DBMS for Business Systems

Evgeniy Grigoriev
RxO project LTD (www.RxO-project.com), Moscow, Russian Federation

Keywords: Relational DBMS, Object-oriented Paradigm, Informational Business System, Business Modelling, Heterogeneous Data Schema, Persistent Object, Encapsulation, Group Operation, Nested Structures, Algorithm Transformation.

Abstract: Many problems of traditional business systems (SAP, Microsoft Dynamics, etc.) are largely caused by their architecture where a tier that implements the business model and data processing (application server or thick client) is separated from the DBMS where the data is stored. A new approach to extend the relational DBMS with a means for creating expressive heterogeneous business models according to principles well-known to users of existing business systems is discussed. With it the business system architecture can be simplified fundamentally because of transferring all data modelling and processing directly to the place of their storage. Principles of a creation and a use of complex business models and data access in these models are discussed. Some questions related to system performance are reviewed.

1 INTRODUCTION

With the advent of the first relational DBMS, it became clear that "that they are inadequate for a broader class of applications" that business data processing applications (Stonebraker et al., 1990). Moreover when we say about necessity to build and use expressive complex business models, the relational DBMSs are not good enough even for the business application (Fowler et al., 2003). This problem was tried to be solved in two ways.

DBMS experts solved it by changing capabilities of the DBMS; in fact, they suggested changes of DBMS technology or even of base DBMS logic, up to a full refusal of relational systems (Atkinson et al., 1990). In the context of this article we will consider only those proposals that involve the evolutionary development of a relational DBMS (e.g. SQL99, Oracle, PostgreSQL). We will denote this group as solutions "inside DBMS".

Application programmers, who answer practical challenges, developed business applications on the basis of traditional principles and technologies of using a relational DBMS. Well-known and successful business systems like SAP, MS Dynamics, etc. are examples of such solutions. We will denote this group as solutions "outside DBMS". Let's note that the way of solutions "outside DBMS" in itself causes significant problems which are inherent in modern business systems.

- Like any multi-tier system, they are very complex and hard to create, modify, and maintain.
- The need to process the data outside of the DBMS leads to catastrophic performance degradation because of non-optimal (and hardly optimized) data exchange between the tiers.
- There are obvious difficulties of integration with third-part programs (e.g. Excel) using standard means of access to data in a DBMS (e.g. ODBC). The DB schema generated by an outer tier is usually incomprehensible to the business users or/and inaccessible because of security reasons. On other hand, the business model existing on the outer tier is inaccessible because this tier usually does not support the standard data access protocols and interfaces.

Both groups showed a rare unanimity, considering the object-oriented approach as a kind of panacea to meet requirements of business modelling. However, there is a fundamental logical difference of how they have implemented this approach with respect to modified/used RDBMS.

Analysis of the solutions "inside the DBMS" shows that all they, one way or another, implement the two principles (many DBMS implement both of these two principles simultaneously).

- An object corresponds to a row of the database table ("object = string"). For

Grigoriev E..
DBMS for Business Systems.
DOI: 10.5220/0005554301030108
In *Proceedings of the 10th International Conference on Software Paradigm Trends* (ICSOFT-PT-2015), pages 103-108
ISBN: 978-989-758-115-1

example, in object-relational DBMS (i.e. Oracle since v.8, PostgreSQL) instances of the object types are stored in a typed table, where each instance corresponds to a single line.

- Object corresponds to a single field in the row of the database table ("object = field"). This principle has been expressed logically in hypothetical "D language"(Date and Darwen,2000). It has got also many technological implementations, when object data are stored in BLOB, JSON, etc. field.

Solutions "outside of DBMS", i.e. the traditional business system, successfully implement other third principle: the business object corresponds to a set of rows from different tables of a relational database. For example, the business object "an invoice" typically corresponds to one row from a table "headers" and few rows from a table "items". This principle is evident and clear for users; it has been being used in successful business systems to build a vast majority of objects for decades. We can assume that it closely matches the needs of the business and that it should be primarily implemented in the DBMS which are adequate for modern business systems. However, attempts to create such DBMS is still absent. Moreover, it seems that this possibility is not even considered theoretically (Date and Darwen, 2000, 2013).

We denote this principle with a logical equation "object = relational database". In this equation we assume that any subset of tuples of relations can be considered as a relation. The object corresponds to the "multiple rows of different tables", i.e. to the set of such relations. In turn, the phrase "a set of relations" is a key element of the formal definition of the relational database.

Further, we will consider our approach on how this principle can be implemented in a relational DBMS. The result is a DBMS, which we will call RxO DBMS. Some of the offered ideas were implemented by us in the prototype "RxO system" (Grigoriev,2013a); its input commands we will use for demonstration.

2 OBJECT-ORIENTED MANAGEMENT SYSTEM FOR RELATIONAL DATA

RxO DBMS implements a new approach to the connection object-oriented and relational ways of managing of data if form of complex objects corresponding to the principle of "object = relational database". Based on the properties of the classical relational data model (Codd, 1970), it is shown by us that all descriptions of such objects and operations on them can be translated into descriptions of relational structures and operations on the last ones (Grigoriev, 2013b).

As a result, it's not necessary to "assemble" such objects from the rows of different tables outside the DBMS to work with them. Instead, it's possible to translate all commands on these objects in actions on the tables (system hides these tables from the user). The user, who uses these commands, keeps a full illusion that he works with complex objects which are formed with the principle that is native for business systems. Thus we offer a sophisticated way to manage traditional relational data, which implements well-known object-oriented principles (Booch, 1991).

3 COMPLEX HETEROGENEOUS DATA SCHEMA

RxO DBMS maintains all usual abilities for working with traditional tables created by the user explicitly. Because of this, in RxO database both relational tables and complex unique objects of different classes can coexist simultaneously (Fig.1); the tables and the classes can be connected by foreign keys.

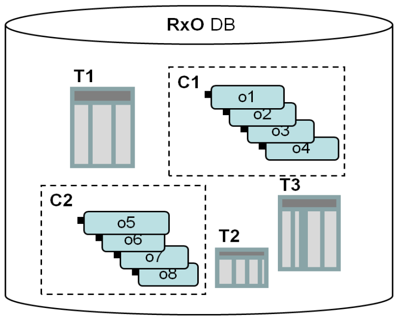

Figure 1: RxO database consists of both tables **T** and object o of classes C.

The possibility to create such heterogeneous data schemes corresponds well to realities of traditional business systems, where the business objects are combined with tables used, for example, for accounting journals, logs, etc.

It also demonstrates that RxO DBMS is an evolutionary extension of existing relational DBMS. Thereby full continuity of existing customer databases and solutions is ensured.

4 CLASSES

Classes of RxO DBMS are an all-sufficient concept for both data modelling and data persistence. This is the fundamental difference from object-relational DBMS (e.g. Oracle), where typed tables must be evidently used to make instances of user-defined type (UDT) persistent.

A set of commands extending the traditional SQL is implemented to describe classes and manage user objects in the prototype "RxO system". Let's review these commands.

The class interface is separated from its implementation. The state of any object is described as a set of values that are native for relational database, namely, scalar values and relation values. Accordingly, the interface of objects is described as set of both scalar and table-valued attributes which represent the state of the object, and methods used to change this state. Relational integrity constraints (keys and foreign keys) can be specified in classes, but the keys are not mandatory for classes, because all objects are unique in database by default.

Classes are created by the `CREATE CLASS` command.

```
CREATE CLASS GOODS{
  GoodsID STRING; //Scalar attr.
  Turnover SET OF //Table attr.
  { Buyer STRING;
    Qty INTEGER;
  }KEY unB (Bayer);//Table field key
}KEY unID (ID)//Class key

CREATE CLASS SALES{
  DocN STRING; //Scalar attr.
  Buyer STRING;// Scalar attr.
  ...
  Items SET OF //Table attr.
  { Art GOODS; //Reference
    Qty INTEGER;
  }KEY uArt (Art); //Table field key
  PostIt(inDate DATETIME); //Method
}KEY uniqDocN (DocN)//Class key
```

The implementation must be defined for each class interface element, both for attributes and for methods. The implementation is defined by the `ALTER ... CLASS IMPLEMENT...` command.

The attributes can be implemented in two ways.

- Values of *stored* attributes are persistent in a system like they are persistent in tables.

```
ALTER CLASS SHIPMENTS
IMPLEMENT DocN, Items AS STORED;
```

- *Calculated* attributes are described in a way similar to descriptions of views or user-defined functions.

```
ALTER CLASS GOODS
IMPLEMENT Turnover AS
BEGIN
  RETURN SELECT
    Buyer, SUM(Items.Qty)
  FROM SHIPMENTS
  WHERE Items.Art = this
     GROUP BY Buyer
END;
```

Both stored and calculated attributes can exist in the class simultaneously.

The method implementation expressions look similar to user-defined functions.

```
ALTER CLASS SHIPMENTS
IMPLEMENT PostIt(inDate DATETIME) AS
BEGIN ... END;
```

The `ALTER CLASS ... IMPLEMENT ...` command allows us to change the implementations of the class interface elements freely in non-empty classes without rebuilding the system. Multiple class inheritance is possible. The class implementation may be changed during the inheritance.

Thus, objects in RxO DBMS unite the properties of tables, views and stored functions, what also corresponds to the principle "object = relational database". The property of data persistence (most important for DBMS) is associated with the implementation of individual attributes and completely separated from the classes interface; it's possible to say that the persistence encapsulated in classes. This feature gives a new way of using and combining stored and calculated data to create expressive business models. Also it gives a new look on some important principles of effective data storages. For example, if the data scheme meant only stored data, then the table attribute `Turnover` of class `GOODS` would be redundant against of the class `SHIPMENTS`. In RxO scheme such redundance is avoided due to the possibility to implement the attribute `Turnover` as calculated from the class `GOODS` data. In this way RxO DBMS allows user to focus on expressive business model first of all, without hard compromises between the expressiveness and an optimization of stored data.

5 OBJECTS

All objects in RxO DBMS are persistent and exist from the moment of creation till moment of explicit destruction. Objects are created with `NEW` command.

```
NEW SHIPMENTS WITH BEGIN
 DocN := "Doc01";
END;
```

Let's note that a reference to the created object is not saved in this example. RxO DBMS implements the principle "a class is a named set of persistent objects". Any object is available as a member of the class, even if there is no reference on it. Because of this, there is no need to explicitly create any additional structures to access the persistent objects ("class extent" of the ODBMS model is an example of such additional structure (Cattel, 2000)).

This feature is a basis for non-procedural objects access with database management commands. The command `FOR` is used to manipulate objects. It operates with objects attributes and other variables visible in class and includes known operations corresponding to types of these attributes and variables. For example, this is a command to change the value of stored table attribute of object created by the previous command (here, the `ART` attribute is initialized with a reference to an object of class `GOODS`).

```
FOR SHIPMENTS[.DocN = "Doc01"]
INSERT INTO Items (Art, Qty)
VALUES(FIRST GOODS[ID = "Tie"], 10);
```

The `FOR` command can be used to execute a sequence of action in objects (batch) or class method.

```
FOR SHIPMENTS[...] BEGIN ... END;
FOR SHIPMENTS[...] EXEC DoShip(...);
```

This command also can be used to manipulate with groups of objects, including whole class.

```
FOR SHIPMENTS[.DocN LIKE ...] ...;
FOR SHIPMENTS EXEC ...;
```

6 ACCESS TO OBJECT DATA

Objects joined with references form a nested data structures (Fig.2). At that, the same objects can form different nested structures through different references.

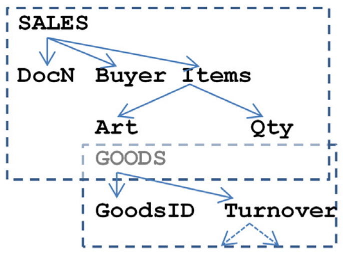

Figure 2: Nested data structure formed by two classes (shown in rectangles).

The data describing the object state can be accessed with traditional relational SELECT queries where the dot-separated path expressions are used. Here are two examples of the queries:

```
SELECT DocN,
       Items.Art.GoodsID,
       Items.Qty
FROM SALES;

SELECT Art.CoodsID,
       Qty
FROM SALES[.DocN = "Doc01"].Items;
```

RxO DBMS analyse these path expressions and automatically transforms the data specified by them to normalized relational view, which processed further in the relational query. We name such a relational view on objects data as "objects views" (O-views). Many O-views can be built from different subsets of the complex structures as on Fig.2.

Name sequences used in queries (for example "Sales" is the name of the O-view, "DocN", "Items.Art.GoodsID", "Items.Qty" - the names of its scalar attributes) unambiguously denote the O-views. Simultaneously, they are processed by the system as a special recording of relational operations required for the calculation this O-view from hidden tables that correspond to objects.

During the calculations, the system performs the late binding of different implementations of the stored and computed attributes of classes. The only requirement is a compliance total sequence of the names used to denote the O-type and each of its attributes; their combination has to give the full path expressions from the root of structure to one of its scalar sheets. Obviously, queries can combine data from classes and tables.

With this, the system can represent all data described in terms of heterogeneous data model including the object data as a set of normalized relations (O-views). These relations are denoted by name sequences, which keep the semantics of original complex structures. Due to this, the automatic transition from the description of complex business models to relational representation of the business model data is imperceptibly for user. This is an important difference from traditional relational and object-relational systems where the only way to represent data in relational way is to evidently describe the data as tables or inside the tables.

For business user it means that traditional reporting on complex business model is possible as soon as the model has been created.

7 COMMENTS ON THE SYSTEM PERFORMANCE

Considering the DBMS used to create an expressive business models, it has a sense to compare its performance not with DBMS, but with existing business system where similar models are possible. Unfortunately, the current prototype does not allow evaluating the performance because it was created to prove the correctness of formal logic and to demonstrate the input OO-commands. However, we afford to make a few comments about it.

Obviously, the main factor that gives a hope of a significant increase of whole system productivity is a transfer of data processing into DBMS, to the place where the data are stored. A practice of manual optimization of business systems shows that the transfer of some operations in the DBMS can speed up these operations tens and even hundreds times (author's practical experience bears out this fact). These dramatic figures are a result of the extremely inefficient organization of data exchange between tiers. The transfer of data processing to the server side removes any issues caused by the data exchange.

Full transfer of the business logic into the database lets us to think about new possibilities to optimize the data processing in multi-user environments.

- An opportunity to perform a preliminary analysis of complex transactions algorithms appears. This analysis can be used, for example, to predict and prevent possible deadlocks.
- A business logic execution on the server side can minimize long transactions; the principle "one command = one transaction" looks like an ideal.
- It is possible to serialize transactions on business-objects level. For example, instead of locking many rows composing some object, the only ID of this object can be locked. With this, the DBMS load because of exact data locking can be reduced.

Another factor, which can improve performance during processing groups of objects, is a possibility to transform algorithms of processing data in objects built on the principle "an object = a relational DB"(Grigoriev, 2013b). In traditional business systems, the only way to process the groups of object is a cycling over the objects; an action on the objects (e.g. some method) is performed again and again for each object of the group (Fig.3).

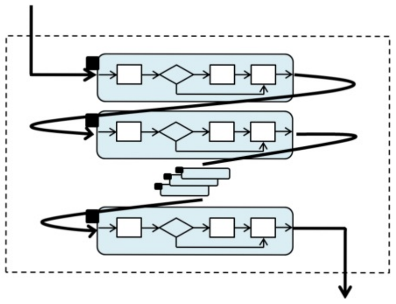
Figure 3: Traditionally, a group of complex objects is processed in a cycle or by means of an iterator, when the each object is processed separately.

If objects consist of rows of different tables, such way of processing leads to repeated access operations to many tables simultaneously to read and write short pieces of data during a long transaction, what is very hard for DBMS.

In RxO DBMS any algorithm used in a calculated attribute or method implementation, can be converted into such algorithm applied to the database tables, that each its step performs an action on all objects of a given group in one relational operation.

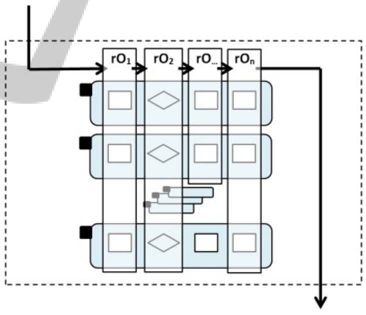

Figure 4: In RxO DBMS operations on a set of objects can be executed in "logical concurrency" mode, when each step of algorithm is executed for all objects inside relational operations rO.

Thus object data can be processed in vector mode (we call it "logical concurrency"). The simplest example is a query that accesses computed table attribute `Turnover`.

```
SELECT GoodsID,
       Turnover.Buyer,
       Turnover.Qty
FROM GOODS
```

This query on the objects data after translation is executed as a single query over the corresponding hidden tables, without any iterators regardless of the number of objects in the `GOODS` class. The described earlier `FOR` command is performed in the same way.

```
FOR SHIPMENTS[.DocN LIKE ...]
BEGIN
  ...
END;
```

Regardless of the number of objects matching the condition given in square brackets, the actions on the object data are performed without cycling over the objects, just by a single call of a stored procedure generated by system after translation code between `BEGIN` and `END`. A technical consequence of this feature is an ability to re-organize and minimize the number of disk access operations data for complex processing of sets of objects.

8 CONCLUSIONS

In the paper an approach is proposed and generally described, which allows relational DBMS to be extended with an ability to create expressive business models inside the DBMS according to principles well-known by existing business systems and clear for business users.

Many important questions are left uncovered, for example language details or the thorough comparison with approaches implemented in existing DBMS. Nevertheless the material presented should be adequate to experienced programmers to visualize the offered approach.

Thinking about possible implementations we understand that many of its features and abilities (including late binding of multiply class implementations, "logical concurrency" mode, preliminary algorithm analysis, object level transaction serialisation) cannot be effectively implemented as a simple "syntax sugar" over existing DBMS and need some modification inside their cores. However we believe that these changes make a sense because they cover most important parts of offered approach and can vastly simplify the business system as a whole, make them faster, and ease access to business models from external programs with standard data access methods.

REFERENCES

Stonebraker, M., Rowe, L., Lindsay, B., Gray, J., Carey, M., Brodie, M., Bernstein, Ph., Beech, D. 1990, *Third-Generation Data Base System Manifesto*, Proc. IFIP WG 2.6 Conf. on Object-Oriented Databases.

Fowler. M., Rice. D, Foemmel, M., Hieatt, E., Mee, R., Stafford, R. 2003, *Patterns of Enterprise Application Architecture*, Addison-Wesley Publishing Company.

Atkinson, M., Bancilhon, F., De Witt, D., Dittrich, K., Maier, D., Zdonik, S., 1990, *The Object-Oriented Database System Manifesto*, Proc. 1st International Conference on Deductive and Object-Oriented Databases, Kyoto, Japan (1989). Elsevier Science, New-York.

Date, C. J., Darwen. H, 2000, *Foundation for Future Database Systems*, Addison-Wesley Publishing Company.

Date, C. J., Darwen. H, 2013, *The Third Manifesto (version dated February 7th, 2013, superseding all previous versions)*, available on www.TheThirdManifesto.com, http://www.dcs.warwick.ac.uk/~hugh/TTM/TTM-2013-02-07.pdf.

Codd, E. F., 1970., *A Relational Model of Data for Large Shared Data Banks*, CACM 13(6).

Booch, G., 1991. *Object Oriented Design with Applications*. the Behjamin/Cummings Publishing Company, Inc.,

Cattel, R. G. G., Douglas K. B., Berler, M., Eastman, J., 2000 *The Object Data Standard: ODMG 3.0*. Morgan Kauffmann Publishers.

Grigoriev, E. 2013a *RxO DBMS prototype*, on-line video, available on www.youtube.com, http://youtu.be/K9opP7-vh18.

Grigoriev, E. 2013b *Object-Oriented Translation for Programmable Relational System*, available on Arxiv.org, http://arxiv.org/abs/1304.2184.