Design patterns of database models as storage systems for experimental information in solving research problems

D.E. Yablokov[1]

[1]*Samara National Research University, 34 Moskovskoe Shosse, 443086, Samara, Russia*

Abstract

The article deals with design patterns of relational databases that are used as storage systems of experimental data. The classification of these patterns based on their complexity and level of detail in the description of entities and their relations is given. For each pattern, specific features of its application in the process of data modeling are shown. Databases created on the basis of simple patterns are less adapted to changes. Their design corresponds only to a context of a solvable task. Databases created using more complex patterns have a more flexible design. It allows considering requirements which can arise in the future that minimizes need for redesign.

Keywords: data model; design pattern; declarative pattern; advanced declarative pattern; contextual pattern; typed contextual pattern; advanced contextual pattern

1. Introduction

An important factor in the design of storage systems for experimental information is the problem of the correct choice of the data modeling strategy. It is a choice of the basic concept which would allow to provide the main context of this information and would be enough flexible for possible extensions [1]. We will mean a set of holistic and systematic ideas can be used to express a certain way of understanding or an interpretation of any objects, events, processes or the phenomena, which have any information value as the concept. Following the definition, we can say that data modeling, as actually selecting the data model, is very important stage in development process of the database. In addition, it lays the foundation of a conceptual framework in terms of which we will work with the storage system. The data model is a kind of database pattern [2], i.e., fixed reproducible means of describing the way to represent and store information. For each pattern, it is necessary to consider the selected abstraction level related to a context of subject domain that in the future will be the data source.

2. Declarative pattern

With this approach [3], the data are described by the principle “as is” (Fig. 1). Many computing applications for experimental research often use a set of elements interrelated by a certain set of connections. For example, an instance of any abstract data structure “graph” may contain a set of entities with the semantics of the graph “vertex” behavior. The relation of these entities to the mentioned above data structure is described by additional entity “graph_vertex”. In addition, these entities can be combined into pairs by using the “edge”, associative entities having semantics of the graph edge behavior. Vertices and edges can represent objects of any kind [4]. Usually they have any characteristic allowing identifying them among a set of similar objects in the description. Moreover, they may have some additional attributes relating, for example, to the description of the position of status of a vertex, weight or direction of an edge. Also they may contain data about the properties of the abstraction, which can be considered as vertex or an edge. Along with the properties directly relating to such concepts as a graph and vertex, the pattern supports attributes of the relations, such as attributes of the associative entities "graph_vertex" and "edge".

Foreign key attributes for relationships between vertices ("from_vertex_id" and "to_vertex_id") indicate their direction in case where data in the form of planar or spatial oriented graph. In the case of undirected graph information on forward edge shall be duplicated only in the backward direction, so that values of "from_vertex_id" and "to_vertex_id" attributes in the row containing information on edge in the backward direction are interchanged. This will allow us to represent data on the connections between vertices of a simple undirected graph in terms of parallel edges of an oriented multigraph.

This pattern is a highly specialized solution, so it is easily to define correspondence between the object of data domain and abstractions used in modeling process. It has the following advantages. First, easy to understand, because a small number of abstractions allows easily model subject domain, with use of simple concepts, which intuitively is clear. Second, easy to support, because of the possibility to manipulate data without the need for knowledge about more difficult features of storage structure. Third, simple queries to fetch data and fast implementation of a data access layer. However, it has some disadvantages, which hinder development of the corresponding databases. The storage structure is initially fixed and has to be changed before adding any new entity or attribute. It will be necessary to introduce the new table as abstraction for the description of some object of data domain or attribute as an analog of its any property. For this purpose, a refactoring procedure like “Introduce New Table” or “Introduce New Column” [5] has to be performed. In case of weak adaptation strategy of the storage system to incoming requirements affecting the content or the quality of an already existing or new information such actions can cause negative consequences.

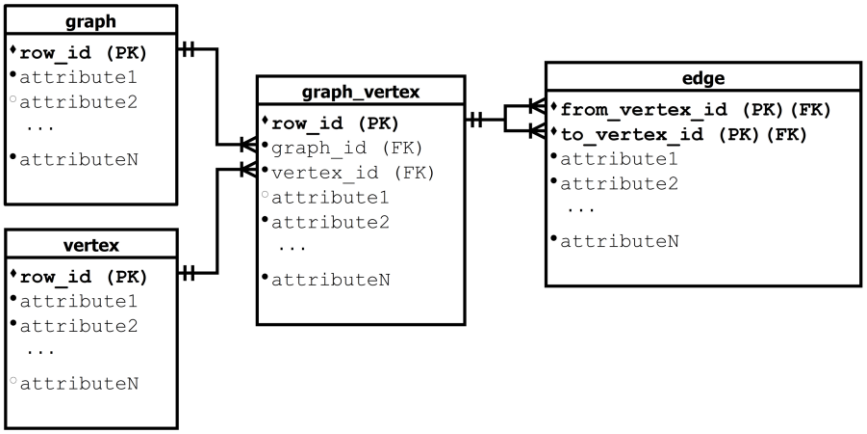

Fig. 1. Advanced declarative pattern.

Using this kind of pattern is passible if change of a data domain context is not expected in the future and development of the storage structure will be done to introduce of sub entities characterizing additional information about the objects already exists in the database.

3. Advanced declarative pattern

This pattern has structure similar to the previous pattern, but realizes the additional indirection level in describing attributes and their values. For example, the data model for storage of crystallochemical information using already mentioned graph abstractions could look as follows (fig. 2).

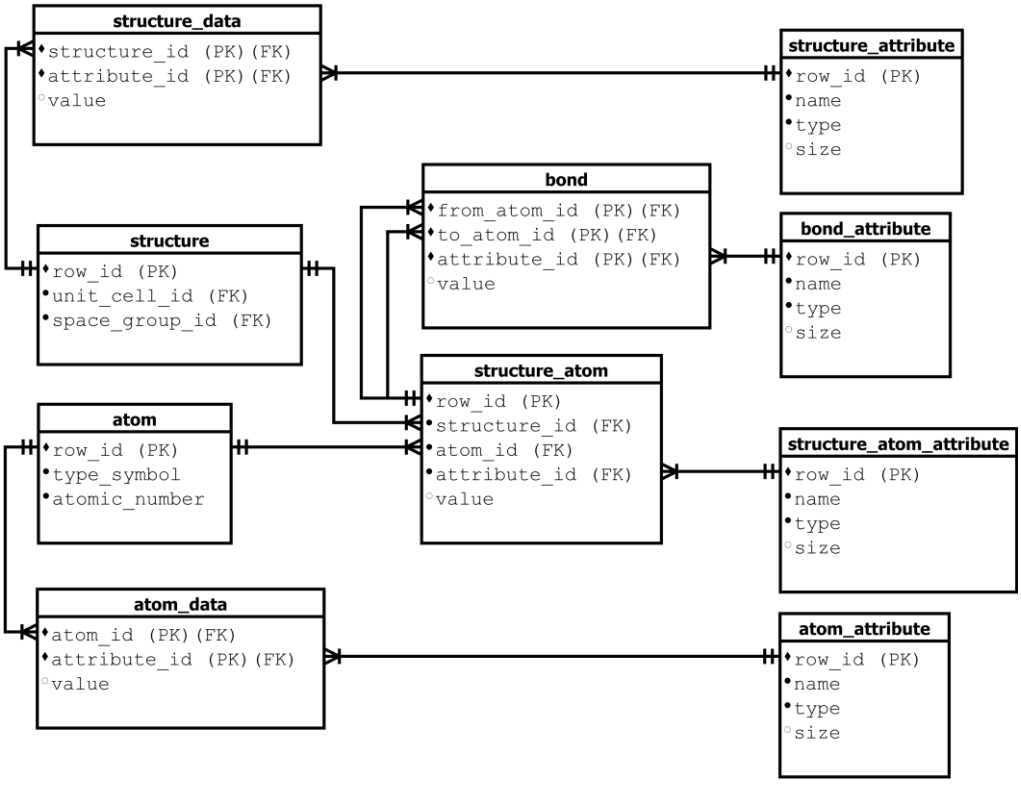

Fig. 2. Advanced declarative pattern.

Foreign key attributes "unit_cell_id" and "space_group_id" of an entity "structure" specify relationships with the entities "unit_cell" and "space_group" describes the concept of unit cell [4] and space group [4]. These data are necessary to unique identification of the chemical structure instance in crystal chemistry. "Type_symbol" and "atomic_number" attributes of an entity "atom" describe the main characteristics of chemical elements from the periodic table. The attribute “value” of the associative entities "structure_data", "atom_data", "structure_atom" and "bond" is needed for store the property values of these entities. Properties without values are described by means the entities "structure_attribute", "atom_attribute", "structure_atom_attribute", "bond_attribute" containing the same set of fields "name", "type" and "size".

This pattern has all the advantages of the declarative approach but has a more developed mechanism for the attribute description. It is allow describing various states of entity instances of different classes and their relationships without

restrictions. If new data about any characteristics or statuses of chemical structures atoms and their relationships are in the future obtained, this pattern allows saving this information without system redesign.

However, even with the flexibility of the storage structure for attributes and their values the introduction of the new entity is impossible without redesigning storage system entire or its specific part. Descriptions of some attributes can be repeated for different entity classes and this indicates the redundancy of attributes data. Because of the data redundancy there is possible to update the attribute description tor only one entity class. The database will contain different descriptions for identical attributes, and it is potential inconsistency when processing or updating data. In data access applications type conversion operations to cast attribute values to the type specified in the attribute description must be implemented. The scenario illustrating the possibility of using this pattern can be the following. If the number of abstractions used in the modeling process to describe domain objects is constant, but the information related to their properties is changed, then it is motivation to use an advanced declarative pattern.

4. Contextual pattern

This pattern [3] implies independence of information context when different objects can be represented differently depending on a situation. Moreover, the change or assignment of their state or behavior can be made even in runtime. Using associative entities for interrelation between objects and attributes (“object attributes”) and also between attributes and relationships (“relationship attributes”) allows assigning to any object or relationship any context-related number of attributes. Based on the conceptual diagram of the contextual pattern (fig. 3) and its simplistic data model (fig. 4) it is possible to conclude that the description of any data domain independently of the context can be represented in terms of objects, their attributes, object attributes, object relationships and relationship attributes.

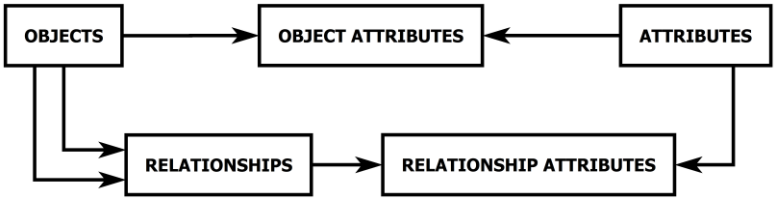

Fig. 3. Contextual pattern (conceptual diagram).

At the description of each object there are “name” and “description” fields for specifying the conceptual context of the instance. For each attribute there is its formal description containing the "name", "type" and "size" fields. Thus, there is a formal declaration of attributes without specifying of the actual values. The actual values of the attributes associated with the object or objects can be defined using the field "value" of the associative entity “object_attribute”. In the same way, the actual values of attributes for entities of "relationship" are defined in the field of "value" of the associative entity "relationship_attribute". Semantics of such approach is very similar to an advanced declarative pattern, but the contextual pattern does not depend on the context of the subject area.

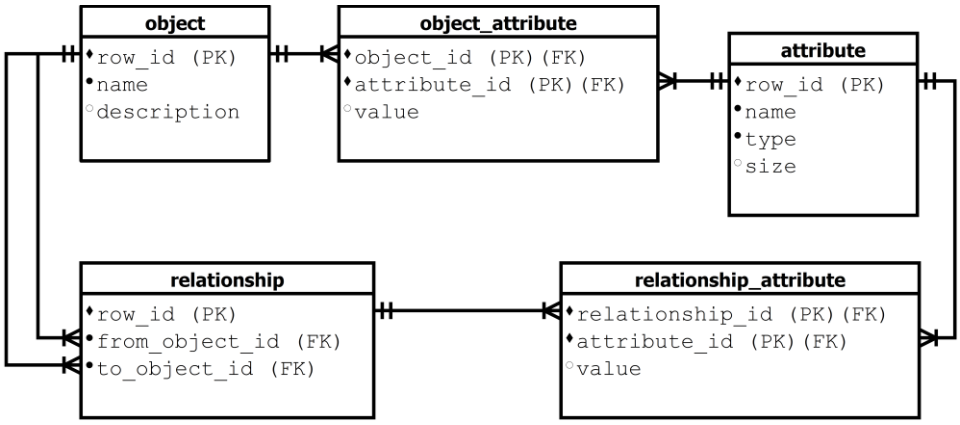

Fig. 4. Contextual pattern (ER-model).

The pattern of this kind is easy to use if you follow a set of implicit rules, but it is necessary to pay for the additional level of flexibility of structure of storage. There is a potential possibility of change the strategy of working with data and there can be difficulties with semantics of storage and data fetching. Not oriented on the subject domain logic, the storage structure will require additional data access level in the form of a set of the user functions or views simplifying access to the information stored in the database. Without the clear specification of the relation of an object to any domain segment, the performance can be reduced when data is fetched because of the large number of objects with an identical set of attributes. The need of information storage about all attributes in one place can lead to data redundancy mentioned in the disadvantages of the advanced declarative pattern.

5. Typed contextual pattern

The pattern solves the problem of missing in the object description the characteristic of its relation to a certain class depending on a context of data domain (fig. 5). By entering user-defined types for objects and their relationships, this kind of pattern allows classifying information about them by means predefined criteria. Creation of such criteria in storage system can be step-by-step, for example, when forming necessary level of understanding of features of data domain. These features could be unknown at the initial stage of working with the database. Moving of information on attribute types to the separate table allows avoiding the problem of data redundancy. The structure of the universal storage [6] created by the principles of the typified contextual pattern can be conceptually partitioned into several main components. First, objects that combine such concepts as object type ("object_type") and an object instance ("object") [7]. Second, attributes ("attribute") and types of attribute values ("data_type") which allow describing signatures of object properties [7] separately. Third, object interrelations associated with types of object relationships ("relationship_type") to formalize the entity class "relationship". Fourth, "object_attribute" and "relationship_attribute" allow storing attribute values of objects and their relationships.

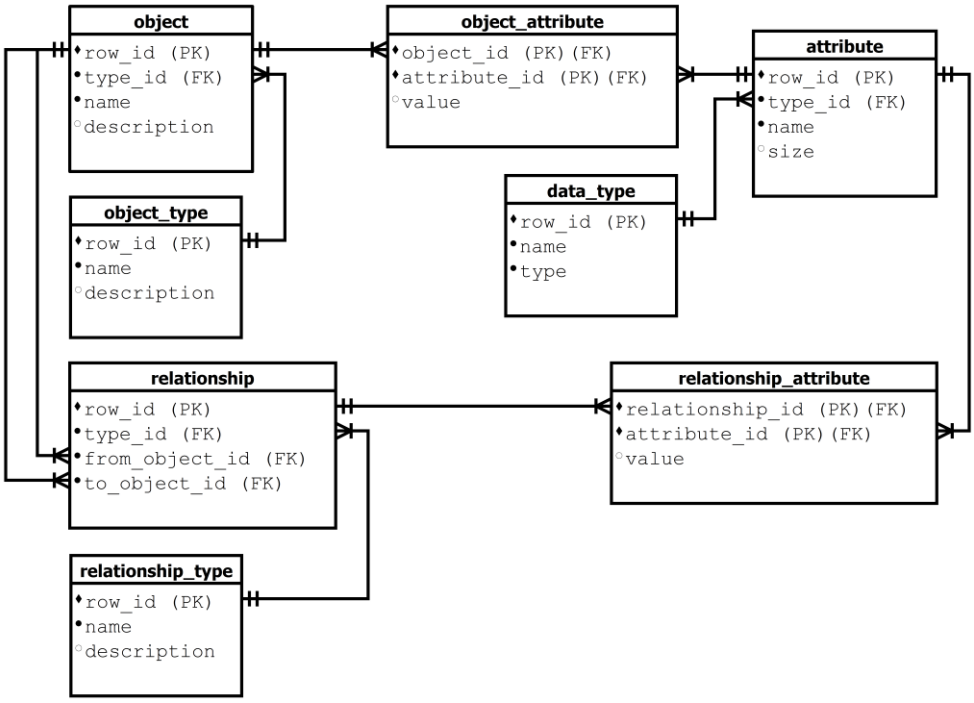

Fig. 5. Typed contextual pattern.

Unfortunately, one of key disadvantages of an advanced declarative pattern complicates use of the typed contextual pattern. Type casting that is necessary for converting attribute values according to their types should still be implemented in the text of queries or views, or in an application at the data access layer.

6. Advanced contextual pattern

In the database created by the principle of an advanced contextual pattern (fig. 6) it is important to define common data for most users primitives based on elementary concepts. This will allow identifying logically associated with these concepts unstructured data independently of the subject area and will provide portability of a data model from one project to another. The detailing level in this approach allows developers of new applications to consider at the early stages of design only the most important issues. All others details of the storage system can be implemented later, when the understanding of problem areas of data domain develops to necessary level.

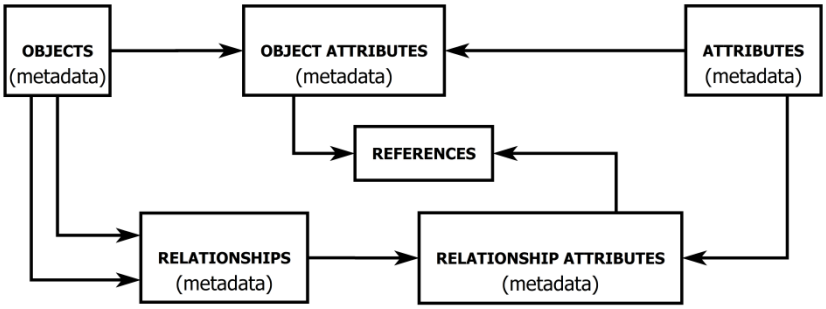

Fig. 6. Advanced contextual pattern (conceptual diagram).

The introduction of additional meta-data level allowed avoiding the disadvantages of the declarative and contextual pattern. The advanced contextual pattern supports all the necessary features to conform to requirements for storage systems based on the universal data model. First, objects description from any subject domain. Second, data representation with using a domain-specific language. Third, data redundancy limitation and support for all possible operations on data processing. Fourth, evolutionary design with adaptation to new requirements and minimal impact on existing data.

Data are described with use of relational approach and bases of object-oriented programming. The main idea is when using a relational kernel of storage system it is extended by the most successful object-oriented technologies. These extensions can be the user-defined type system and the means of describing hierarchical data, such as inheritance and composition which allow to represent relations of objects according to the principles “is a” (similar behavior) or “has a” (part of). Object-oriented approach allows you to represent data in the form of a set of interacting objects, each of them associated with the specific entity class. It promotes the correct and more effective structuring storable information and makes possible to perform an object-oriented decomposition to form the conceptual boundaries of the data model.

As with the typed contextual pattern, the storage structure created by the principles of an advanced contextual pattern can be conceptually partitioned into several main components. The following is a short description of these components is provided, and explanations to use of some categories of the ideas on the basis of which the proposed solutions and methodologies for working with data.

6.1. Objects

Any object is an instance of an entity class and considered as pure abstraction without binding to subject domain (fig. 7). Specification of properties of objects according to the object-oriented approach to the relational storage model is made at the level of object type ("object_type"). According to storage semantics each object type inherits to any base type ("meta_type") which in terms of the elementary primitives to define a context as a criteria for possible classification of all child data elements. The special attention needs to be focused on the field "parent_row_id" in the description of "object_type". This field is needed to create tree-like hierarchical structures, which in the storage semantics of the advanced contextual pattern means inheritance.

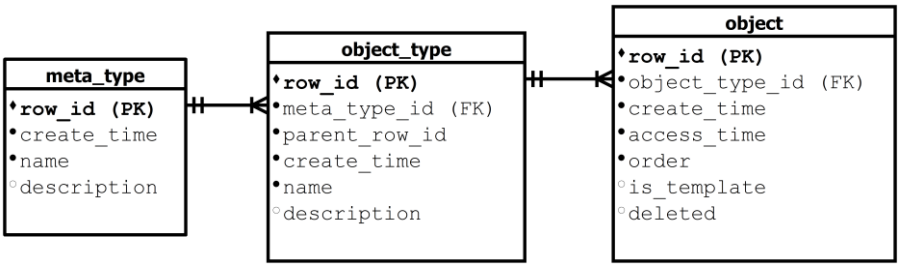

Fig. 7. Advanced contextual pattern (objects).

6.2. Relationships

Each instance of "relationship" is associated with a specific instance of "object_type". Thus, the association of a relationship with a type of objects is set (fig. 8). For example, the bonds type between atoms (“object_type”) may be “HB” (Hydrogen Bond) and base type ("meta_type") can be the "Edge". In the analysis or decomposition of this atomic bond it can be considered in terms of primitive graph abstractions. The description of each relationship contains the link to the “relationship_type” defining the necessary abstraction level for specifying of criteria that separate a concrete relationship instance from other relationships.

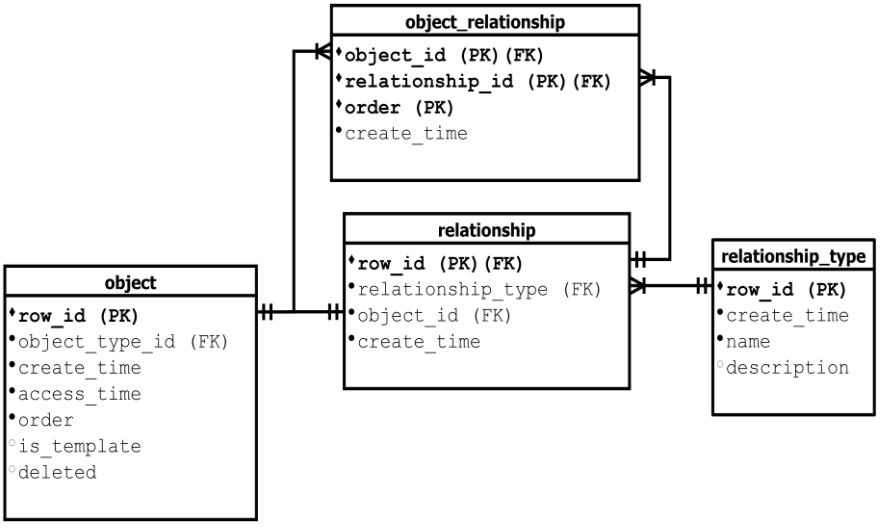

Fig. 8. Advanced contextual pattern (relationships).

For example, the “relationship_type” can specify the form of the relationship between objects on levels such as acquaintance, aggregation or composition. The associative entity “object_relationship” allows creating correlation between parent and descendants, for example, specifying a subset of the graph elements using only the edges or vertices.

6.3. Attributes

Each “attribute” contains the “structural_code” field in the description, specifying the semantics of its storage. It can be as primitive attributes, associated with data types, and composite, consisting of primitive or same composite attributes (fig. 9). The associative entity "attribute_data_type" contains a reference to the data type ("data_type_id" field), and also the "attribute_exid" field, which is used to construct a table alias for storing the values of primitive attributes.

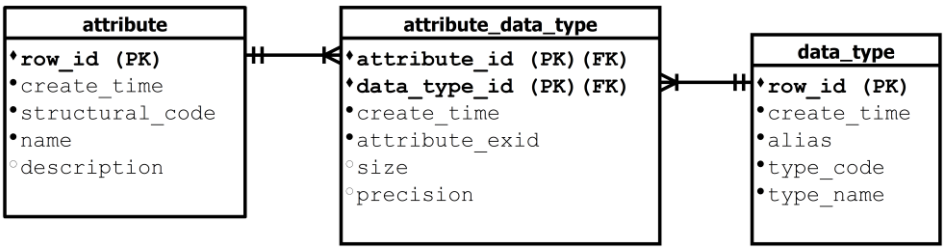

Fig. 9. Advanced contextual pattern (relationships).

6.4. Objects and relationships attributes

Each "object_type" can be assigned any number of attributes (fig. 10). The associative entity “object_type_attribute” allows to specify by means of the “parent_row_id” field what attribute is composite and what of attributes are child of it. By default the access specifier to properties of ancestor always will be public. It means that in the description of object hierarchies the Liskov Substitution Principle (LSP), one of the basic principles for working in object-oriented style is used. Excluding attributes from the descendant scope is possible when the value of the field “not_inherited” is set to “true”, the access level is defined to private.

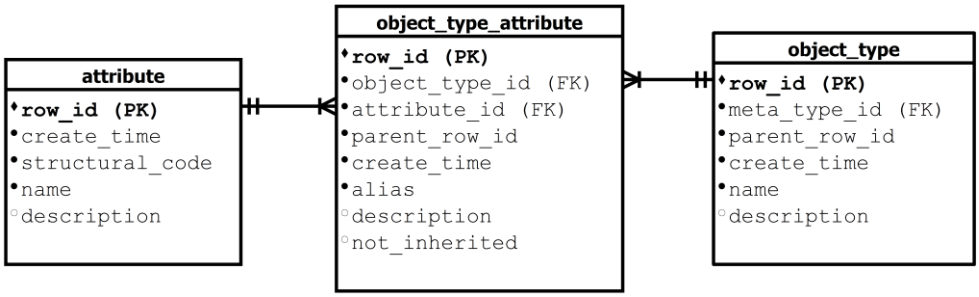

Fig. 10. Advanced contextual pattern (objects and relationships attributes).

6.5. *Attribute values*

The values of primitive attributes are stored in separate tables for example “attribute_value_18F19DCCA4F24B27B3D0BAB50AAE740B” (fig. 11). It allows during the query design not worry about converting of attribute values. Aliases of these tables contain the ID got from value of the field “attribute_exid” declared in the description of “attribute_data_type”.

6.6. Attribute references

The introduced mechanism of references (“attribute_reference”) allows defining the relative value for the attribute of another object or relationship (fig. 12). When organizing dictionaries containing any information or a set of constants used in describing experimental data, references to the values of dictionary records can be used as relative values for attributes of other objects or relationships. This will reduce duplication of information and make the data more normalized without any problems with referential integrity.

7. Conclusion

The selection of a data model pattern is in fact selection of a paradigm of working with data on the basis of the domain model and the used abstraction level. When using any of patterns we get a lot of technologies corresponding to the complexity level of a pattern allow controlling the complexity of subject domain and complexity of storage system as well as the complexity of software for data processing. Whatever was the selected approach it does not replace the experience and style of thinking necessary to choose the correct strategy for working on data processing and analysis projects.

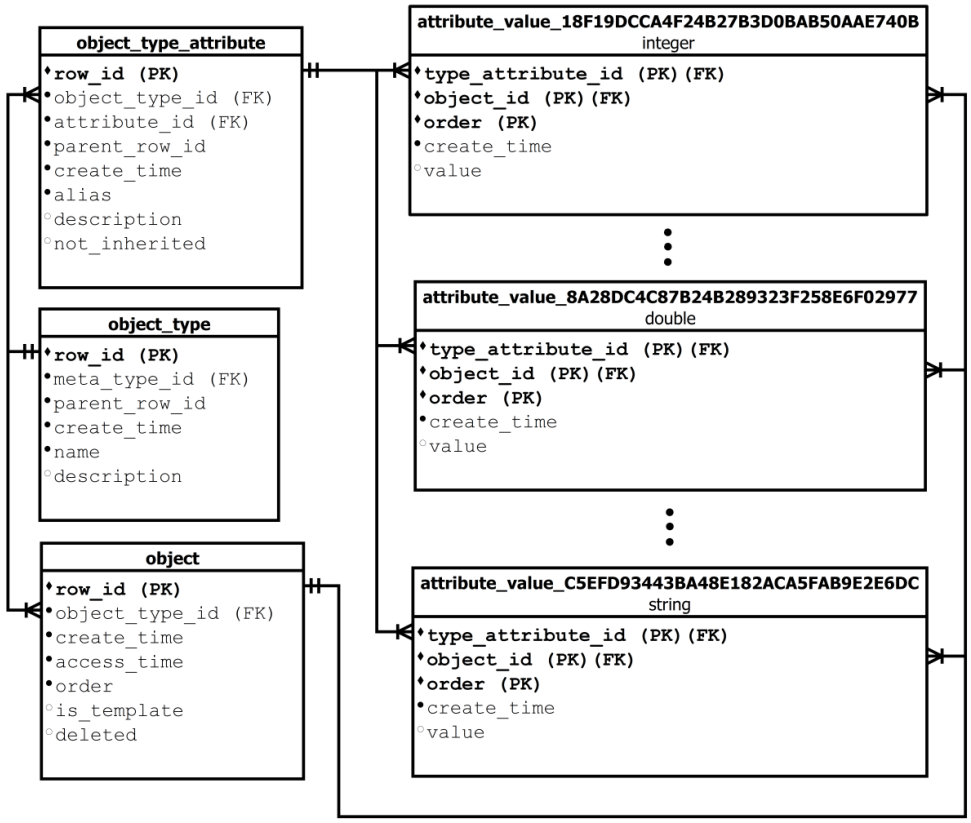

Fig. 11. Advanced contextual pattern (attribute values).

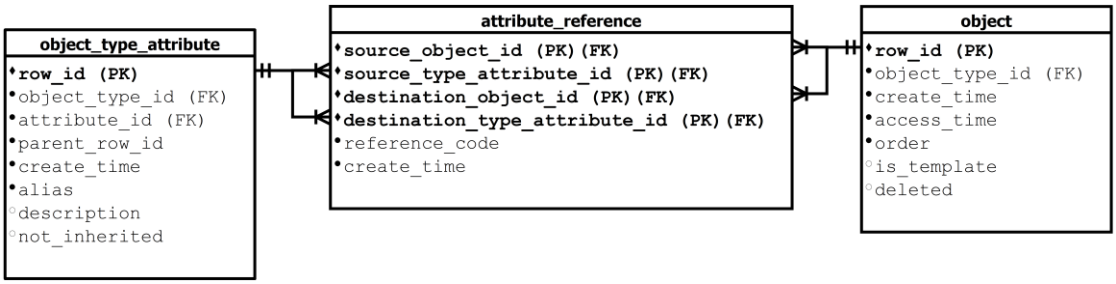

Fig. 12. Advanced contextual pattern (attribute references).

Acknowledgements

This work was supported by the Russian government (Grant 14.B25.31.0005).

References

[1] Simsion GC, Witt GC. Data Modeling Essentials, Third Edition. Morgan Kaufmann Publishers, 2005; 560 p.
[2] Hey DC. Data Model Patterns: Conventions of Thought. Dorset House Publishing, 1996; 288 p.
[3] Silverstone L. The data Model Resource Book. Vol. 3: Universal Patterns for Data Modeling. Wiley Computer Publishing, 2009; 648 p.
[4] Blatov VA, Proserpio DM. Periodic-Graph Approaches in Crystal Structure Prediction. Edited by Oganov AR. Modern Methods of Cristal Structure Prediction. Wiley-VCH, 2011; 1–28.
[5] Ambler S, Sadalage PJ. Refactoring Databases: Evolutionary Database Design. Addison-Wesley, 2006; 384 p.
[6] Silverstone L. The Data Model Resource Book. Vol. 1: A Library of Universal Data Models for All Enterprises. Wiley Computer Publishing, 2001; 542 p.
[7] Fowler M. Patterns of Enterprise Application Architecture. Addison-Weatley, 2003; 736 p.