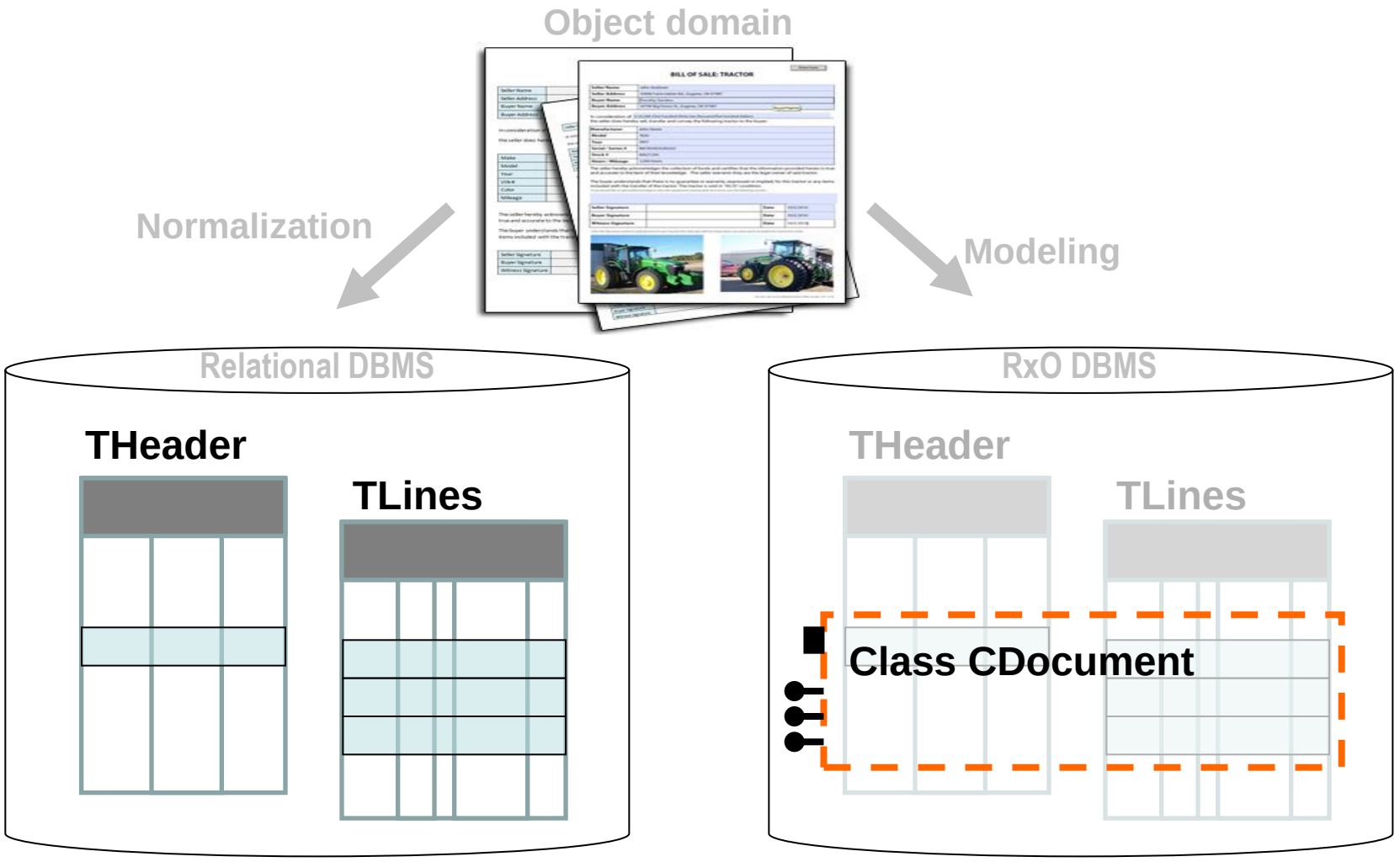
RxO project
Grigoriev.E@RxO-project.com

OBJECT-ORIENTED MANAGEMENT SYSTEMS FOR RELATIONAL DATABASES (RxO DBMS)

Slide list:

- RxO DBMS manages relational data in object-oriented way.
- Complex heterogeneous data scheme
- Complex objects
- Class interface implementation
- How the objects are accessed in RxO DB
- Relational representation of object data
- Architectural metaphor “OO translator for relational target machine”
- Group object processing
- New possibilities: dynamic object reclassification
- Project status and references
- RDBMS evolution towards servers for object domain modeling

RxO DBMS manages relational data in object-oriented way.



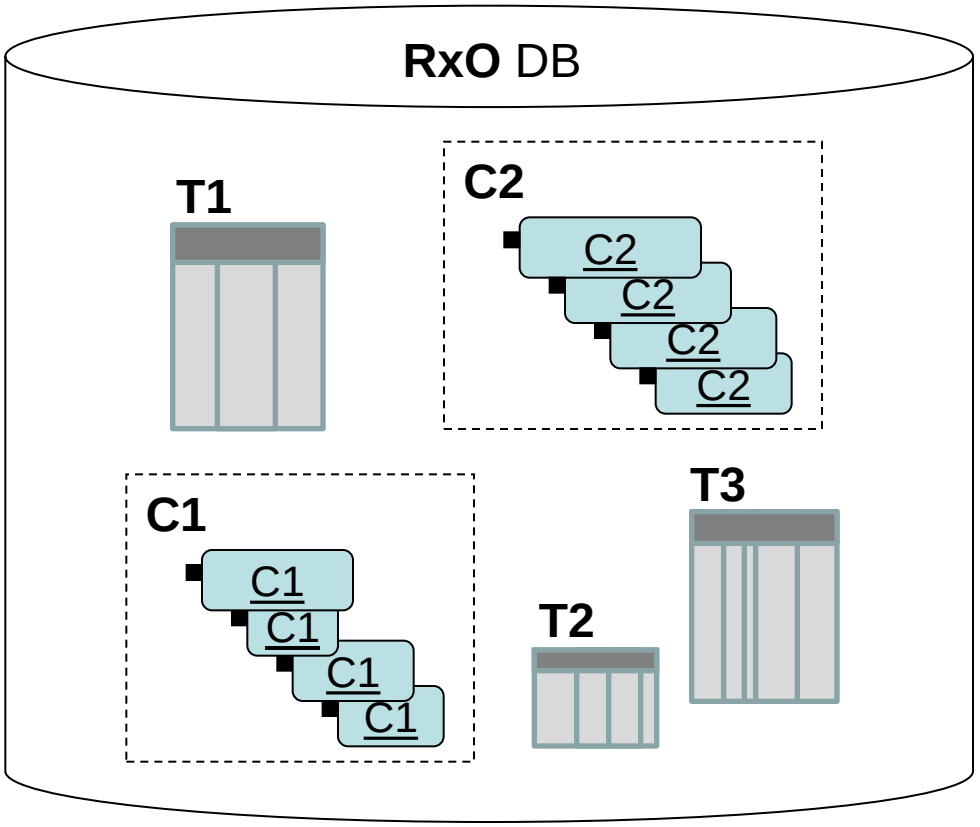
In traditional RDBMS user must keep in mind how complex object data are distributed to normalized tables to manage data.

RxO DBMS can manage <u>associations</u> of relational records of different table structures, which look for users as complex objects

Complex heterogeneous data scheme (or “not only tables”)

For users RxO DB consists of both traditional normalized tables **T** and complex classes **C**, united with both relational keys and object references.

A set of declarative commands, extending SQL, is used to create and change classes and to manage object data. With this, no external OO programming languages are necessary to create model of complex object domain.



Complex objects

In RxO DB state of any object are presented with set of values native for relational systems, exactly domain values and relational values.
Thus, class object interface is defined as a set of scalar (inc. reference) and table properties, which together present the object state, and methods, which are used to change the state.
Optional constraints (keys and foreign keys) can be set for classes.

```
CREATE CLASS SHIPMENTS
{
  DocN STRING;                  //Scalar property
  Cntr CONTRACTORS;             //Reference property
  DocDate DATETIME;             //Scalar property
  Txt STRING;                   // -||-
  PostIt(inDate DATETIME);      //Method
  PostDate DATETIME;            //Scalar property
  Items SET OF                  //Table property (отношение)
  {
    Art STRING;                 //Attribute
    Qty INTEGER; //pieces       // -||-
  }KEY uniqArt (Art);           //Key of table property (class local constraint)
}KEY uniqDocN (DocN)            //Class key (global constraint)
REFERENCE ToUniqArt             //Foreign key
Items.(Art) ON GOODS.UniqArt; //
```

Example of `CREATE CLASS` command (prototype “RxO system”[1] syntax is used here and later)

By means of this command new classes can added in working system when it’s necessary .
Multiple inheriance is possible.

Class interface implementation

Class interface is strongly separated from implementation. Each element defined in interface must be implemented, both properties and method.
Properties (both scalar and table ones) can be implemented as
- stored. Stored properties store their values persistently, like tables do.
- calculated. Calculated properties act like views. This implementation uses SELECT or UDF.

Method implementation is similar to UDF or UDP.
Thus the classes unite possibilities of tables, views and stored functional of traditional RDBMS.

Implementation examples:

```
ALTER CLASS SHIPMENTS
REALIZE DocN STRING AS
STORED;
```

Values of the properties will be stored

```
ALTER CLASS SHIPMENTS
REALIZE PostDate DATETIME
AS
{
  RETURN SELECT ...;
}
```

Value of the property will be calculated

```
ALTER CLASS SHIPMENTS
REALIZE PostIt(inDate DATETIME) AS
{
  IF(PostDate IS NULL) THEN
  {
    PostDate := inDate;
    Txt := "DONE " + DocN;
    IF( DocDate < '2009.01.01') THEN
      Txt := "(OLD!)" + Txt;
  }
};
```

method implementation

By means of this command classes can be re-implemented during inheritance. Also class implementation can be changed in existing non-empty classes without total system rebuilding.

How the objects are accessed in RxO DB

An object in RxO DB exists since it was created till it will de destroyed evidently. All objects can be accessed as elements of their classes (so a class can be defined as “a named set of existing objects of the class”). Due this feature any object is accessible, even there is no reference on it, so references are not obligatory in management commands and non-navigation access to objects is easy possible. References are only necessary to produce complex object structures.

```
NEW SHIPMENTS WITH SET
  .DocN := "In01",
  .Cntr := ONE OF CONTRACTORS[.Name = "Levis"],
  .DocDate :='1999.01.01',
  .Txt := " 1st supply";
```

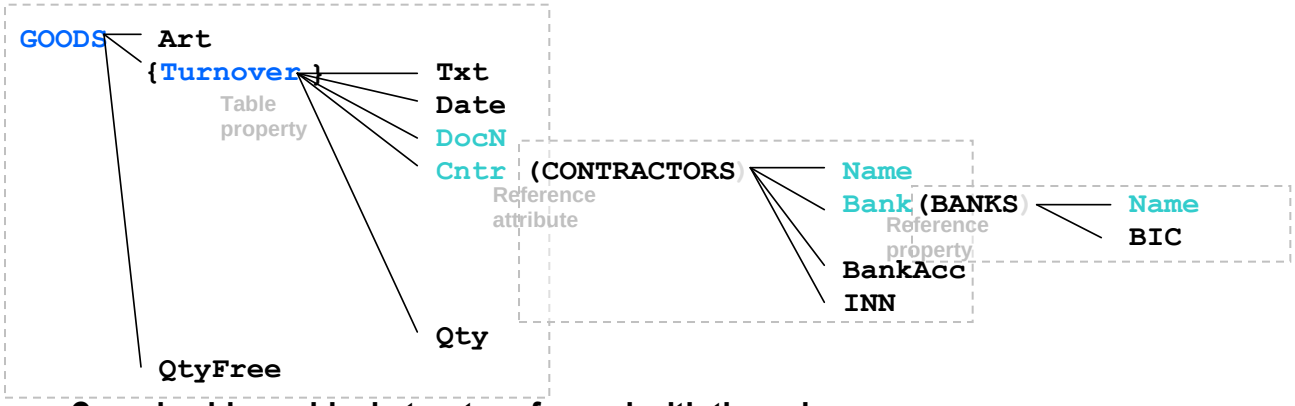
Example of declarative NEW command. Here reference property `Cntr` are initialized with the reference on existing object of class `CONTRACTORS`.

```
IN SHIPMENTS[.DocN = "In01"] INSERT INTO Items (.Art,
.Qty) VALUES("Tie", 10);
```

Example of declarative command change a state of the `SHIPMENT` object created in previous example.

Relational representation of object data

Users can get object data using in relational queries path expressions which describe complex nested object structures.



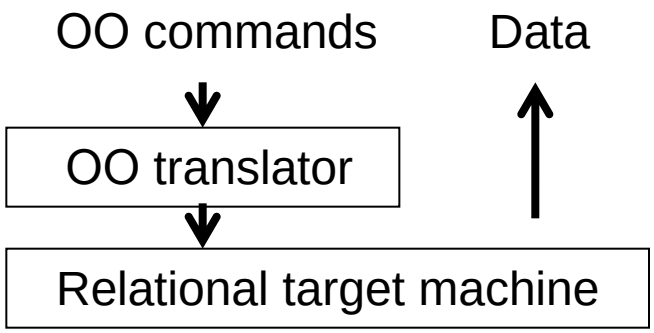
Complex hierarchical structure formed with thee classes.

```
SELECT DocN, Cntr.Name, Cntr.Bank.Name
FROM   GOODS.Turnover;
```

Example of relational query to this complex structure.

RxO system builds relational representation of object data according to used dot-separated name sequences before a calculation of the query itself, in a manner like tables are JOINen manually by key fields in traditional RDBMS [2]. The only requirement is to keep in query the name sequences which form together a full path expression from a root of hierarchical structure to its scalar leafs. Generally, only names is important for users to get data. During such calculation system binds implementations of properties, so result can contains both stored and calculated values simultaneously. If the implementation is changed the query will return new result. So, relational data representation instantly depends on class internals. Queries can combine data from both classes and traditional tables.

Architectural metaphor “OO translator for relational target machine”



OO-translator performs transformation of income OO commands into relational command (or equal actions on data in DBMS).

RDBMS implements both data storage and manipulations on the data and by fact is a target machine in this

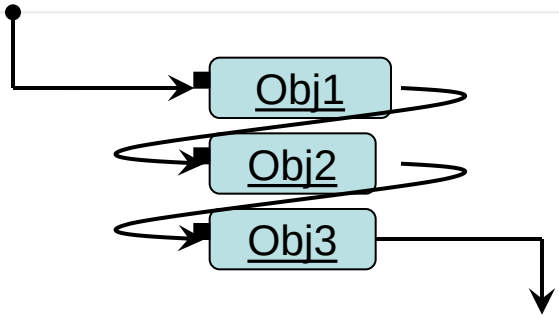
The essence of OO-translation for relational target mashine[2]:

- complex object structure description is mapped into some relational structures descriptions (tables and views). The mapping info are stored into DBMS catalogue to be used as a symbol table during all translation operations,
- all the operations on objects are translated immediately into operations on the relational structures data, based on the mapping info.

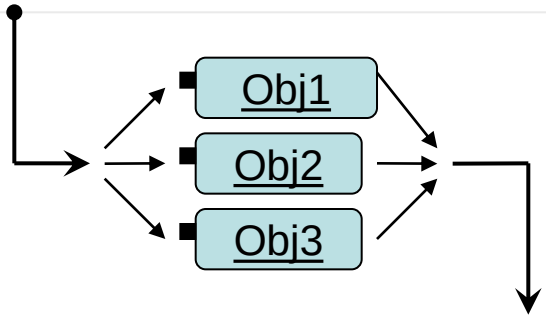
The translation has formal description in terms of classical relational data model. It can be implemented in different ways: in a middleware program independent from RDBMS or by means of additional functional added inside existing RDBMS.

With this, complex data are processed directly in a storage, without data exchange between DBMS and object domain modeling application, what is usual in traditional system architecture. As a result the complexity of informational system is reduced and their rate is rising.

Group object processing



Traditionally, a group of objects is processed in a circle or by means of an iterator, when the action is executed for each object separately.



In RxO DBMS any action on object can be performed for a group of object in "vertical" mode.

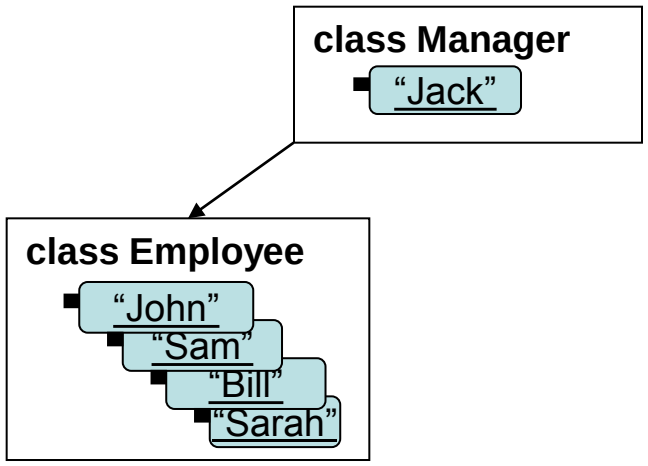
It's shown formally[2] that any implementation of a calculated property or a method can be transformed into a procedure on the underlying relational structures in a way that each step of any source algorithm given in the implementation is performed simultaneously (in single relational operation) on data of group of objects for any group of objects. In this way any data processing given in object term is translated into relational processing.

As an advantage, this possibility lets re-order and minimize disk operations during complex processing of sets of objects, so the processing performs faster. Also it possible seems to be "native" for relational storages with vertical data organization.

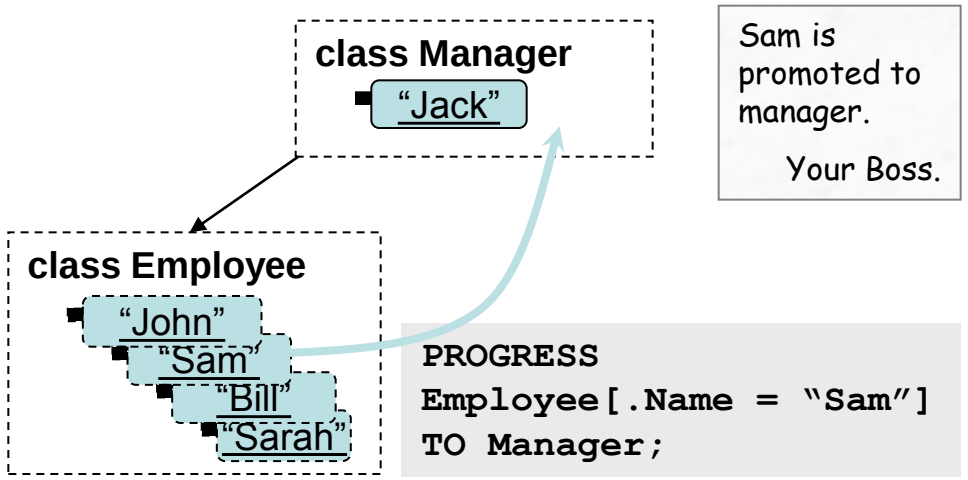
```
EXEC SHIPMENTS[.DocN = "…"].PostIt(…);

SELECT
  #g.PricePL,
  #g.QtyFree      //  ← calculated property
FROM GOODS #g;
```

Examples of commands where "vertical" object processing is used.
1) Method is executed for `SHIPMENTS` objects satisfied to a condition.
2) Values of property `QtyFree` are calculated for all objects of class `GOODS`.

New possibilities: dynamic object reclassification



In traditional OO approach all objects statically belong to classes which there were created in all their life.



In RxO DBMS object can be moved to other class with special commands. The system controls if this movement is possible according to data scheme and existing links between objects.

With metaphor "OO translator for relational target machine" it's easy possible to manipulate with structure of data in memory of the "relational machine", keeping associations of the data. As a result a new level of flexibility of persistent complex objects is reached, when the objects can be moved and transformed inside existing data scheme according to processes existing in modeled object domain [3]. It makes the model more accurate and expressive and gives new possibilities to develop informational systems during their life.

RDBMS evolution towards servers of object domain modeling

RxO approach makes possible an RDBMS smooth evolution towards new kind of systems which fully combine properties of DBMS and rich object domain modeling tools.

For DBMS users RxO DBMS is just further development of existing RDBMS (as a new version), it fully keep all their features, including SQL dialects, access protocols, user control, transactions management, etc. RxO version of existing DBMS can be easily used instead of previous ones in existing user systems and databases. Users get full succession of systems.

For DBMS vendors the RxO approach saves and maximally uses functionalities of existing DBMS. Realization of new features doesn’t require significant changes in system cores. The use of existing functional allows using proven solutions and saving reliability of the relational DBMSs.

At last RxO approach doesn’t try to change relational data model, but uses it as a formal base and implements it in a new way. With this, it’s possible to be sure of correct results of the system activities.

References:

[1] Grigoriev,E., “RxO system. Simple semantic approach for representation complex objects data in table form.” http://odbms.org/download/RxO1.pdf
[2] Grigoriev,E., “Object-Oriented Translation for Programmable Relational System (DRAFT)” http://arxiv.org/abs/1304.2184
[3] Grigoriev,E., ”On PROGRESS operation. How to make object-oriented systems more object-oriented.” http://arxiv.org/ftp/arxiv/papers/1304/1304.3140.pdf

THANK YOU VERY MUCH!

www.RxO-project.com

Grigoriev.E@RxO-project.com
Shevlokov.M@RxO-project.com