sum_{r \leq \text{Time}} (\{M(\{d, h, r, d\}\})\), i.e., the sum over all the \((d, h, r, d)\) combinations that \((d, h)\) is a part of. Note that the set on the right-hand side of the equation is a multi-set, or bag.

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 SDB \(S = (S, D, M)\), categories \(C_{j1}, ..., C_{jm}\) in dimensions \(D_{j1}, ..., D_{jm}\) with dimension types \(T_{j1}, ..., T_{jm}\), and measures \(M_1, ..., M_p\) the result of the SumQL statement:

\[
\text{SELECT } M_1, ..., M_p \text{ INTO } S' \text{ BY CATEGORY } C_{j1}, ..., C_{jm} \text{ FROM } S \text{ WHERE } p = \]

\[
S' = \pi[T_{j1}, ..., T_{jm}, M_1, ..., M_p](\alpha[C_1, ..., C_m][\sigma[p](S)]), \text{ where } C_i = \text{if } (i \in \{j_1, ..., j_m\}) \text{ then } C_{j_i} \text{ else } T_{D_i}.
\]