Tarski algebraic operations on the frame database model (FDB)

Yannakoudakis E. J., Nitsiou M., Skourlas C., Karanikolas N. N.

Abstract

The Tarski model allows for the simulation of various database models, using an algebraic formulation, persistent. We shall show how the **F**rame **D**ata**B**ase (FDB) model together with its language **C**onceptual **U**niversal **D**atabase **L**anguage (CUDL) can be expressed in the Tarski model. We model an FDB database by a set of mathematical relations and manipulate the database by performing Tarski algebraic operations on these relations. Our main emphasis is on the data manipulation language, using the extended Tarski algebra that is proven to be a computationally complete language. We also extended the Tarski algebra in order to meet all the FDB model requirements.

Keywords: Tarski algebra operations, FDB model, CUDL, data manipulation, dynamic databases

Introduction

Previous research work has shown that the FDB model [Yannakoudakis E.J et. Al. (1999)] [Yannakoudakis E.J et. Al. (2001)] and its associated CUDL language [Yannakoudakis E.J et. Al. (2006)] can form the basis for the creation and maintenance of dynamically evolving database environments. There has been a clear definition of all FDB objects as well as the CUDL language constructs and semantics. The authors have shown the advantages of the FDB model along with the CUDL language that is used to manipulate FDB data sources. The authors have also shown the problems that can be solved, using the FDB data model and the CUDL language, in comparison with the most well-known relational or object-oriented data models.

Specifically, there has been an investigation of dynamically evolving database environments and corresponding schemata, allowing storage and manipulation of variable length data, a variable number of fields per record, variable length records, manipulation of authority records and links between records and fields, and dynamically defined objects (relations in the traditional sense). This has resulted in a new framework for the definition of a unified schema that eliminates completely the need for reorganisation at both logical and internal levels. Retrieval of data is optimised through self-contained storage chunks that also vary dynamically.

Also, there has been an implementation of CUDL, addressing this new framework, including a discussion of the design philosophy of CUDL, and a specification of the language through examples of the constructs and the corresponding queries. It has been shown [Yannakoudakis E.J et. Al. (2006)] that CUDL offers a focused, flexible, efficient and highly expressive environment along with its formal basis with a targeted set of structural properties, rules, and processes, allowing the development of applications in a continuously evolving database system. It also provides the ability to define and manipulate database information and changes that can be easily navigated, ensuring full maintainability of all applications. The formal definition of the syntax and the semantics of CUDL with example interpretations have been presented, in order to illustrate its use. The basic features, query specification and interpretation, object manipulation, query language constructs, and query processing techniques used in the language have also been discussed.

Let us analyse the FDB model and the CUDL language a bit more. The user conceives the data of an FDB system via CUDL as an extension of the relational model. What the user conceives gives him tables (entities in the FDB model), rows (frames in the FDB model) and columns (tag_attributes in the FDB model). More specifically, it gives him the sense of the organisation of data under the form of tables with certain moreover extensions. The management however and operation of this model is laborious and time-consuming and it requires from the user a very good acquaintance of the proposed model. For this reason we wanted to create a language which will help the user to manipulate the applications that have been created based on the proposed model [Yannakoudakis E.J et. Al. (2006)].

Nevertheless there is an obvious lack of an algebraic formulation for both the FDB model as well as the CUDL language, given that algebraic formulations are better platforms from which to build real database systems [Gyssens M. et. Al.(1994)], since they concern the study of structure, relation and quantity and therefore can be used to explore all aspects of a database system. Indeed mathematics has served in defining various database models and languages [Beeri C. et. Al. (1992)], [Codd E. F. (1970)], [Codd E. F. (1979)], [Gyssens M. et. Al. **(**1990**)**], [Sarathy M. et. Al. (1993)], [Abiteboul S. et. Al. (+1991)], [Su S.Y.W. et. Al. (1993)], [Sarathy V. et. Al..(1992)], [East D., et. Al. (2006),], [Benedikt M., et. Al. (2003),].

Given this situation, we considered a simple algebra, the Tarski algebra, that is appropriate to support binary relation-based database schemata [M. Gyssens et. Al. (1994)]. In our case, the Tarski algebra operates on binary relations with tags (keys in the FDB model), in that it is at the level of abstraction of relation based database models and is thus more natural and effective than other algebras for such database models. We chose the Tarski algebra since we saw that it allows every possible complex manipulation of our model and is thus more natural and effective than other algebras and it is a computationally complete language [Gyssens M. et. Al.(1994)]. It

is important to stress here that these binary relations are conceptual. In fact, the Tarski algebra is shown to support physical data independence [Sarathy V. et. Al. (1992)].

Using the Tarski algebra for data manipulation

The object base schemata and instances we are working with are a collection of binary relations. In order to manipulate these relations it is necessary to develop an algebra that is closed with respect to the class of binary relations and expressive enough to handle all reasonable queries. The kernel of the basic Tarski algebra [Tarski A. (1941)] consists of four well-known operators on binary relations:

- ✓ The union of two relations r and s, denoted $r \cup s$
- ✓ The composition of two relations, denoted $r \bullet s$
- ✓ The inverse of a relation r, denoted r^{-1}
- ✓ The complement of a relation e, denoted $\tilde{e}$.

This algebra was then extended [Gyssens M. et. Al. (1994).], [Sarathy V. et. Al.(1991)] to enable representation of complex objects by adding certain object-id creation operators. The algebra was also extended with some simple selection operators. The selection operators are simple and select pairs from a relation based on certain selection conditions involving constants. The object-id creation operators are more fundamental and allow the creation of object identifiers for ordered-pairs. Specifically, the full Tarski algebra has, besides the four basic operators, two constant selection the left- and right-selection operators, and two ordered-pair (oid) creation the left- and right-oid creation operators. This algebra was further extended [Sarathy V. et. Al.(1992)] to facilitate parallel execution of queries by allowing mechanisms to horizontally partition relations and accumulate sub query results.

First, we have to state that we have the tag notion in our relations, that is a set of values (keys) that can uniquely identify each set of values (tuples) in our model. The type of coding used in the FDB model uniquely identifies an element within a record without ambiguities which might be caused by the use of names, such as ‘title’ of an element. Therefore, we do not have to create new relations with new tags, preserving the links between the attributes.

Thus, we can define an FDB database as a set of mathematical relations each one having its own tag that can uniquely identify its values.

Le us explain this. In our model we have a set of mathematical relations ($s_1 \ldots s_m$) for which every value (tuple) in s_i, $1 \le i \le m$, is uniquely identified. Indeed, for every set s of our model we have a collection of values that consists of an infinite set of data values where some of them are used to represent the tags in our model. Our tags are not “auxiliary” values, but they also represent “data” in our model.

We have to state that in our model we use the notion of universal tagging [Gyssens M. et. Al. (1994)], meaning that in every set s in our model no value is introduced more than once as a tag. Therefore, this tag is always "new" i.e., it is not a component of a pair in s and it has not already occur in the context of s, therefore every set of values in our model never gets the same tag.

For example, let us consider the relation messages:
`messages (message_id, language, message).`
This is uniquely identified by (message_id, language) as it is defined to be the primary key in the relation and as it is shown this relation is in 3NF [Yannakoudakis E.J et. Al. (2001)].

The same happens with the relation entities:
`Entities (frame_entity_id, title).`
This relation is uniquely identified by (frame_entity_id) as it is also defined to be the primary key in the relation and it is also shown that this relation to be in 3NF [Yannakoudakis E.J et. Al. (2001)]. In the appendix of this paper we show how all the relations in an FDB database are uniquely identified.

Therefore from definition 8.5 [Gyssens M. et. Al. (1994)] we can conclude that an FDB database can be simulated in the Tarski algebra. In this definition authors prove that by using the encoding of Codd relations into mathematical relations, they can simulate a relational query in the Tarski algebra. In other words they show that when we have relations with tags these relations can form queries that can be simulated using the Tarski algebra.

We now turn to the CUDL language. This language is defined over the FDB data model in order to offer focused, flexible, and efficient management of the model, expressing full use of it's capabilities by containing rules, processes, that can serve any applications based on that model. The language has the ability to smoothly combine schema and data querying while exploiting all FDB modelling features. It is therefore a programming language that offers a set of built-in data definition and manipulation operations, as well as direct support for the fixed FDB data model [Yannakoudakis E.J et. Al. (2006)]. Therefore all CUDL functions rely on operations on the binary relations that consist the FDB data model.

So we can approach the description of CUDL constructs by using the extended Tarski algebra and we can translate CUDL queries into equivalent Tarski algebra expressions.

In the following part of this paper we are going to deal with the most commonly used (and maybe most important) CUDL functions defined, which are:

- ✓ Data manipulation: CUDL is able to handle requests to update or delete existing data in the data source or to add new data to the data source.

- ✓ Data queries: CUDL is able to handle requests to retrieve existing data from the data source.

Before we proceed we would like to introduce the nested "if then…" construct, which is a derived construct that can be simulated in the basic Tarski algebra and is specified as follows:

```
If (tarski-expression1) then
      (tarski-expression2)
        If (tarski-expression3) then
            (tarski-expression4)
                  .
                  .
             If (tarski-expressionm) then
                 (tarski-expression(m+1))
```

Generally speaking every CUDL statement includes a projection and (or) a selection, as defined in the Tarski algebra, in the set messages. This set holds all the information that the user can actually understand. After that the values in the set messages are combined with the sets in question, using the Extended Tarski algebra και the construct that we have introduced above, in order to give the user the final result.

In other words every CUDL query begins with a mapping of the values of the set in question to their corresponding values in the set messages in order to select tuples that satisfy a given predicate and (or) copy values for some specified attributes only. Since we are dealing with sets, duplicate rows are eliminated. Afterwards we do the following:

- perform efficient searching on the sets of the data source that focuses on the collection of values (information) that match user queries
- organize this information into collections (virtual subsets) each of which contains all or part of the information originally asked by the user
- construct a final (virtual) subset based upon the created collections that holds the information wanted
- return a result to the user

The following are chosen to cover different types of CUDL statements, in order to show the range of the algebra.

Query

Here we want to find which tags appear under the entity x. In the CUDL language we would provide the following statement:

- ✓ `Find tag_attributes when entity = 'x'`

To solve this query using the extended Tarski algebra we have:

```
Query entities_tags
      begin
            Message_id_entity= Π(message_id) (σ(message=x)(messages))
            Entity_id= Π(frame_entity_id)
            (entities∩message_entity_id)
            While (tag_attributes∩entity_id )≠ □
            do
                  Tags_temp = Π(title) (tag_attributes∩entity_id)
                  Tags = Π(message) (messages∩tags_temp)
            Od
            Return tags
      End;
```

Here, we first have to find the "message_id" value of the value x. Thus, we have a subset of the set "messages" called "Message_id_entity". In the next step, we take the "frame_entity_id" value in the set "entities∩message_entity_id" and we have a new subset called "entity_id". Then we perform an iterative query in the set "tag_attributes∩entity_id" that is, we take each value appearing under 'title', having another subset called "tag_temp". Then for each value found under title we take the values appearing under "message" in the set "messages∩tags_temp".

Addition

Let us suppose that we want to add a new value x in the set entities. In the CUDL language this is done by using the following statement:

- ✓ `Add entities title = 'x'`

In the Tarski algebra this is achieved as follows:

```
            Message_id = Π(message_id)(σ(messages=x(messages))
            If message_id = □
            then
                  New_message = {(message_id,language,message)
                  | (message_id,language,message) □ messages}
                  Message_id = Π(message_id) (new_message))
            New_entity = {(frame_entity_id,title) | (
            frame_entity_id, title) □ entities ⋀ title□
            message_id }
            Entities = entities ∪ new_entity
```

First we have to look for the "message_id" value in the set messages that corresponds to the value x. If there is no such value in the set messages then a new value x is added in this set. Now we can take the "message_id" value for the value x and thus create a new subset called "message_id". Then we create a new set called

"new_entity" containing the new value. The last step is to add this new value the set entities.

Deletion

In the following we show how we can delete an entity x from the set "entities". In the CUDL language we provide the statement:

- ✓ `delete entities when entity = 'x'`

In the Tarski algebra this is achieved as follows:

```
Message_id_entity= Π message_id (σmessage=x(messages))
If  Message_id_entity ≠ □
then
      Entity_id_temp= Π(frame_entity_id,,title)
      (entities∩message_id_entity)
      If  Entity_id_temp ≠ □
      then
            Entities = entities – entity_id_temp
```

Again we have to look for the "message_id" value in the set messages that corresponds to the value x. If this value exists in the set messages we have a new set called "message_id_entity" (which is not empty). Then we perform a search in the set "entities∩message_id_entity" and take a new set called "entity_id_temp". If this last set is not empty then we can have a new set "entities" not containing the value in question (x).

Replacement

This example demonstrates how we can replace the value y in the set "entities" with a new value x. In the CUDL language we have:

- ✓ `alter entities  title =  'y' with 'x',`

whereas in Tarski algebra we have:

```
Message_id1 = Π message_id(σmessage=y(messages))
      If message_id1 ≠ □
      then
            Entities_temp = Πframe_entity_id (entities ∩
            message_id1)
            If entities_temp ≠ □
            then
                  Entities = entities – entities_temp
                  Message_id2 =
                  Πmessage_id(σmessage=x(messages))
                  If message_id2 = □
                  then
```

```
                    New_message =
                    {(message_id,language,message) |
                    (message_id,language,message)  
                    messages}
                    Message_id = Π message_id,message
                    (new_message)
                    New_entity =
                    {(frame_entity_id,title) |
                    (frame_entity_id, title)  
                    entities ⋀ title  message_id}
                    Entities = entities ∪ new_entity
             Else
                    New_entity =
                    {(frame_entity_id,title) |
                    (frame_entity_id, title)  
                    entities ⋀ title  message_id2}
                    Entities = entities ∪ new_entity
```

Here we have to look for the "message_id" value in the set "messages" that corresponds to the value y. If the value actually exists in the set "messages' then we look for this value in the set entities. If this value exists there as well we must proceed into removing this value from the set "entities". Then we have to look for the "message_id" value in the set "messages" that corresponds to the value x. If the value does not exist in the set "messages" then this value is added. Finally the new value is added to the set entities (see Addition statement). If the value exists in the set "messages" this value is added only in the set "entities".

Conclusions

Previous research work has shown that the FDB model and its associated CUDL language can form the basis for the creation and maintenance of dynamically evolving database environments. However we noticed that there is an obvious lack of an algebraic formulation for both the FDB model as well as the CUDL language, and we decided to provide such an algebraic formulation, since algebraic formulations are better platforms from which to build real database systems. Given this situation, we considered a simple algebra the Tarski algebra, that is appropriate to support binary relation-based database schemata. We modeled an FDB database by a set of mathematical relations and manipulated the database by performing Tarski algebraic operations on these relations. Therefore, we have shown that the Tarski algebra provides a strong and simple algebraic foundation for the definition of the data FDB model as well as data manipulation and queries in the CUDL language. This helped us to understand the theoretical foundations of the query language based on that data

model. Also, we have extended the Tarski algebra in order to meet the FDB model requirements.

In the future we need to investigate how this algebra can be extended. In this direction, we intend to provide more extensions in this algebra in order to meet more advanced features of the FDB model and the CUDL language (such as aggregates, grouping, ordering and mathematical operations). We also intend to consider in detail the application of the Tarski algebra in the formulation of more complex manipulations of the FDB data model.

References

Abiteboul S. and Grumbach S. (1991), *A Rule-Based Language with Functions and Sets*, ACM Transactions on Database Systems, Vol 16, No 1, pp. 1-30.

Beeri C., Milo T. (1992), *Functional and Predicative programming in OODB'S*, in Proceedings of the eleventh ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems, San Diego, California, United States.

Benedikt M., Libkin L., Schwentick T., Segoufin L. (2003), *Definable relations and first-order query languages over strings,* Journal of the ACM (JACM)**,** Volume 50 Issue 5, pp. 694 – 751.

Cood E. F. (1970), *A Relational Model of Data for Large Shared Data Banks*, Communications of the ACM 377, Volume 13, Number 6, pp. 377-387.

Codd E. F. (1979), *Extending the Database Relational Model to Capture More Meaning*, ACM Transactions on Database Systems, Vol. 4, No. 4, pp. 397-434.

East D., Truszczyn´ski M. (2006), *Predicate-Calculus-Based Logics for Modeling and Solving Search Problems,* ACM Transactions on Computational Logic, Vol. 7, No. 1, January, Pages 38–83.

Gyssens M., Paredaens J., Van Gucht D. (1990)**,** *A graph-oriented object database model*, in Proceedings of the ninth ACM SIGACT-SIGMOD-SIGART: symposium on Principles of database systems ,Nashville, Tennessee, United States.

Gyssens M., Saxton L., and Van Gucht D. (1994), *Tagging as an Alternative to Object Creation. Query processing for advanced database systems*, Morgan Kaufmann Publishers, Printed in the USA, ISBN: 1-55860-271-2.

Sarathy M., Van Gucht D. (1993), *Implementation of a Graph Oriented Query Language: IUGQL,* Technical Report No 376, Computer Science Department, Indiana University.

Sarathy V., Saxton L. and Van Gucht D. (1992) *An Object Based Algebra for Parallel Query Processing and Optimization*, Technical Report, No 368, Dept. of Computer Science, Indiana University.

Sarathy V., Saxton L. and Van Gucht D. (1991), *Translating Query Graphs intoTarski Algebra Expressions,* Technical Report No 342, Dept. of Computer Science, Indiana University.

Su S.Y.W. , Guo M., Lam H. (1993), *Association algebra: a mathematical foundation for object-oriented databases*, IEEE transactions on knowledge and data engineering, vol. 5, n°5, pp. 775-798.

Tarski A. (1941), *On the calculus of relations*, Journal of symbolic logic, vol. 6, pp. 73-89.

Yannakoudakis E.J., Tsionos C.X. and Kapetis C.A. (1999), *A new framework for dynamically evolving database environments*, Journal of Documentation, Vol. 55, No. 2, pp. 144-158.

Yannakoudakis E. J., Diamantis I. K. (2001), *Further improvements of the Framework for Dynamic Evolving of Database environments*, in Proceeding of the HERCMA 2001, 5th Hellenic – European Conference on Computer Mathematics and its Applications, Athens, Greece.

Yannakoudakis E. J., and Nitsiou M. (2006), *A new conceptual universal database language (CUDL)*, in 2nd IC-SCCE: 2nd International Conference From Scientific Computing to Computational Engineering, Athens, Greece.

Appendix

We present the binary relations used in the FDB model with their unique identifiers (tags in the Tarski algebra, keys in the FDB model). The identifiers are underlined.

Languages	(language_id,lang_name)
Datatypes	(datatype_id,datatype_name)
Messages	(message_id, language, message)
Entities	(frame_entity_id,title)
Tag_attributes	(entity,tag,title,occurrence,repetition,authority,language, datatype, length)
Subfield_attributes	(entity,tag,subfield,Title,occurrence,repetition,language, datatype, length)
Catalogue	(Entity,Frame_object_number,Frame_object_label, Temp_stamp**)**
Tag_data	(Entity ,Frame_object,Tag,Repetition,Chunk,Tdata)
Authority_links	(From_entity,Auth_tag1,To_entity,Auth_tag2)
Subfield_data	(Entity,Frame_object,Tag,Tag_repetion,Subfield, Subfld_Repetition,Chunk,sdata)
Bit_data	(entity,tag,format_type,byte_size,bit_image)
Coded_subfields	(entity,tag,subfield,start_char,end_char,cods_value)
Coded_tags	(entity,tag,start_char,end_char,codt_value)
Sys_interface	(sys_int_id,language,sys_message)