Object Identity Set Algebra
for Object-Oriented Database Systems

Shugang Wang
China Lottery Online
Beijing, China
e-mail: wmywangsg@yahoo.com.cn

Abstract—Object identity set algebra is presented in the paper. The algebra is efficient to express queries to objects and gains support from underlying relational algebra and formal language theories. The algebra lies on top of relational algebra in the structure of layered mathematical models guiding object-oriented database systems. Object identity set algebra is defined precisely to apply mathematical set functions for querying objects in database systems.

Keywords—object identity set, mathematical set algebra, object-oriented, database.

I. INTRODUCTION

The paper presents object identity set algebra as a mathematical model for querying objects in object-oriented database systems (OODB). The algebra is expressed with two assumptions and three basic functions. The paper analyzes the mathematical models for OODB, and gives a structure of layered mathematical models.

The algebra is efficient to express queries to objects and gains support from underlying relational algebra and formal language theories. The algebra lies on top of relational algebra in the structure of layered mathematical models guiding object-oriented database systems.

The relational model is today the primary data model for commercial data-processing applications. It has attained its primary position because of its simplicity, which eases the job of programmer, as compared to earlier data models such as the network model or the hierarchical model.

The relational algebra is one of the mathematical models for database systems, which bears precise definition comparing to the Turing complete languages. Following on from the successes of the relational algebra in terms of their precise definition, the object identity set algebra was designed around the basic notions of mathematical set.

The remainder of this paper is organized as follows. We show some background on query algebras for OODB in Section II. Then, in Section III and IV, which constitute the major part of this paper, we present the object identity set algebra. In Section V, we outline the structure of mathematical models in the OODB area and show the position of this algebra. In section VI, we give business examples to show the application of the object identity set algebra. Finally we conclude the paper in Section VII.

II. RELATED WORKS

We outline the structure of mathematical models for OODB in section V. Programming languages has developed the concepts and functions on object level. Relational algebra has developed the concepts and functions on data collection level.

With the knowledge of structure of mathematical models discussed in section V, we can form some evaluation on the algebras cited here. The algebras combine the concepts together on the object level and the data collection level. The algebras regenerate the concepts, which exist on object level and data collection level, in the application scope of OODB.

The query calculi are cited as the references, although they do not focus on presenting a particular algebra, but rather to show the general reduction rules to form the algebras.

A. A Query Algebra

Reference [8] presents a query algebra for object-oriented databases. The query algebra provides type specific operations against collections of complex objects with identity. The algebra assume collection objects have type Set[T] or Multiset[T], although semantics of the operations could be modified to support most other collection types. The algebraic operations support the notion of abstract data type, and all access to objects in a collection having type Collection[T] is through the public interface defined for type T.

A query over a Set[T] returns a Set[Q]. Q is defined by the query and is either an existing type or a tuple type. The result of a query is a new typed database collection, with the collection type determined statically using the type structure. The result of a query using the algebra presented thus far is a new collection of objects. The collection will be a newly identified object in the database, and thus there cannot be identical responses to a query. The creation of new objects means that multiple objects with unique identifiers may be created when a single object is desired.

Most operations return collections of existing database objects. The algebra additionally provides operators to create new relationships, possibly by joining objects from different collections.

Select operation creates a new collection object containing
the identifiers of all members of collection S satisfying the predicate.

The Image and Project operations are used primarily to return components of objects in the collection being queried.

The Ojoin operator is an explicit join operator used to create relationships between objects from two classes in the database.

The algebra includes set operations Union, Difference, and Intersection, operation Flatten (for sets of sets) and operations Nest and UnNest (for sets of tuples). The first three operations are used to create new collections of objects, the latter three generally re-structure collections of objects.

The algebra defines operations DupEliminate and Coalesce to handle situations where equal objects are created by a query.

B. O-Algebra

The output of each O-Algebra [4] operation is a new collection of objects.

Given an object type T, a CO (collection of objects) of type T is a collection of instances of the type T. The algebra uses the same name for both a type and its extents. From the definition above, each object in the model has the form (o, v), where o is an Oid, and v is a tuple of with at most one level of nesting.

O_Agebra is an algebra (S, Σ), where S is the set of operands, which includes all COs, and Σ is the set of operators, which includes a set of core operators and some additional operators. The set of core operators consists of map, select, Cartesian product, group, ungroup, project, extend, union, difference, DelDup which define the expressive power of O_Agebra. Additional operators remap, join, outerjoin, cap are also defined.

C. AQUA

One of the primary goals of AQUA [5][6] is to provide a model general enough to simulate the constructs of any object-oriented data model (and most value-oriented models), no matter what choices it makes with respect to certain features (bulk types, encapsulation, identity versus value, notions of equality, inheritance, and operations).

AQUA characterizes the distinction between ‘objects’ and ‘values’ as the difference between entities (objects) with mutable and immutable semantics; this provides a much cleaner formalism.

Operators for sets are separated into several groups, including the unary set operators, the binary set operators, the LFP operator, the set restructuring operators, and the various join operators.

Multisets support nearly all the same operations as sets, with very similar semantics in most cases. Most multiset operators are quite similar to the corresponding set operators, except for the fact that the input and output types are multisets instead of sets. Besides sets and multisets, the algebra supports a host of other types.

D. QAL

This section presents the main features of the object-oriented data model serving as a framework for the definition of the QAL algebra [7]. The data model provides, in addition to the basic constructs of the object-oriented data model, a uniform view of the database by treating classes as abstract objects.

An object identifier (abbr. oid) is a unique symbol which represents an entity from the real world. An object identifier is either an individual identifier which represents a concrete entity, or a class identifier which represents an abstract concept.

An object is defined as a pair (i; v), where i is an object identifier and v its corresponding value. An object identifier is a reference to an object, and a value represents the state of object realized by o-value. Similarly as in the case of object identifiers, the algebra differentiates between two types of objects: class objects and individual objects. A class object represents an abstract concept and acts as a representation of a set of individual objects which share similar static structure and behavior.

The model-based operations are closely related to the concepts introduced by the data model formalization. They are intended for inquiring about: the associations among the individual objects, the relationships between the individual objects and class objects, and the relationships among class objects themselves.

The following model-based operations are defined: valuation operations, extension operations, comparison operations based on the o-value poset, closure operations based on the partial ordering relationship, operations for finding the nearest common more general or more specific objects of a set of objects, and equality operations.

The set of declarative operations comprises three groups of operations. The first group includes the generalisation of the basic operations of the (flat) relational algebra for the manipulation of o-values. This group contains the operations select, union, differ and intsc. The second group of operations extends the functionality of the restructuring operations. These are the QAL operations unnest and group. The last group of QAL operations is a set of higher-order functions that evolved from functional query languages. This set includes the QAL operations apply, tuple, close and apply_at.

E. Towards an object-oriented query algebra

Reference [9] has sketched how a minimal set of bulk primitives, parameterized by expressions containing arbitrary operators, can support a wide variety of traditional database operations and optimizations.

Much of what the model has described revolves around a single family of operators named fold. With inductive reasoning and some laws, operators such as select, map, and join can be expressed in terms of fold. One advantage of reducing multiple bulk operators to a single one is that the effort required to build an evaluator for the algebra is thereby reduced as well.

The paper has discussed bulk operators on lists, sets, multisets, arrays, tree structures handled by fold. From fold one can derive all the traditional relational operators and more.

Up to this point all the constructors the model has mentioned yield pure-value types that do not admit updates.

A value is an object, but it is an immutable object and, unlike mutable objects, it is not guaranteed distinct from all other objects. In the model, every object is a value, and every value is
an object. Whether or not an object (or value) is tagged with type information depends on whether it belongs to a union sort.

**F. Optimizing Object Queries Using an Effective Calculus**

The model [3] focuses on a very important optimization problem, query unnesting (sometimes called query decorrelation) and proposes a practical, effective, and general solution.

The monoid calculus is based on the concept of monoids from abstract algebra. A monoid of type \( T \) is a pair \((+, Z)\), where \(+\) is a binary function that takes two values \( T \) and return a value \( T \), called the accumulator or the merge function of this monoid, and \( Z \) of type \( T \); called the zero element of the monoid, is the left and right identity of \(+\). That is, the zero element satisfies \( Z+x = x+Z=x \) for every \( x \).

A monoid comprehension is defined by some reduction rules. The calculus has a semantic well formedness requirements that a comprehension be over an idempotent or commutative monoid if any of its generators are over idempotent or commutative monoids.

The basic algebraic operators, including \( \text{join} \), \( \text{selection} \), \( \text{unnest} \) and \( \text{reduce} \), can be defined in terms of the monoid calculus.

**III. ASSUMPTIONS**

**A. First Assumption, Class Defined**

The class declaration and definition are needed for the relative business entities, before discussing queries to the object-oriented database (OODB). This is a trivial requirement because queries are made to the object-oriented applications on OODB. It is clear from the assumption that queries will not create new classes.

For comparison, relational algebra allows new schema to be created instead, such as a schema with fewer attributes by the projection operation or a schema with more attributes by the Cartesian product operation [1]. Declaring a class type before creating a variable of the class is a property of compilation type programming languages, and this improve the application program quality by type checking. Creating classes at run time is not practicable for most compilation type programming languages.

**B. Second Assumption, Object Universal Set Prepared**

Object base is the collection of all the objects of the class defined by the first assumption. Object universal set is the collection of all the object identities to the objects in the object base. Object identity is the unique key to retrieve an object in the object base and have one to one relationship to the object.

The universal set for objects must be prepared before any query can take place. The universal set for objects is the restriction for any query could be expressed to the object base, including query sources which could be recognized and the query results which are possible to be produced.

It is stated in the mathematical text books that we will assume that for each discussion there is a universal set \( U \) (which will vary with the discussion) containing all the objects for which the discussion is meaningful. Any other set mentioned in the discussion will automatically be assumed to be a subset of \( U \).

**IV. OBJECT IDENTITY SET ALGEBRA**

Object Identity Set Algebra is an algebra \((S, \Sigma)\), where \( S \) is the set of operands, which includes all object identity sets, and \( \Sigma \) is the set of operators, which includes a set of core operators and some additional operators. The set of core operators consists of \( \text{select}, \text{union}, \text{difference} \), which defines the expressive power of the algebra. Additional operators \( \text{intersect}, \text{add_entry}, \text{remove_entry} \) are also defined.

The expressive power of object identity set algebra is boosted by the mathematical set functions.

**A. Object Identity Set**

Object identity set is the mathematical set of the object identity to the object in the object base. The operations upon one or two object identity set will create a new object identity set.

Object identity set is defined as a subset of the universal set stated by the second assumption in section III-B. Object identity set is the key concept to implement the queries to the object-oriented database. Query to the OODB is expressed by operations defined on the object identity set.

\[
\text{Object Identity Set} \subseteq \text{Object Universal Set} \quad (1)
\]

**B. The Set Union Operation**

Set union operation generates a set which contains all the object identities in the first object identity set and those in the second object identity set.

\[
\text{set1} \cup \text{set2} = \{x \mid x \in \text{set1 or } x \in \text{set2}\} \quad (2)
\]

where, \( x \) is an object identity, \( \text{set1} \) and \( \text{set2} \) are object identity sets.

**C. The Set Difference Operation**

Set difference operation generates a set which contains the object identities in the first object set but are not in the second object identity set.

\[
\text{set1} - \text{set2} = \{x \mid x \in \text{set1 and } x \notin \text{set2}\} \quad (3)
\]

where, \( x \) is an object identity, \( \text{set1} \) and \( \text{set2} \) are object identity sets.

**D. The Select Operation**

The select operation selects objects that satisfy a given predicate, while a predicate is a function that returns a Boolean (true/false) value. The select operation will generate an object identity set as a subset of the source object identity set.

The predicate is used as an argument in the select operation. Following the first assumption, the predicate must be expressed by one or more methods defined on the object class.
set1.query (P) = {x | x ∈ set1 and P (x) is true} \hspace{1cm} (4)

where, set1 is an object identity set, and P is the predicate on the objects in the object base.

E. The Compatible Object Set

Two object identity sets can take the union or difference operation only when they belong to the same object base, which prepares the universal set for any object identity sets in the queries. Object identity sets belong to different object base are not compatible for the union and difference operation.

The object base prepares the universal set stated in the mathematical text books. Although the compatibility requirement seems obvious, it does make some subtle difference between the object identity set algebra and relational algebra [1]. It will be illustrated in the latter query examples in section VI.

V. MATHEMATICAL MODEL LEVELS

This section we outline the levels of mathematical models for object-oriented databases. The levels are separated by the scope of functions in the mathematical models.

A. Object Level Mathematical Model

Objects are the elements discussed in the mathematical model of this level. The functions upon objects deal with the relations between individual object by mapping one object class to another object class.

The functions on this level include the basic programming concepts like data structures, functions and control flow, and include the object-oriented programming concepts like class, instance, object data, object methods, and class inheritance. This level of mathematical model includes the formal languages and computational theories. Recursively enumerable languages are known as type-0 languages in the Chomsky hierarchy of formal languages.

B. Data Collection Level Mathematical Model

Data collections, which are collections of internal data of object in the above level, are the elements discussed in the mathematical model of this level. The functions upon data collection map one data collection to another.

Relation is one of the representations for data collection. The functions in relational algebra [1] map one relation to another relation, including six basic operations of select, project, Cartesian product, set union and set difference, as well as additional derived operations.

This level of functions is of value type collection algebra, and the data collections are subsets of a universal set formed by Cartesian product of the object data domains defined on the object level model. The universal set defines the limit to the expressive power of relational algebra.

C. Object Identity Set Level Mathematical Model

This level of mathematical model discusses object identity sets, which are the subsets of a universal set containing identities to objects in the data collection of the above level.

A relation implies a universal set, which is the collection of row identity of the relation. Object identity sets are the subset of the universal set. Object identity set algebra, introduced in this paper, supplies three operations of select, union and difference defined upon object identity sets.

The two assumptions show that the object identity set algebra is based on the mathematical model of object-oriented programming languages and relational algebra. The mathematical model receive the gifts from the existing achievements in the formal languages and relational algebra researches, and focus on a new area with stricter definitiveness than relational algebra as well as programming languages.

D. Definitive Precision vs. Expressive Power

It is known that the query languages based upon relational algebra are not Turing complete, thus the expressive power of relational algebra is less than the Turing complete languages.

The object identity set algebra does not contain the operations of project and Cartesian product, which expand the expressive power of relational algebra. We can conclude that relational algebra has greater expressive power than the object identity set algebra.

In Fig. 1, we show that the expressive power becomes less, from the object level to data collection level, then to object identity set level. Relational algebra lies in the data collection level, while object identity set algebra lies in the top level.

On the other hand, object identity set algebra gains finer definitive precision for databases. Released of the burden of object concepts and data collection concepts, object identity set algebra grasp mathematical set functions strictly. As a result,
queries expressed with object identity sets are light weighted, in comparison to the queries expressed in value type algebras, both in relational algebra or in object-oriented query algebras in section II.

VI. BUSINESS EXAMPLES

As stated in the above section, object identity set algebra is less powerful than the relational algebra, which belongs to the mathematical models in the data collection level, but it is clear that object identity set algebra can simulate the relational algebra with the assistance of the object level models. We can express the queries in relational algebra [1] [2] with the object identity set algebra on object-oriented programming language. Although we do not include the rigorous proof here, we give some business examples to illustrate the application of the algebra.

A. Business Object Examples

Lottery transaction system is of the architecture of the host and the terminal, and the host implements lottery business objects such as tickets and game products on OODB. Lotto, Digit and Keno are of the popular gaming products in the world.

Game Product object base is used to control the ticket selling and paying in the lottery transaction system and also governing summary accounts such as sold amount and paid amount. Game Product is a good example to illustrate the queries on the OODB.

Define Game Product class, as required by the first assumption stated in section III-A.

Program 1

```cpp
class Game_Product : public OODB_Object
{
public:
    bool open (GameProduct id, string name);
    bool sell (Pennies proposal);
    bool pay (Pennies proposal);
    bool is_open () const;
    Pennies get_sold () const;
    Pennies get_paid () const;

private:
    GameProduct product_id;
    string product_name;
    Pennies sold_amount;
    Pennies paid_amount;
    bool status_open;
};
```

Define Game Product base, containing all the Game Product objects, as required by the second assumption. Predicate classes are also defined here to assist the queries to the object base by select operation.

Program 2

```cpp
class Game_Product_Base :public OODB_Object_Base
{
public:
    struct Is_Open : public OODB_Object_Query
    {
        bool query (const Game_Product * obj)
        {
            return obj->is_open ();
        }
    }

    struct Get_Sold : public OODB_Object_Query
    {
        Pennies min;
        Pennies max;
        Get_Sold (Pennies min = 0, Pennies max = 0);
        bool query (const Game_Product * obj)
        {
            bool result =
            obj->get_sold () >= this->min &&
            obj->get_sold () <= this->max;
            return result;
        }
    }

    struct Sold_Sum : public OODB_Object_Query
    {
        Pennies sold_amount;
        Sold_Sum (Pennies s = 0);
        bool query (const Game_Product * obj)
        {
            this->sold_amount += obj->get_sold ();
            return true;
        }
    }
};
```

B. Business Object Queries

Create object base and enter a Game Product entry.

Program 3 (part 1)

```cpp
Game_Product_Base game_base ();
Game_Product *game = game_base.add_entry ();
game->open (101, "Lotto");
game->sell (100);
game->pay (50);
```

These operations will enter a Game Product entry with id of 101. Repeat the operations and enter two additional Game Products with Id of 102 and Id of 103.

Program 3 (part 2)

```cpp
id 101, name Lotto, status_open True, sold_amount 100, paid_amount 50;
id 102, name Digit, status_open True, sold_amount 200, paid_amount 150;
id 103, name Keno, status_open True, sold_amount 1000, paid_amount 800.
```

There are altogether three Game Products in the object base now. We give some query examples on the object base with different query conditions.

1 Find all the open game products. After query, the object set temp_set contains the three Game Products in the object base.

The predicate object Is_Open is defined in the class of Game_Product_Base in program 2.

Program 3 (part 3)

```cpp
OODB_Object_Set game_set (game_base);
OODB_Object_Set temp_set, result_set;
temp_set = game_set.query
(Game_Product_Base::Is_Open ());
```

2 Find those Game Products with sold amount between 100 and 500. After query, the object set temp_set contains Game
Product 101 and 102. The predicate object Get_Sold is defined in the class of Game_Product_Base in program 2.

Program 3 (part 4)

```
Game_Product_Base::Get_Sold sold (100,500);
temp_set = game_set.query (sold);
```

3 Total up the sold amount in the object set. After query, the total sold amount of 1300 is calculated out and is stored in the predicate object sold_sum. The predicate object Sold_Sum is defined in the class of Game_Product_Base.

Program 3 (part 5)

```
Game_Product_Base::Sold_Sum sold_sum;
temp_set = game_set.query (sold_sum);
```

4 Find all the open Game Products and further require the sold amount between 100 and 500, at last total up the sold amount in the object set. Object set result_set includes Game Product 101 and 102, and the total sold amount is 300 for the two Game Products in the set.

Program 3 (part 6)

```
Game_Product_Base::Is_Open is_open;
Game_Product_Base::Sold_Sum sold_sum2;
result_set = game_set.query (is_open)
. query (get_sold)
. query (sold_sum2);
cout << sold_sum2.sold_amount << endl;
```

Another way to form the last query is to use set intersect operation, which will produce the same query result.

Program 3 (part 7)

```
Game_Product_Base::Is_Open is_open;
Game_Product_Base::Sold_Sum sold_sum3;
OODB_Object_Set open_set, sold_set, sum_set;
open_set = game_set.query (is_open);
sold_set = game_set.query (get_sold);
sum_set = open_set * sold_set;
result_set = sum_set.query (sold_sum3);
cout << sold_sum3.sold_amount << endl;
```

VII. CONCLUSION

Following on from the successes in terms of the precise definition, the object identity set algebra was designed around the basic notions of mathematical set, which represents the fundamental mechanic for the queries to the database system. Tightly grasping the mathematical set, the object identity set algebra has released part of the burden on the data collection algebras, which is typically represented by the relational algebra.

The algebra in this approach does not follow the way of the previous works on object-oriented algebras that repeat the works on object level and data collection level. Instead we create a new mathematical model level for OODB and receive the supports from the underlying works both on object level and data collection level, represented by formal languages and relational algebra respectively.

The simplicity of object identity algebra will ease the job of application programmer. As a result, queries expressed with object identity sets are light weighted, as compared to earlier data models such as relational model or the object-oriented models in the related works.

It will be helpful to apply the object query model to the various object-oriented database applications and database management systems. The project, which implemented the object identity set algebra, and the lottery application examples are introduced in the paper to help understand the mathematical model.

ACKNOWLEDGMENT

I have benefited immensely from the work that was performed in the Keno project and in the ATLAS systems. We have learned valuable lessons by looking at the experiences with those systems. Thanks go to Mingchen Wang and Fei Ye for their detailed comments and assistance on the paper. I would also like to thank Lebao Liu, Gary Elliott and the colleagues in the Keno group that have adopted the research results.

REFERENCES