An Integrity Constraints Language for a Conceptual Multidimensional Data Model.

Fernando Carpani, Raul Ruggia
Instituto de Computación. Facultad de Ingeniería.
Universidad de la República. Montevideo, Uruguay.
e-mail: {carpani,ruggia}@cing.edu.uy

Abstract

A number of models for conceptual design of multidimensional databases have been proposed but only a few of them support the notion of integrity constraints.

This work introduces a Conceptual Multidimensional Data Model, which supports a generic constraint language and a high-level data structure. Also, a graphical notation for structures and constraints is provided.

We think that this language allows descriptions to be as precise as necessary by means of the integrity constraints it provides. These integrity constraints can be seen as load program specification for a multidimensional database.

Keywords: Conceptual Design, Multidimensional Databases, Integrity Constraints.

1. Introduction.

A Data Warehouse is a database with summarized information oriented to decision support. Usually, from the user point of view, the Data Warehouse is a multidimensional database.

In a data model, some good properties are achieved by means of high-level integrity constraints. These constraints help the analyst to fulfill the 100% principle and the conceptualization principle ([Lou92]). In addition, from a practical perspective, the constraints simplify the maintenance of the Data Warehouse design.

For the best of our knowledge, does not exist any proposal for a multidimensional data model including a generic constraint language.

In this work, a conceptual data model called CMDM is proposed. This model provides a many-sorted logic language for defining integrity constraints. Also, the model provides a high-level structure name Dimensional Relation, which is oriented to the definition of subsets of cubes with common properties.

With CMDM, it is possible to specify portions of the real world in multidimensional terms by means of a graphical language and textual constraints definitions. The model supports a "classical" vision in the sense of describing data structures and integrity constraints with a high abstraction level.

The rest of the paper is organized as follows. Section 2 presents an overview of the related works. Section 3 introduces the notions of multidimensional data structures. Section 4 gives a characterization for CMDM data structures, integrity constraints and graphical notation. Finally, section 5 presents some conclusions including a comparison with other models and future work.

2. Related Work

In order to describe multidimensional databases, based on relational, logical and conceptual data models were proposed.

Examples for the relational based approach are [Li96] and [Gys97].

Between the logical models, a precursor work supporting the independence of implementation and generic dimensionality ([Cod93]) is found in [Agr97]. However, the model does not make difference between schema and instance, and some relevant properties are not represented in a structural way. In [Cab97] and [Cab98], a logical multidimensional data model is proposed. The model is explained in multidimensional terms and has clear differences between schema and instance. It also supports generic dimensionality and some query languages. However, the model is oriented to express manipulation and not data structures or relations between data. Its features, makes the model suitable for specify multidimensional databases at logical level and CMDM takes some ideas from it.

There are many works on conceptual multidimensional data models. In [Leh98a], a data model with a formal characterization of multidimensional domains and some issues on summarizability ([Len97]) is exposed. However, this model does not support generic dimensionality. In [Sap99a], a multidimensional extension to E/R model is proposed. This model has a clear graphical language and a clear semantics but does not support generic dimensionality or summarizability. In [Gol98a], a graphical model is proposed. This model supports some constraints including some summarizability specifications.
but it does not support generic dimensionality and it
doesn’t let the drill across paths be identified in a clear
way. The last issue is solved in [GoI99] or [GoI98]
suggesting a method which seems to be adequate for
multidimensional schema integration. In these works,
some notions of workload are supported. Finally, [Fra99]
and [Fra99a] expose the DWQ proposals about
conceptual multidimensional data modeling. Since this
model is based on Description Logics ([CaI98, Ho99,
Ho99a]), some reasoning methods over the model can be
implemented.


This section presents the basic notions on
multidimensional data structures and integrity constraints.
Detailed information can be found in [KeI96a] or
[Tho97].

3.1. Multidimensional Data Structures.

A Multidimensional Database is a set of Cubes (Fig.
1 [a]). Each Cube is a multidimensional matrix or a
multiple variable function. The domains and ranges of
these functions are finite sets.

Each domain of a cube is called a Dimension Level,
and a range is called a Measure (Fig. 1 [b]). There is only
one difference between a dimension level and a measure:
the role in a cube. Therefore, a measure is also a
dimension level. A tuple built with one element from each
dimension level of a cube is called a Coordinate. Thus, a
cube is a function from a set of coordinates into a
measure. Note that, the set of coordinates of a cube is a
finite set because all domains are finite.

Fig. 1. (a) A Multidimensional Database as set of cubes. (b) Elements in a Cube.

Some cubes in the database could be related by a Drill
relationship (up or down) (Fig. 2 [a]). This relationship is
induced by a DrillUp relationship among some domains
from each one of the involved cubes (Fig. 2).

A set of Dimension Levels with its DrillUp
relationship is called a Dimension.

In Fig. 2 (b), a dimension exposing its DrillUp relation
is represented. This relation has two faces: One between
levels (the intension) and one between level elements (the
extension). In the levels face, the DrillUp relationship is a
partial order between levels called Level Hierarchy or
DrillUp Hierarchy. This hierarchy may present alternative
paths through different levels. In the level elements face,
an element in a level is related with at most one element in
each higher level in the hierarchy.

The DrillUp relationship over cubes is induced by the
DrillUp relationship over the involved levels. In Fig. 2 (a),
the cube (3) is related to (or “is computed from”) the cube
(1) because the Salesman level is under the City level in the
Salesmen dimension level hierarchy.

Fig. 2. (a) Some Drill relationship between cubes. (b) A
dimension for Salesmen.

All measures from cube (1) grouped by this
“movement” in the hierarchy of Salesmen, must be
summarized by the sum operation to compute the measure
in cube (3).

3.2. Integrity Constraints

Over the multidimensional structures described in the
previous sections, many integrity constraints may be
defined.

For example, on levels we may define the following
constraints:
• “A salesman must be greater than 18 and less
than 61”.
• “There cannot be two different salesmen with the
same document number”.

Note that the first condition is referred to each
element in the level while the second is referred to the set
of elements in the level. Something similar occurs on a
dimension:
• “In every city there are at least two salesmen and
at most five ones”.

- “The enterprise sells vehicles of the following types: sedan, pick-up, van, wagon or truck. These vehicles can be classified in two categories: familiar or utilitarian. All sedan and wagon must be familiar, and all trucks must be utilitarian.”

These constraints are referred to different levels in a dimension so are referred to values in the dimension.

In the context of a database, a constraint may state conditions about a level, about a dimension or about different dimensions simultaneously:

- “Each salesman can sell vehicles of only one category (familiar or utilitarian)”.

In the vehicles dimension, there may be category and type levels (also Brand, Model and Vehicle). Note that in the cubes of Fig. 2 (a), there is no reference to category level. However, these cubes must verify the constraint.

In summary, the constraints in a multidimensional model must be focused on the determination of which cubes can belong to the specified database and which not.

4. Integrity Constraints and Multidimensional Structures in CMDM.

Usually, a data model is introduced by explaining the way to specify in the model the following three elements:

- How the data and data relation mechanism may be described. This is given by the data structures schemas.

- How much must be understood such schemas. This is given by the description of the possible instances of each structure schema.

- What data that there must be and there must not be in the structures. This is given by a description of an integrity constraints mechanism.

In the next sections, CMDM will be introduced according to these guidelines using natural language. However, a formal description for the schemas, instances and general constraints language can be found in [Car00].

4.1. Levels.

The intuition behind the level notion is a set of elements. Therefore, a Level Instance is a set of elements of a type; it is such as an entity set in the E-R model. A Level Schema has a name, a type definition and a set of constraint expressions. In Fig. 3 (a), examples of graphical notation for level schemas are presented, showing the name and the type definition. The constraint expressions are explained in section 4.5. In this version of CMDM, the types of the level elements are assumed as record types.

The instance of a level must satisfy its constraints.

4.2. Dimensions.

The intuition behind the dimension is a hierarchy of elements from each level. Thus, a Dimension Schema is a tuple with a name, a set of level names, a partial order between the previous level names and a set of constraint expressions (section 4.5). The partial order describes how the levels must be related.

A Dimension Instance is a pair with two functions. The first function, maps each level name to its instance. For each pair of levels in the schema partial order, the second function returns a strict function from the instance of the first level to the instance of the second one.

In this way, the second function of the pair represents the drill-up relation as a high order function. Given an element in a level, any related element in a higher level may be obtained through the drill-up function. Thus, the result of applying the drill-up function to the levels day and quarter is a function that maps the instance of the level day to the instance of the level quarter. The result of this function applied to the element from the level day with value 01/02/00, must be the element from the level quarter with value 1 (Fig. 3 [b]). Since the result of a partial application of the drill-up is a strict function, there must be assumed the existence of a distinguished element \( \bot \) in each level, which will be returned when the related element does not exist. These distinguished elements are linked between each other.

Every function involved in a dimension instance, must satisfy the dimension constraints.

![Fig. 3](image-url)  (a) Some level schema. (b) A dimension Time with alternative hierarchies.

Fig. 3 (b) shows a possible instance for the dimension time and Fig. 2 (b) shows a possible instance for Salesmen dimension.
4.3. **Dimensional Relations.**

As defined above, a multidimensional database is a set of cubes built on a set of dimensions. The idea of **Dimensional Relation** is a subset of cubes from the multidimensional database. A **Dimensional Relation Schema** has a name, a set of dimensions and a set of constraint expressions. A **Dimensional Relation Instance** is a set of cubes. Such cubes have at least one level from each dimension in the schema in any role (coordinate or measure) and satisfy the constraints in the schema.

These cubes are indeed generic ones (abstracts in [Cab97] terminology). The cubes in CMDM are functions from tuples with one element from each level to Booleans. The tuples in the cube domain include all the levels in spite of the role the level plays in the cube.

Fig. 5 (b) shows a possible instance of the transactions dimensional relation. Rectangles represent dimensions and the oval represents the dimensional relation in itself. Fig. 5 (c) shows a possible instance of the dimensional relation depicted in figure (a). Note that:

- Every dimension is involved as an axis or as a measure.
- A dimension (vehicles) involved as an axis in one cube is involved as a measure in another. This is a form of general dimensionality.

4.4. **Multidimensional Databases.**

A multidimensional database is a set of cubes built over some dimensions, which are built over some levels. Therefore, a **Multidimensional Database Schema** in CMDM has four elements:

- A set of schema levels used to build the dimensions.
- A set of Dimension schemas built on the previous level schemas.
- A set of Dimensional Relation schemas built on the previous dimension schemas.
- A set of constraint expressions such each constraint involves any previous set or element in such sets.

A Multidimensional **Database Instance** in CMDM is the set of cubes that satisfies every constraint and is determined by the following tree functions that map each level name, each dimension schema and each dimensional relation to its own instance.

4.5. **Integrity Constraints in CMDM.**

In the Relational Model, there is a set of clear ad-hoc constraints to use: the dependencies (functional, join, etc.).

In a multidimensional model like CMDM, it is not clear which constraints must be included in a set like the previous and which not. In this sense, the use of a generic logic language appears as a good election. Nevertheless, it may be restricted to add tractability. In addition, in order to simplify its use, a graphical notation may be added. This is the approach followed in CMDM.

The proposed language is a many-sorted logic language, which supports terms representing sets and relative quantification over these sets. There must be in the language an instance constant for each schema element in the model. Each one of these constants represents the instance of the schema element with this name. In addition, there are another constants representing functions such that given an instance constant, the function returns its schema. For example, there is a function **Dim_Level** that returns the set of level names in a dimension. Our proposal makes an intensive use of macro definitions to add readability.

As explained in the previous sections, every structure has a set of constraint expressions. Such constraint expressions are written in this language and it is assumed.
that the related structure is referenced by them.

For example, the first two constraints over the salesman level described in the section 3.2 can be represented in the language with the following sentences:

\[ \forall x \in Salesman. (x(\text{age}) > 18 \land x(\text{age}) < 61) \]

\[ SKey(Salesman,Document) \]

The first constraint states that every salesman in the dimension has the age between these limits.

The second constraint is an application of the following macro definition:

\[ SKey(Lvle,Att) \equiv \forall x \in Lvle. \forall y \in Lvle. (x(Att) = y(Att) \rightarrow x = y) \]

This macro says that the second parameter is a key in the instance of the level in the first parameter. This macro does not support compound keys but can be replaced by the following:

\[ Key(Lvle,AttSet) \equiv \forall x \in Lvle. \forall y \in Lvle. ((\forall a \in AttSet. (x(a) = y(a)) \rightarrow x = y) \]

### 4.6. Constraints in a Graphical Notation.

As a general guideline, in the schema graphical representation of any item can appear some constraints written in logical language. However, some constraints may have a graphical representation and these representations must be translated into the logical language.

In a level, the graphical representations are simple. For example, to show that an attribute is a key in a level, its name must be underlined in the level definition. However, in dimensions and dimensional relations the graphical representations may be more complex and expressive.

The next subsections explain some of these representations.

---

**Fig. 6.** Constraints in a Dimension.

### 4.6.1. Graphical Constraints on Dimensions.

The first kind of graphical constraints on dimensions follows the general guideline.

Fig. 6 shows some constraints. The constraints from the bottom of the picture are associated to the Salesman level. The middle rectangle contains dimension constraints. In this case, this macro can be only used in constraints in this dimension. The top rectangle contains a set of constraints, which are applications of the macro defined in the middle rectangle. This macro states that the drill-up from every element in the City level whose name value is received in the first parameter must be an element in the County level with the value received in the second parameter as county name.

### 4.6.2. Graphical Constraints on Dimensional Relations.

The possible instances of a dimensional relation are sets of cubes. The graphical expressions for constraints over these sets can be induced by the logical language.

Fig. 7 shows this idea. This constraint says “There are at least one cube in the instance of Transactions with Price as measure and the others levels as coordinates”. If the arrow over Price does not appear then the constraint says, “There is at least one cube in the instance of transactions involving these levels as coordinates or measures”. The symbol in the cube can be \( \exists, \forall, \neg \) or a name. The symbol “\( \neg \)” must be understood as “not exist” and a name must be understood as “exist one cube with name...”. This kind of constraint can be translated into the logical language. For example, Fig. 7 constraint can be translated as follows:

\[ \exists c \in Transactions. \text{Cube}(c,[\text{Salesman, Salesman, Time, Day, Vehicles, Vehicle, Sales, Price}]) \]

Where “cube” is the following macro

\[ \text{Cube}(c,L) \equiv (\text{Dom}(c) = L \land f(l)) \land \text{Measure}(c,L) \]

This macro says “\( c \) is a cube with the set of levels \( L \) as coordinates and the level \( L \) as measure”. The expression \( \text{Dom}(c) \) represents the set of levels involved in the cube \( c \). The expression \( \text{Measure}(c,L) \) is a macro that says “the level \( L \) is a measure in the cube \( c \)”:

\[ \text{Measure}(c,L) \equiv (\exists f \in \text{Coord}(c). f_1 \in \text{Coord}(c). (\forall l_1 \in \text{Dom}(c). (l_1 \in f(l_1) = f_1(l_1)) \land f(l_1) \neq f_1(l_1))) \]

This macro states that the relation of the remainder levels in the cube and \( I \) is a functional relation.

Another constraint that can be defined on Dimensional Relations is a Roll-up constraint. The roll-up constraint in Fig. 8 states the same that the previous constraint, but adding a condition that says “There exists another cube such that is an aggregation from the previous constraint computed over the drill-up from Salesman to City in the Salesmen dimension, and the measure must be resumed by the sum operator”.

---

**Fig. 8.** Constraints in a Dimension.
5. Conclusions.

A new multidimensional conceptual data model supporting general constraints definitions and high level structures is proposed. This model (CMDM) presents a graphical language for defining structures and constraints, and a logical language with macros for constraints definition.

The use of a logical language for the definition of constraints has some advantages over a set of ad-hoc constraints. This approach helps to analyze which constraints are interesting and which not. The macros can assist the analyst to do this task.

The graphical language is oriented to simplify the definition of constraints and must be understood as “graphical macros” over the logical language.

The model presents basic multidimensional structures such as levels and dimensions and a high level structure (Dimensional Relations). The last structure is oriented to allow the specification of set of cubes with common properties. If the only way to define cubes is by extension, then some interesting cubes might be hidden for the designer. The approach based in Dimensional Relations can help the analyst in the identification of new points of view over the data (new interesting cubes).

The conjunction of this features, allows the model to support some properties:

- The model is independent of any implementation.
- The dimensions hierarchies can be defined in a static way (in the structure of dimension) and in a dynamic way (by aggregations).
- The dimensional relations give supports for Generic Dimensionality.
- The constraints language allows the definitions of Summarizability, constraints over values and roll-up.

Fig. 9 presents a comparison between some models including CMDM. The column Model Type says, “query” for models with a special emphasis in express queries, “logical” for logical models and “conceptual” for conceptual models. The semantics of the remainder columns is clear. The comment “No explicitly” appears in a column when there exists some feature in the model that might be used to support the column feature in an indirectly way.

Some works are in course about CMDM.

A CASE over CMDM is in developing. This CASE is conceived as a graphical editor with facilities oriented to CMDM ([Font00]).
In addition, we are developing a way to build a relational data warehouse schema ([Per00]). In this work, a way to define a mapping between an operational relational database and a specification in CMDM is suggested. With this mapping defined, a method to build a relational data warehouse based on Schema Transformations ([Mar00]) is developed. This data warehouse must be agreed with the original CMDM specification.

A lot of work is still waiting in CMDM.

In order to avoid the generation of contradictory data warehouses, it might be interesting to develop (semi-) automatic methods to check constraints satisfaction. For example, it might be used automated reasoning methods such as Tableaux or Resolution ([Gou97]).

By other hand, the constraints can be thought as specification of load or extraction and cleaning programs for these data warehouses that agree with the CMDM specification. With this approach, might be possible to apply some techniques of formal systems such as [Nor89], oriented to derive such programs in an (semi-) automatic way.

To do these works, the constraint language may need a redefinition or restriction in order to simplify its handling.

In order to make a “fine tuning” on the model from the analyst perspective, more work on industrial applications must be developed.

<table>
<thead>
<tr>
<th>Model</th>
<th>Model Type</th>
<th>Relational Independence</th>
<th>Hierarchy Definition</th>
<th>Generic Dimens.</th>
<th>Summ.</th>
<th>Generic Const. Language</th>
</tr>
</thead>
<tbody>
<tr>
<td>[Lj96]</td>
<td>Query</td>
<td>No</td>
<td>Relations</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>[Agr97]</td>
<td>Query</td>
<td>Yes</td>
<td>By Operations.</td>
<td>By oper.</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>[Cab97]</td>
<td>Logical</td>
<td>Yes</td>
<td>Structural</td>
<td>By oper.</td>
<td>No</td>
<td>No expl.</td>
</tr>
<tr>
<td>[Leh98a]</td>
<td>Logical</td>
<td>Yes</td>
<td>Structural</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>[Sap99a]</td>
<td>Concept.</td>
<td>Yes (ER Extension)</td>
<td>Structural</td>
<td>No</td>
<td>No</td>
<td>No expl.</td>
</tr>
<tr>
<td>[Go98]</td>
<td>Concept.</td>
<td>Yes</td>
<td>Structural</td>
<td>No</td>
<td>Yes</td>
<td>No expl.</td>
</tr>
<tr>
<td>[Fra99a]</td>
<td>Concept.</td>
<td>Yes (ER extension)</td>
<td>Structural (No expl.)</td>
<td>No</td>
<td>No</td>
<td>No expl.</td>
</tr>
<tr>
<td>CMDM</td>
<td>Concept.</td>
<td>Yes</td>
<td>Structural</td>
<td>Yes</td>
<td>Yes (Constr.)</td>
<td>Yes</td>
</tr>
</tbody>
</table>

![Fig. 9. Comparison of CMDM with others models.](image)

**Bibliography.**


