Математика и информатика | *Volume 65, Number 1, 2022* | *Mathematics and Informatics*

https://doi.org/10.53656/math2022-1-6-inn

Educational Technologies
Образователни технологии

INNOVATIVE PROPOSALS FOR DATABASE STORAGE AND MANAGEMENT

Yulian Ivanov Petkov, PhD student,
Dr. Alexandre Ivanov Chikalanov, Assoc. Prof.
University of Library Studies and Information Technologies – Sofia (Bulgaria)

Abstract. At present, the problem of storing large data sets as a source of artificial intelligence acquires a geopolitical and strategic character. The most well-known and used type of databases so far are the relational (SQL databases) and nonrelational (NoSQL databases. The both approaches have some principle problems, which are described below. That publication presents two original approaches to overcoming some of these shortcomings. First one is Object-oriented model for storing data in a relational database. The second is Storage of non-relational data in a relational database according to previously freely created by the user models. Presented models were used as base for software development of more than ten middle and large size national and European scientific and industrial projects.

Keywords: relational; SQL; non-relational; NoSQL; object-oriented model

Introduction

The most well-known and used type of database so far is the relational (SQL database). The other type of database that uses a non-relational data model is called NoSQL (non SQL / Not only SQL).

SQL databases are widely used – from small amounts of information, for example from a two-page website to large web or mobile applications, blogs, online stores and more. The most famous ready-made content management systems (CMS) support and use relational databases - WordPress, Joomla, Drupal, Magento and others. However, fewer are those that support NoSQL databases (such as Drupal) (Thein and Thwin 2019). A major problem with relational databases is that the presence of NULL values cannot be avoided, which in some cases can reach a significant percentage of the total number of elements. Another problem with relational databases is the difference between the relational and object-oriented data modeling approaches.

NoSQL databases are the common name for various database technologies created for modern applications and the vast amount of information they work with. NoSQL databases solve various SQL constraints for:

– easy scalability on server clusters (horizontal dialing);

– support for different types of data structures;

– use in development with flexible methodologies (agile development).

In this publication, the authors propose two innovative approaches for data storage and management for facilitating database related web portals. Some NoSQL databases may not fully comply with the ACID transaction model. Some NoSQL databases may also not support join operations used in relational databases. The proposed two innovative logical models for modeling of large data sets in general solve the above listed problems(Sharma and Meenu 2012), (Stonebraker 2010), (Ziqi Li 2018).

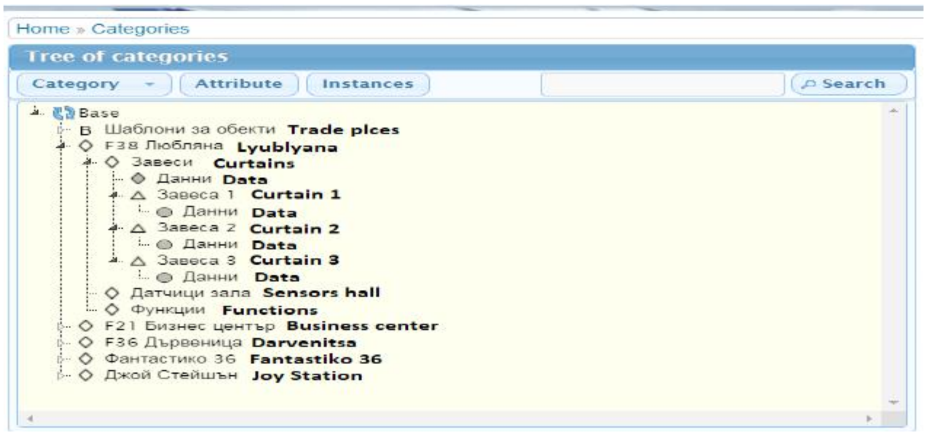
Object-oriented model for storing data in a relational database. That model is based on the so-called root tree of discrete mathematics, strictly following the rules of object-oriented programming. The idea of the model is that a table stores data describing the abstract model of objects and the relationships between them, as well as the type of objects. Each object is represented as an instance of a class. The names of the objects must be unique along the entire branch of tree. A separate table is created for each main class, and the inheriting instances use the table of the parent class. Each class can contain any number of attributes of freely selected types that the database supports. The description of attributes is stored in a separate table. The proposed model supports three types of classes, but everyone is free to use them partially or to enrich and develop models with new types, at their discretion. The relations between classes are:

– Has an empty relation – his successors do not include his attributes. Marked as „◊".

– Has a full relationship – his successors include all attributes. Marked as „♦".

– There is a relationship – includes brothers of type „◊" and „♦". They also inherit the attributes of their fathers. Marked as „Δ".

Fig. 1 shows a graphical presentation of above described relations. The presentation is from a full scaled working industrial project for management of thermo curtains.



```
Base
  B Шаблони за обекти  Trade plces
  ◇ F38 Любляна  Lyublyana
    ◇ Завеси  Curtains
      ◆ Данни  Data
      △ Завеса 1  Curtain 1
        ● Данни  Data
      △ Завеса 2  Curtain 2
        ● Данни  Data
      △ Завеса 3  Curtain 3
        ● Данни  Data
    ◇ Датчици зала  Sensors hall
    ◇ Функции  Functions
  ◇ F21 Бизнес център  Business center
  ◇ F36 Дърленица  Darvenitsa
  ◇ Фантастико 36  Fantastiko 36
  ◇ Джой Стейшън  Joy Station
```

Figure 1. Graphical presentation of part of classes involved in an application for thermo curtains management

Descrition of table forming the model follows:

(1) **category** – Basic table of classes. Contains all class information, as identifier, position in tables tree, relations to neighboring tables in that tree etc.

(2) **attribute** – Basic table for class attributes. Contains all information about an attribute, attribute identifier, class identifier to which it belongs, is that attribute mandatory or optional, attribute name etc.

Storage of non-relational data in a relational database according to previously freely created by the user models. The model allows the storage of complex data structures, as well as complex structures of structures or their projections, as the data is stored in a Relational database. Data structures can be considered as forms. The abstract logical model described in this way is represented in a relational scheme, as only the attributes that have value, such as rows in the table, are stored in the database and thus lead to optimization in their storage. The type of attributes can be arbitrarily chosen from the possible ones, which we will consider below:

(1) **Text field** – a field of type Textarea for storing multi-line text data;

(2) **Hidden field** – field of type Hidden for storage of official data invisible in the forms;

(3) **File** – File type field for storing files. Only the name and the actual path to the storage location of the information carrier are stored in the database;

(4) **Image** – Image type field for storing images. Only the name and the actual path to the storage location of the information carrier are stored in the database;

(5) **Date** – Date field for storing data in date format (YYYY-MM-DD);

(6) **Time** – field of type type for storing data in hour format (HH: mm);

(7) **Date and time** – field of type DateTime for data storage in date and time format (YYYY-MM-DD HH: mm);

(8) **Check box** – a field of type Checkbox for storing selected from the list of possible choices;

(9) **Radio** – field of type Radio for storage of data for single selection from the list of possible choices;

(10) **Select** – a field of type Select for storing data from a list. Both single selection and multiple selection from the list of possible choices are supported.

Each attribute can be assigned a default value as well as a comment field. Attribute names must be unique throughout the database. Bookmark, Radio, and Selection attributes can also use so-called external values selected from standard application tables. There can be multiple attributes of the same type in the database. The data is stored in the database by means of forms freely created by the user (models) with attributes selected by him. This contributes to the ability to create arbitrary forms without the need to add program code and create new tables in the database. The forms are subject to grouping by type. The type of forms is a classifier. For example, Industry, Sports, Finance, etc. Each type can contain many categories in itself, which appear as its specialization. For example, for Sports, the category includes

Football, Basketball and others.

The data is fully indexed. Data items are extracted through complex queries. It is permissible to build complex relationships between different forms for statistical analysis or other purposes without the need to create SQL queries.

To better explain the model, we will start with a complete description of the tables that make it up:

(1) **tool** – Basic table for attributes. Contains all information about an attribute, such as tool_id (unique automatically generated number), name (unique attribute name), external (indicates whether an external table is used), value (default value), type (attribute type from described above) and status (attribute status);

(2) **tool_value** – Additional table with values for the attributes of type "Bookmark", "Radio" and "Selection". Contains all value information, such as tool_value_id (unique automatically generated number), tool_id (attribute relation), name (unique value name), value (default value), sort_order (visualization order) , status;

(3) **type** – Basic table for types. Contains all information about a type, such as type_id (unique automatically generated number), name (unique type name), sort_order (order of visualization), status (type status);

4) **category** – Basic table for categories. Contains all information about a category, such as category_id (unique automatically generated number), type_id (information about which type it belongs to), name (unique category name), sort_order (order of visualization), status (status of category);

5) **repetend** – Basic table for cyclically recurring periods. Contains all information about them, such as repetend_id (unique automatically generated number), name (unique name of the period), value (value for the period), sort_order (order of visualization), status (status of the period);

6) **tab** – Basic table for sections. Contains all information about a section, such as tab_id (unique automatically generated number), name (unique section name), status (section status);

(7) **form** – Basic table for forms. Contains all information about a given form, such as form_id (unique automatically generated number), type_id (information about which type it belongs to), category_id (information about which category it belongs to), repetend_id (information about this to which period it belongs to), name (unique name of the form), tab_order (order of tabs when formatting for text attributes), max_value (indicates the maximum value for the attributes in it), sort_order (order of order when visualizing the form), status (status of the form);

(8) **form_tab** – Additional table for forms. Contains all the information about the sections in a given form, such as form_id (information about which form it belongs to), tab_id (information about which section it includes), multy (information about whether the section includes multiple pages), col_num (information about this how many columns form the partition when visualizing the shape);

(9) **form_value** – Additional table for forms. Contains all information about

the attributes in a given form, such as form_id (information about which form it belongs to), tab_id (information about which section it belongs to), tool_id (information about which attributes it includes), position (information about which column it belongs to), numrows (indicates the number of rows for text attributes when formatting for whether the attribute allows comments (max_value) (specifies the maximum value for the attributes in it), sort_order (sort order in form preview), sort_order (sort order in form preview);

(10) **form_data** – Basic table for records. Contains all information about a record, such as form_data_id (unique automatically generated number), form_id (information about which form it belongs to), customer_id (information about which user added the record), for_customer_id (information about which user refers to-record), pscore (indicates the value of the selected attributes in the record), date_added (indicates the time of adding the record), status (status of the record);

(11) **form_data_list** – Additional table for records. Contains all information about a page in a record, such as form_data_id (information about which record it belongs to), tab_id (information about which section it belongs to), body_id (information about page number), customer_id (information about which user has added the page), text (page title), date_added (indicates the time of adding the page);

(12) **form_data_value** – Additional table for records. Contains all information about the attributes with a value in a record, such as form_data_id (information about which record it belongs to), body_id (information about which page it belongs to), tool_id (attribute number information), tool_value_id (information about attribute value number), value (data according to the attribute type);

13) **statistics** – Basic table of forms for statistical analysis. Contains all information about them, such as statistics_id (unique automatically generated number), name (unique name of the statistical analysis), sort_order (order of visualization), status (status of the statistical analysis);

14) **statistics_value** – Additional table for statistical analysis. Contains all information about the sections of the statistical analysis, such as statistics_value_id (unique automatically generated number), statistics_id (information about which statistical analysis it belongs to), name (unique name of the section of the statistical analysis), value_avg (information for average section value (value), value_max (information about the maximum value of the section), sort_order (order of visualization);

15) **statistics_to_tool** – Additional table for statistical analysis. Contains all information about the attributes forming sections, such as statistics_value_id (information about which section it belongs to), statistics_id (information about which statistical analysis it belongs to), tool_id (information about attribute number);For greater clarity, we will illustrate the principle of operation in the following example, considering the possible variants between Relational Databases, Non-Relational Databases and the Proposed Model.

Suppose we have 30 custom forms with 50 attributes, some of which contain multiple-choice attributes.

Non-relational database

Contains one table in which the data is recorded and one row for recording. Disadvantages – each record must contain a complete description of the format to which it belongs and a complete set of attributes to maintain data integrity. Also, when searching, each record must be uncompressed and checked, as indexing and relation are not supported.

Recording example.

{ID:1, form:1, attr1:{aID:11, aID:4, aID:8}, attr2:"…", attr3:"…", ….., attr50:"…"}

Relational databases

Contains 30 tables with 51 columns, the first being for the unique record ID. Each table has an additional table with 3 columns, due to the multiple-choice attributes. The first column contains the record ID, the second contains the attribute ID, and the third contains the ID of the selected value. Easy and fast search, but only when it comes to searching in one table. The union and the section is a complex case.

Pre-freely created by the user model

First we create the attributes we need, and if there are attributes that are repetitive in type and value, we have only one instance. For example, in 10 of the forms there is the attribute "State", we create it once and use it in the 10 forms. Then we add values to the "Bookmark", "Radio" and "Selection" attributes. We create the types we need and the categories belonging to the types. We create sections and cyclically repeating periods as needed. We can now proceed to the description of the abstract model.

We create the first form, giving it a name, then add the sections and define their arrangement. We specify for each section how many columns it has and whether it supports multiple pages (instances). We add to each column the necessary attributes and their properties in the appropriate row.

Once the model is fully described, we can retrieve data and store it in the database.

This action is repeated until we describe the other 29 forms.

As can be seen from the description, we use only tables 1) to 12) to describe all 30 tables and 50 attributes. The same goes for several thousand tables with several hundred attributes.

Let's look now at how the data that enters the database is actually stored. For each incoming record, a row is added in Table 10). In Table 11) rows are added according to the number of sections and pages that contain attributes with value. In Table 12), only the attributes that contain a value are added.

In conclusion, we can say that the main advantage over non-relational databases is the ability to search without having to unpack the records, which requires a large amount of memory and on the other hand the speed of queries, as the fields are indexed. Compared to relational databases, the main advantage is that it is not necessary to write complex search queries when merging multiple tables (where merging is sometimes even impossible). The number of tables and the number of columns in them is also not insignificant.

Conclusion

After the short presentation and description of mentioned two innovative models for data storage and management, as well as the examples attached to them in this chapter, we can draw the following conclusions:

– The developed object model for data presentation realizes on a logical level the main relations of the Object-Oriented Programming. These relationships are realized by avoiding the use of object-oriented databases, which were experimented at the beginning of the millennium, but were rejected by the industry due to the complexity of their use and maintenance. The proposed logical model can be physically implemented through all Relational databases. An intuitive visual tool for managing objects of user classes, which are included in the above-described relations, has also been developed.

– The developed model for presenting non-relational data combines the good practices of Relational Databases and Non-Relational Databases, while omitting some of their shortcomings. The model allows the storage of complex data structures, as well as complex structures of structures or their projections, as the data is stored in a Relational database. An architecture has also been developed for a rapid transition from a visual interface to a data warehouse and vice versa. The proposed model eliminates data with NULL value, which increases the capacity per unit of long-term memory, as well as the speed of data access.

A short list or European projects implemented on the base of proposed models includes:

(1) H2020 BOWI: Boosting Digital Innovation in Europe (bowi-network.eu), AgeWare Project.

(2) EIT Climate KIC TRANSFORM Project: Smart Climate KIC City Transformation.

(3)H2020 ACTIVEAGE: ACTivating InnoVative IoT smart living environments for AGEing well (SofiaPilot), (www.activageproject.eu).

(4) H2020 Cross4Health project (cross4health.eu).

(5) FP7 ICT “Experiential Living Labs for the Internet of Things” – ELLIOT.

REFERENCES

SHARMA, V. & MEENU, D., 2012. SQL and NoSQL Database. 2012. *International Journal of Advanced Research in Computer Science and Software Engineering*. **2**(8), ISSN: 2277 128X.

STONEBRAKER, M., 2010. SQL databases v. NoSQL databases. *Computer Science Commun. ACM.*

THEIN, M. & THWIN, M., 2019 Relational Databases. Available from: doi.org/10.2307/j.ctvc77jrc.15

ZIQI LI, NoSQL Databases 2018. Available from: DOI: 10.22224/gistbok/2018.2.4.

✉ **Yulian Ivanov Petkov, PhD student**
University of Library Studies and Information Technologies
Sofia, Bulgaria
E-mail forexformat@abv.bg

✉ **Dr. Alexandre Ivanov Chikalanov, Assoc. Prof.**
ORCID ID: 0000-0002-4281-1376
University of Library Studies and Information Technologies
Sofia, Bulgaria
E-mail a.chikalanov@unbit.bg