Journal of Universal Computer Science, vol. 15, no. 1 (2009), 273-303
submitted: 29/6/08, accepted: 2/11/08, appeared: 1/1/09 © J.UCS

A Formal Framework of Aggregation for the OLAP-OLTP Model

Hans-J. Lenz
(Free University Berlin, Institute of Production, Information Systems and Operations Research, Garystr. 21, 14195 Berlin, Germany
hjlenz@wiwiss.fu-berlin.de)

Bernhard Thalheim
(Department of Computer Science, Olshausenstr. 40, D-24098 Kiel, Germany
thalheim@is.informatik.uni-kiel.de)

Abstract: OLAP applications are widely used in business applications. They are often (implicitly) defined on top of OLTP systems and extensively use aggregation and transformation functions. The main OLAP data structure is a multidimensional table with three kinds of attributes: so-called dimension attributes, implicit attributes given by aggregation functions and fact attributes. Domains of dimension attributes are structured and thus support a variety of aggregations. These aggregations are used to generate new values for the fact attributes. In this paper we systematically develop a theory for OLAP applications. We first define aggregation functions and use these to introduce an OLAP algebra. Based on these foundations we derive properties that guarantee or contradict correctness of OLAP computations. Finally, for pragmatical treatment of OLAP applications the OLTP-OLAP specification frame is introduced.

Key Words: aggregation functions, OLAP, cube operator, OLTP-OLAP specification frame
Category: H.4, D.2.10

1 Introduction

1.1 OLAP versus OLTP Computations

While OLTP systems are defined in a rigorous mathematical way (e.g. [Schewe and Thalheim, 1993]) OLAP technology lacks of a rigorous mathematical framework and an engineering methodology for sound application. OLAP functionality is based on cube operations [Gray et al., 1997] that provide an intuitive way for data analysts to navigate through various levels of summary information in the data warehouse. Data cubes can be generated by application of the basis operations crossing and nesting [McCullagh and Nelder, 1983] (also see [Lenz, 1994]). OLAP approaches have been re-introducing this generation for the cube but in a different and more complex way. In a data cube, attributes are categorized into dimension attributes and measure attributes. A number of pitfalls with respect to usage of OLAP databases [Lehner et al., 1998; Lenz and Shoshani, 1997; Lenz and Thalheim, 2001a] may occur when cube operations are executed. For example, OLAP operations are often not completely defined, the formal treatment of transformations within OLTP databases and OLAP operators is contradictory or consequences of inherent (stochastic) dependency structures between dimensions are unknown, and the innocent user does not know about implied side effects.

SQL and database systems use aggregation operators heavily. These operators are applicable in the case that null values are not used in the database and computation is based on disjointness, completeness, and extraction of values [Lenz and Thalheim, 2001a]. The average function as well as other aggregation functions are not well-behaved or not meeting intentions in the presence of null values for SQL aggregation since summarisation is based on existing values but counting considers all objects despite from null values. Since OLAP computation uses aggregation functions and aggregated values to a large extent, the danger of integrating obtained wrong values into subsequent applications is even higher.

Furthermore, a systematic treatment of OLTP-OLAP-transformations has not yet been developed. The cube operators have been introduced in an ad-hoc manner, e.g., [Gray et al., 1997], without systematic treatment of its potentials and deficiencies. There are, however, a number of corrections [Franconi and Kamble, 2004; Gyssens and Lakshmanan, 1997; Thalheim, 2000; Vassiladis and Skiadopoulos, 2000].

Finally, the cube operations are given either in an intuitive and informal way [Rafanelli, 2003; Rafanelli, 2005] or are entirely neglected during definition of the cube. Efficient processing of operations must be taken into consideration at the moment of schema design and definition of database structures.

1.2 Aggregation Functions

OLAP functionality and analysis is based on aggregation functions. Thus, aggregation functions applied to a data cube need to be well-defined in order to get correct results. Relational database aggregation functions are widely used but not well-applied in SQL. They may lead to unexpected results and paradoxes such as the Simpson and the hierarchy paradoxes [Lenz, 1993; Lenz and Neiling, 1998; Lenz and Shoshani, 1997; Lenz and Thalheim, 2001b]. Aggregation must consider dependencies, e.g., stochastic dependencies, among the dimensions, and must obey the laws of associativity, commutativity and perfect aggregation [Lenz and Thalheim, 2006].

Aggregation can be viewed as a mapping of data to data defined on an associated simple schemata (typically a relationship type) and with a higher-level view of the data. The simpler description by a schema is then used instead of the original data collection. The aggregation can be done at multiple granularities and for different schemata.

A classification of aggregation functions has been introduced in [Gray et al., 1997] and extended in [Lenz and Thalheim, 2001b]. We distinguish between distributive or inductive aggregation functions that can be defined using a structural recursion schema, algebraic aggregation functions that are expressed by finite bounded algebraic expressions, and holistic aggregation functions that are not bound on the size of storage needed to describe the sub-aggregates.

Aggregation functions may have properties [Lenz and Thalheim, 2005] such as being idempotent, min/max-invariant, continuous, homogeneous, additive, or associative and may have the Lipschitz property. The last property becomes crucial if data are noisy or null-valued. The classical simple aggregation operations *min, max, sum, avg, count* obey this property. *sum* is for instance not idempotent, not shift-invariant, and not self-identical. The invalidity of one of these properties limits the applicability of aggregation functions.

The aggregation problem can be formulated as follows: *Given a database $\mathcal{DB}$ and a database schema $\mathcal{S}$. We need conditions for a separation of applicability of SQL aggregation functions and data cube operations. They either can be applied either without restrictions or must be used under strict consideration of restrictions given by the metadata in the repository and under consideration of integrity constraints. In the last case, we are interested in a transformation of these operations and functions that maintains their intention and allows to correctly compute their result.*

This problem is closely related to the pragmatical aggregation problem of value collections that requires the existence of an approximation function which retains the essential features of the collection but is simpler to represent for large or huge collections of values.

1.3 Outline of the Paper

This paper extends the ideas found in [Lehner et al., 1998; Lenz and Shoshani, 1997; Lenz and Thalheim, 2001a] by providing a formal basis for defining OLAP schemata that provide a basis for correct computations. We develop a theory of aggregation functions, introduce OLTP-OLAP transformations, generalize the notion of the OLAP cube and derive properties for correct OLAP computations. We introduce in this paper a novel definition of the data cube by generalizing the approaches proposed in [Gyssens and Lakshmanan, 1997; Thalheim, 2000; Vassiladis and Skiadopoulos, 2000]. Our general definition of the OLAP cube avoids problematic computations and anomalies.

2 OLTP Schemata and Aggregation Operations

2.1 OLTP Schemata

Definition 1. A database attribute type is based on a type system $\mathfrak{T}_{\mathcal{A}}$ that is defined by the type definition $t = b \mid (A_1 : t_1, \ldots, A_n : t_n) \mid \{t\} \mid [t] \mid \ell : t$ where

ℓ is a collection of labels, A_i is an attribute name and b is an arbitrary collection of *base types*, e.g., base types such as $BOOL = \{\mathbf{T}, \mathbf{F}\}$, $\mathbb{1} = \{\mathbf{1}\}$, $TEXT$, PIC, $MPIC$, $CARD$, INT , $REAL$, $DATE$, URL, and $MAIL$. The union of all base types that are numeric (real, cardinal, integer, complex, rational or other number types) is denoted by NUM.

Definition 2. A (relation or) entity database type E is defined by a named relation (or tuple) type expressions on $\mathfrak{T}_{\mathcal{E}}$, i.e., $E \triangleq ((A_1 : t_1, \ldots, A_n : t_n), \Sigma_E)$[1] for a (relation or) entity type name E, attribute names $A_i \in \mathfrak{N}$, attribute types t_i on $\mathfrak{T}_{\mathcal{A}}$ and a set of integrity constraints Σ_E defined on $(A_1 : t_1, \ldots, A_n : t_n)$.
A relationship type R is defined by a name $R \in \mathfrak{N}$, a relationship type expression t_R and a set of integrity constraints Σ_R defined on t_R, i.e., $R \triangleq (t_R, \Sigma_R)$.
A cluster type C is defined by a name $C \in \mathfrak{N}$, a cluster type expression t_C and a set of integrity constraints Σ_C defined on t_C, i.e., $C \triangleq (t_C, \Sigma_C)$.
A relationship type expression t_R is based on a type system $\mathfrak{T}_{\mathcal{R}}$ that is defined by the type definition $t_R = (\ell_1 : t_R \mid \ell_1 : t_E \mid \ell_1 : t_C, \ldots, \ell_m : t_R \mid \ell_m : t_E \mid \ell_m : t_C,\ A_1 : t_1, \ldots, A_n : t_n)$ for relationship types t_R, entity types t_E, cluster types t_C and attribute types t_i.
A cluster type expression t_C based on a type system $\mathfrak{T}_{\mathcal{C}}$ that is defined by the type definition $t_C = \ell_1 : t_R \mid \ell_1 : t_E \mid \ell_1 : t_C \dot\cup \ldots \dot\cup \ell_m : t_R \mid \ell_m : t_E \mid \ell_m : t_C$ for the disjoint union $\dot\cup$ operation, for relationship types t_R, for entity types t_E and for cluster types t_C.

Definition 3. An entity-relationship database schema (also called object-relational database schema) $\mathcal{S}$ consists of a finite set of entity, relationship and cluster types on a type system $\mathfrak{T}_{\mathfrak{E}\mathcal{R}} = \mathfrak{T}_{\mathfrak{E}} \cup \mathfrak{T}_{\mathfrak{R}} \cup \mathfrak{T}_{\mathfrak{C}}$ that extends $\mathfrak{T}_{\mathcal{A}}$ by type names $\mathfrak{N}$ and by integrity constraints $\Sigma_{\mathcal{S}}$ defined on $\mathfrak{T}_{\mathfrak{E}\mathcal{R}}$.

We typically assume that names are unique, i.e., there is only one assignment of a type expression to a name and labels or attributes are unique in type expressions. The ER model language abbreviates the type assignment by the names of the types used if names are unique. The ER schema is typically closed, i.e., each type used in a type expression is defined within the schema. The cluster type can also be defined by other operations beside $\dot\cup$ such as join, selection, or quotient. We restrict cluster type expressions to the disjoint union operation..

The *database schema* $\mathcal{S}$ is used to define (well-structured) *objects* , the *database* $\mathcal{DB}$ over a database schema $\mathcal{S}$, the *query algebra*, and general operations through structural recursion [Buneman et al., 1994].

Definition 4. The set $DOM(t)$ of all objects defined on type t is called carrier set. A (entity, relationship or cluster) class N^C of a type $N \triangleq (t_N, \Sigma_N)$ consists

[1] We use $\triangleq$ for the explicit assignment of a name to a structure expression or to a type.

of a finite collection of objects on $DOM(t_N)$ that obeys Σ_N. The database $\mathcal{DB}$ over $\mathcal{S}$ consists of a set of collections N_i^C for each type N_i that satisfies $\Sigma_{\mathcal{S}}$.

Collections are often to be considered bags (or multisets) of objects of the given type. We also may consider other collection types such as sets and lists. Typically, sets are used for collections. Aggregation functions are defined on bags. We typically associate a type with one kind of collections, e.g. the type K^T for finite sets of objects defined on T. Each type has thus its specific (finite) collection type K^T. The set of all collections on T forms an algebra with the generalisations of the set operations such as generalized union $\cup_T$, generalized intersection $\cap_T$, and generalized empty elements $\emptyset_T$ on T.

Operations and queries are definable through structural recursion.

Definition 5. [Thalheim, 2000] Given types T, T', the collection types K^T on T and operations $\cup_T$, $\cap_T$, $\emptyset_T$ on K^T. Given further an object h_0 on T' and two functions defined on the types $h_1 \quad : \quad K^T \;\rightarrow T'$ and $h_2 \quad : \quad T' \times T' \;\rightarrow\; T'$
.
We define the structural recursion $srec_{h_0,h_1,h_2}$ on T and T' as follows:
$srec_{h_0,h_1,h_2}(\emptyset_T) \;:=\; h_0$
$srec_{h_0,h_1,h_2}(\{\!|s|\!\}) \;:=\; h_1(s)$ for singleton collections $\{\!|s|\!\}$ and objects s on T
,
$srec_{h_0,h_1,h_2}(T^C \cup_T \{\!|s|\!\}) := h_2(srec_{h_0,h_1,h_2}(T^C), h_1(s))$ iff $T^C \cup_T \{\!|s|\!\} \neq T^C$.

Typical examples of functions defined by structural recursion are `sum` (defined through $srec_{undef,id,+}$), *max* (i.e., $srec_{undef,id,max}$), *min* (i.e., $srec_{undef,id,min}$), and *count* (i.e., $srec_{0,1,+}$) (also called size or cardinality) for the identical mapping *id* of values of components of objects defined on T and a subcomponent type T' of T or the type $\mathcal{N}_0$ of natural numbers with 0 for the size function. Structural recursion is a specific kind of general recursion schemes [Malzew, 1965] (or *recursion* in brief).

This definition uses insertion of objects into a collection. We may also define structural recursion using disjoint union of collections on T. If the collection is a set then the condition for $srec_{h_0,h_1,h_2}(T^C \cup_T \{\!|s|\!\})$ is equivalent to $T^C \cap_T \{\!|s|\!\} = \emptyset_T$.

Each query q has a result type *type*(q) and is defined on an input type *inp_type*(q). A query can be defined by structural recursion on $\mathcal{S} = inp\_type(q)$ and *type*(q). A *view* V on $\mathcal{S}$ is given by defining query q_V that defines the view schema $\mathcal{S}_V = type(q)$ on $\mathcal{S}$.

We notice that view schemata may consist of more than one type. Relational database schemata typically use singleton view schemata.

2.2 Database Aggregation Functions

Database aggregation functions map a collection of objects defined on a type to numeric values. The typical collection considered is the bag. An aggregation function is defined as an infinite family $\mathcal{F} = (f_k : Bag_k \rightarrow NUM | k \in \mathbb{N}_0)$ with functions

$$f_k \ : \ Bag_k \rightarrow NUM$$

that map a bag Bag_k on a type T with k elements to a numerical range. The type T may be a type of the database schema or simply a numerical type. Database types have a number of ordering schemes.

Equivalently, instead of functions defined on bags with k elements, we take symmetric k-ary functions $f_k : T^k \rightarrow NUM$. A function f_k is called *symmetric* if $f_k(x_1, \ldots, x_k) = f_k(x_{\rho(1)}, \ldots, x_{\rho(k)})$ holds for any permutation ρ on $\{1, \ldots, k\}$.

Database research has been defining aggregation functions for bags of any size. In this case the bag algebra must be considered as well for function definition. Our definition defines an aggregation function as an infinite family of bag functions. The shortcut $f \stackrel{\circ}{=} f_{\mathcal{F}}$ can be used for convenience.

Definition 6. A family of symmetric functions $\mathcal{F} = (f_k : Bag_k \rightarrow NUM | k \in \mathbb{N}_0)$ for bags of size k on T is called database aggregation function f on T if each f_k is defined through recursion on the basis of $f_1, ..., f_{k-1}$ for $k \geq 2$ and if they are monotone according to one order of T and to the order of NUM.

$[0, 1]$-interval aggregation functions are surveyed in [Calvo et al., 2002]. Database aggregation functions differ from those by symmetry, constructivity and are not limited to the $[0, 1]$-interval. Constructivity by recursion on the basis of $f_1, ..., f_{k-1}$ within the family is defined by the existence of a recursion function $g_{\mathcal{F}}$ such that $f_k = g_{\mathcal{F}}(f_1, ..., f_{k-1})$. The recursion function can be defined on a subset $f_{i_1}, ...f_{i_m}$ of $f_1, ..., f_{k-1}$. The recursion function is called compositional [Malzew, 1965] if for any r a function $g_r : NUM^r \rightarrow NUM$ exists such that $f_q = g(f_{k_1}, ..., f_{k_r})$ for $q, k_1, ..., k_r$ with $q = k_1 + ... + k_r$.

Typically, the domain $DOM(T)$ is a partially ordered set with minimal and maximal elements and has an operation $+_T$ (sum, union or concatenation of objects) and $\cdot_T$ (product of objects).
Aggregation functions from $\mathcal{F}$ may have the following properties:

Idempotent: $f_k(x, ..., x) = f_1(x)$ for all $x \in DOM(T)$ for all k-bags consisting only of x,

Min/Max-invariant: $f_k(min_T, ..., min_T) = f_1(min_T)$ and $f_k(max_T, ..., max_T) = f_1(max_T)$ for the minimal and maximal elements in $DOM(T)$ and for all k-bags consisting only of min_T,

Self-identical: $f_k(x_1, ..., x_k) = f_{k+1}(x_1, ..., x_k, f_k(x_1, ..., x_k))$ for any $x_1, ..., x_k$, Shift-invariant: $f_k(x_1 +_T b, ..., x_k +_T b) = f_k(x_1, ..., x_k) + f_1(b)$ for all k-bags $x_1, ..., x_k$ and $b \in DOM(T)$,

Homogeneous (of degree 1): $f_k(b \cdot_T x_1, ..., b \cdot_T x_k) = f_1(b) \cdot f_k(x_1, ..., x_k)$ for all k-bags $x_1, ..., x_k$ and $b \in DOM(T)$,

Additive: $f_k(x_1 +_T y_1, ..., x_k +_T y_k) = f_k(x_1, ..., x_k) + f_k(y_1, ..., y_k)$ for all k-bags $x_1, ..., x_k$ and $y_1, ..., y_k$,

Associative: The recursion function $g_{\mathcal{F}}$ is compositional.

Min/Max-invariance is defined in a general setting. One should however be aware whether the data type T is ordinal or metric. Some systems allow to compute minimal values that do not make sense, e.g. min\{male, female\} = female. Associativity is called summarizability in [Lenz and Shoshani, 1997]. The recursion function $g_{\mathcal{F}}$ can be based on a separation of input variables, on a value extraction function $h_k : T^k \to NUM^k$ and separated computation, e.g., $g_{\mathcal{F}}(g_{\mathcal{F}}(h_{k_1}(\underline{x}_1)), ..., g_{\mathcal{F}}(h_{k_r}(\underline{x}_r))) = g_{\mathcal{F}}(h_{k_1+...+k_r}(\underline{x}_1, ..., \underline{x}_r))$ for tuples $\underline{x}_1, ..., \underline{x}_r$ with $k_1, ..., k_r$ elements, correspondingly. If the aggregation is idempotent then it is min/max-invariant.

We distinguish between distributive, algebraic and holistic aggregation functions:

Distributive or inductive functions are defined by structural recursion. A typical example are the simple statistical functions of SQL-92: `count` (absolute frequency, size), `sum` (total), `min, max`.

Distributive functions are associative. They preserve partitions of bags or sets, i.e. given a bag X with k elements and a partition $X = X_1 \cup X_2 \cup ... \cup X_n$ of X into pairwise disjoint subbags with k_i elements correspondingly. Then for a distributive function f there exist a function g such that $f_k(X) = g(f_{k_1}(X_1), ..., f_{k_n}(X_n))$. Functions such as *count, sum, min, max* are distributive.

Algebraic functions can be expressed by finite algebraic expressions defined over distributive functions. Typical examples of algebraic functions in database languages are `average` and statistical *covariance*. The average function for instance can be defined on the basis of an expression on *count* and *sum*.

Holistic functions are more complex aggregation functions used in order or temporal statistics which relate data subsets to other subsets or supersets, e.g. growth rates, changes in an aggregate value over time or any dimension set (banded reports, control break reports, OLAP dimensions). For holistic functions there is no bound on the size of the storage needed to describe a sub-aggregate. Typical examples are *mostFrequent, rank* and *median* .

Usually, their implementation and expression in database languages requires tricky programming. Holistic functions are computable over temporal views. We will not discuss these functions in detail within this paper.

Theorem 7. *Aggregation functions have the following properties:*
`max`, `min` *are idempotent, min-/max-invariant, self-identical, additive, homogeneous, and associative.*
`sum` *is homogeneous, additive, associative, is not idempotent, not self-identical, and not shift-invariant,*
`avg` *is idempotent, shift-invariant, homogeneous, additive, is not self-identical, and not associative,*
`count` *is associative, not idempotent, not self-identical, not shift-invariant but cardinality preserving, not homogeneous, and not additive.*

The proof of the theorem is straightforward and thus omitted. It uses the definition given above. For instance, $srec_{undef,id,max}$ is idempotent, as $max\{id, ..., id\} = id$ by the definition of structural recursion. If null values are used then maximum and minimum are undefined. The sum function can be defined in four different ways in the presence of null values in $DOM(T)$, e.g., by $sum^l := srec_{undef,h^l_{1,1},+}$ with $h^l_{1,1}(x) = Id(x)$ for $x \neq NULL$ and $h^l_{1,1}(NULL) = l$ for $l \in \{0, undef, default, expect\}$, the default value in $DOM(T)$, the statistical expectation in $DOM(T)$ and the undefined value that extends T'. In the case of $h^{undef}_{1,1}$ we may propagate the undefined value through $+$. Similarly, $count^{l'} = srec_{0,h^{l'}_{1,2},+}$ and $h^{l'}_{1,2}(x) = 1$ for $x \neq NULL$ $h^{l'}_{1,2}(NULL) = l'$ for $l' \in \{1, undef, default, expect\}$. The function $\mathtt{count}^1$ is abbreviated by `count`. The average function avg can be defined in sixteen different ways [Lenz and Thalheim, 2001b] depending on the treatment of null values. SQL chooses $avg^{SQL} := \frac{\mathtt{sum}^0}{\mathtt{count}^1}$. It is better to use $avg := \frac{\mathtt{sum}^{undef}}{\mathtt{count}^{undef}}$. An aggregation function is cardinality preserving if it equals to the cardinality of the bag.

Depending on these properties, the behavior of aggregation functions varies. For instance, if the aggregation function is not associative then roll-up (defined below) may falsify the result [Lenz and Shoshani, 1997].

The existence or the non-existence of null values in NUM is not only a design issue. It heavily influences the behavior of aggregation functions. For instance, as noted in [Lenz and Thalheim, 2001a] the `min` and `max` functions will not remain to be idempotent, the average function can be defined in at least nine different ways. SQL is using a less convenient notion for the average function.

3 OLAP Schemata as Derived Schemata

The data cube [Gray et al., 1997] also known as the multidimensional table in OLAP systems is designed to provide an intuitive way for data analysts to navigate through various levels of summary information. In a data cube, attributes are categorized into dimension attributes and measure (or fact) attributes. Measure attributes of those records with the same values are combined (mainly summed up) into an aggregate value. The intuition behind the cube is intriguing. Since it seems so simple to combine values the cube operator is applied whenever a summary is requested. But intuition may fail or may be misled. It is a matter of fact that all OLAP operators have to be carefully re-defined [Meo-Evoli et al., 1992] because they operate on multi-way tables (data cubes) but not on flat tables ("data matrices"). The starting point of our investigation is given by a number of known pitfalls, which correspond to unsound OLAP operations.

The data cube is mainly queried by selection and navigation or equivalently by subcubing. Selection is based on a criterion that is evaluated against data or levels of dimension in order to restrict the set of retrieved data. Possible navigation operations which can be applied to a cube are roll-up (aggregation of data from a lower level to a higher level of granularity within a dimension hierarchy), drill-down (the inverse of roll-up), slice (selection by subset of values of the dimension), and dice (by table projection or equivalently by grouping of data with respect to a subset of dimensions of a cube).

There have been a number of proposals for the data cube definition, e.g., [Agrawal et al., 1997; Gray et al., 1997; Lehner et al., 1998]. But each approach presents its own view of multidimensional requirements, terminology and formalism. The first precise operational definition can be found in [Lenz and Thalheim, 2005]. These functions are analogues of functions of the object-relational or entity-relationship algebra [Gyssens and Lakshmanan, 1997; Thalheim, 2000; Vassiladis and Skiadopoulos, 2000]. We revise these notions and the proposals in [Franconi and Kamble, 2004; Rafanelli, 2003; Rafanelli, 2005] and in [Lenz and Thalheim, 2006] by a novel introduction of a rigorous mathematical cube model.

Any OLAP specification language must satisfy a number of requirements.

- ◦ Analytical or OLAP data need a conceptually generic data type, i.e. a data cube together with well-defined methods.
- ◦ The formalism must be implementation independent, must be based on a separation of schema definition (e.g., the cube itself) and real data, and must support a generic definition of the query algebra.
- ◦ Dimensions must support representation by different structures or schemata in order to allow flexible querying of cubes.
- ◦ The data model supports views or so-called complex structured cell values or measures.
- ◦ The query language must have a formal description.

It is surprising [Lenz and Thalheim, 2006; Molnar, 2007; Molnar and Thalheim, 2007] that none of the proposed OLAP models satisfies at least three of the requirement groups. For instance, the usual prosaic specification of OLAP queries is highly ambiguous. Drill-down and other OLAP operations are often only verbally explained on trivial three-dimensional examples.

3.1 The Mathematical Cube Model

The data cube operator has informally been introduced in [Gray et al., 1997]. Despite its partial realisation in SQL it is surprising that the pitfalls of this definition have not yet been repaired [Lenz and Thalheim, 2006]. The operator is typically introduced in an intuitive or example-based way that is based on an intentional use of a relationship type (called a fact type) with component types (called dimension types) that are typically entity types. This understanding [Lewerenz et al., 1999] has been neglected in research. Classical treatment leads to an informal or vague introduction of main functions such as roll-up, drill-down, pivoting, dice and slice functions.

We define the cube using the partition model [Molnar, 2007; Molnar and Thalheim, 2007]. The mathematical cube model is based on a layered schema definition. We start with a re-definition of types. Types have their carrier sets or domains. Carrier sets have their value partitions. Some of the partitions may be used for grouping of values. For instance, the values in $DateTime$ may be grouped into hours, days, months and quarters. Groups can be extended by a name for each group of values. This multi-layer model is the basis for introducing a new and simple cube model.

Let us consider first an arbitrary domain $DOM(T)$ of a type T. The set $2^{DOM(T)}$ of all subsets of $DOM(T)$ constitutes a Boolean lattice. At the same time, any equivalence relation on $DOM(T)$ defines a set of *equivalence classes* on $DOM(T)$. Let us assume that each of these equivalence classes can potentially be named. We denote by $\mathfrak{P}(DOM(T))$ the set of all pairs consisting of equivalence classes on $DOM(T)$ together with their names. Let us for convenience assume that each equivalence class is uniquely named. In this case we can use a function *name* that assigns to a name n_c its equivalence class c for the pair $(c, n_c) \in \mathfrak{P}(DOM(T))$. In the opposite direction, the function *group* assigns an equivalence class c to a name n_c.

Definition 8. A subset $\hat{\mathfrak{P}}$ of $2^{DOM(T)}$ is called a partition of $DOM(T)$ if the union of the subsets covers $DOM(T)$ and the elements of $\hat{\mathfrak{P}}$ are pairwise disjoint. We may also consider the trivial partitions $\perp = \{\{d\} | d \in DOM(T)\}$ and $\top = \{DOM(T)\}$. A named partition consists of a partition together with the names for its classes.

A named partition is denoted by $\mathfrak{p}$. The names used for a named partition $\mathfrak{p}$ are denoted by $names(\mathfrak{p})$. The partitions of $\mathfrak{p}$ are denoted by $partitions(\mathfrak{p})$. Trivial partitions are not assigned to names. The element relation $\in$ may be generalised to named partitions. We write $c \in \mathfrak{p}$ instead of $c \in partitions(\mathfrak{p})$.

Partitions may be ordered by inclusion.

Definition 9. The canonical order of partitions on $DOM(T)$ relates two named partitions $\mathfrak{p}, \mathfrak{p}'$. We define $\mathfrak{p} \preceq \mathfrak{p}'$ iff for all $(c, n_c) \in \mathfrak{p}$ there exists one element $(c', n_{c'}) \in \mathfrak{p}'$ such that $c \subseteq c'$.

We also may consider non-canonical orderings such as the *majority order* $\preceq_m^{\texttt{choice}}$ that relates two named partitions iff for all (c, n_c) from $\mathfrak{p}$ there exists one and only one element $(c', n_{c'}) \in \mathfrak{p}'$ such that

either $|c \cap c'| > max\{|c \cap c''| \mid (c'', n_{c''}) \in \mathfrak{p}',\ c'' \neq c'\}$

or $(c', n_{c'}) \in \mathfrak{p}'$ is determined by a (deterministic) `choice` operator among

$\{c^+ \in \mathfrak{p}' \mid |c \cap c^+| = max\{|c \cap c''| \mid (c'', n_{c''}) \in \mathfrak{p}'\}\}$.

If the last case does not appear then we omit the choice operator in $\preceq_m^{\texttt{choice}}$. The choice operator must be deterministic. Otherwise, computation of values would lead to different values in any case of computation.

Example 1. Let us consider the domain $DateTime$ as an example. The named partitions $\perp$, $Days$, $Weeks$, $Months$, $Quarters$, $Years$, and $\top$ denote the named partition of highest granularity (i.e., equivalence classes are singleton), the named partitions of $DateTime$ by days, by weeks, by months, by quarters, by years, and the trivial no-granularity named partition, correspondingly. We observe $\perp \preceq \mathfrak{p}$ and $\mathfrak{p} \preceq \top$ for any named partition in this list. We notice too that $Days \preceq Months \preceq Quarters \preceq Years$.
$Weeks \preceq_m Months$ is a different ordering that causes a lot of confusion.

Naming of partitions is also necessary if we want to distinguish different treatment of values. For instance, $GMT{-}Time$, $BusinessWeek$, $WeekInSemester$ form partitions over $DateTime$ which can or cannot be ordered by inclusion in the same way.

[Hurtado and Mendelzon, 2002] introduced dimension constraints. They extend the notion of split constraints introduced by the same authors, of path constraints [Thalheim, 2000], and of disjunctive existence constraints [Thalheim, 1991]. Partitions can be expressed through dimension constraints. Dimension constraints can be completed to form partitions. Typically they do not form a cover. Dimension constraints may be used to restrict partitions to meaningful ones.

Example 2. The selection of partitions may depend on the application. The dimensions $DateTime$ and $Area$ may vary in dependence of the application. For instance, the partitions in Figure 1 display the order of regions in Germany and Hungary [Molnar, 2007] (Land(county)-Kreis(region)-Gemeinde(municipality)-Gemeindeteil(district) and Regio-Megye-Kisterseg-Települes-Kerület).

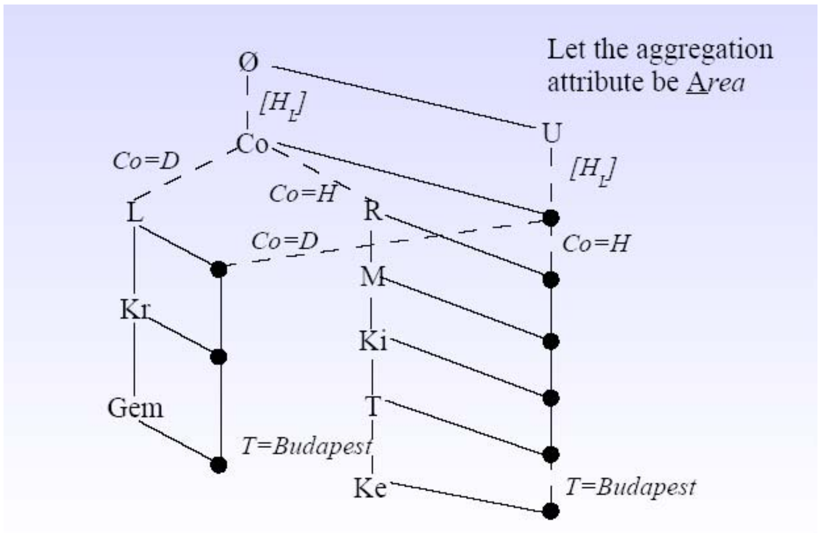

Figure 1: Application dependent partition lattices for *Area*

The cube definition can be based on the ER model [Lewerenz et al., 1999]. The fact table is a relationship collection R^C defined on a relationship type R. We assume that entity types $E_1, ..., E_k$ are component types of R. The relationship type may be extended by additional derived attributes $A_1, ...A_m$ which are associated to aggregation functions.

Definition 10. A cube schema $\mathcal{C}$ consists of a relationship type R that is defined on entity types $E_1, ..., E_k$ called "dimensions" and on pairs (A_i, f_i) of "fact" attributes $A_1, ..., A_m$ and aggregation functions $f_1, ..., f_m$, i.e.,

$$R = (E_1, ..., E_k, (A_1, f_1), ..., (A_m, f_m)).$$

We notice that a cube schema can be defined as a relationship type that is extended by derived attributes which use aggregation function for their derivation. Relationship types may also be used instead of entity types. Most applications base the cube schema on a relationship type that only uses entity types.

Definition 11. Given a cube schema $\mathcal{C}$, a collection (or relationship class) R^C on R, and partitions $\mathfrak{p}_i$ of $DOM(E_i)$ for any component $E_1, ..., E_k$. A cell $R^{(c_1,...,c_k)}$ of R^C is a non-empty set $\sigma_{E_1 \in c_1, ...E_k \in c_k}(R^C)$ for $c_i \in \mathfrak{p}_i$ and for the classical relational selection operation σ_α.

Definition 12 (Cube). Given now partitions $\mathfrak{p}_1, ..., \mathfrak{p}_k$ for all component types of $\mathcal{C}$.
A **cube** $cube^{\mathfrak{p}_1,...,\mathfrak{p}_k}(R^C)$ on R^C and on $\mathfrak{p}_i$, $1 \leq i \leq k$ consists of the set

$$\{\sigma_{E_1 \in c_1, ... E_k \in c_k}(R^C) \neq \emptyset \,|\, c_1 \in \mathfrak{p}_1, ..., c_k \in \mathfrak{p}_k\}$$

of all cells $R^{(c_1,...,c_k)}$ of R^C for the partitions $\mathfrak{p}_i$, $1 \leq i \leq k$.
If $\mathfrak{p}_i = \top$ then we may omit the partition $\mathfrak{p}_i$ from the cube superscripts.

Example 3. Typically, empty cells are removed from a cube. Data within a cube are not uniformly distributed. Therefore, a cube may have cells that are not uniform. Figure 2 illustrates a three-dimensional cube with nonuniform cells.

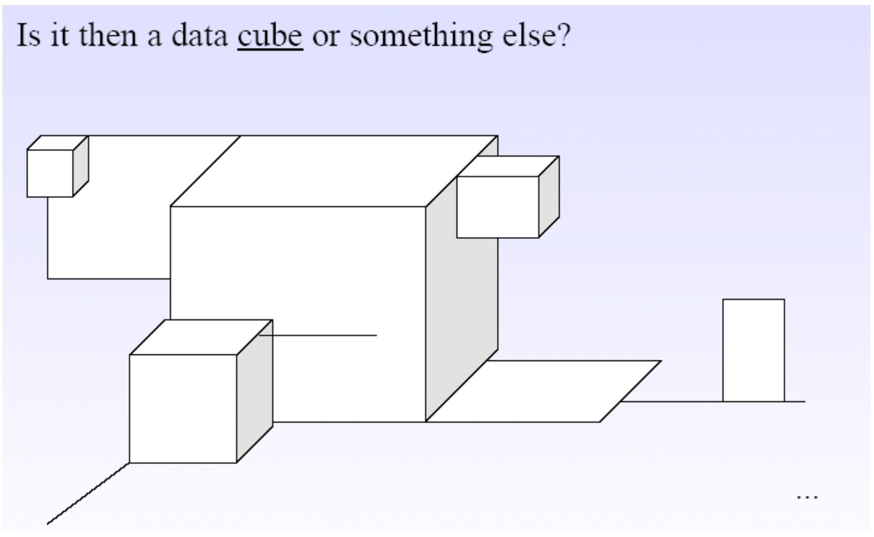

Figure 2: An abstract example of a cube with nonuniform cells that should not be considered

The cube is defined as a class over a relationship type. The extended cube additionally consists of the derived attributes and the aggregation functions. Aggregation functions are assigned to attributes $A_1, ..., A_m$ and applied to cells of R^C. For instance,

$$sum(\pi_{\text{Sales}}(\sigma_{\text{SellingDate} \in Week_x}(R^C)))$$

computes the total turnover in week x based on the projection function π_{Sales}, the selection function $\sigma_{\text{SellingDate} \in Week_x}$ and the relationship class R^C.

Population	Place		Inhabitants	
	Municip	Region	(#Inhabitants, sum)	(Income, avg)
	Kiel	KI	232270	16.300
	Plön	PLÖ	12988	14.270
	Preetz	PLÖ	15800	17.900
	Raisdorf	PLÖ	7583	11.340
	...	...	...	...

$\pi_{\text{Region},\#\text{Inhabitants}}$(Population)	Region	(#Inhabitants, sum)
	KI	232270
	PLÖ	133624
	...	...

$\sigma_{\text{Region=PLÖ}}(\pi_{\text{Region},\#\text{Inhabitants}}$(Population))	Region	(#Inhabitants, sum)
	PLÖ	133624

$\sigma_{\text{Municip=Raisdorf}}$(Population)	Place		Inhabitants	
	Municip	Region	(#Inhabitants, sum)	(Income, avg)
	Raisdorf	PLÖ	7583	11.340

$\pi_{\text{Region},\#\text{Inhabitants}}(\sigma_{\text{Municip=Raisdorf}}$(Population))	Region	(#Inhabitants, sum)
	PLÖ	7583

Definition 13. An **extended cube** consists of cells $R^{(c_1,...,c_k)}$ and values $f_i(R^{(c_1,...,c_k)})$ for each of the attributes $A_1, ..., A_m$.

Our definition of the cube is based on cells and on names. These names are sometimes necessary if we need to distinguish attributes or components of the cube depending on varying granularity.

Example 4. Let us consider a cube that aggregates data about the population of a certain area.

We may apply different aggregation operations to this cube. The following tables show that aggregating first over region and the number of inhabitants and then restricting this new cube to a certain region will not give the same result as in the case if we first restrict the cube to the region and then apply the aggregation. Therefore, the classical equivalences used for optimisation of query processing do not commute with aggregation operations. Therefore, summarisability is only a necessary condition for correctness of cube computations.

The meaning of attributes is different depending on the way of aggregation. For instance, *#Inhabitants* means at the same time (1) the number of inhabitants of certain municipality and (2) the number of inhabitants in a certain region after aggregation. Therefore, we should use our naming or annotation concept if we want to differentiate the meaning.

Our cube model allows to consider the changing meaning of the dimensions, components and attributes depending on the level of the aggregation. The operations we use for the computation of the aggregations must be explicitly considered if we need a more meaningful interpretation. For instance, the attribute *#Inhabitants* can be adorned by the operation that has been applied to the cube: $\#\text{Inhabitants}^{\pi_{\text{Region},\#\text{Inhabitants}}}$ and $\#\text{Inhabitants}^{\sigma_{\text{Municipality}=\text{`Raisdorf'}}}$ carry a meaning that is different from the original #Inhabitants.

A cube and an extended cube can be *materialised*. Then each cell is recorded with its corresponding aggregations for the attributes.

Definition 14. Spreadsheet cubes are defined for sequences $\mathfrak{p}_1 \preceq ... \preceq \mathfrak{p}_n$ of partitions for one or more dimensional components where $\preceq$ denotes the canonical order.

For instance, the partitions $Days$, $Months$, $Quarters$, $Years$ define a spreadsheet cube for components defined over $DateTime$.

Definition 15. A named (extended) cube consists of names for each cell instead of the sets for the cells.

Our notion of the cube also results in natural definitions of main OLAP operations without additional special consideration of summarisability and non-summarisability. If partition orders are used then summarisability comes for free.

3.2 Operations of the OLAP Schema

Our definition of the cube can be now easily used for a precise mathematical definition of the main operations for cubes and extended cubes. In literature cube operations are typically given in an informal way. For instance, [Rafanelli, 2003; Rafanelli, 2005] defines the drill-down operation as a "specific operation (analytical technique) for looking at data going from the most summarized view to the most detailed view". Often cube operations are not defined at all but only illustrated by examples. Users who would like to learn more about cube operations are linked to existing implementations. We may adapt the definitions of [Gyssens and Lakshmanan, 1997; Thalheim, 2000; Vassiladis and Skiadopoulos, 2000].

A cube algebra is given by a cube schema $R = (E_1, ..., E_k, (A_1, f_1), ..., (A_m, f_m))$ and an algebra consisting of at least navigation, selection, projection and split functions.

We introduce now the main query operations. Selection is based on a criterion that is evaluated against data or levels of dimension in order to restrict the set of retrieved data. Roll-up is an aggregation of data from a lower level to a higher level of granularity within a dimensions hierarchy. Drill-down is the inverse of roll-up. Dice can be defined by roll-up to ALL. Slice groups data with respect to a proper subset of dimensions of a cube. The last four operations may be considered to be navigation operations. Thus, the data cube is mainly queried by selection and navigation.

Given a cube with partitions $\mathfrak{p}_i, \mathfrak{p}'_i$ for the dimensional component E_i with $\mathfrak{p}_i \preceq \mathfrak{p}'_i$, the `drill-down` operation transfers a cube defined on $\mathfrak{p}'_i$ to a cube defined on $\mathfrak{p}_i$. `Roll-up` transfers a cube defined on $\mathfrak{p}_i$ to a cube defined on $\mathfrak{p}'_i$. The `slice` operation is nothing else but the object-relational selection operation. The `dice` operation can be defined using $\top$ partitions for all dimensional components that are out of scope.

Definition 16. Given a cube $cube^{\mathfrak{p}_1,...,\mathfrak{p}_k}(R^C)$ on the cube schema

$$R = (E_1, ..., E_k, (A_1, f_1), ..., (A_m, f_m)),$$

a dimension i and (named) partitions $\mathfrak{p}_i \preceq \mathfrak{p}'_i \preceq \top_i$.
Basic drill-down functions map the cube $cube^{\mathfrak{p}_1,...,\mathfrak{p}'_i,...,\mathfrak{p}_k}(R^C)$ to the cube

$$cube^{\mathfrak{p}_1,...,\mathfrak{p}_i,...,\mathfrak{p}_k}(R^C).$$

Basic roll-up functions map the cube $cube^{\mathfrak{p}_1,...,\mathfrak{p}_i,...,\mathfrak{p}_k}(R^C)$ to the cube

$$cube^{\mathfrak{p}_1,...,\mathfrak{p}'_i,...,\mathfrak{p}_k}(R^C).$$

Basic slice functions map the cube $cube^{\mathfrak{p}_1,...,\mathfrak{p}_k}(R^C)$ to the cube

$$\sigma_\alpha(cube^{\mathfrak{p}_1,...,\mathfrak{p}_k}(R^C)).$$

Basic dice functions map the cube $cube^{\mathfrak{p}_1,...,\mathfrak{p}_i,...,\mathfrak{p}_k}(R^C)$ to the cube

$$cube^{\mathfrak{p}_1,...,\top_i,...,\mathfrak{p}_k}(R^C).$$

Roll-up functions are the inverse of drill-down functions. Basic slice functions are similar to selection of tuples within a set. The slice function is nothing else then the object-relational selection operation. Basic dice functions are similar to projection in the first-order query algebra. Basic dice functions are defined as special roll-up functions. We also may omit the dimension i. In this case we loose the information on existence of this dimension.

These operations may be combined using staggering of functions. We, thus, obtain drill-down functions by superposing basic drill-down functions.

Slice function can also be defined through cells. Let $dimension(\alpha)$ be the set of all dimensions that are restricted by the selection criterion α. Let further

$$\sigma^{\partial}_{\alpha}(c_i) \quad = \quad \begin{cases} \emptyset \text{ if } R_i \in dimension(\alpha) \wedge \sigma_\alpha(c_i) \neq c_i \\ c_i \text{ otherwise} \end{cases}$$

a modified selection function that considers only those cells that entirely fulfill the selection condition α.
Close slice functions restrict the cube cells to those that entirely fulfill the selection criterion α, i.e., $\{\sigma_{R_1 \in \sigma_\alpha^\partial(c_1),\ldots R_n \in \sigma_\alpha^\partial(c_n)}(R^C) \neq \emptyset \,|\, c_1 \in \mathfrak{p}_1, ..., c_n \in \mathfrak{p}_n\}$.
Liberal slice functions restrict the cells to those that partially fulfill the selection criterion α, i.e. to cells $\{\sigma_{R_1 \in \sigma_\alpha(c_1),\ldots R_n \in \sigma_\alpha(c_n)}(R^C) \neq \emptyset \,|\, c_1 \in \mathfrak{p}_1, ..., c_n \in \mathfrak{p}_n\}$.
Lazy and eager slice functions apply the selection functions directly to values in the cells.

We observe that the aggregation functions must be additive along the dimension for drill-down and dice functions.

Generalizing the first-order query algebra, [Thalheim, 2000] defines additional OLAP operations such as

join functions for mergers of cubes,

union functions for union of two or more cubes of identical type if union is defined,

rotation or pivoting functions for rearrangement of the order of dimensions, and

rename functions for renaming of dimensions.

The definition of these operations is the same as in the classical case [Thalheim, 2000].

Our new definition of the cube allows one to generalize a large body of knowledge obtained for object-relational databases to cubes. The *integration* of cubes can be defined in a similar form [Molnar, 2007].

We observe that the slice, drill-down, roll-up, union, rotate, and rename functions form a relationally complete query algebra of OLAP operations. The proof is based on the relational completeness of the corresponding operations of the first-order query algebra and straightforward following the proof scheme of [Paredaens et al., 1989].

3.3 Properties of Cube Aggregations

Definition 17. Given a query function q, a database $\mathcal{DB}$, and aggregation functions $f, g \in \mathcal{F}$, the function q is called $\mathcal{F}$-invariant in $\mathcal{DB}$ if $f(q(\mathcal{DB})) = g(\mathcal{DB})$.

Definition 18. An aggregation function f preserves partitions of sets if a function g exists such that $f(X) = g(f(X_1), ..., f(X_n))$ for $X = X_1 \cup X_2 \cup ... \cup X_n$ with $X_i \cap X_j = \emptyset$, $i \neq j$.

Proposition 19. *Distributive aggregation functions preserve partitions of sets.*

$$\begin{tikzcd}
\mathcal{DB} \arrow[r, "q"'] \arrow[d, "g"] & q(\mathcal{DB}) \arrow[dl, "f"] \\
g(\mathcal{DB}) = f(q(\mathcal{DB})) &
\end{tikzcd}$$

Figure 3: Invariance of aggregation functions for query transformations

Distributive aggregation functions are defined through structural recursion. We may equivalently use the union approach to definition of structural recursion instead of the insertion approach.

This proposition implies an important statement for distributive aggregation functions. Its proof is based on the definition of distributive aggregation functions and the disjointness of equivalence classes of partitions.

Theorem 20. *Distributive aggregation functions are invariant for dice, roll-up, and drill-down aggregation functions.*

We may, however, conclude also a number of negative properties: We may directly conclude the following properties:

Proposition 21. *Roll-up functions are neither sum-invariant nor avg-invariant in general.*

This behaviour has already been reported in [Lenz and Thalheim, 2001b; Lenz and Thalheim, 2006]. The counterexamples are based on the hierarchy and Simpson paradoxes [Lenz and Thalheim, 2006]. The average function is already an algebraic function and depends on the weight of partitions under consideration. The sum-non-invariance is observed for partial roll-up functions or for bag-based roll-ups.

Proposition 22. *Rearrangement functions are min-, max-, count-, and sum-invariant. They are avg-invariant along partition orders.*

Rearrangement functions provide a different representation of the same cube, e.g., using another order in the domains. The proof of this proposition is based on the definition of distributive functions. The average function is invariant as long as the partition is not concerned.

A number of invariance properties can be observed for classical relational database functions. For instance, the `Bag2Set` operation (DISTINCT) is `min`-, `max`-invariant, not `sum`-invariant but `sum`-invariant for sets, not `avg`-invariant but `avg`-invariant for sets without NULL's. The `Project` function is invariant for all distributive functions. The `Select, Join` functions are not invariant for most aggregation functions. The `Union, Difference, Intersection` functions are not invariant in most cases. The `Renaming` function is invariant for all aggregation functions.

Selection functions are typically not invariant for most cubes. This property is based on the partiality of selection, i.e. the subset generation.

Definition 23. An aggregation function f is called subset-sensitive if a database $\mathcal{DB}$ and a selection condition α exist such that $f(\mathcal{DB}) \neq f(\sigma_\alpha(\mathcal{DB}))$.

Definition 24. A query q is object-preserving iff for any database $\mathcal{DB}$ objects of $q(\mathcal{DB})$ remain to be distinguishable from each other [Thalheim, 2000]. A query q is value-set-preserving if for any database $\mathcal{DB}$ $allValues(q(\mathcal{DB})) = allValues(\mathcal{DB})$, i.e., none of the values appearing in $\mathcal{DB}$ is lost through $q(\mathcal{DB})$ for the function $allValues$ that collects all values occurring in $\mathcal{DB}$.

Selection, join, intersection, and difference are subset-sensitive operations.

Proposition 25. *All classical aggregation functions are subset-sensitive.*
Object-invariant queries are `min`- *and* `max`-*invariant.*
Value-set-preserving queries are `min`- *and* `max`-*invariant.*

The proposition is obvious.

The exclusion/inclusion principle is one of the important properties for counting. Given classes $C_1, ..., C_m$. The intersection of C_{i_1},..., C_{i_n} is denoted by $I(C_{i_1}, ...C_{i_n})$.

Definition 26. A function f can be computed based on the exclusion/inclusion property if
$f(\mathcal{D}) = \sum_{i=1}^{m} f(C_i) - \sum_{j=1}^{m-1} \sum_{k=j+1}^{n} f(I(C_j, C_k)) + - ...$

$$+(-1)^{r-1} \sum_{1 \leq j_1 < ... < j_r \leq m} f(I(C_{j_1}, ...C_{j_r})) + \ldots + (-1)^{m-1} f(I(C_1, ..., C_m)).$$

Theorem 27. *Distributive aggregation functions can be computed based on the exclusion/inclusion principle.*

The proof of the theorem is based on the definition of distributive functions through structural recursion based on the union approach, is straightforward and can be thus omitted.

3.4 Enriching OLAP Specifications By Metadata Information

We use annotation schemes for characterisation of data types. The following characteristics are extended to any data used for exploration and analysis:

– Ordering of data may be applied using a number of ordering schemata at the same time. Novel ordering schemata not applied in database management must be developed since ordering of historical data is often based on associated data such as history, region, and important events.

– Defaults change over time. They are also context-dependent. Therefore, the time and context must also be represented whenever defaults are used.

– Representation systems vary for different platforms, for historical periods and for different regions. They depend on agreements of the society at that time and at that place. Additionally data representation may have been changed by the chroniclers.

– Casting of data is necessary whenever we are interested in normalised data that can be used for comparisons. The casting schema may have to be changed in the case that the new data are replacing data currently used.

– Classification are dependent on the habits of the region and the historical period. Classification schemata carry an important part of the semantics and must thus be maintained also in the case of data aggregation.

– Data types may have their specific hierarchical ordering of partitions (e.g., within time, space). These partitions may also be used for approximation by generalisation of data.

Enriching OLAP data by type information enhances the correctness of analysis. Data types can typically be characterised by their properties [Mansmann et al., 2007; Thalheim, 2000].

3.5 The OLTP-OLAP Transformation

We define a layered OLTP-OLAP architecture by introducing an OLTP schema, by characterizing aggregations, and by introducing OLTP-OLAP transformations. Based on this framework we derive a specification frame that is useful for guaranteeing correctness of OLAP applications.

Definition 28. An OLTP-OLAP transformation is defined through

– a family of grouping functions G,

– a family of aggregation functions F, and

– a family of transformations T.

Grouping functions are used for defining lattices of partitions. Aggregation functions are used to associate values computed through application of aggregation functions. Transformations are used to transform the fact attributes. An example for a nonlinear transformation is the conversion of fuel consumption $\frac{l}{100km} \mapsto \frac{miles}{gallon}$ [Hand, 1994].

It is now well understood that OLAP applications can be built on top of OLTP systems. An enriched OLTP-OLAP transformation consists of

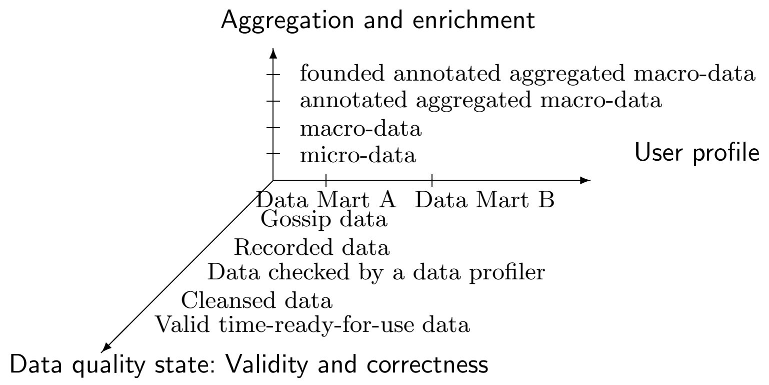
Aggregation and enrichment
founded annotated aggregated macro-data
annotated aggregated macro-data
macro-data
micro-data
User profile
Data Mart A
Data Mart B
Gossip data
Recorded data
Data checked by a data profiler
Cleansed data
Valid time-ready-for-use data
Data quality state: Validity and correctness

Figure 4: Three dimensions of the OLAP space: Data layers, user profiles, and quality of data

an ***enriched OLTP scheme*** that provides partitions together with domain types, contains a schema of the OLTP database, and a set of permitted aggregation functions applicable for the fact attributes and

a ***transformation function*** defined by a view on the enriched OLTP scheme, by attributes and derived attribute where the latter are defined through aggregation functions applied to the cells of the cube and by a number of partitions for some of the dimensions.

The ***result of the OLTP-OLAP transformation*** is a (materialised) (extended) (spreadsheet) cube that typically defines a *star* or a *snowflake schema.*

The **OLAP schema** consists of an OLTP-OLAP transformation and a set of *modelling assumptions* [Lenz and Thalheim, 2005]. Typical assumptions taken into consideration for an OLAP schema are $\mathcal{P}_1$-subset invariance, $\mathcal{P}_2$-superset invariance, $\mathcal{P}_3$-update invariance, $\mathcal{P}_4$-union invariance, and equidistance. Subset invariance states that the fact values remain fixed if the the underlying OLTP database is restricted to a sub-database guided by a policy $\mathcal{P}_1$. Superset, update and union invariance is defined in a similar way. Equidistance requires that the transformation functions used are linear.

OLAP applications use data in a variety of ways, in different granularity, with different scope and of different quality. Conceptual development of such applications must take this into account. We thus use for conceptual development an additional characterisation depicted in Figure 4 and discussed in detail below.

3.6 The OLTP-OLAP Specification Frame

We already observed that the correctness of computations within the cube depends on the aggregations and OLAP query functions, and on properties of the domains of fact attributes $A_1, ..., A_m$. If we can restrict the application to some OLAP query functions then correctness of computation may be achieved more easily. If some of the aggregation functions are not of interest in the given application we may exclude them. The domain types $Dom(M_j)$ of the fact attributes M_j may preserve a set Ψ of properties. Furthermore, the correctness depends on the cube under consideration. Therefore, we propose specification frames restricting OLAP applications.

Various modelling assumptions can be applied to cubes. Some of them are:

– Disjointness: OLTP-OLAP transformations are restricted to groupings which generate disjoint groups.

– Completeness: Groupings used for OLTP-OLAP transformations cover the entire set of database objects.

– $\mathcal{P}$-subset invariance: Fact values are not changed if the OLTP database is restricted to objects based on the policy $\mathcal{P}$.

– $\mathcal{P}$-union invariance: Fact values are not changed if the OLTP database is extended by new objects which depend on the policy $\mathcal{P}$.

– Equidistance: Transformations $T \in \mathcal{T}$ which are used are linear.

A policy restricts the modification of a database. Policies are used to automatically enforce OLTP and OLAP constraints integrity. Integrity enforcement is based on the constraints management supported by systems (checking mode, statement or row level, preor postconditions, scope conditions, matching conditions, reference types), integrity constraint modules execution (scheduling, conflict resolution, and granularity of triggers or procedures; order of execution), level of consistency during integrity control, and level of specification (declarative, imperative, interface-backed).

Domain types may be restricted by properties such as precision and accuracy, granularity, and ordering. Furthermore, domains can be based on scales, can represent classifications and can contain default values.Domain values can be extended by measures, e.g., relative, absolute, linear and non-linear. Domain values can be transformed by casting functions to values of other domain types. [Thalheim, 2000] distinguishes nominal, absolute, rank, ratio, atomar, complex, and interval types.

Typically application do not need the full expressive power for their computation. Instead we may restrict the application to a specific set of aggregation functions, to OLAP query functions defined with the OLAP algebra, by some constraints or properties and to some modelling assumptions.

Definition 29. A specification frame $\mathfrak{F} = (F, \mathfrak{O}, \Psi, \mathfrak{M})$ consists of a set F of aggregation functions, a set $\mathfrak{O}$ of OLAP query operations, a set Ψ of properties, and a set $\mathfrak{M}$ of modelling assumptions.
The cube C is called $\mathfrak{F}$-correct if the OLTP-OLAP transformations are restricted to the functions in F, fact domains fulfill Ψ, and the modelling assumptions are valid for the cube C.

Definition 30. An OLAP application consists of an OLTP schema $\mathcal{S}$ and of a specification frame $\mathfrak{F}$ and of a set of $\mathfrak{F}$-correct cubes.

Next we show how to avoid incorrect application cases. Then we characterize OLAP schemata by elaborating modelling assumptions.

4 Correct OLAP Applications

4.1 Incorrect OLAP Computations and OLAP Paradoxes

Several observations concerning wrong OLAP computations have already been made but are not well considered in the current practice of OLAP computations. This fact is surprising since mathematics provides an appropriate background to avoid these paradoxes. We hint at some of them without defining them or the operations. These paradoxes have been reported in [Lehner et al., 1998; Lenz and Shoshani, 1997; Lenz and Thalheim, 2001a] and are based on [Fisher, 1962; Schneeweiß, 1965; Sondermann, 1973].

Summarization over one-way tables: The first category of incorrect OLAP applications is related to grouping [Lenz and Shoshani, 1997]. Roll-up operations become incorrect if hierarchies used for the cube are not based on the disjointness property for groupings.

The ***Simpson Paradox*** [Simpson, 1951] appears if one dices or summarizes over a separator Z of a binary join dependency which is a type of MVD $Z \rightarrow\rightarrow X|Y$ or binary join dependency $\bowtie (XZ, YZ)$ of a cube scheme with component sets $X \cup Y \cup Z$.

Non-commutative operators: It is well known in mathematics but not so well known in the OLAP community that the transformation T and the arithmetic mean are not generally commutative. This fact is essential even in the phase of popularization of a data warehouse as part of ETL (Extraction-Transformation-Loading). Let O be a given set of numeric operators. Let $o_1 \in \mathsf{O}$ be a linear operator and $o_2 \in \mathsf{O}$ a non-linear operator. Then it is generally not true that $o_1 \circ o_2 = o_2 \circ o_1$.

Collapsing hierarchies: Hierarchies or classifications may be asymmetric and unbalanced. Such hierarchies may collapse whenever groups are combined without preservation of the asymmetry. The inhomogeneous granularity of attributes involved causes incoherence of information computed by the cube operator.

Imperfect aggregation: Cube materialisation is sometimes senseless. There exists an appropriate homomorphism H only if a pseudo-inverse of the aggregations is used, e.g, [Sondermann, 1973].

Loss of identifiability in OLAP schemata: OLAP schemata use aggregated values. After aggregation detailed computations based on identifiable database objects may be impossible, be infeasible or lead to superficially complex computations. These computations become simpler if the OLTP schema is well-designed and the OLAP schema is properly based on the OLTP schema.

These paradoxes can be avoided if operations and transformations are well-defined and conditions for their application are provided.

4.2 Correctness Conditions for OLAP Operations

Cube operations may be still correct for some data, whereas in other cases incorrectness becomes obvious. The development of guards is a appropriate way to avoid incorrectness. We may include these guidelines into the specification of OLAP schemata depending on the functions used.

Based on the theory of aggregation functions, our cube schema definition and its properties we can now identify a number of properties[2]. Drill-down functions are used for decomposing groups of data along a hierarchy.

Proposition 31. *Drill-down functions are well defined*

if the cube construction is based on disjointness and completeness modelling assumptions and

if data granularity is guaranteed at leaf level L_1 and no structural null are used at any level L_i $(i > 1)$ in between.

The proof of this observation is straight-forward and based on direct application of the definitions.

Data granularity allows to identify each object used by its values. We notice that the conditions are only necessary ones. For instance, the *avg* function may be incorrect if the domain is ordinal.

Roll-up functions are used for merging groups along a hierarchy. Problematic results are observed for collapsing hierarchies especially in the case of algebraic and holistic aggregation functions.

[2] We omit their proofs since they are straightforward checks of the function application.

Proposition 32. *Roll-up functions are only well-defined if data granularity is guaranteed at leaf level L_1 and no structural null are used at any level L_i $(i > 1)$ in between.*

The proof is based on properties of roll-up functions and the completeness property.

This proposition directly leads to a restriction for roll-up functions.

Proposition 33. *Roll-up functions must be based on disjointness and completeness modelling assumptions.*

Dice functions are unproblematic for distributive functions. Algebraic aggregation functions may be combined with repairing functions[Lenz and Thalheim, 2001a].

Proposition 34. *The Simpson paradox is observed if for groups at dice level $L_p \preceq L_r$ or at roll-up level $L_p \preceq L_r$
$(\forall j(o(g_{ij}) < o(g_{kj}))) \not\Rightarrow o(\sum_{j=1}^{n} g_{ij}) < o(\sum_{j=1}^{n} g_{kj})$ for given k and i.*

Proposition 35. *Dice functions can only correctly be applied if the cube construction is based on union invariance, i.e.*

$$f(\sqcup^*_{o \in g_i} value(o)) = \sqcup^*_{o \in g_i}(f(value(o)))$$

*holds for groups g_i along dimensions for the generalized union $\sqcup^*$.
Distributive aggregation functions are union invariant. Otherwise repair functions must be applied for algebraic aggregation functions.*

Proposition 36. *Dice functions can only be used along dimensions for which constraints among cube dimensions are not lost, i.e. if the constraint set that is shrunk to the new dimensions implies all constraints within these new dimensions.*

Slice functions are similar to selection of tuples within a set. They are subset operation and equivalent to conditioning in statistics.

Proposition 37. *Slice functions and roll-up functions must be query-invariant, i.e. for the slice or roll-up function o' and the query function q: $q(\underline{x}_1, ...\underline{x}_n) = q(o'(\underline{x}_1), ..., o'(\underline{x}_n))$.*

Figure 2 illustrates a cube that is not query-invariant.

Proposition 38. *Slice functions must be subset invariant.*

If the case that a slice function is subset-dependent then the slices under consideration change with partitions $\mathfrak{p} \preceq \mathfrak{p}'$.

Constraints invalidated by subset construction are those integrity constraints that have to be expressed through $\forall\exists$-constraints[Thalheim, 2000], e.g., inclusion dependencies, multivalued dependencies, tuple-generating constraints.

OLTP-OLAP transformations can cause paradoxes or lead to problematic OLAP schemes. Statistics and the theory of mathematical functions have developed a rich theory that must be considered for OLTP-OLAP transformations. For instance, the arithmetic average is very sensitive to extreme values such as outliers and may be distorted by them.

Proposition 39. *Application of median instead of mean average functions for aggregation leads to a robust OLAP query operation with respect to outliers.*

4.3 Repairing Approaches

We distinguished between distributive, algebraic, and holistic aggregation functions. Distributive aggregation functions are gracious functions in the sense that OLAP application are correct. We can, however, also extend this approach to algebraic functions.

Definition 40. An aggregation function f can be be expressed by the aggregation function f^* if functions g, h exist such that $f((\mathcal{DB})) = h(f^*(g(\mathcal{DB})))$ for all databases $\mathcal{DB}$.

We also say that the function f can be extended to the function $g \circ f^* \circ h$.
h is called a forgetful function and g is called a pre-computation functor.
Applying the construction to the average function directly leads to functions
$g(\{s\}) = \{(1, s)\}$, $g(M) = \sqcup_{s \in M} g(s)$ for bags,
$f_1^*(i, k) = (i, k)$,
$f_2^*((i, k), (j, l)) = (i + j, \frac{i \cdot k \, + \, j \cdot l}{i + j})$,
$f_{p+1}^*((i_1, k_1), ..., (i_p, k_p), (i_{p+1}, k_{p+1})) = f_2^*(f_p^*((i_1, k_1), ..., (i_p, k_p)), (i_{p+1}, k_{p+1}))$,
and $h(i, k) = k$.

Algebraic aggregation functions are defined through expressions over distributive aggregation functions. Expressions of functions are built [Poeschel and Kaluznin, 1979] using an algebra of function construction operators consisting of identification of arguments, permutation of arguments, introduction of fictive arguments and superposition
$g(f_1(x_{11}, ..., x_{1n_1}), ..., f_k(x_{11}, ..., x_{1n_1}))$ of functions $f_1, ..., f_k, g$.
We can use now the construction of algebraic functions for derivation of the extensions of these functions and conclude:

Theorem 41. *Algebraic aggregation functions can be extended to distributive functions.*

This theorem seems to be surprising. We may however use the expression that has been used for the definition of the algebraic aggregation function. This expression is used for the introduction of an extended schema. The algebraic aggregation function can thus be removed without loss of information.

The average function is quasi-associative, i.e. there are functions f, g, h and associative operators such that

$$avg(x_1, ..., x_n) = f(B(g(x_1), ..., g(x_n)), C(h(x_1), ..., h(x_n)))$$

for $B \equiv C \equiv sum$, $f(x, y) = \frac{x}{y}$, $g(x) = x$, and $h(x) = 1$.

Quasi-associativity can be extended to bisymmetry that is an important property for aggregation over dimensions:
$\forall n, m \in, \forall x_{11}, ..., x_{mn} f_{m \cdot m}(x_{11}, .., x_{mn}) = f_m(f_n(x_{11}, .., x_{1n}), .., f_n(x_{n1}, .., x_{mn}))$.
Typical bisymmetric functions are the classical distributive aggregation functions and the geometric mean.

Theorem 42. *The average function is well-defined on hierarchies only if the hierarchy is well-balanced.*

The bisymmetry property of the average function proves the theorem. Each cell of the cube has the same number of values. In this case we can use the *sum* function together with the size values instead of the average function.

5 Conclusion

OLAP computations may lead to correct results but may also lead to incorrect ones. The main reason for this unpleasant behavior is the ambiguity in the definitions used for the OLAP cube and OLAP operations.

Our definition of the cube and the cube schema is based an a two-layer specification frame for OLTP schemata. The base layer extends the definition of the base data types by partitions of the domains used for the data types of the OLTP schema. The OLTP specification layer is similar to the specification of object-relational or entity-relationship schemata. Cube can be then defined as being structured according to the partitions of the base layer. Cube definitions given in the literature do not use this base layer. In this case, special structuring schemata or lattices of hierarchical structures are introduced. This way of defining a cube becomes cumbersome and is difficult to understand, to apply and to evolve.

As far as we known, a systematic treatment of properties of aggregation functions has not yet been made in database research. Aggregation functions have their own properties. Their application is restricted by these properties.

The paradoxes in the last section of this paper demonstrate the importance of systematic treatment.

We thus decided to introduce the two-layer specification frame for OLTP specification. It should be noticed that stream processing would also be simplified if such layered specifications are used. OLAP schemata can be derived from OLTP schemata. They typically form a view on the OLTP database. The OLAP algebra consists of a number of specific operations such as roll-up and drill-down functions. It is surprising that our definition is the first formal one in the literature. Typically OLAP functions are given on the basis of an illustration of their computation and of examples. The literature then refers to existing systems and their decisions for implementation.

The OLTP-OLAP specification frame we introduced in this paper can be extended to a theory of database aggregations and database abstractions. Furthermore, we did not discuss the impact of constraints to OLTP-OLAP specification frames.

Most of the propositions in this paper give only necessary conditions for correctness. The development of a correctness theory is one of the main issues for future research. We concentrated the paper on distributive and algebraic aggregation functions. Holistic aggregation functions are very important in applications. The existence of a theory for these functions would be beneficial.

References

[Agrawal et al., 1997] Agrawal, R., Gupta, A., and Sarawagi, S. (1997). Modeling multidimensional databases. In Gray, A. and Larson, P.-Å., editors, *Proc. 13th Int. Conf. on Data Engineering - ICDE'97*, pages 232–243, Birmingham. IEEE Computer Society Press.

[Buneman et al., 1994] Buneman, P., Libkin, L., Suciu, D., Tannen, V., and Wong, L. (1994). Comprehension syntax. *SIGMOD Record*, 23(1):87–96.

[Calvo et al., 2002] Calvo, T., Mayor, G., and Mesiar, R. (2002). *Aggregation operators - New trends and applications.* Physica, Heidelberg.

[Fisher, 1962] Fisher, W. D. (1962). Optimal aggregation in multi-equation prediction models. *Econometrica*, 30:744–769.

[Franconi and Kamble, 2004] Franconi, E. and Kamble, A. (2004). The gmd data model and algebra for multidimensional information. In *CAiSE*, volume 3084 of *Lecture Notes in Computer Science*, pages 446–462. Springer.

[Gray et al., 1997] Gray, J., Chaudhuri, S., Bosworth, A., Layman, A., Reichart, D., and Venkatrao, M. (1997). Data cube: A relational aggregation operator generalizing group-by, cross-tab, and sub-totals. *Data Mining and Knowledge Discovery*, 1(1):29–53.

[Gyssens and Lakshmanan, 1997] Gyssens, M. and Lakshmanan, L. V. S. (1997). A foundation for multidimensional databases. In Jarke, M., Carey, M. J., Dittrich, K. R., Lochovsky, F. H., Loucopoulos, P., and Jeusfeld, M. A., editors, *Proc. 23rd Int. Conf. on Very Large Databases - VLDB'97*, pages 106–115, Athens. Morgan Kaufmann, San Francisco.

[Hand, 1994] Hand, D. J. (1994). Deconstructing statistical questions (with discussion). *Journal of the Royal Statistical Society, Series A*, 157:317–356.

[Hurtado and Mendelzon, 2002] Hurtado, C. and Mendelzon, A. (2002). OLAP dimension constraints. In *ACM PODS*, pages 169–179.

[Lehner et al., 1998] Lehner, W., Albrecht, J., and Wedekind, H. (1998). Normal forms for multivariate databases. In *SSDBM X, Capri.*

[Lenz, 1993] Lenz, H.-J. (1993). Contribution to the discussion on "Deconstructing statistical questions" by David J. Hand. Read before The Royal Statistical Society , London.

[Lenz, 1994] Lenz, H.-J. (1994). The conceptual schema and external schemata of metadatabases. In *SSDBM*, pages 160–165. IEEE Computer Society.

[Lenz and Neiling, 1998] Lenz, H.-J. and Neiling, M. (1998). Zur Summierbarkeit von Häufigkeiten in multi-dimensionalen Tabellen. Technical Report 1998/28, Diskussionsbeiträge des Fachbereichs Wirtschaftswissenschaft der Freien Universität Berlin.

[Lenz and Shoshani, 1997] Lenz, H.-J. and Shoshani, A. (1997). Summarizability in OLAP and statistical databases. In *SSDBM IX, 1997, Washington.*

[Lenz and Thalheim, 2001a] Lenz, H.-J. and Thalheim, B. (2001a). OLAP Databases and Aggregation Functions. In *Proc. 13th Intern. Conf. on Scientific and Statistical Database Management, Jul 18-20, 2001, George Mason University, Fairfax, Virginia, USA*, pages 91–100. IEEE Computer Society.

[Lenz and Thalheim, 2001b] Lenz, H.-J. and Thalheim, B. (2001b). OLAP databases and aggregation functions. In *13th SSDBM 2001*, pages 91 – 100.

[Lenz and Thalheim, 2005] Lenz, H.-J. and Thalheim, B. (2005). OLTP-OLAP schemes for sound applications. In *TEAA 2005*, volume LNCS 3888, pages 99–113, Trondheim. Springer.

[Lenz and Thalheim, 2006] Lenz, H.-J. and Thalheim, B. (2006). Warning - cube may mislead. In *CSIT'06*, volume 2, pages 7–16, Amman, Jordan.

[Lewerenz et al., 1999] Lewerenz, J., Schewe, K.-D., and Thalheim, B. (1999). Modeling data warehouses and OLAP applications by means of dialogue objects. In *Proc. ER'99, LNCS 1728*, pages 354–368. Springer, Berlin.

[Malzew, 1965] Malzew, A. (1965). *Algorithms and recursive functions*. Nauka. (in Russian).

[Mansmann et al., 2007] Mansmann, S., Neumuth, T., and Scholl, M. (2007). Multidimensional data modeling for business process analysis. In *Proc. ER'2007, LNCS*, page .. Springer.

[McCullagh and Nelder, 1983] McCullagh, P. and Nelder, J. A. (1983). *Generalized Linear Models*. Chapman and Hall, London.

[Meo-Evoli et al., 1992] Meo-Evoli, L., Ricci, F. L., and Shoshani, A. (1992). On the semantic completeness of macro-data operators for statistical aggregations. In Hinterberger, H. and French, J. C., editors, *Proc. of the 6th Int. Working Conf. on Scientific and Statistical Database Management, Monte Verita, Switzerland, 1992*, pages 239–258. Institut f. Wissenschaftliches Rechnen Eidgenössische Technische Hochschule Zürich.

[Molnar, 2007] Molnar, A. (2007). *A general partition data model and a contribution to the theory of functional dependencies*. PhD thesis, Eötvös Loránd University, Faculty of Informatics, Budapest.

[Molnar and Thalheim, 2007] Molnar, A. and Thalheim, B. (2007). Conceptual development of OLAP applications. In *Business Intelligence: Methods and Applications*, pages 27 – 38. Klöden-Verlag.

[Paredaens et al., 1989] Paredaens, J., Bra, P. D., Gyssens, M., and Gucht, D. V. (1989). *The structure of the relational database model*. Springer, Berlin.

[Poeschel and Kaluznin, 1979] Poeschel, R. and Kaluznin, L. A. (1979). *Funktionen- und Relationenalgebren: Ein Kapitel der diskreten Mathematik*. Birkhaeuser, Basel.

[Rafanelli, 2003] Rafanelli, M. (2003). Operators for multidimensional aggregate data. In *Multidimensional Databases*, pages 116–165.

[Rafanelli, 2005] Rafanelli, M. (2005). Basic notions on multidimensional aggregate data. In Khosrow-Pour, M., editor, *Encyclopedia of Information Science and Technology (I)*, pages 211–216. Idea Group.

[Schewe and Thalheim, 1993] Schewe, K.-D. and Thalheim, B. (1993). Fundamental concepts of object oriented databases. *Acta Cybernetica*, 11(4):49–81.

[Schneeweiß, 1965] Schneeweiß, H. (1965). Das Aggregationsproblem. *Statistische Hefte*, 6:1–26.

[Simpson, 1951] Simpson, C. H. (1951). The interpretation of interaction in contingency tables. *JRSS, series B*, 13:238–241.

[Sondermann, 1973] Sondermann, D. (1973). Optimale Aggregation von großen Gleichungssystemen. *Zeitschrift für Nationalökonomie*, 33:235–250.

[Thalheim, 1991] Thalheim, B. (1991). *Dependencies in relational databases.* Teubner, Leipzig.

[Thalheim, 2000] Thalheim, B. (2000). *Entity-relationship modeling – Foundations of database technology.* Springer, Berlin.

[Vassiladis and Skiadopoulos, 2000] Vassiladis, P. and Skiadopoulos, S. (2000). Modeling and optimization issues for multidimensional databases. In *Proc. CAiSE'2000*, LNCS 1789, pages 482–497. Springer, Berlin.