



Available online at [www.sciencedirect.com](http://www.sciencedirect.com)

SCIENCE @ DIRECT®

Computer Standards & Interfaces xx (2005) xxx–xxx

COMPUTER STANDARDS  
& INTERFACES

[www.elsevier.com/locate/csi](http://www.elsevier.com/locate/csi)

## An ontological approach to describe the SQL:2003 object-relational features

C. Calero <sup>a,\*</sup>, F. Ruiz <sup>a</sup>, A. Baroni <sup>b</sup>, F. Brito e Abreu <sup>b</sup>, M. Piattini <sup>a</sup>

<sup>a</sup> *ALARCOS Research Group, Dept. of Computer Science, University of Castilla-La Mancha, Spain*

<sup>b</sup> *QUASAR Research Group, CITI/DI/FCT, New University of Lisbon, Portugal*

Received 29 April 2005; received in revised form 27 June 2005; accepted 19 September 2005

### Abstract

The SQL (Structured Query Language) is currently available in most database management systems and is the focus of an intense standardization process resulting in the latest version of the SQL:2003 standard. Standards are fundamental, but often they are difficult to use, due to their lack of understandability and the occurrence of inconsistencies. An ontology is useful for clarifying the elements of a standard, along with their interrelationships, as well as for detecting inconsistencies. In this paper we propose an ontology for the object-relational features of the new SQL:2003 standard, formalized with UML 2.0 class diagrams and OCL well-formedness rules. The ontology is instantiated with an example in which most of the new object-relational features of the SQL:2003 standard are presented.

© 2005 Elsevier B.V. All rights reserved.

*Keywords:* Databases; SQL:2003 Standard; Ontology; Object-Relational Features; CWM; UML; OCL

### 1. Introduction

After four decades of continuous maturation, databases became a crucial component of information systems, playing a strategic role in the support of organizational decisions. A symptom of that crucial

role is the growth of the relational and object-relational database management systems (DBMS) market, which reached \$13.6 billion last year and is expected to come close to \$20 billion by 2008 [1]. The cornerstone of any DBMS is its query language.

Among the plethora of query languages present in the earliest DBMS, SQL (Structured Query Language) has emerged both as a de jure and de facto standard. SQL has been the focus of an intense process of standardization throughout the years [2–8]), where most DBMS vendors have been actively involved. The latest version of the standard, named SQL:2003, presents some important improvements upon its predecessor,

\* Corresponding author. E.S. Informática, Paseo de la Universidad, 4. 13071 Ciudad Real Spain. Tel.: +34 926295300; fax: +34 926295354.

*E-mail addresses:* coral.calero@uclm.es (C. Calero), francisco.ruizg@uclm.es (F. Ruiz), alinebaroni@di.fct.unl.pt (A. Baroni), fba@di.fct.unl.pt (F. Brito e Abreu), mario.piattini@uclm.es (M. Piattini).

the SQL:1999. The SQL standard has become a very large document, currently with several hundredths of pages. Besides its length and large amount of technical intricacies, also several inconsistencies, such as those shown in Section 3.4, hamper its understandability. Those inconsistencies are probably due to misconceptions on the problem domain, among the many contributors of the standardization process.

When talking about a given domain, the relevant concepts should be expressed by means of a common vocabulary that allows us to:

- (i) overcome barriers created by ambiguous discourses, approaches, representations, and tools in their respective contexts;
- (ii) share the meaning of terms;
- (iii) achieve a common agreement within a community.

This vocabulary can be organized in a glossary, taxonomy, thesaurus or ontology [9].

In a standard like the SQL:2003 there are a considerable number of interwoven concepts, where the semantics of the interrelationships are often very complex. In such a case, an ontology can effectively complement the standard, by mitigating inconsistency problems, thus increasing its understandability. In this paper we propose an ontology for the object-relational features of the SQL:2003 standard. The final goal of this ontology is to show what is, and what is not, in the SQL:2003, its main elements and how they can be combined when describing an object-relational database schema, thereby giving a general overview of the most significant concepts of the SQL:2003.

The remainder of this paper is organized as follows. In Section 2, the evolution of the SQL standard is recalled. We present, in Section 3, the proposed SQL:2003 ontology, assuming that readers are familiar with the concepts of UML<sup>1</sup> (*Unified Modeling Language*), including OCL (*Object Constraint Language*) [10,11]. In Section 4 we validate the ontology, first by mapping its concepts against the SQL:2003 schemata and then by instantiating it with an example. A comparison between the proposed ontology and the Common Warehouse Metamodel (CWM) [12] is

given in Section 5. Finally, we draw some conclusions and give a brief outline of our plans for future work.

## 2. Evolution and current situation of the SQL

The relational model came about as a result of E. Codd's research<sup>2</sup> at IBM during the sixties. The SQL, originally named SEQUEL (*Structured English QUery Language*), was implemented in an IBM prototype (SEQUEL-XRM), during the mid-seventies. Some years later, a subset of this language was implemented in IBM's System-R.

In 1979, ORACLE emerged as the first commercial DBMS based on SQL, followed by several other products (e.g., SQL/DS, DB2, DG/SQL, SYBASE, INTERBASE, INFORMIX, UNIFY). Even those which had not originally implemented SQL as their base query language, offered SQL interfaces (e.g., INGRES, ADABAS, SUPRA, IDMS/R). As a result of this process, SQL became a de facto standard.

In late 1982, ANSI H2<sup>3</sup> began to standardize a version of the relational data model through the IBM donated language, SEQUEL [13]. Renamed SQL by H2, basic SQL was completed and became an American National Standard in 1986 [14] and soon an ISO standard [2].

In 1989, the first version of the SQL standard was revised and an addendum [3], which included main improvements on referential integrity issues, was published. Meanwhile, ANSI brought out a standard for embedded SQL [16].

In the early nineties, a new version, known as SQL2 or SQL-92, was published by ISO [4]. Both the semantic capabilities of the language and error management were then considerably improved. That standard was complemented a few years later, with the approval of SQL/CLI (*Call-Level Interface*) [5] and SQL/PSM (*Persistent Stored Modules*) [6]. SQL became a com-

<sup>1</sup> The latest UML version (2.0) is used, although readers familiar with version 1.\* will be at ease.

<sup>2</sup> A summary of Codd's research can be found in [15].

<sup>3</sup> H2, the ANSI National Committee on Information Technology Standards (NCITS) Technical Committee on Database, was founded in 1978 and was known as X3H2 until 1996, when the Accredited Standards Committee X3 changed its name to NCITS (National Committee for Information Technology Standards). In 2001, NCITS became INCITS (International Committee for Information Technology) and the H2 committee was known as the ANSI INCITS H2 Technical Committee on Database.

Table 1  
Evolution of SQL

70s	Relational model DBMS prototypes (SEQUEL XRM) First relational DBMS
80s	ANSI SQL-86 standard ISO SQL-87 standard SQL-89 addendum ANSI embedded SQL
90s	SQL 92 SQL/CLI SQL/PSM SQL:1999
2003	SQL:2003

plete computational language, with features such as control structures and exception handling.

During the last half of the nineties, SQL was extended by the inclusion of object-oriented capabilities. The resulting standard was divided into several parts. This version, formerly known as SQL3 and then

finally called SQL:1999, incorporated features such as new basic data types (e.g., very large objects), user-defined data types, recursive query operators, sensitive cursors, tables generalization and user roles.

The latest version of the standard, the SQL:2003 [8], is the result of major revisions and extensions to most parts of the SQL:1999 standard. This version includes SQL/XML (XML related specifications), new basic data types (*bigint*, *multiset* and XML), enhancements to SQL-invoked routines, extensions to the CREATE TABLE statement, a new MERGE statement, a new schema object (the sequence generator) and two new sorts of columns (*identity* and *generated*). Table 1 summarizes the evolution of SQL.

The SQL:2003 standard is composed of nine parts, which are briefly described in Table 2. The numeration of parts is not contiguous due to historical reasons: some parts have disappeared (e.g., SQL:1999's part 5 – SQL/Bindings – was included in part 2 of SQL:2003)

Table 2  
Structure and summary of the SQL:2003 standard

Part	Name	Description
1	<i>Framework</i> (SQL/Framework)	Overview of the standard. It describes the conceptual framework used in other parts to specify the grammar of SQL and the result of processing statements in that language by an SQL-implementation. It also defines terms and notation used in the other parts.
2	<i>Foundation</i> (SQL/Foundation)	This part defines the data structures and basic operations on SQL-data. It provides functional capabilities for creating, accessing, maintaining, controlling, and protecting SQL-data. This part also specifies the syntax and semantics of a database language. It deals with the portability of data definitions and compilation units between SQL-implementations and the interconnection of SQL-implementations.
3	<i>Call-Level Interface</i> (SQL/CLI)	It defines the structures and procedures that may be used to execute SQL statements from within an application written in a standard programming language, such that used procedures are independent of the SQL statements to be executed.
4	<i>Persistent Stored Modules</i> (SQL/PSM)	This part specifies the syntax and semantics of a database language for declaring and maintaining persistent database language routines in SQL-server modules.
9	<i>Management of External Data</i> (SQL/MED)	Extensions to Database Language SQL are defined, in order to support management of external data, through the use of foreign-data wrappers and datalink types.
10	<i>Object Language Bindings</i> (SQL/OLB)	It defines extensions to support embedding of SQL statements into programs written in the Java programming language, commonly known as “SQLJ”. This part specifies the syntax and semantics of SQLJ, as well as mechanisms to ensure binary portability of resulting SQLJ applications. In addition, it specifies a number of Java packages and their classes.
11	<i>Information and Definition Schema</i> (SQL/Schemata)	This part specifies an Information Schema and a Definition Schema that describes the SQL object identifier, the structure and integrity constraints of SQL-data, the security and authorization specifications related to SQL-data, the features, sub-features and packages of this standard, and the support that each of these has in an SQL implementation. It also includes SQL-implementation information and sizing items.
13	<i>Routines and Types Using the Java Programming Language</i> (SQL/JRT)	It specifies the ability of invoking static methods written in the Java programming language as SQL-invoked routines and of using classes defined in the Java programming language as SQL structured user-defined types.
14	<i>XML-Related Specifications</i> (SQL/XML)	This part defines ways in which SQL can be used in conjunction with XML.



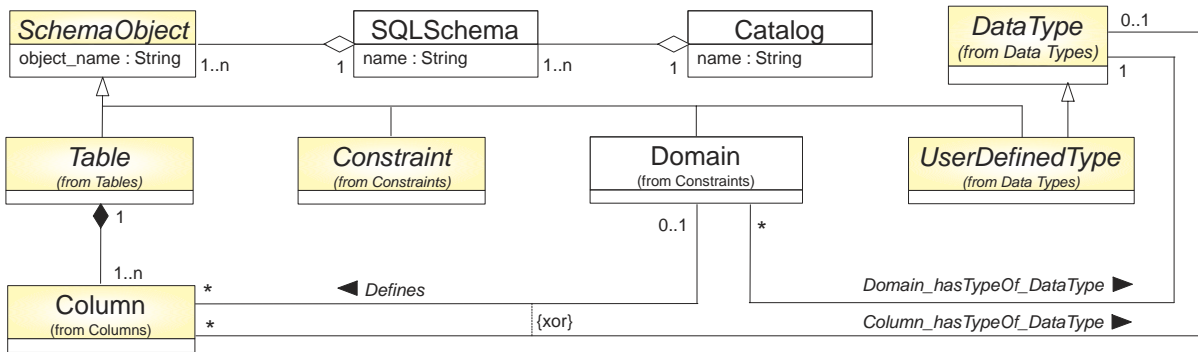


Fig. 2. The schema objects sub-ontology (General view).

diagrams [10,19] have also been used by several authors as an ontology representational language [20–22]. The required formality can be obtained by adding OCL<sup>4</sup> well-formedness rules. UML-based ontologies have the advantage of being more widely understandable (especially by the RDBMS-SQL practitioners that are more familiarized with UML than with the ontological languages) and of being aligned with the MDE (*Model Driven Engineering*) movement [24].

For the development of the ontology we used a process similar to that proposed in [25]. In fact, we combined a top-down and a bottom-up approach through several iterations, using the information of parts 1 and 2 (*Framework* and *Foundation*, respectively) of the standard and reengineering the metadata of part 11 (*Information and Definition Schema*). To clarify some concepts regarding the elements of the ontology and the associations among them, we have also used the books of Melton and Simon [26,27].

For the sake of simplicity, the ontology was divided into two sub-ontologies: *DataTypes* and *SchemaObjects*. The first contains the aspects related to data types, which are detailed on Fig. 1. The second sub-ontology was split in four parts: a general view (Fig. 2), the *Tables* view (Fig. 3), the *Constraints* view (Fig. 4) and the *Columns* one (Fig. 5). Each

class represents a concept (whose properties are represented as class attributes). The associations among classes represent interrelationships among the corresponding concepts, with their semantics specified by the relationship type (generalization, aggregation, association ...) and cardinalities.

The graphical abstractions of UML class diagrams do not allow conveying several types of restrictions that occur in the modelled domain, such as well-formedness rules for the ontological concepts. However, those rules can be expressed as OCL invariants (constraints which must be satisfied at all times), as presented in the following sections. For expressing an invariant in OCL, we start by indicating its context (the class to whose instances the invariant is applied), then the invariant identifier (preceded by the *inv* keyword) and, finally, the invariant expression itself (after the colon sign). That expression takes the current instance of the context class as a starting point (represented by the *self* keyword). The dot operator (“.”) is used either to access attributes, operations or to navigate through an association. In the latter case the identifier used is the role name of the target class, or, if undefined, the actual name of the target class, starting by a lower case letter. Whenever we navigate through an association whose target cardinality is more than 1, then the resulting expression evaluates to a collection type (*set*, *bag* or *sequence*). To apply an operation to that collection, we use the “->” operator. The collection operations used in the following sections are:

```
source->excludes (object:T)
```

This Boolean function is true iff *object* is not an element of the *source* collection.

<sup>4</sup> OCL is a textual language, defined within the UML standard [11], that allows specifying invariants, pre-conditions, post-conditions, guard conditions, and other types of constraints, which cannot be expressed using the UML diagrammatic notations. OCL is underpinned by mathematical set theory and logic, like in formal languages, but was designed for usability and is easily grasped by anybody familiar with object-oriented modeling concepts in general, and the UML notation in particular [23].

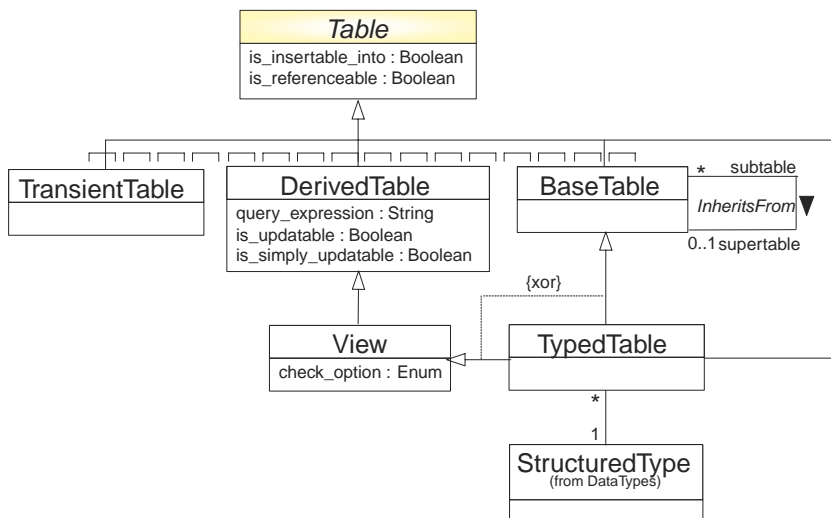


Fig. 3. The schema objects sub-ontology (Tables view).

`source->isUnique (iterators | body)`

This Boolean function is true iff *body* evaluates to a different value for each element in the *source* collection, iterated by the named iterator variables.

`source->notEmpty()`

This Boolean function is true iff the *source* collection is not empty.

### 3.2. Data types sub-ontology

Fig. 1 shows three different kinds of *Data Types*: *Constructed*, *Predefined* and *User-Defined Types (UDTs)*. The constructed types can be *Composite Types* and *Reference Types*. The composite types can be *Collection Types (Arrays and Multisets)* composed of *Elements*, and *Row Types* composed of

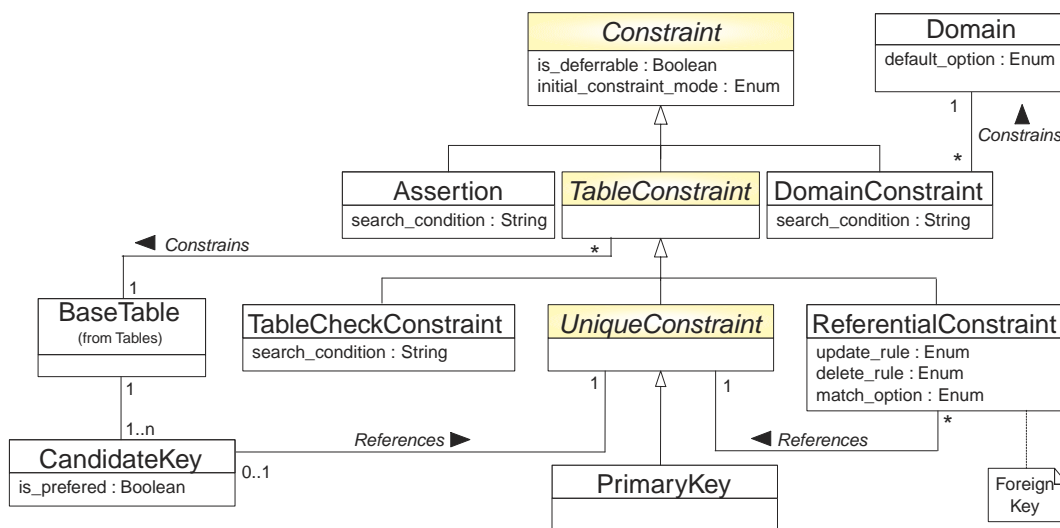


Fig. 4. The schema objects sub-ontology (Constraints view).

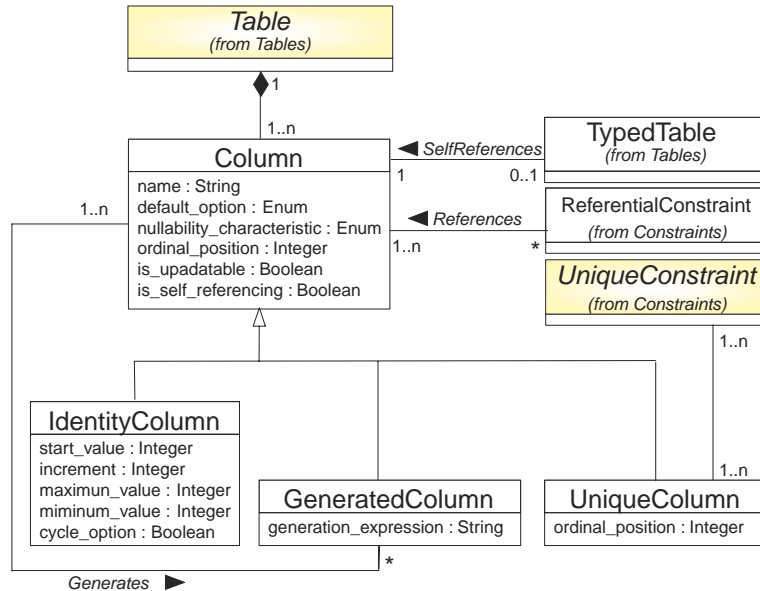


Fig. 5. The schema objects sub-ontology (Columns view).

*Fields*. While in collections all the elements have the same type (thus the corresponding data type is defined for the collection concept), in rows, fields may have different types (thus the corresponding data type is defined for the field concept).

The UDTs can be *Structured Types* and *Distinct Types* (the latter are defined over a predefined data type). Structured types are composed of one or more *Attributes* and optional methods (*Method Specifications*). Each attribute has one data type. Direct inheritance among structured types, row types or reference types, may occur.

The well-formedness rules that complement the *Data Types* diagram (Fig. 1) are the following:

- A *Reference Type* cannot inherit from itself
 

```
context ReferenceType
inv ReferenceTypeCanNotInherit:
self.supertype -> excludes (self)
```
- A *Row Type* cannot inherit from itself
 

```
context RowType
inv RowTypeCanNotInherit:
self.supertype -> excludes (self)
```
- A *Structured Type* cannot inherit from itself
 

```
context StructuredType
inv StructuredTypeCanNotInherit:
self.supertype -> excludes (self)
```
- Attributes of a *Structured Type* must have unique names
 

```
context StructuredType
inv UniqueAttributeName:
self.attribute -> isUnique (a: Attribute| a.name)
```
- An attribute cannot be of the type of the corresponding *Structured Type*

```
context Attribute
inv NonRecursiveAttributeType:
self.structuredType <> self.dataType
```
- Fields of a *Row Type* must have unique names
 

```
context RowType
inv UniqueFieldName:
self.field -> isUnique (f: Field| f.name)
```
- A field cannot be of the type of the corresponding *Row Type*

```
context Field
inv NonRecursiveFieldType:
self.dataType <> self.rowType
```
- Elements of a *Collection Type* must be in a distinct position.
 

```
context CollectionType
inv UniqueElementPosition:
self.element -> isUnique (e: Element| e.ordinal_position)
```

- A *Collection* cannot be of the type of its corresponding *Data Type*

```
context CollectionType
inv NonRecursiveCollectionType:
self.dataType <> self
```

### 3.3. Schema objects sub-ontology

#### 3.3.1. General view

Fig. 2 illustrates the four different *Schema Objects* related to object-relational features: *Tables*, *Constraints*, *Domains*, and *UDTs*. Tables are composed of *Columns*, which are defined over a domain or a data type. Each domain is defined over a data type.

A *Catalog* (a set of *Schemas*) is an additional way of qualifying the names of the elements (*catalog\_name*, *schema\_name* and *schema\_object\_name*). An SQL environment can contain one or more catalogs and, similarly, a catalog may contain one or more schemas.

Details on tables, constraints and columns are shown on the next sub-sections and on Figs. 3–5.

The well-formedness rules that complement the Schema Objects diagram (Fig. 2) are:

- Within a *Catalog*, *SQL Schema* names are unique

```
context Catalog
inv UniqueSchemaName:
self.SQLSchema -> isUnique (s: SQLSchema|
s.name)
```
- Within a *Schema*, *Schema Object* names are unique

```
context SQLSchema
inv UniqueSchemaObjectName:
self.schemaObject -> isUnique (o: SchemaOb-
ject| o.object_name)
```
- A *Column* has either a *Domain* or a *Data Type*, but not both. This invariant is expressed graphically on Fig. 2. It could also be expressed as:

```
context Column
inv ColumnDomainOrDataType:
(self.domain.dataType->notEmpty()) xor(self.
dataType -> notEmpty())
```

#### 3.3.2. Tables view

Fig. 3 shows the taxonomies or hierarchies for *Tables*. On one hand, a *Table* can be *Transient*, *Derived* (including *Views*) or *Base*, and on the other hand there are *Typed* and “*Not Typed*” tables. A typed table is a specialization of a base table or a view, but is not both at

the same time (see the exclusive or constraint between both specializations). Rows of a typed table are instances of one associated structured type. A typed table has a self-referencing column (see Fig. 5), which is the way to implement the object identifier, common in object-oriented languages. Base tables can be part of an inheritance hierarchy as supertables and subtables.

The only well-formedness rule for the *Tables* diagram (Fig. 3) is:

- A *BaseTable* cannot inherit from itself.

```
context BaseTable
inv BCannotInherit:
self.supertable -> excludes (self)
```

#### 3.3.3. Constraints view

Fig. 4 reveals that *Constraints* can be *Assertions*, *Table Constraints* and *Domain Constraints*. A table constraint affects one base table and can be of three possible kinds: *Table Check Constraint*, *Unique Constraint* (including *Primary Key*) and *Referential Constraint* (for representing the foreign keys). A *Table Check Constraint* only specializes its parent class so that this includes the search condition. A *Unique Constraint* is defined over a *set of columns*, while a *Referential Constraint* references only one unique constraint (see Fig. 5). A domain constraint affects one *Domain*. Finally, a *Base Table* has one or more *Candidate Keys* and each one of them corresponds to a unique constraint.

There are no well-formedness rules for the diagram on Fig. 4.

#### 3.3.4. Columns view

Details regarding *Columns* are showed on Fig. 5. Three kinds of columns are identified: *Identity Column* if it is used for implementing the object identifier, *Generated Column* if its values are derived from other columns and *Unique Column* if it is included in a unique constraint.

The well-formedness rules that complement the *Columns* view diagram (Fig. 5) are:

- A *GeneratedColumn* cannot be calculated from itself

```
context GeneratedColumn
inv NonRecursiveGenColumn:
self.column -> excludes (self)
```

- Within a *Table*, *Column* names are unique context *Table*  
 inv UniqueColumnName:  
 self.column -> isUnique (c: Column| c.name)

### 3.4. Mitigating inconsistencies

During the development process of the ontology, some inconsistencies were detected in the SQL:2003 standard. As a consequence, several plausible, yet conflicting, ontological representations may arise. Mitigating those inconsistencies, in the realm of the proposed ontology, requires deciding which is the most appropriate representation. We report below some of the inconsistencies found, and explain how we have tackled them:

- *Inheritance among data types* — The claim that all data types can be specialized is contradicted by the fact that predefined types can not (page 11, in part II: *Foundation*). Our representation decision, based on pages 11, 40, 43 and 44 of part II, was that the inheritance is only possible in structured types, reference types or row types but it has no sense in others, such as distinct and collection types (see Fig. 1).
- *Inheritance among tables* — The claim that all data types can be specialized is also contradicted by the fact that not all tables can (page 55, in part II: *Foundation*). Our representation decision, based on the text on page 13 of part I: *Framework*, was to limit the inheritance to the typed base tables (see Fig. 3).
- *Scope of method specification* — The standard claims at a certain point (page 37, of part II: *Foundation*) that a method specification should be associated with a UDT. In our opinion, it is better to associate the method specification with a structured type because a distinct type is defined over a predefined type which has no methods (see Fig. 1).

## 4. Validating the ontology

### 4.1. Validation by mapping against the SQL:2003 Schemata

Part 11: *Schemata* of the SQL:2003 standard includes several hundred lines of SQL code contain-

ing a set of tables that describe all the elements used by the SQL:2003. It can therefore be considered as a metamodel that may be used to instantiate any SQL schema. However, this large metamodel does not make explicit the relationships among the elements of the standard in a synthesized fashion. Instead, they are hidden amongst the large amount of the schemata code and implementation descriptions (more than 300 pages). In our ontology, the main concepts and their relationships are clearly provided in five Figs. (1–5), thus improving the understandability of the standard. A similar situation appears when one tries to recognize the characteristics of a complex information system by reading the SQL schema code (logical model), instead of looking at the corresponding conceptual model.

We have used the *Schemata* as a starting point for carrying out a mapping process that allowed us to grasp the concepts from the tables. Through this process, we acquired new information for improving and complementing the knowledge extracted from other parts of the standard. Notwithstanding, the most important outcome of this mapping process was the validation of the ontology itself.

In Table 3, the *Definition Schema* base tables and the corresponding elements of the ontology are presented. The page number within the standard, where each concept is defined, is indicated in the Page column.

Additionally, some of the concepts are not derived from the *Definition Schema* tables, but we have included them in the ontology, because they are important when attempting to grasp the model underlying SQL:2003. For instance, the *SchemaObject* concept, which generalizes the *Table*, *Constraint*, *Domain* and *UserDefinedType* concepts, does not appear in any of the Schemata tables. These concepts are generalizations of others that are implemented throughout several tables. The opposite happens too: concepts of the ontology are “camouflaged” into a table that is used to implement other concepts. *DerivedTable*, *TransientTable* and *BaseTable* were all implemented as *Tables*. In general, this situation occurs when the former are specializations of the latter (a more generic concept).

Correspondences among the tables of the *Definition Schema* and the elements of the ontology con-

Table 3  
The definition\_schema base tables and the elements of the ontology

Base table	Page	Corresponding ontology concept/Association mapping
<i>Assertions</i>	121	<i>Assertion</i> concept (Fig. 4).
<i>Attributes</i>	123	<i>Attribute</i> concept and its association with the <i>StructuredType</i> concept (Fig. 1).
<i>Catalog_names</i>	126	<i>Catalog</i> concept (Fig. 2).
<i>Check_Column_Usage</i>	131	Not included for improving the ontology readability. It represents the columns used within the check clause of an assertion, table check constraint or domain constraint.
<i>Check_Constraints</i>	133	<i>Constraint</i> concept (Figs. 2 and 4).
<i>Check_Table_Usage</i>	134	“Constrains” association between <i>TableConstraint</i> and <i>BaseTable</i> (Fig. 4).
<i>Column_Column_Usage</i>	137	“Generates” association between <i>GeneratedColumn</i> and <i>Column</i> (Fig. 5).
<i>Columns</i>	140	<i>Column</i> concept and its associations “Defines” with the <i>Domain</i> concept and “Column_hasTypeOf_DataType” with the <i>DataType</i> concept (both on Fig. 2). Also corresponds to the <i>IdentityColumn</i> and the <i>GeneratedColumn</i> concepts (in the Schemata all these data are put together). In our ontology we decided to separate these concepts (Fig. 5).
<i>Data_Type_Descriptor</i>	145	<i>DataType</i> concept (Fig. 1). Specific attributes of some predefined types (e.g., character_maximum_length) have not been included. Properties of several special kinds of data types (predefined, multiset and array) are implemented in this table.
<i>Direct_Supertables</i>	153	“InheritsFrom” association between <i>BaseTable</i> and <i>BaseTable</i> (Fig. 3).
<i>Direct_Supertypes</i>	155	“InheritsFrom” association between <i>StructuredType</i> and <i>StructuredType</i> (this is different as in Schemata where this relationship is between two <i>User_Defined_Types</i> ) (Fig. 1).
<i>Domain_Constraints</i>	157	<i>DomainConstraint</i> concept and the “Constrains” association between <i>DomainConstraint</i> and <i>Domain</i> concepts (Fig. 4).
<i>Domains</i>	159	<i>Domain</i> concept and the “Domain_hasTypeOf_DataType” association with <i>DataType</i> concept (Fig. 2).
<i>Element_Types</i>	160	<i>CollectionType</i> concept (Fig. 1). The name was changed because the original was inexact (instances of this concept are collections instead of element types of collections). Also the <i>Element</i> concept and its association with <i>CollectionType</i> were implemented with this table (Fig. 1).
<i>Fields</i>	162	<i>Field</i> and the <i>RowType</i> concepts (Fig. 1). The latter is required in order to represent that a row is an aggregation of fields.
<i>Key_Column_Usage</i>	164	The association between <i>UniqueConstraint</i> and <i>UniqueColumn</i> concepts (Fig. 5) and also to the “References” association between <i>ReferentialConstraint</i> and <i>Column</i> concepts (Fig. 5). Consequently, this table implements the set of columns that constitutes a unique, primary or foreign key.
<i>Method_Specifications</i>	169	<i>MethodSpecification</i> concept and its association with the <i>StructuredType</i> concept (Fig. 1).
<i>Referenced_Types</i>	177	<i>ReferenceType</i> concept and the “References” association with <i>StructuredType</i> concept (Fig. 1). The name has been adapted to better represent the data contents in this table, with their semantics.
<i>Referential_Constraints</i>	179	<i>ReferentialConstraint</i> concept and its “References” association with the <i>UniqueConstraint</i> concept (Fig. 4).
<i>Schemata</i>	200	<i>SQLSchema</i> concept and its aggregation with the <i>Catalog</i> concept (Fig. 2).
<i>Table_Constraints</i>	214	<i>TableConstraint</i> concept and the “Constrains” association between <i>BaseTable</i> and <i>TableConstraint</i> concepts (Fig. 4).
<i>Tables</i>	222	<i>Table</i> concept (Figs. 2 3 4 and 5). The <i>TypedTable</i> concept and its associations with the <i>StructuredType</i> concept (Fig. 3) and with the <i>Column</i> concept (Fig. 5) are also implemented with this table.
<i>User_Defined_Types</i>	245	<i>UserDefinedType</i> concept and its specializations <i>StructuredType</i> and <i>DistinctType</i> (Fig. 1). For this last case the “HasSource” association with <i>PredefinedType</i> concept was included.
<i>View_Column_Usage</i>	249	Not included for improving the ontology readability. It implements the columns used, explicitly or implicitly, in the query expression of a view.
<i>View_Table_Usage</i>	251	Not included for improving the ontology readability. Similarly to the previous one, it implements the tables contained in a table reference included into a query expression of a view.
<i>Views</i>	252	<i>View</i> concept (Fig. 3).

firm that a conceptualization of some kind (such as this ontology) is required. For instance, on the *Schemata* it is possible to implement the fields and the rows together in the same table, but in the ontology we have split the concepts, a *Row* being an aggregation of *Fields*. Again is confirmed the interest of having a purely conceptual view (the ontology) over another oriented to the implementation (the Schemata metamodel).

#### 4.2. Validation by instantiating the ontology

To further validate the ontology, we chose some sample schemas based on those found in [26,27], and tried to prove that all their elements (concepts and its relationships) were in the ontology. The sample covers a wide set of concepts of the SQL:2003 and it can be understood better through the table diagram presented on Fig. 6.

Appendix A displays the complete SQL code and Appendix B shows its corresponding ontology instantiation, represented as a UML object diagram. Appendix A illustrates not only the SQL code for the *video\_and\_music* schema, including the four base tables illustrated on Fig. 6, but also other schema objects of several categories. On the tables in this

schema it is possible to identify the following ontology instantiations:

- *Data Type*:
  - *Distinct*: money;
  - *Structured Type*: movie, music\_distributors\_st, address, US\_address;
  - *Array*: genre;
  - *Multiset*: languages;
  - *Reference*: dist\_ref;
  - *Row*: zip;
  - *Field*: basic, plus4;
  - *Predefined*: integer, character, ...;
- *Column*: cust\_address, cust\_last\_name, stock\_number, ...;
- *Identity Column*: nr\_of\_customer;
- *Generated Column*: cust\_complete\_name;
- *View*: problem\_customers;
- *Domain*: price;
- *Attribute*: street, city, distributor\_id, title, ...;
- *Method Specification*: zipcode and length\_interval;
- *Constraint*:
  - *Assertion*: limit\_total\_movie\_stock\_value;
  - *Domain Constraint*: price;
  - *Unique Constraint*: movie\_stars\_unique;
  - *Referential Constraint*: movies\_stars\_fk\_movie;

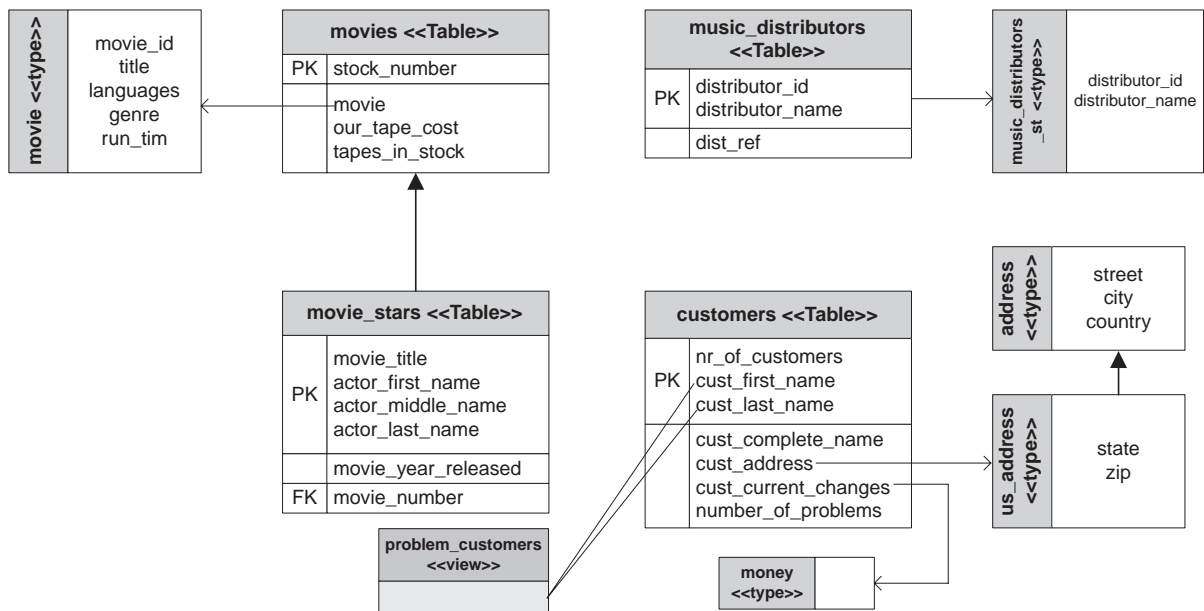


Fig. 6. Representation of the video\_and\_music schema.

- *Table Check Constraint*: movies\_stock\_number\_not\_null, customers\_cust\_last\_name\_not\_null, ...;
- *Primary Key*: movies\_primary\_key, stock\_number, distributor\_name, ...;
- *Foreign Key*: movie\_number.

Other characteristics of the example are:

- table movies uses the domain price and the structured type movie;
- table movie\_stars has a foreign key constraint referencing to table movies;
- music\_distributors is a typed table based on the structured type with equal name but ending in “\_st”;
- table customers has an identity column and a generated column and uses the money distinct type and the US\_address structured type.

## 5. Related work

### 5.1. Introduction to the CWM

The proliferation of data management and analysis tools has resulted in almost as many different representations and treatments of metadata as there are tools. Those representations are proprietary and often not entirely exposed in an API. This hampers

tools interoperability and reduces the potential for QVT (Query, View, Transformation) operations [28].

The main purpose of the CWM (Common Warehouse Metamodel), a standard for interchange of warehouse metadata, proposed by the Object Management Group [12], is to solve the previous problems. The CWM is a framework for representing metadata about data sources, data targets, transformations and analysis, and the processes and operations that create and manage warehouse data. The CWM uses packages to control complexity and create groupings of logically interrelated meta-classes, organized to minimize package dependencies. There are five top-level packages, summarized in Table 4.

We are only concerned here with the CWM subset that represents object-relational features, which is mainly contained in the *Relational* package, within the *Resource* package. In our opinion this is the most representative analogous effort to the ontology proposed in this paper, although based on a previous version of the SQL standard. In fact, the CWM *Relational* package is based on the SQL:1999 standard section concerning RDBMS catalogs and describes data accessible through a relational interface such as a native RDBMS, ODBC, or JDBC. The *Relational* package uses constructs of the *ObjectModel* package to describe the object extensions added to SQL and, since it also addresses the issues of indexing, primary keys and foreign keys, it

Table 4  
Structure and summary of the 5 top-Level packages of CWM

Package	Description
Object Model	This package provides basic constructs for creating and describing metamodel classes in the other CWM packages. For that purpose, only a subset of the UML metamodel, which mainly includes its structural and instantiation aspects (related to class and object diagrams), is required. The other packages are designed to maximize the reuse of this Object Model, thus sharing modelling constructs whenever possible.
Foundation	This package includes a collection of metamodel sub-packages that contain model elements representing concepts and structures that are specific to the goals and purposes of CWM, providing a common foundation which other packages can extend as necessary to meet their specific needs. Consequently, Foundation model elements often have a more general-purpose nature than model elements found in other packages.
Resource	This data resources package includes a collection of constructors (sub-packages) that allow the representation of relational, record, multidimensional, and XML data resources. Where applicable, key elements of these sub-packages from the same model elements in the Object Model.
Analysis	This package includes a collection of sub-packages that represent data transformations, OLAP (On-line Analytical Processing), data mining, information visualization, and business nomenclature.
Management	This warehouse management package includes a collection of sub-packages that represent warehouse processes and results of warehouse operations.

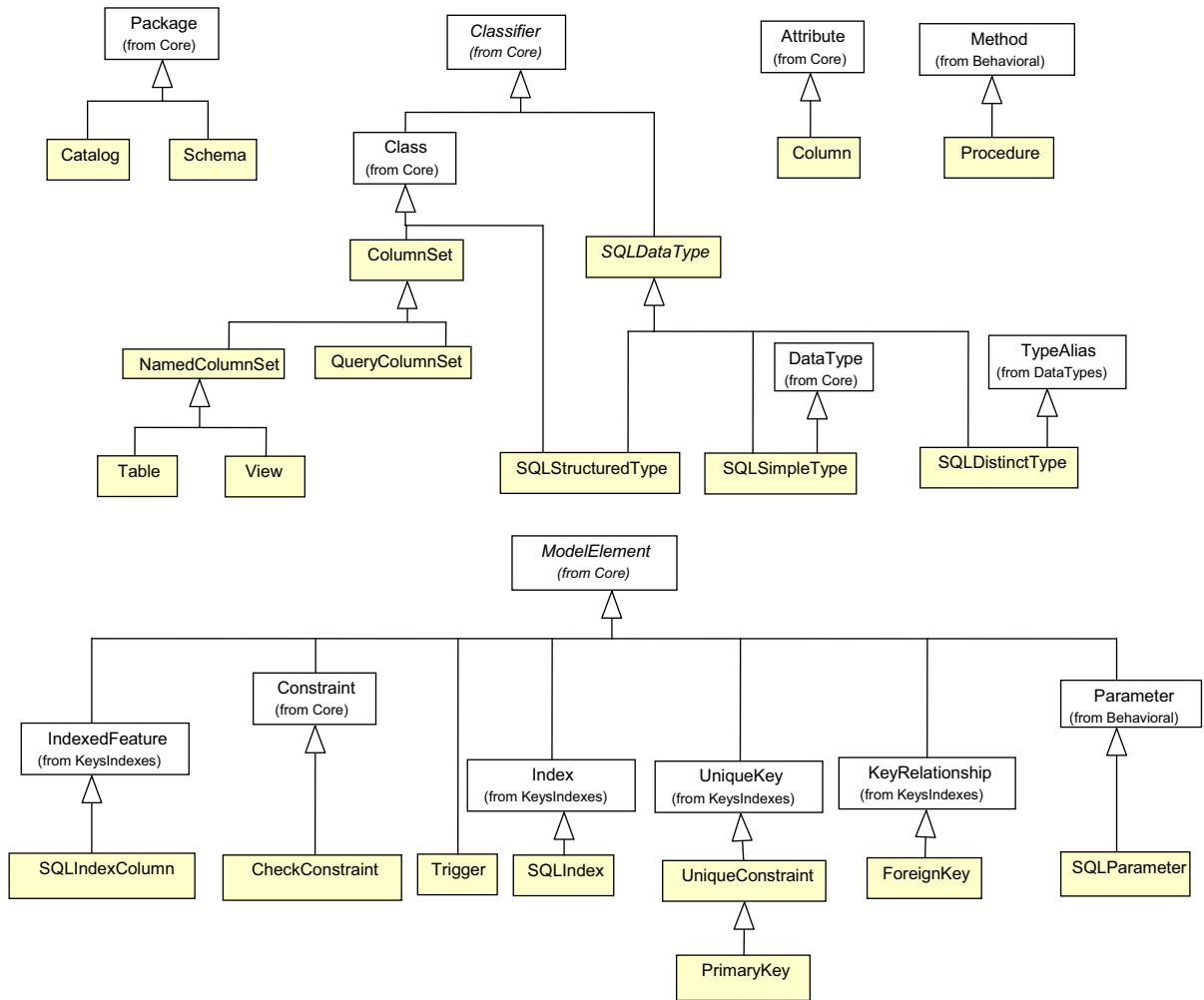


Fig. 7. Meta-class specializations in the CWM relational package.

extends the corresponding concepts from the *Foundation* package.

These dependencies are mostly due to the extension (specialization) of concepts contained in the target packages. A summary of those specializations is represented on Fig. 7. Two top-level containers, *Catalog* and *Schema*, extend the *ObjectModel::Package* meta-class. *ColumnSet* and *SQLStructuredType* extend *Class*. The *Columns* contained in the *ColumnSet* are extensions of the *ObjectModel::Attribute*. The data type of a column (*SQLDataType*) extends the *ObjectModel::Classifier*. Indexes, keys, triggers and constraints are all direct or indirect specializations of the *Core::ModelElement* concept.

## 5.2. CWM vs SQL:2003

The object-relational part of CWM has some inconsistencies with respect to the SQL:2003 standard, and more specifically, with reference to the base tables of the *Definition\_Schema* (part 11). Some of the most representative inconsistencies are the following<sup>5</sup>:

### (a) CWM *ObjectModel* package:

- In the “Core” sub-package, CWM defines “Feature” specialized into “StructuralFeature”

<sup>5</sup> CWM elements are written between quotation marks, while the ontology elements are in italic.

which is also specialized into “Attribute”. Both “Feature” and “Structural Feature” do not exist in SQL. “StructuralFeature” is a generalization of *Column*, *Field* and *Attribute* and “Feature” is a generalization of *Table*, *Row* and *UserDefinedType*.

- In the Behavioral sub-package of CWM, a “Method” is a specialization of “Behavioral-Feature”, which is a specialization of “Feature”. *MethodSpecification* exists in SQL but “BehavioralFeature” does not.

(b) CWM *Foundation package*:

- In the “Keys and Indexes” sub-package, CWM defines “Index” that does not exist in SQL. “UniqueKey” (similar to *UniqueConstraint* in SQL) and “KeyRelationship” (similar to *ReferentialConstraint* in SQL) are indirectly related through “StructuralFeature” and “Indexed-Feature” to “Index”.

(c) CWM *Relational sub-package* within the resource package:

- CWM defines “ColumnSet” specialized into “NamedColumnSet” and “QueryColumnSet”. “NamedColumnSet” is also specialized into “Table” and “View”. This is not concordant with the SQL hierarchy of table types. For instance, *TransientTable* does not exist, or “QueryColumnSet” is not equivalent to *DerivedTable* in SQL because the first does not include views and the second does.
- The data types hierarchies in CWM and SQL are different. In CWM there are three kinds of sub-types (“SQLStructuredType”, “SQLSimpleType” and “SQLDistinctType”) and in SQL there is also the *UserDefinedType* which could not be found within the CWM.
- CWM does not consider the “UniqueKey” (including “UniqueConstraint” and “PrimaryKey”) and the “KeyRelationship” (including “ForeignKey”) as “Constraints” considering only as a “Constraint” the “CheckConstraint”. So the constraint hierarchy of CWM does not fulfill the SQL:2003 requirements.
- In CWM “Procedures” is a kind of “Owned-Element” (equivalent to *SchemaObject*) but this has no equivalence in SQL.

- “Row” is a subtype of “Object” instance (content) in CWM whereas in SQL it is a data type category.
- In CWM a “SQLStructuredType” can only be a column aggregation, a *StructuredType* can be an aggregation of columns (when it is associated to a typed table) or of other user-defined data types.
- CWM establishes that “KeyRelationships” can be (is specialized in) “ColumnSet”, “ForeignKey” or “Column”. This is not in accordance with the SQL:2003 because in the standard neither the columns nor the tables (that supposedly are the “ColumnSet” of CWM) are constraints.
- CWM considers a “column” as a subtype of “attribute”, which is not in line with the concepts presented in SQL:2003.

The previous list of discrepancies can be justified mainly upon two facts. First, CWM is based on SQL:1999 and not on SQL:2003. Consequently, it fails to cover the new features of the SQL:2003 standard. Second, CWM is based on the MOF (Meta-Object Facility), an OMG attempt to produce a general purpose framework for expressing metamodels [29]. The price for this greater generality is the increased complexity of metamodels, because they incorporate concepts that are not “native” to the problem domain (object-relational database schemas, in this case).

## 6. Conclusions and future work

In this paper we have introduced an ontology for increasing the understandability of the SQL:2003 concepts. We have used the SQL:2003 Schemata (part 11 of the standard) in a mapping exercise against the proposed ontology, as a means of validating and refining it. We have used an example from the literature to demonstrate, by instantiation, that the ontology comprises all the relevant concepts. We have compared our proposal with the object-relational part of the CWM. The latter (i) fails to cover some new features of the SQL:2003 standard because it was based on a previous version of the standard and (ii) has some semantic overhead because the metamodel

is “polluted” with concepts imported from the underlying MOF framework.

The version of the ontology presented here covers the object-relational aspects of a database schema. The full coverage of the standard was a too ambitious endeavour for this first version of the ontology. The following parts of the SQL:2003 were left out: part 3 (Call Level Interface), part 9 (Management of External Data), part 10 (Object Language Bindings), part 13 (Routines and Types Using the Java Programming Language) and part 14 (XML-Related Specifications). Many concepts used (but not defined) in those parts are included in the ontology. Following our initial effort to capture the most important and generic features in the ontology, we plan to extend it with other features of the SQL:2003 standard, such as triggers, stored procedures and access-control features.

The availability of this ontology allowed us to start exploring some new research tracks, the first of these being the formalization of a set of complexity mea-

asures for object-relational schemas [30], that were first defined in [31]. That set was formalized using a technique, originally proposed in [32], that uses OCL clauses upon the ontology. We have provisions to define new measures, incorporating the more recent features of the SQL:2003 standard, using the same formalization approach.

### Acknowledgements

The authors thank Prof. Pedro Guerreiro (from FCT/UNL) and the anonymous reviewers for their valuable comments and suggestions.

This work was partially funded by the Fundação para a Ciência e a Tecnologia (FCT / MCES, Portugal: <http://www.fct.mces.pt/>) and by projects ENIGMAS (PBI-05-058) from the Consejería de Educación y Ciencia (JCCM-Spain) and CALIPO (TIC2003-07804-C05-03) from the Ministerio de Educación y Ciencia (Spain).

## Appendix A. SQL:2003 Code Example Based on Melton and Simon [26,27]

```

CREATE SCHEMA video_and_music
AUTHORIZATION m_s_enterprises
DEFAULT CHARACTER SET "Latin_1"

CREATE DOMAIN price DECIMAL (7,2)
CHECK (VALUE IS NOT 0);

CREATE DISTINCT TYPE money AS DECIMAL (9,2);

CREATE TYPE movie AS(
movie_id INTEGER,
title CHARACTER VARYING (100),
languages MULTISET ['English', 'French', 'Spanish',
                    'Portuguese', 'Italian'],
genre CHARACTER VARYING (20) ARRAY [10],
run_time INTEGER
)
INSTANTIABLE
NOT FINAL

METHOD length_interval ()
RETURNS INTERVAL HOUR (2) TO MINUTE

CREATE INSTANCE METHOD length_interval ()
RETURNS INTERVAL HOUR (2) TO MINUTE FOR MOVIE
RETURN CAST (CAST (SELF.run_time AS INTERVAL (4) )
AS INTERVAL HOUR (2) TO MINUTE);

CREATE TABLE movies (
stock_number CHARACTER(10)
CONSTRAINT movies_stock_number_not_null NOT
NULL,
movie movie,
our_tape_cost price,
tapes_in_stock INTEGER
CONSTRAINT movies_primary_key PRIMARY KEY
(stock_number)
);

CREATE TABLE movie_stars (
movie_title CHARACTER (30)
CONSTRAINT movies_stars_movie_title_not_null NOT
NULL,
movie_year_released DATE,
movie_number CHARACTER (10),
actor_last_name CHARACTER (35)
CONSTRAINT movies_stars_actor_last_name_not_null
NOT NULL,
actor_first_name CHARACTER (25)
CONSTRAINT movies_stars_actor_first_name_not_null
NOT NULL,
actor_middle_name CHARACTER (25),
CONSTRAINT movies_stars_unique
UNIQUE (movie_title, actor_last_name, actor_first_name,
actor_middle_name)
NOT DEFERRABLE,
CONSTRAINT movies_stars_fk_movie
FOREIGN KEY (movie_number)
REFERENCES movies (stock_number)
ON DELETE CASCADE
ON UPDATE CASCADE
);

CREATE TYPE music_distributors_st AS (
distributor_id CHARACTER (15),
distributor_name CHARACTER (25)
);

```

```

CREATE TABLE music_distributors OF
music_distributors_st (
REF IS dist_ref SYSTEM GENERATED,
distributor_id WITH OPTIONS
CONSTRAINT
music_distributors_distributor_id_not_null
NOT NULL,
distributor_name WITH OPTIONS
CONSTRAINT
music_distributors_distributor_name_not_null
NOT NULL,
);

CREATE TYPE address AS(
street CHARACTER VARYING (35),
city CHARACTER VARYING (40),
country CHARACTER (3)
);

CREATE TYPE US_address UNDER address AS(
state CHARACTER (2),
zip ROW (
Basic INTEGER,
Plus4 SMALLINT)
)

METHOD zipcode ()
RETURNS CHARACTER VARYING (10);

CREATE INSTANCE METHOD zipcode ()
RETURNS CHARACTER VARYING (10)
FOR US_address
BEGIN
IF SELF.zip.plus4 IS NULL
THEN RETURN CAST (SELF.zip.basic AS
CHARACTERVARYING (5));
ELSE RETURN CAST (SELF.zip.basic AS
CHARACTER VARYING (5))
|| '-' || CAST (SELF.zip.basic AS
CHARACTER VARYING (4))
ENDIF;
END;

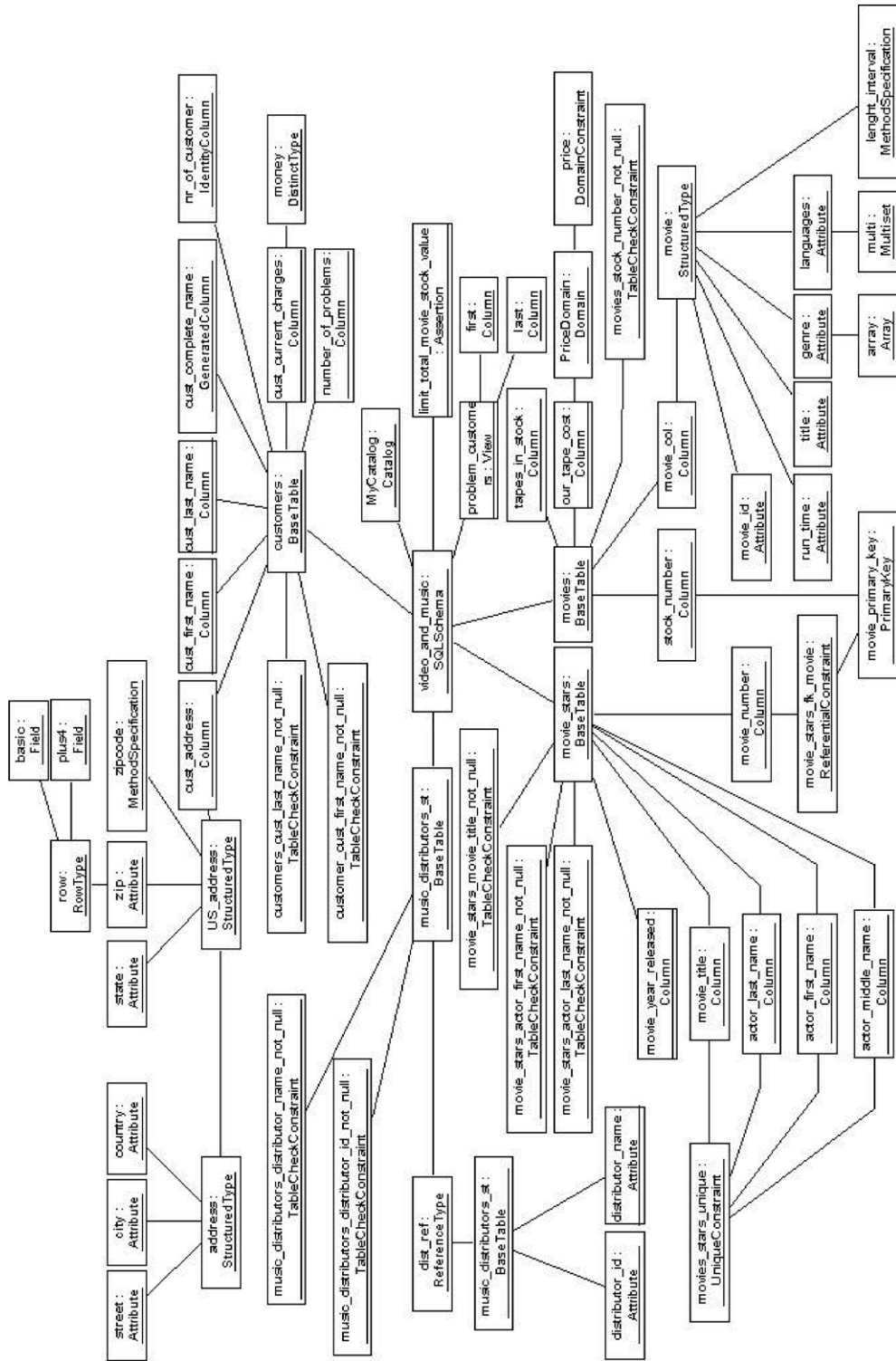
CREATE TABLE customers(
nr_of_customer INTEGER
GENERATED ALWAYS AS IDENTITY
(START WITH 1 INCREMENTED BY 1 MINVALUE 1),
cust_last_name CHARACTER (35)
CONSTRAINT customers_cust_last_name_not_null
NOT NULL,
cust_first_name CHARACTER (35)
CONSTRAINT customers_cust_first_name_not_null
NOT NULL,
cust_complete_name GENERATED ALWAYS AS
(cust_first_name || cust_last_name),
cust_address US_address,
cust_current_charges mone y,
number_of_problems SMALLINT );

CREATE VIEW problem_customers ( last, first)
AS
SELECT cust_last_name, cust_first_name
FROM customers
WHERE number_of_problems >
0.8 * (SELECT MAX(number_of_problems)
FROM customers);

CREATE ASSERTION limit_total_movie_stock_value
CHECK ( ( SELECT COUNT(*)
FROM customers
WHERE number_of_problems > 5
AND cust_current_charges > 150,00
AND cust_current_charges < 1000,00) <10 );

```

Appendix B. Ontology Instantiation Example (UML Object Diagram)



## References

- [1] IDC, Worldwide Relational Database Management Systems 2005–2009 Forecast: An Initial View. Electronic document, Available electronically at: <http://www.idc.co/getdoc.jsp?containerID=33078> (accessed in July, 2005), 2005.
- [2] ISO, Database Language SQL, ISO/IEC 9075 International Standard, 1987.
- [3] ISO, Database Language SQL with Integrity Enhancement. ISO/IEC 9075 International Standard, 1989.
- [4] ISO, Database Language SQL, ISO/IEC 9075 International Standard, 1992.
- [5] ISO, SQL/CLI-Call Level Interface, ISO/IEC 9075-3: 1995 International Standard, 1995.
- [6] ISO, SQL/PSM-Persistent Storage Module, ISO/IEC 9075-4: 1996 International Standard, 1996.
- [7] ISO, Information Technology – Database languages – SQL: Parts 1 to 5. ISO/IEC 9075-1:1999 to 9075-5:1999 International Standards, 1999.
- [8] ISO, Information Technology – Database languages – SQL: Parts 1 to 4 and 9 to 14. ISO/IEC 9075-1:2003 to 9075-14:2003 International Standards, 2003.
- [9] W. Pidcock, What are the differences between a vocabulary, a taxonomy, a thesaurus, an ontology and a meta-model? Electronic Document, 2003 Available electronically at: <http://www.metamodel.com/article.php?story=20030115211223271>, accessed in July, 2005.
- [10] OMG, UML 2.0 Superstructure Specification, October, Object Management Group Specification, 2004, Available electronically at: <http://www.omg.org/cgi-bin/doc?ptc/2004-10-02> accessed in July, 2005.
- [11] OMG, UML 2.0 OCL Specification, October, Object Management Group Specification, 2004, Available electronically at: <http://www.omg.org/cgi-bin/doc?ptc/2003-10-14> accessed in July, 2005.
- [12] OMG, Common Warehouse Metamodel (CWM) Specification. Version 1.1, March, Object Management Group Specification, 2003, Available electronically at: <http://www.omg.org/technology/documents/formal/cwm.htm> accessed in July, 2005.
- [13] ANSI, Standard relational language, ANSI.X3 Standard, 1982.
- [14] ANSI, Structured query language, ANSI.X3 Standard, 1986.
- [15] C. Date, Edgar F. Codd. A Tribute and Personal Memoir, ACM SIGMOD Record, vol. 32 (4), 2003, pp. 4–13.
- [16] ANSI, Database language embedded SQL, ANSI.X3.168 Standard, 1989.
- [17] T. Gruber, Towards principles for the design of ontologies used for knowledge sharing, International Journal of Human-Computer Studies 43 (5/6) (1995) 907–928.
- [18] W3C, Web ontology language. Technology and society domain, W3C Specification, 2004, Available electronically at: <http://www.w3.org/2004/OWL/> accessed in July, 2005.
- [19] J. Rumbaugh, I. Jacobson, G. Booch, The Unified Modeling Language Reference Manual, 2nd edition, Addison-Wesley Professional, ISBN: 0321245628, 2004.
- [20] X. Wang, C.W. Chan, Ontology modeling using UML, 7th International Conference on Object Oriented Information Systems Conference (OOIS'2001), Springer, Calgary, Canada, 2001, pp. 59–68.
- [21] A. Gomez-Perez, O. Corcho, M. Fernandez-Lopez, Ontological Engineering: With Examples from the Areas of Knowledge Management, e-Commerce and the Semantic Web, Springer, ISBN: 1852335513, 2004.
- [22] F. Ruiz, A. Vizcaíno, M. Piattini, F. García, An ontology for the management of software maintenance projects, International Journal of Software Engineering and Knowledge Engineering 14 (3) (2004) 323–349.
- [23] J. Warmer, A. Kleppe, The Object Constraint Language: Getting Your Models Ready for MDA, 2nd edition, Addison-Wesley Professional, ISBN: 0321179366, 2003.
- [24] PlanetMDE, MDE (Model Driven Engineering), Community Portal, 2005, Available electronically at: <http://www.planetmde.org> accessed in July, 2005.
- [25] R. Kishore, H. Zhang, R. Ramesh, A Helix-Spindle Model for Ontological Engineering, Communications of the ACM 47 (2) (2004) 69–75.
- [26] J. Melton, A.R. Simon, SQL: 1999. Understanding Relational Language Components. ISBN 1-55860-456-1. Morgan Kaufmann Publishers, Academic Press, USA, 2002.
- [27] J. Melton, SQL:1999-Understanding Object-Relational and Other Advanced Features, ISBN 1-55860-677-7, Morgan Kaufmann Publishers. Academic Press. USA, 2003.
- [28] OMG, Revised submission for MOF 2.0 Query/View/Transformation RFP, QVT-merge group, Object Management Group Specification, 2005 Available electronically at: <http://omg.org/docs/ad/05-03-02.pdf> accessed in July, 2005.
- [29] OMG, Meta-Object Facility (MOF) core specification, Version 2.0, October, Object Management Group Specification, 2003 Available electronically at: <http://www.omg.org/cgi-bin/doc?ptc/2003-10-04>, accessed in July, 2005.
- [30] A. Baroni, C. Calero, F. Ruiz, F. Brito e Abreu, Formalizing object-relational database metrics, Proceedings of the 5th Conference of Portuguese Association of Information Systems (CAPSI), Associação Portuguesa de Sistemas de Informação, Lisbon, 2004, 3–5 November 2004.
- [31] C. Calero, M. Piattini, M. Genero, Defining Complexity Metrics for Object-Relational Databases, Proceedings of the Object-Oriented Information Systems Conference (OOIS 2000), London, 18–21 December, Springer, ISBN: 1-85233-420-7, 2000, pp. 391–400.
- [32] F. Brito e Abreu, Using OCL to formalize object-oriented metrics definitions, Technical Report ES007/2001, Software Engineering Group, INESC, 2001, June 2001. Available electronically at: <http://ctp.di.fct.unl.pt/QUASAR/>.

**Coral Calero** has a PhD in Computer Science (University of Castilla–La Mancha, Spain) and an MSc in Computer Science (University of Seville, Spain). Aside from that, she is an Associate Professor in the Computer Science Department at the University of Castilla–La Mancha (UCLM). She is a coeditor/coauthor of several international books, among others: *Metrics for software conceptual models*, 2005, Imperial College Press, ISBN 1-86094-497-3, *Auditing Information Systems*, 2000, Idea Group Publishing, USA, ISBN 1-878289-75-6; *Advanced Databases Technology and Design*, 2000, Artech House, USA, ISBN 0-89006-395-8; *Information and Database quality*, 2002, Kluwer Academic Publishers, Norwell, USA, ISBN 0-7923-7599-8. Her research interests are as follows: software quality metrics, quality models, web and portal quality, databases and data warehouse quality.

**Francisco Ruiz** has a PhD in Computer Science (University of Castilla–La Mancha, Spain) and an MSc in Chemistry–Physics (University Complutense of Madrid). He is also associate professor in the Computer Science Department at the University of Castilla–La Mancha (UCLM) and Vice-Director of the Alarcos Research Group. He has been Dean of the UCLM Computer Science School for 7 years and Data Processing Director at the same University for 4 years. He has also worked for several companies as an analyst-programmer and a project manager. He is a coeditor/coauthor of several books, among others: *Metrics for Software Conceptual Models* (2004, Imperial College Press, UK), *Advances in Software Maintenance Management: Technologies and Solutions* (2002, Idea Group, USA), *The Guide to IT Service Management* (2002, Addison-Wesley, UK), *Information Modeling in the New Millennium* (2001, Idea Group, USA), *Advanced Databases Technology and Design* (2000, Artech House, UK), *Auditing Information Systems* (2000, Idea Group, USA), *World Class IT Service Management* (2000, Ten Hagen & Stam P., Netherlands), etc. His current research interests include business process management systems, software process technology and modeling, software maintenance, and software projects planning and managing.

**Aline Baroni** has an MSc in Computer Science granted by the Vrije Universiteit Brussel, Belgium and École des Mines de Nantes, France. Currently, she is doing her PhD, whose topic is the quantitative evaluation of UML behavioural models. She is also an

assistant professor in Brazil, where she has taught several disciplines concerning software engineering and object-orientation. Moreover, she is author and co-author of several papers and she has worked within several academic and industry projects around the world. Her research interests are mainly in the areas of software metrics, software quality, software evolution and reuse, formal methods, databases quality and human computer interfaces, and software refactorings.

**Fernando Brito e Abreu** holds a PhD on Computer Science (IST/UTL) and is a professor at the Computer Science Department of the FCT/UNL (Portugal). He leads the QUASAR research team and the Portuguese IT&T Quality Commission (CS03) of the National Council for Quality (CNQ). He is a member of the EOQ Software Group and of the editorial board of the *Software Quality Professional* journal (ASQ). He has co-organized and chaired several scientific events worldwide and is author or co-author of many papers on diverse topics such as resource estimation, software evolution, complexity metrics, modularity reengineering and refactoring, evaluation of OR schemas, CBD assessment or AOD meta-modeling. He is the creator of the MOOD metrics set and of the Metamodel-Driven Measurement (M2DM) approach.

**Mario Piattini** has an MSc and a PhD in Computer Science (Polytechnical University of Madrid) and an MSc in Psychology (UNED.). He is also a Certified Information System Auditor and a Certified information System Manager by ISACA (Information System Audit and Control Association) as well as a Full Professor in the Department of Computer Science at the University of Castilla–La Mancha, in Ciudad Real, Spain. Furthermore, he is the author of several books and papers on databases, software engineering and information systems. He is a coeditor of several international books: *Advanced Databases Technology and Design*, 2000, Artech House, UK; *Information and Database Quality*, 2002, Kluwer Academic Publishers, Norwell, USA; *Component-based Software Quality: Methods and Techniques*, 2004, Springer, Germany; *Conceptual Software Metrics*, Imperial College Press, UK, 2005. He leads the ALARCOS research group of the Department of Computer Science at the University of Castilla–La Mancha, in Ciudad Real, Spain. His research interests are advanced databases, database quality, software metrics, security and audit, and software maintenance.