Encyclopedia of Business Analytics and Optimization, J. Wang (Editor), IGI-Global, 2014, 512-522.

Concept-Oriented Query Language

Alexandr Savinov, *Technische Universität Dresden, Germany*

INTRODUCTION

With the explosion of data volume and the variety of data sources (Cohen et al., 2009) – two aspects of the *big data* problem - we observe quite significant difficulties in applying conventional *data analysis* methodologies to real world problems. The existing technologies for data management and analytics were pushed to the limits of their ability to solve more and more complex analysis tasks:

- **Agile analytics.** Perhaps the most widely used methodology for data analysis during several decades is based on the multidimensional metaphor where data is viewed as existing in a multidimensional space. A problem of this approach is that it is based on application-specific scenarios with predefined roles of dimensions, measures, cubes and facts. Changing such scenarios is a quite difficult task because they are embedded in both database systems and client software. The goal of agile analytics consists in going beyond standard OLAP analysis by facilitating exploratory ad-hoc analytics where the user can freely vary all data processing and visualization parameters.
- **Self-service analytics.** The conventional approach to analysis is to approach IT department which however has several drawbacks: business frequently does not trust data provided by IT, IT is unable to understand the needs of the user (and this leads to frustration and low motivation), IT might not be able to respond to user requests as quickly as is desirable (and the requirements may well change during the response time), existing BI tools are not intended for non-professional users. Self-service analytics is one of the most significant trends in the BI industry over the last few years and these tools aim to give non-professional users the ability to solve analytical tasks with little or no help from IT.
- **Near real time analytics.** It may take days to generate a BI report in a typical enterprise system and there is strong demand in reducing the time between data acquisition and making a business decision. One of the core problems is that traditional systems are based on two separate technology stacks: for transactional workload and for analytical workload. The design principles and techniques of these two subsystems are quite different and they cannot provide the necessary response time and agility of decision making on large volumes of data (Chaudhuri, Dayal, & Narasayya, 2011; Thiele & Lehner 2012). Although modern hardware provides a basis for a new generation of in-memory, columnar databases (Boncz, 2012; Larson, 2013) with potentially higher query performance on analytical workloads, it is important to understand that real time analytics is not a hardware problem - new data models, new query languages, new analysis scenarios, new analysis algorithms are needed.
- **Semantic analysis.** The conventional approach is that it is the task of the human analyst to understand the meaning of data while the system has to only execute precise queries. However, a typical enterprise system can contain tens of thousands data tables and open systems can involve numerous external data sources. In this situation it is extremely difficult to get meaningful results manually. Existing solutions add semantics via a separate layer which is based on quite different data modeling and analysis techniques. This leads to complex mappings and translations at all levels of the system architecture.
- **Reasoning about data.** The goal of this type of analysis is to answer questions by automatically deriving them from the available data. This task has been a prerogative of the systems based on formal logic which have several drawbacks: formal logic is not natural for expressing analysis tasks, formal logic is not very suitable for numeric analysis, formal logic requires a separate system because it is not directly compatible with available data storage, queries in formal logic are computationally expensive.
- **Analytical computations.** Analysis is not limited by the operations of grouping and aggregation. Now analysts need to embed arbitrary computations in their analysis tasks. Such tasks are

normally expressed as batch jobs where data is exported from one or many databases and then processed using an analysis program. Executing arbitrary analysis tasks close to the data (ideally directly where data resides) is still a big problem. It is actually a new incarnation of the old problem of incompatibility between programming and data modeling (impedance mismatch) because data is modeled and manipulated differently in programming languages and databases.

These fundamental challenges require a principled solution rather than yet another specific technique. Many of the above problems can be solved at the level of a unified data model which should be general enough to cover major analytical patterns of thought, and at the same time should it be simple and natural. In this article we describe a novel query language, called the concept-oriented query language (COQL), which addresses the above issues and is aimed at radically *simplifying* typical data analysis and data modeling tasks. COQL is a syntactic description of the concept-oriented model (COM) (Savinov, 2009, 2011) and it has the following distinguishing features:

- COQL replaces joins as a means of connectivity by a novel *arrow notation* which can be viewed as a set-oriented analog of dot notation
- COQL replaces group-by operation by a novel operation of de-projection
- COQL introduces a novel mechanism of inference based on the multidimensional structure of data instead of using logical inference
- COQL inherently supports dimensions as a basic construct rather than treating them as something optional that is added for specific kinds of analysis
- COM and COQL support several data modeling and analysis paradigms (relational, multidimensional, entity-relationship, semantic and conceptual, object-oriented) by resolving many incompatibilities and controversies as well as increasing semantic integrity of data models and analysis tasks
- COQL relies on a novel data typing construct, called concept, and two relations: inclusion and partial order.

BACKGROUND

A language reflects main principles of the underlying model or paradigm using syntactic constructs and rules. It is valid for programming languages, query language, conceptual languages and for other areas where a theory can be described syntactically. Below we describe major language categories used for data querying with the focus on data analysis, connectivity and set operations.

Join-based languages. It is probably the most wide spread class of query languages which rely on the join operation as connectivity means. The most wide spread version of this class is SQL query language developed within the relational model of data (Codd, 1970). Yet, it is important to understand that join is a variant of a more general approach used in formal logic and consisting in binding free variables. This method can be also characterized as "common value" approach: two things are supposed to be related if they share some common value. It has numerous advantages but probably the most important one in the context of data management is that join is an operation on sets. It also has quite significant drawbacks (Savinov, 2012a): join is a low level and error-prone operation which requires high expertise (Atzeni, 2013), joins can easily produce meaningless results because it lacks semantics, join exposes the mechanics of connectivity at the level of business logic and is a typical cross-cutting concern, join is not analytics-friendly.

Deductive query language. These languages are designed to be as close as possible to the underlying formalism (normally first-order predicate logic) and their goal is to provide a full-featured mechanism of inference. They were traditionally used in deductive databases (Ullman & Zaniolo, 1990) but currently an interest to these languages grows with the development of such technologies as Semantic Web and Linked Open Data which are based on Description Logics. This approach has the following drawbacks: it is not very intuitive and requires high expertise, it lacks structure because all predicates and sets have the same level, it is not very suitable for multidimensional and numeric analysis.

Graph-based languages. These languages (Wood, 2012) belong to the class of navigational languages because their connectivity mechanism is based on the notion of path in some structure.

These paths can be followed in different directions and used for retrieving graph nodes and edges. The most common operations in graph databases is graph traversal which plays approximately the same role as joins. These languages have always been very natural and very easy to use but they are not very compatible with the dominating relational model, provide limited structuring mechanisms, have limited semantics and not directly designed for numerical and multidimensional analysis.

Object-oriented query languages. The main focus of this approach (Dittrich, 1986; Atkinson et al., 1990) is to support programming models and its development was driven by the need to decrease or even eliminate the differences between data modeling and programming. Object-oriented query languages rely on class descriptions and dot notation for navigation purposes. However they are more instance-oriented and provide weaker support for set operations in comparison to other models. In particular, very natural and convenient dot notation is not suitable for set operations.

Multidimensional query languages. Languages like MDX are used in the context of standard OLAP models (Li & Wang, 1996; Pedersen & Jensen, 2001) for solving analytical tasks. This approach is based on the notions of dimension, measure, facts and cube. It is intended for numerical and multidimensional analysis but not directly compatible with other paradigms. Therefore, data management is normally broken into two areas: managing transactional data (relational database) and managing analytical data (data warehouse).

Other analytical query languages. There are several new empirical approaches to self-service agile analytics implemented in software products. One of them is a novel Tabular data model and its native query language, called Data Analysis eXpressions (DAX). These technologies are part of the Business Intelligence Semantic Model (BISM) implemented in Microsoft SQL Server 2012 Analysis Services (Russo, Ferrari, & Webb, 2012) and supported in some client products like PowerPivot. The goal of this approach is to provide an analytical model for all user experiences by significantly simplifying typical analysis tasks and facilitating self-service analytics. This technology eliminates the need in and limitations of the conventional multidimensional models by providing an expression language for arbitrary computations over available data. There exist also quite many empirical query languages developed within NoSQL approach (Mohan, 2013). However, they are normally either too specific implementations based on the features of the underlying database or adaptations of other approaches.

CONCEPTS AND INCLUSION RELATION

Concepts

COM introduces a novel type modeling construct, called *concept*. Concept is defined as a couple of two classes: one *identity class* and one *entity class*. The main difference between them is that instances of an identity class are passed by-value and instances of an entity class are passed by-reference. Passing an entity by-reference means that the coupled identity is copied instead of the entity. There are two particular cases of concepts which produce traditional typing constructs:

- if entity class is empty then concept describes values (identity instances are values)
- if identity class is empty then it is equivalent to traditional classes instances of which are objects represented by primitive references

For example, if colors are represented as values then they are described by a concept with the empty identity class:

```
CONCEPT ColorValue //Instances are values
  IDENTITY
    INTEGER red, green, blue
  ENTITY // Empty
```

If colors have to be passed by-reference (in order to be shared among many other elements) then their fields are described in the entity class:

```
CONCEPT ColorObject //Instances are objects
  IDENTITY // Empty
  ENTITY
    INTEGER red, green, blue
```

In the general case, both constituents are not empty so that elements may have arbitrary domain-specific identity and arbitrary entity structure. For example, existing data models do not allow us to define colors as elements identified by name (passed by-value) and having three constituents passed by-reference:

```
CONCEPT Color
  IDENTITY
    CHAR(10) name
  ENTITY
    INTEGER red, green, blue
```

Once a concept has been defined, it can be used as a type of variables, fields, parameters and other (typed) elements. These elements will store a value in the format of the identity class and provide access to the values stored in the entity. Concepts are also used to define sets of elements. A set in COQL is written as some concept name in parentheses. For example, a set of colors is written as `(Color)`. A subset is specified by providing a constraint on its elements which is written as a logical expression separated by bar symbol. For example, the following set contains only red colors: `(Color | green=0 && blue=0)`.

One of the most important benefits of the concept-oriented approach is that it generalizes and simplifies the object-relational model by unifying domain modeling and relation modeling. In COM, concepts define only one kind of sets where elements (concept instances) are identity-entity *couples* while existing approaches use two different kinds of sets: domains consisting of values and relations consisting of tuples. Note also that identities are different from the mechanism of (primary) keys. Identities can be viewed as surrogates with arbitrary domain-specific structure while keys are entity attributes with a special role. A concept can be thought of as a description of computer memory where user-defined addresses are used to access user-defined cells. Concepts are also informally analogous to complex numbers in mathematics which also have two constituents but are manipulated as one whole.

Inclusion

COM provides a novel relation, called *inclusion*, which is applied to concepts and is used instead of the classical inheritance. Any concept (except for a root) is included in some other concept so that all concepts exist in a hierarchy. A parent concept in the inclusion hierarchy is referred to as a *super-concept* and child concepts are referred to as *sub-concepts*. Inclusion relation among concepts is interpreted as IS-IN relation which means that many instances of a child concept are contained within one instance of the parent concept.

Elements are still identified by their identities but now these identities extend parent identities which in turn extend their parent identity and so on up to the root which represents the whole space of elements. Inclusion in COM is similar to the hierarchical XML structure where parent elements contain child elements directly by-value. A fully-specified identity of an element is a value which consists of several segments starting from the root and ending with the identity of this element. Thus inclusion relation allows us to model containment with hierarchical address space. For example, if bank accounts are identified with respect to their bank (one bank has many accounts) then concept `Account` has to be included in the `Bank` concept:

```
CONCEPT Account IN Bank // Exists in bank
  IDENTITY
    CHAR(10) accNo // Extends identity of Bank
  ENTITY
    DOUBLE balance
```

If entity class is empty then inclusion is equivalent to value extension and can be used to define more specific value types. If identity class is empty then inclusion is equivalent to classical inheritance where additional fields can be added to an existing entity. When applied to relations, inclusion provides a mechanism for their extension by defining more specific relations on the basis of more general ones.

The use of concept inclusion significantly diminishes the differences between relational and object-oriented modeling because concepts provide a common mechanism for modeling simultaneously value domains and relation types. In the general case, both concepts and their instances exist in a hierarchy where parents are shared parts of children (like in prototype-based programming). For example, a bank has many accounts and an account might have many sub-accounts like savings accounts. Thus COM inclusion eliminates the asymmetry between classes and instances so that a concept hierarchy directly models instance hierarchies. Note also that IS-A relation in COM is a particular case of IS-IN relation which means that to be included in some set means to inherit the properties and behavior of this set (members of a set are more specific elements than the set they are in).

DATA MANIPULATION AND ANALYSIS

Logical Navigation and Arrow Notation

COQL uses *arrow notation* (Savinov, 2012a) for navigating through the data structure and retrieving sets of elements. Arrow notation can be viewed as a set-oriented analogue of the conventional dot notation. If dot operation is applied to instances then arrow operations are applied to sets. Another difference is that arrow notation provides two versions: for moving along attributes and for moving in the opposite direction against attributes. Since all concepts and elements in COM are partially ordered, this is interpreted as moving up or down in the partially ordered set of elements. Also, they are interpreted as moving between layers of detail.

The operation of *projection*, denoted by right arrow, is applied to a set of elements and returns a subset of elements which are referenced from the source elements by the specified attribute. For example (Fig. 1), given a set of books we can find all their publishers by projecting them along the publisher attribute:

```
(Book | date > '01.01.2005')
  -> publisher -> (Publisher)
```

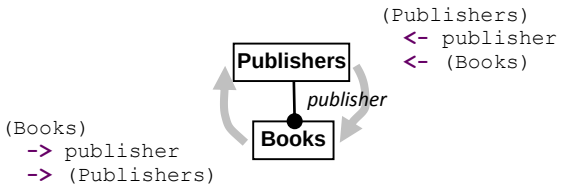

Figure 1. Projection and de-projection.

De-projection is the opposite operation denoted by left arrow. It returns a set of elements from the target set which reference the elements of the source set. For example (Fig. 1), given a set of publishers we can de-project them and find all the books they published:

```
(Publisher | name = 'XYZ')
  <- publisher <- (Book)
```

These operations serve for logical navigation in COQL (Savinov, 2006a). The general idea is that constraints are imposed in some part of the schema and then propagated to another part of the schema using a zig-zag attribute path composed of projections and de-projections. For example (Fig. 2), we could easily find all writers of a publisher by applying two de-projections followed by projection:

```
(Publisher | name = 'XYZ')
    <- publisher <- (Book)
    <- book <- (BookWriter)
    -> writer -> (Writer)
```

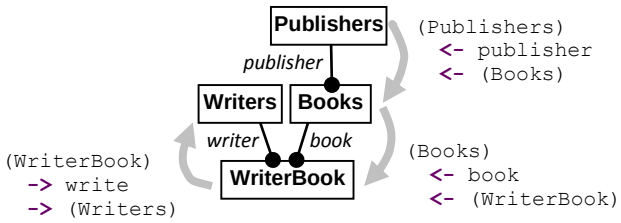

Figure 2. Logical navigation.

Taking into account semantic interpretations of partial order in COM, projection means getting all coordinates of the source points, finding all more general elements than the source elements, and all containers for the source elements. De-projection is interpreted as getting all points having the source coordinates, finding all more specific elements for the source elements, and all members of the source containers.

One of the main benefits of these two operations is that they eliminate the need in join and group-by operations. Joins are not needed because sets are connected using multidimensional hierarchical structure of the model. Group-by is not needed because any element is interpreted as a group consisting of its lesser elements. Given an element (group) we can get its members by applying de-projection operation. For example, if it is necessary to select only publishers with more than 10 books then it can be done it as follows:

```
(Publishers |
  COUNT(publisher <- (Books)) > 10)
```

Here de-projection `publisher <- (Books)` returns a group of books of this publisher and then their count is compared with 10.

Constraint Propagation and Inference

An important application of projection and de-projection operations consists in propagating arbitrary constraints through the model structure. Constraints are specified as a source set of elements. The propagation path is specified as a sequence of attributes and intermediate sets leading to the target set the elements of which have to be retrieved. In many cases intermediate attributes and sets can be omitted and then the system will reconstruct the propagation path. Such projection and de-projection with an undefined dimension path is denoted by `'*->'` and `'<-*'` (with star symbol interpreted as *any* dimension path). For example (Fig. 3), given a collection of `Books` we can find all related `Addresses` using the following projection query:

```
(Books | price < 10) *-> (Addresses)
```

This query is translated into the following query where the propagation path is written explicitly:

```
(Books | price < 10)
  -> publisher -> (Publishers)
  -> address -> (Addresses)
```

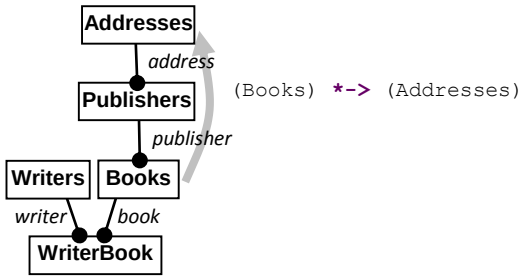

Figure 3. Constraint propagation.

This query can be reversed and then it will return all `Books` related to the selected `Addresses`:

```
(Addresses | country == 'DE') <-* (Books)
```

The above queries are examples of propagating constraints in only one direction and they are restricted versions of inference. In the general case, source and target sets may have arbitrary locations in the schema and then this approach does not work because they are not connected by a projection or de-projection path. To solve this problem of propagating constraints between arbitrary sets, COQL propose a special solution which is based on *inference operator* denoted as `'<-*->'` (de-projection step followed by projection step via an arbitrary dimension path). It connects two sets from the database and finds elements of the second set which are related to the first one. This approach assumes that source and target sets have some common lesser set which is treated as a relationship between them (or a fact set in OLAP terms). Constraints imposed on the source set are de-projected down to this lesser set by selecting a subset of its elements. And then the selected elements are projected up to the target set.

For example (Fig. 4), given a set of young writers we can easily find related countries by using only one operator:

```
(Writers | age < 30)
  <-*-> (Addresses) -> countries
```

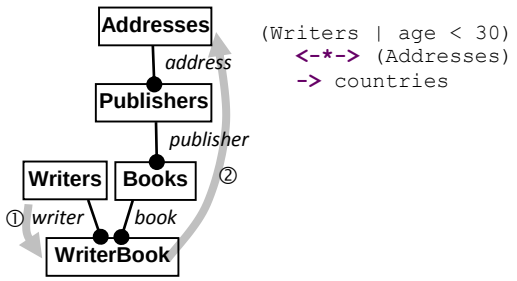

Figure 4. Inference.

To answer this query, the system first chooses a common lesser collection, `WriterBooks` in this example, and then transforms this query to two operations of de-projection and projection:

```
(Writers | age < 30)
  <-* (WriterBooks) // De-project
  *-> (Addresses) -> countries // Project
```

After that, the system reconstructs the complete constraint propagation path:

```
(Writers | age < 30)
  <- writer <- (WriterBooks)
  -> book -> (Books)
  -> publisher -> (Publishers)
  -> address -> (Addresses) -> countries
```

If there are no common lesser sets or there are more than one such set then it is possible to provide more information in the query that the system can use for inference as described in (Savinov, 2006b, 2012b).

Product Operation for Multidimensional Analysis

Navigational operations of projection and de-projection are intended for retrieving data that already exists in the database. However, in many cases it is necessary to generate new data elements. This task is supported by the product operation which returns all combinations of elements from source sets specified as parameters. In COQL, product is written as a number of sets enclosed in parentheses and optionally prefixed by the `CUBE` or `PRODUCT` keyword. For example (Fig. 5), given two source collections with countries and product categories we can produce a 2-dimensional set where every element is a combination of one country and one produce category:

```
ResultCube =
  CUBE (Countries, Categories)
```

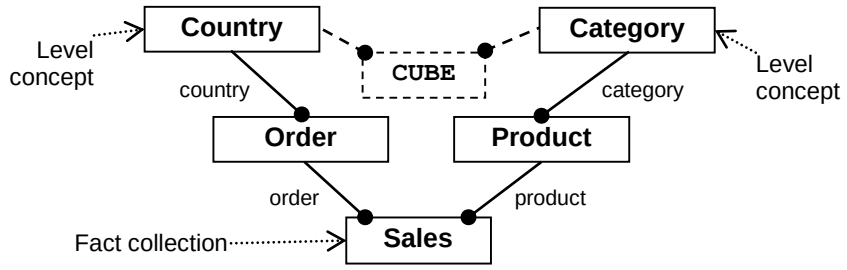

Figure 5. Multidimensional analysis.

The main application of the product operation is multidimensional analysis where the task is to group facts over cells of the cube and then compute some numeric aggregated characteristic (measure) for this group (Savinov, 2005). Grouping is performed by de-projecting a cell down to the facts. This group is then passed to an aggregation parameter and all these operations are performed for each element of the cube in the BODY block. For example, sales over countries and categories can be computed using the following query:

```
CUBE(Countries co, Categories ca)
BODY {
  cell = co <-* Sales AND
         ca <-* Sales
  measure = SUM(cell.amount)
}
RETURN co.name, ca.name, measure
```

The group of sale facts belonging to one cell (one country and one category) is computed by de-projecting the current country and category down to the `Sales` fact collection and then finding their intersection (denoted by AND). Then all sales are summed up within one cell of the cube using only one numeric dimension for aggregation. Finally, the value computed for this country and this category is returned in the result.

FUTURE RESEARCH DIRECTIONS

Currently there is strong demand in more expressive query languages providing not only some built-in analytical operators but rather inherently supporting analytics as their primary goal. COQL follows this long term goal by trying to revisit how analytics is done and by radically simplifying typical analytical and other data management tasks. Such a next generation query language should be much closer to traditional programming languages which allow for implementing arbitrary business logic and arbitrary data manipulations. Such a unification of programming and query languages could be done on the basis of concept-oriented programming (Savinov, 2012c) which is also based on the same principles as COQL.

Another direction for future research consists in using COQL for conceptual modeling. Here the challenge is to eliminate differences between logical and conceptual levels of modeling. COQL already provides significant support for modeling relationships and in future more mechanisms and patterns for conceptual modeling should be integrated into it. Representing data semantics as well as reasoning about data are also considered important directions for future research.

CONCLUSION

The concept-oriented model and query language is a step towards developing a unified approach to data management and analytics which provides equal support for transactional, analytical and conceptual views on data as well as addresses other issues: agile and self-service analytics, semantics and reasoning, near-real-time analytics and analytical computations close to the data. COM and COQL take a holistic view on data by unifying a wide range of existing data modeling approaches and reducing them to only three major principles: duality, inclusion and partial order. COQL is a rather simple and natural language in comparison to existing query languages but on the other hand it is a rather powerful language which introduces several significant changes to the way data is represented, queries and analyzed:

- COQL removes such predefined roles as dimension, measure, fact, and cube from the model. At the same time, these roles can still be assigned to elements depending on the necessary result. The benefit is that the model remains inherently multidimensional and analytical but without constraints imposed by the OLAP approach to analysis.
- COQL introduces arrow notation as the main connectivity mechanism replacing joins. The benefit is that the details of connections are effectively hidden by retaining the set-orientation of joins and simplicity of dot notation.
- COQL relies on partial order relation for representing da semantics. This makes the whole approach very natural because the data modeling constructs directly correspond to the real things and business notions they represent. It also makes this approach very general because partial order can represent many existing semantic constructs and models: entity-relationship modeling, multidimensional modeling, levels of detail, object-oriented modeling and inheritance, containment relation. The main benefit is that this significantly simplifies data management and analytics by eliminating or reducing many incompatibilities which stem from a large number of diverse data modeling techniques and patterns.
- COQL provides inference capabilities as an inherent part of the query languages without adding new constructs but rather relying only on the multidimensional setting and semantic interpretation of the partial order relation. The benefit is that it allow for not only doing complex numeric analysis but also performing tasks which have always been a prerogative of logic-based models.

REFERENCES

Atkinson, M., Bancilhon, F., DeWitt, D., Dittrich, K., Maier, D., & Zdonik, S. (1989). The object-oriented database system manifesto. In *Proc. 1st Int. Conf. on Deductive and Object-Oriented Databases* (pp. 223-240).

Atzeni, P., Jensen, C.S., Orsi, G., Ram, S., Tanca, L., & Torlone, R. (2013). The relational model is dead, SQL is dead, and I don't feel so good myself. *ACM SIGMOD Record*, *42*(2), 64-68.

Boncz, P. (Ed.) (2012). Column store systems [Special issue]. *IEEE Data Eng. Bull.*, *35*(1).

Chaudhuri, S., Dayal, U., & Narasayya, V. (2011). An overview of Business Intelligence technology. *Communications of the ACM*, *54*(8), 88-98.

Codd, E.F. (1970). A relational model of data for large shared data banks. *Communications of the ACM*, *13*(6), 377-387.

Cohen, J., Dolan, B., Dunlap, M., Hellerstein, J.M., & Welton, C. (2009). Mad skills: New analysis practices for big data. In *Proc. 35th International Conference on Very Large Data Bases (VLDB 2009)* (pp. 1481-1492).

Dittrich, K.R. (1986). Object-oriented database systems: The notions and the issues. In *Proc. Intl. Workshop on Object-Oriented Database Systems* (pp. 2-4).

Larson, P. (Ed.) (2013). Main-memory database systems [Special issue]. *IEEE Data Eng. Bull.*, *36*(2).

Li, C., & Wang, X.S. (1996). A data model for supporting on-line analytical processing. In *Proc. Conference on Information and Knowledge Management* (pp. 81-88). Baltimore, MD.

Mohan, C. (2013). History repeats itself: Sensible and NonsenSQL aspects of the NoSQL hoopla. In *Proc. EDBT 2013* (pp. 11-16).

Pedersen, T.B., & Jensen, C.S. (2001). Multidimensional database technology. *IEEE Computers*, *34*(12), 40-46.

Russo, M., Ferrari, A., & Webb, C. (2012). *Microsoft SQL Server 2012 Analysis Services: The BISM Tabular Model*. Microsoft Press.

Savinov, A. (2012a). References and arrow notation instead of join operation in query languages. *Computer Science Journal of Moldova (CSJM)*, *20*(3), 313-333.

Savinov, A. (2012b). Inference in hierarchical multidimensional space. In *Proc. International Conference on Data Technologies and Applications (DATA 2012)* (pp. 70-76).

Savinov, A. (2012c). Concept-oriented programming: Classes and inheritance revisited. In *Proc. 7th International Conference on Software Paradigm Trends (ICSOFT 2012)* (pp. 381-387).

Savinov, A. (2011). Concept-oriented query language for data modeling and analysis. In L. Yan & Z. Ma (Eds), *Advanced database query systems: Techniques, applications and technologies* (pp. 85-101). IGI Global.

Savinov, A. (2009). Concept-oriented model. In V.E. Ferraggine, J.H. Doorn, & L.C. Rivero (Eds.), *Handbook of research on innovations in database technologies and applications: Current and future trends* (2nd ed., pp. 171-180). IGI Global.

Savinov, A. (2006a). Query by constraint propagation in the concept-oriented data model. *Computer Science Journal of Moldova (CSJM), 14*(2), 219-238.

Savinov, A. (2006b). Grouping and aggregation in the concept-oriented data model. *ACM Symposium on Applied Computing (SAC 2006)* (pp. 482-486).

Savinov, A. (2005). Hierarchical multidimensional modelling in the concept-oriented data model. *Proc. 3rd international conference on Concept Lattices and Their Applications (CLA'05)* (pp. 123-134).

Thiele, M., & Lehner, W. (2012). Real-TimeBIand Situational Analysis. In M.E. Zorrilla, J.-N. Mazón, Ó. Ferrández, I. Garrigós, F. Daniel, & J. Trujillo (Eds.), *Business Intelligence Applications and the Web: Models, Systems and Technologies* (pp. 285-309). IGI Global.

Ullman, J.D., & Zaniolo, C. (1990). Deductive databases: achievements and future directions. *ACM SIGMOD Record*, *19*(4), 75-82.

Wood, P.T. (2012). Query languages for graph databases. *ACM SIGMOD Record*, *41*(1), 50-60.

KEY TERMS & DEFINITIONS

Arrow Notation is an approach to data access where fields are used to navigate through a structure. The main difference from dot notation is that arrow notation is a set-oriented approach and arrow operators are applied to and return sets of elements rather than individual elements. Another difference from dot notation is that arrow notation uses two opposite operators for navigating in both directions. In COQL, arrows denote projection and de-projection operators.

Concept is a syntactic construct which is used to describe a data type and generalizes conventional classes. Concept is defined as a couple of one identity class and one entity class. Thus concepts can model both values (if entity class is empty) and objects (if identity class is empty).

Concept-Oriented Query Language (COQL) is a syntactic embodiment of the concept-oriented model (COM). It is a join-free query language which uses references and model multidimensional structure for connectivity. At the same time, it is a set-oriented approach because its operators manipulate sets of data elements rather than individual elements. It is also a semantic language because its constructs reflect and rely on basic semantic relationships existing in the model. Main operations of this query language are projection, de-projection and product (cube).

De-projection is an operation applied to a set of elements and returning all their lesser elements in the partially ordered set. In terms of references, it returns a set of elements which reference the source elements along the specified dimension.

Inclusion is relation between concepts which generalizes classical inheritance. One difference of inclusion from inheritance is that it describes a hierarchy of data elements where child elements share their parent element. Another difference is that it also models containment relation where child elements exist within their parent element.

Inference is a procedure where constraints imposed on some source sets of elements are automatically propagated to some target set by returning related data elements as a result set. In COQL, inference operator is implemented as an access path consisting of two parts: de-projection part and projection part.

Logical Navigation is an approach to querying where the result is specified via a path in the model structure which leads from source elements to the elements from the result set. In COQL, logical navigation is supported by projection and de-projection operations. A sequence of projection and de-projection operations is referred to as a logical access path.

Projection is an operation applied to a set of elements and returning all their greater elements in the partially ordered set. In terms of references, it returns a set of elements which are referenced by the source elements along the specified dimension.