Different data storage built on ERP objects

A. Selmeci, T. Orosz
Óbudai Egyetem AMK, Székesfehérvár, Hungary
E-mails: selmeci.attila@amk.uni-obuda.hu, orosz.tamas@amk.uni-obuda.hu

***Abstract*— The ERP systems use tables for storing configuration and application data. The application data consist of master- and transactional data. When an HR expert would like to read employee data, the programs should read many corresponding master data tables and in some cases transactional tables as well. Afterwards the application builds up the employee object and the corresponding objects, like absences, qualifications, etc. Another example could be an invoice object, which contains the item objects and the invoice can be approved, rejected, parked, etc. The running systems are using object oriented thinking, but they convert the data from tables into object or vice versa.**

With our novel approach this conversion can be omitted. In our pilot environment we have defined some low level objects for test purposes. We plan to build some basic functions as well in the close future.

Keywords: **Object orientation, Data storage, ERP, Fast access, Object model**

I. INTRODUCTION

The ERP systems are well-formed, stabile, general solutions. The data models and data storage are different, but similar in that case that each of them are using RDBMS for the persistent layer and the business logic is defined in a procedural manner. The programming environment can be object-oriented, but the business elements, which are part of the daily process, are simple components, modularization units, but not real objects. We discuss in the chapters the object-orientation concept and its effect on ERP systems and present the object-oriented thinking. There is parallelism in data model relations and object relations, which are important to make a possible change from procedural to object based implementation.

II. OBJECT ORIENTATION AND RELATIONS

The object-oriented concept is deeply involved in software programming. However the paradigm can be treated on higher levels as well. The object-orientation is mainly built on some basic principles as pillars. These are the following:

- **Abstraction:** It is the simplification of the elements in the nature to be able to model them in an artificial environment. It means that we dismiss many unnecessary attributes, functions, which are not important for us. If we consider an object like Employee or CoffeeMachine, they have several properties and we can name some functions, which can be used to set, change the status of an Employee or CoffeeMachine. In abstraction we have to know which attributes of an object are important to be able to model the natural object and which functions are needed to simulate the natural behavior. On the other hand we don't have to know how these functions are working internally and we don't know who the attributes and functions are implemented (we do not know how the coffee machine work really internally).
- **Encapsulation:** It bounds the attributes and functions of an object together into one unit to be able to handle the internal state organically. The encapsulation is used to hide the state (property values of the object) and the implementation from the world. This helps to store specific information about the object and control the access to it.
- **Inheritance:** The objects can have several relationships among each other. One special relation between them is the inheritance. The new object can be derived from another object, so they can build a hierarchy. In this hierarchy some of the properties and functions of the original object are shared or inherited. Standard example can be a lamp and a street lamp, which is the descendant of the lamp object itself. The street lamp is a lamp as well, but it has some additional "features" (properties and functions).
- **Polymorphism:** This means several forms or shapes. This principle describes in object-oriented world that some objects having the same function with different implementation or a function occurs in an object multiple times with different signature (so-called overloaded methods). In programming the first one is a useful tool in many cases. We can imagine a booking company, where we can book hotels, flights, or cars for rental. Each of the booking procedures requires the same data, so the interface of the objects for booking is the same, but they execute internally totally different procedures. In the code we can call the same booking function and the environment can decide according to the given object, which implementation should be executed.

These principles are part of the core concept of object-orientation. We did not describe, but the similar objects can be classified and managed together. So we call the

same type of objects object-oriented class and the instances of such a class is an object [1].

We can also define other relations than inheritance between objects, like part-of or has-a. (The inheritance can be written as is-a relation. As we discussed the street lamp *is a* lamp.) The main relationships are:

- Association (has-a): a very loos relationship, but the most popular one. This is the standard connection between objects in the real World as well, like a doctor-patient, man-car.
- Aggregation (part-of, whole/part): special association type, where the objects belong to each other as part of the whole. E.g. wheels and car, employees and departments. The wheels and employees have their own lifetime. If the car is destroyed, the wheels can be used for another car, or if a belonging department is closed, the employee is still employed in the company.
- Composition (part-of, death): the objects belongs together, they are interdependent object, like brain is part of the body. (If the body object is deleted, the brain object should be destroyed as well.) It can be observed as special type of aggregation, but the lifetime of parts are dependent on the whole.

These principles and relationships can be used in data models as well for designing or defining tables in a relational database management system (RDBMS). The classes with the properties can be thought as entity types in data modeling. Each object is an entity from that type. E.g. people class or entity type, and man or woman as instances of class, objects or entities. The connection between these entity types can be defined with entity relationship model using connection types and cardinalities (see later). Here we should think about the severity of connection types and the available relationship between objects or classes. These differences should be handled and described during design. This is another form of abstraction. We can consider two level of abstraction:

1. Real World to data model
2. Data model to object model

The RDBMS implementation can be established using the association between classes and tables, and objects and records. This kind of representation is not object-oriented, it holds only the data model, but it can establish the relations as well. In the entity relationship modeling we can distinguish the relations with two main characteristics:

- Type or category
- Cardinality

The connection types are similar to the object relations, but here we can add more information:

- Hierarchical: an entity type is dependent on the existence of exactly one other entity type. In this case the ID of the referenced entity type is inherited ("used") by the dependent entity type. E.g. Sales office and offered goods. (The good is assigned to the given sales office. It cannot exist without the sales office.)
- Aggregating/Associative: an entity type is dependent on the existence of multiple other entity types. This type is used to resolve many-to-many relationships. The full ID of the referenced entity types is inherited by the dependent entity type.
- Referential: an entity type refers to another entity type but is not identified by the relationship. It is possible to change or eliminate the relationship, but as long as the relationship exists, it is non-optional. The primary key of the referenced entity type is passed onto the dependent entity type as a non-key field.
- Conditional-Referential: the same as a referential relationship, but the relationship is optional.
- Specialization: an entity type that represents a subset of the referenced entity type but also has additional attributes. In this case the referenced entity type is called the generalization, and the dependent entity type is called the specialization. Generalizations and specializations have the same primary key fields but are used to store different information in non-key fields.

The cardinality (n:m) describes the possible entities from the dependent entity type (target) from the viewpoint of the independent entity type (source). Referring to *n:m* relation the value of *n* is generally 1, which means that there is exactly one independent (on Figure 1. left side) entry.

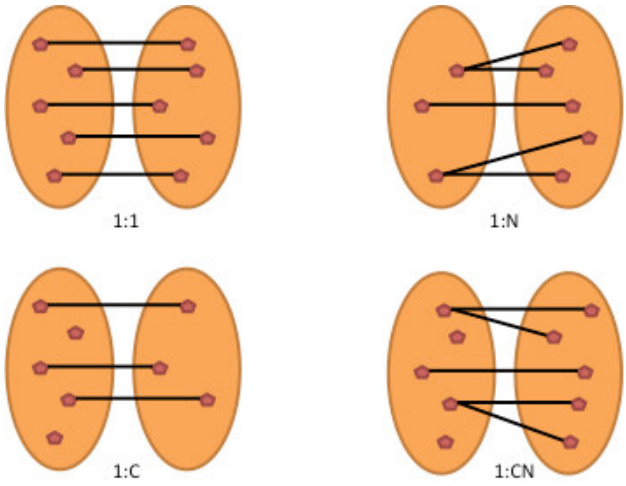

Figure 1 - General Cardinalities

The m value, as the Figure 1 shows can be 1,C,N or CN. Easy (but not really realistic) examples can be for 1:N the mothers – child assignment, because a woman is a mother when she has at least one child, but can have more as well. The 1:CN cardinality is the most popular one. As an example we can take person – car assignment, where one person can own one or more cars, but there are persons, who has no car at all. The 1:C cardinality is an interesting one. A technical implementation can be the ID and the description in a given language. It could be that there is no translation for the given language of some entities, but the others have. The 1:1 cardinality means that each dependent entity is assigned to exactly one independent entity, which would mean in RDBMS area two tables with same key having the same number of

entries, where the corresponding keys values are the same. It would be an extension of the first table.

These cardinality levels are connected with the relation type as well. We have discussed the cardinalities, where each of the dependent (rights) entities is connected exactly one independent (left) entity (n=1). If the n value is C, that means a dependent (right) entity can exist without an independent (left) entity. This kind of cardinality can occur only in case of referential connection type.

As we see the data model is almost similar to the object model in creating relations between entities. Certainly there should be a functional model in case of data model, but the objects are different. The object are not so flat type as the tables in RDBMS, but they contain the functions to be set or read the state of the corresponding data.

III. ERP OBJECTS AND MODELS

We can bring this abstraction from the very deep technical level to higher floor as well and we can define the objects / classes in ERP systems. In ERP system we can speak about objects of business. Some example can be employee, invoice, material, purchase order, etc. These objects in business level have more than only attributes (properties) and functions (methods). There could be a special element of object, the event. The standard, we can say static model knows who should be informed about changes. E.g. a price of a good has been changed, the invoice content should be changed before sending it to the customer (of course it depends on the business logic and policy as well). In the static model the price object should deliver information to each of the objects, which use this object. It means that the object should keep a list of connected object at least to fulfill this requirement. If we create a proper object model, this data mustn't be stored in the object. The events can help here, because the object just publish the state change via this event and any object interested in can catch and handle the event. It should only register for the event. This is a simple publish-subscribe method. In business world we can imagine a process where the employee is hired and some internal orders are created for the different group to prepare the office, laptop, phone, network connection, users in systems, etc. The equipment (like phone, laptop) should be assigned to the employee (association).

Within an ERP system we have to consider the different business modules, which contain several classes connected to each other and some (not object-oriented) interfaces connected to other modules. It is not easy to find the right granularity of objects and define the right functions and properties (abstraction) for them.

The older systems like SAP have their own data and functional models, which is based on the standard procedural thinking. There are many tables containing the master and transactional data with relations according to the RDBMS rules. The functions and many relationship types, categories (explained above) cannot be implemented in the RDBMS, or using an easier way, they are implemented in the applications.

The Figure 2 shows very simplified tables containing sales orders. Two tables represent the order itself with the item entries. These tables refer to the other two separate tables, which contain the material data and the customer list. There is no information on the assets or warehouse in this simple example. The only interesting is that we have more tables for an order. In the ERP HCM (Human Capital Management) or HR (Human Resources) area there are several different entries, which cannot be stored within one single table, because the different characteristics like addresses, positions, children, qualifications, etc. have varying structures. The varying structures cannot be stored within one single table if we do not want to mask the different record entries according the given structure in the table. (It requires a huge programming work during design time and processing resources during run time.)

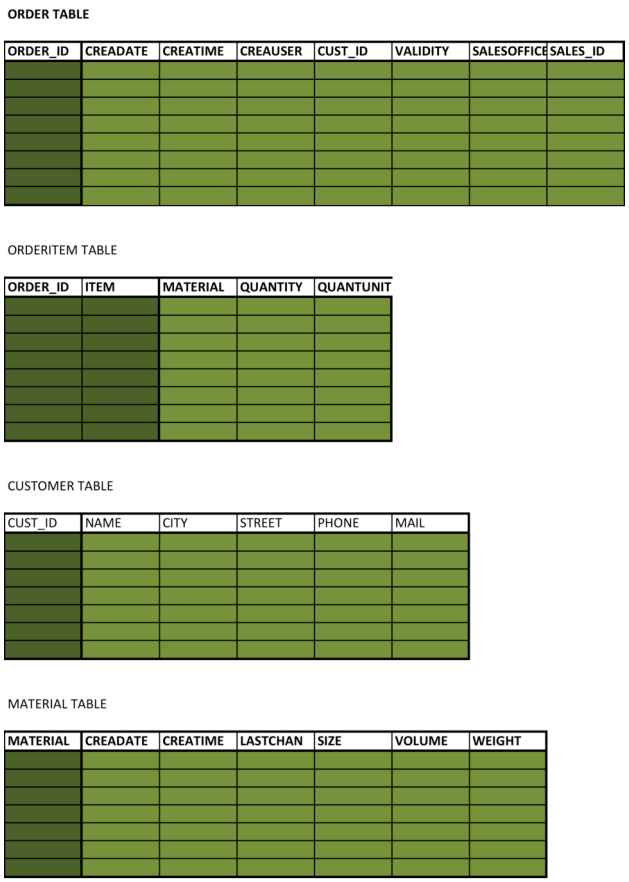

Figure 2 - Simple example from ERP tables

The newer ERP systems are not always built in an object-oriented thinking even if they are implemented using object-oriented programming. Some of them are having real objects from business, like sales order or employee. We referred to the old fashion SAP ERP system already as a system using procedural thinking with data and functional model. On the other hand SAP has implemented a higher level above the data and functional implementation layer. They are called so simple business objects. (These are not the same as the product called BusinessObjects®, which has similar name, but it is a Business Intelligent solution suite by SAP.) These are special programming units written mainly in procedural tools, but having object-oriented behaviors. The naming is not so coherent, because SAP does not use the class word to refer to the structure, but they have the name "business object type". (The instances are the business object.)

These objects have several types of components, like an ordinary object:

- Attributes: there are some attributes for identification, which are called *key fields*
- Methods
- Interfaces: Similar to the standard OOP interfaces containing only definitions without implementation. It is used to add the same attributes, methods, events to different business object types, so they can have the same "interface" for outside callers. The methods should be implemented within the business object.
- Events: for publishing the status change.
- Basic data: Technical data for internal usage.

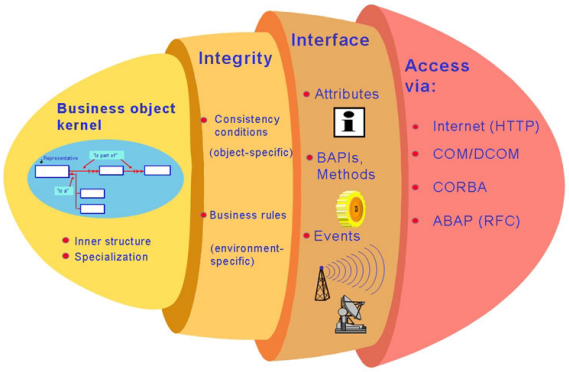

Figure 3 - SAP Business Object Type (c)

As the Figure 3 show, there is a multi-level encapsulation in this case. In the OO world there are visibility sections of an object. These handle the access to the attributes, methods, etc. from other objects (or outside). There are 3 main different sections: private (only internal), public (for everyone), and protected (private from outside, but public for inherited classes). With SAP business objects we have only private and public methods. The other components do not have access-level settings; they are "public" (the attributes are read only). The Figure 4 and 5 show some parts of the standard SAP Sales Order business object as an example. As earlier referred the attributes are divided to two parts, highlighting the key attributes. The Figure 5 shows the methods, where the little green light in the middle sings whether the method is public or not. The green ones are public, which could be called from outside the SAP system as well. These methods are called BAPIs [3]. So SAP did not really implement a real object oriented business model, but tried to define this layer offering an OO thinking and usage.

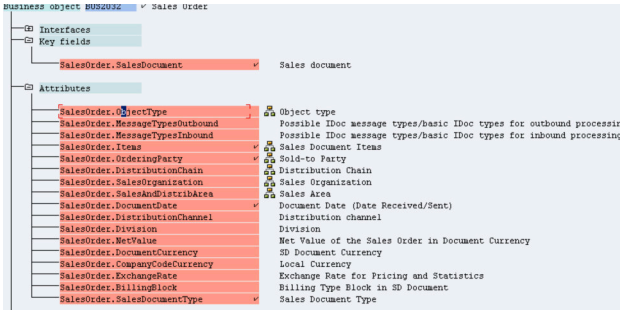

Figure 4 - Sales Order SAP object attributes (c)

The Figure 3 shows some access possibilities from outside. There are libraries for C++, Java and .NET to be able to build externally the object imaging and call the methods of the remote SAP business object via the local object [2].

Figure 5 - Sales Order SAP object methods (c)

With these BAPI methods (signed with green on Figure 5) the external program can get a list of orders (list of the instances of this business object type), create one, confirm one or even change it.

This example helps to understand our purpose in this paper.

IV. DATA ACCESS

The ERP systems store data in RDBMS, which means they use the original data model, like on Figure 2. If we think about accessing the data we have mainly two kind of technique:

- Direct access: creating, changing or displaying (or sometimes deleting) entities by ID or key
- Sequential access: listing data (e.g. sales order list)
- Searching: subset of sequential access, but needs to read according to the attribute values

An ERP system is an OLTP (Online Transaction Processing) type system, so it mainly uses the direct access. The others are interesting only for value helps or consistency. You may say that there should be many reports and searches (queries) in an ERP as well. You are rights, but according to the nowadays technologies for reporting it is better to use data warehouse or analytics above big data environments, so we may skip those access requirements for a while.

Let us suppose that we would like to get the data of a sales order. The simplified database (on Figure 2) requires two reads. One read from the head and another from the item table. In case of SAP business object the standard way to read the status of a sales order from database accesses at least 5 tables. If we consider an employee the system should read more than 10 tables. If we check the opposite way, when we create an object (sales order, employee), so many or more tables are accessed directly.

It means when an ERP system would like to access a business object it should read several tables and build the object itself. This is the point where we see the possible object-oriented thinking for ERP systems. Logically we work with objects, but we always have to separate the content and store them in several tables, or select the corresponding rows from different tables to be able to set up the object.

If check again the parked sequential read problem we have to recognize that we always have functions to generate a list from instances of a class or object type.

Unfortunately these lists are not really useful for queries, because they do not contain the attribute differences, we cannot filter on specific attributes.

V. Possible way

According to our study at the first level we recommend to store objects directly. To store the object we have to have an object store. There are several possibilities to store object directly in a database. These are the Object Oriented database systems (OODBMS) or object-relational database systems (ORDBMS). Both are related to store and handle objects [5].

During our study we have faced some issues. The main problem is the searching. We cannot easily search for objects if we would like to filter on specific attribute values. There are no general tools to do that, because the objects are different. One may have read only attribute, which can be access directly, but other gives the value only via get methods. It could be even worse when more attribute values should be used for a complex search (e.g. those sales orders generated in a time interval are interesting, where the customer lives in a given city. This is a kind of join, but it could work only in ORDBMS environments as join. The other problem is the authorization or access control. According to a query the database can generate the list corresponding to the conditions, but it could contain those object references as well, which are not allowed to touch via the user. It goes beyond our paper currently.

Indexing could be a good solution, but for that we have to build up metadata and create index on them. The access control generates another issue of re-indexing on access change.

As we have wrote already we tried to step forward with the object oriented data storage. The general ERP system architectures have at least 3 layers: database, application layer and the presentation (or user interface) layer. Under the database system there are some infrastructure elements for real, technical data stores. As we see the database layer is handling complex data access, manages files, makes available multiple accesses, manages indexes, etc. [6]. We started in our study to work with an object-storage to make a persistent object layer without any database. Currently we have installed a single node Ceph (https://ceph.com/) object store, where we could figure out the way, how we can leave out the database layer.

We have defined some basic objects and pools. We can handle the data objects and the defined methods as well. Unfortunately the library does not support too much capability of object relationship, indexing or inheritance. Using some metadata can approximate all these functions. According to our study our choosen object store solution (Ceph) does not use metadata server of object store technique. We should consider other solutions as well.

VI. Conclusion

There are not too many OODBMS or ORDBMS solutions in the market. During the last par years the other NOSQL database solutions decreased the popularity of these database systems. We see that the ERP systems should be object oriented, because the information stored there, are mainly object based ones and not function based ones. For statistical and data mining purposes the RDBMS or object-oriented databases are not useful and the World is going to this direction with the big data platforms.

We have faced many issues concerning object-oriented databases. Some were listed in this paper, which are also referred in other papers. The usage of the object store as different data storage is a new direction of ERP systems, which open a new research area.

References

[1] Igor Barbaric, “Design Patterns in Object-Oriented ABAP” *Galileo Press Bonn-Boston*, *2010*

[2] Rich Heilman, Thomas Jung, “Next Generation ABAP Development” *Galileo Press Bonn-Boston*, *2007*

[3] Selmeci, A., Orosz, I., Orosz, T., “SAP BAPI as a break-through and future communication enabler”, 2011. Dec. AIS2011, ISBN:978-615-5018-07-7; 978-615-5018-22-0

[4] Priyanka J. Pursani, Prof. A. B. Raut, “*ITFOOD: Indexing Technique for Fuzzy Object Oriented Database*”, March 2014, International Journal of Advanced Research in Computer Engineering & Technology (IJARCET) Volume 3 Issue 3

[5] Georghe and Bucharest, “*Comparison of RDBMS, OODBMS and ORDBMS*”, Journal of RevistaInformatica Economică, 2007, Vol.4 , No.44

[6] A. Selmeci, T. Orosz, “*Usage of SOA and BPM changes the roles and the way of thinking in development*”, 2012, In: IEEE 10th Jubilee International Symposium on Intelligent Systems and Informatics (SISY), Subotica Szerbia, ISBN 978-1-4673-4751-8