Metamodel describing a relational database schema

Matus Chochlik, Jozef Kostolny, Penka Martincova

***Abstract*—** In this paper we present the design of the foundation of an expert system for clinical interpretation of "-omics" data using methods of big data processing. Such an expert system should be able to classify tumors processing vast volume of input data of different nature (demographic, genomic, clinical, epidemiological, diagnostic) and to be ready to append data in real-time, to update itself, to learn, and ultimately to be an instrument for clinical oncologists in evaluating predispositions and disease prognosis and for selecting the most appropriate therapy.

***Keywords*—** metamodel, metaprograming, database, genetic data

I. Introduction

Metadata-driven programming and metaprograming can be used in a broad range of programming tasks. We have implemented a metamodel which contains metadata describing various aspects of a relational database and can be used for several di erent purposes like dynamic generation of application user interfaces, dynamic generation of SQL, visual exploration and visualisation of the database, automatic generation of the documentation and of visual diagrams of the database model, etc. The metamodel is stored in the database which it describes and the metadata is accessed through the same interface as the regular data. Some of the metadata is automatically generated during the generation of the source SQL script, by out custom build system, some is extracted from the standard database catalogs and the rest, for example the localized descriptions, has been filled-in manually. The metamodel also implements several functions which simplify the usage of the metadata (some are described below). Using the metadata instead of hard-wiring database model-related logic into applications has the advantage that the applications handle changes in the database much more gracefully and it makes the maintenance or further development of the system easier.

Large national and international studies are currently dealing with an individualized approach in the prediction, diagnosis and therapy of cancer. Individualized approach laid the foundations for the development of a new discipline Personalized medicine. In the past it was customary for all oncological diseases to apply a uniform standard treatment methods -regardless of whether the individual patient's symptoms and peculiarities of the disease. In the recent years, research has produced many changes in medicine - today we can far more accurately characterize cancer disease in each individual and it is still more accurate to describe individual molecular biological tests. Rapid advances in human genomics and postgenomic studies of comprehensive molecular data, collectively called "omics" such as transcriptomics, proteomics and metabolomics, give rise to new possibilities in medicine. These methods are widely used in molecular diagnosis of inherited diseases, infectious diseases, prenatal diagnosis, pharmacogenomics, and not least in the molecular diagnosis of cancer and prognosis. The tumor disease is caused by the accumulation of mutations in different genes in a single cell. These mutations, which may be genetic (inherited from parents) and somatic (obtained in the course of life) have a fundamental effect on oncogenes, suppressor genes or genes that are responsible for DNA repair, allow escape control cellular mechanism leading ultimately to tumor formation.

M. Chochlik,, Faculty of Management science and Informatics, University of Zilina (e-mail: matus.chochlik@fri.uniza.sk).
J. Kostolny,, Faculty of Management science and Informatics, University of Zilina (e-mail: jozef.kostolny@fri.uniza.sk).
P. Martincova, Faculty of Management science and Informatics, University of Zilina (e-mail: penka.chochlik@fri.uniza.sk). .

In the last few years have new generation sequencing (NGS) technologies made possible to produce a large number of studies providing a comprehensive molecular characterization of cancer and led to the discovery of new genes associated with particular tumor types and to identify a large number of recurrent mutations (identical mutations detected in a large number of samples) that have been missed by standard cytogenetic techniques. Post-genomic data provide detailed information on the structure of molecular evolution diseased cells. This is a clinically and pathologically unobservable information that can be used for detailed classification. Structure of gene expression not only provides information about the microscopic molecular activity of diseased cells, but also shows a close relationship to the macroscopic manifestation of the disease, especially the prognosis of the disease, which can not be correctly predicted purely based on clinical and pathological findings. It is therefore necessary to be able to make the best use of genomic, analytical and clinical data in clinical medicine, so we can innovate conventional medicine and open the possibility of new medical care that we could call post-genomic medicine [1].

II. Overview of the database model

The database contains clinical and other medical examination data and a large amount of genomic data. The it provides input for analysis using data mining techniques (decision trees) and other traditional and modern methods of Big data processing (Graph theory and MapReduce). In order to search for hidden dependencies, derived data are used for the monitoring of various types of intergenic interactions and interactions at the phenotype level. For this purpose it is necessary to provide software support for testing a constructing hypotheses associated with prediction, prognosis and therapy of cancer diseases.

The database has been designed and implemented with the following considerations in mind:

- it should be able to handle large amounts of data (in the order of 109 of records) in terms of storage space requirements and database operation performance,
- it should efficiently provide the stored data in a form suitable for statistical analysis and data mining,
- it should provide strong support for semi-automated application GUI generation (further work by the authors related to metadata-based semi-automated GUI generation was described in [2], [3], [4]). and support for dynamic, user-driven exploration and visualization of the database,
- texts in the type-tables (like `diagnosis`, `gene`, `attribute_type`, `examination_type`, `value_type`, `enumeration_type`, etc.), should be translated to database user's preferred language, in order to enable international cooperation in the future,
- physical units of the stored values should be properly handled (see [5] for details on how this can be accomplished in a relational database).
- storing sensitive personal information should be minimized and all such data should be protected from unauthorized access.

The database consists of roughly 70 main relations (tables or views), most of which are shown (with the exception of the relations belonging to the metamodel) in Figure 1. These relations can be divided into several groups:

- *database users* - basic information about the database users and their privileges.
- *localization* - related to the localization of the texts in the database.
- *dbSNP* - data imported from the single nucleotide polymorphism reference database [6].
- *polymorphisms* - list of all known DNA polymorphisms, their classification and relation to genes.

- *basic* biochemistry - nucleic acids, nucleobases and aminoacids referenced by other parts of the database.
- *patients* - basic information about the patients and the members of the control groups.
- *examinations* - physical and medical examinations of the patients (mostly numeric values or intervals including the proper units of measurement).
- *attributes* - various attributes (mostly text or pre-defined enumerated values) of the patients and their anamneses, including personal data which is stored encrypted in the database.
- *ontology* - a simple graph-based categorization of the records in the database.
- *metamodel* - contains metadata describing various aspects of the database itself.

Most of the basic business logic is implemented directly in the database by approximately 180 PL/SQL functions.

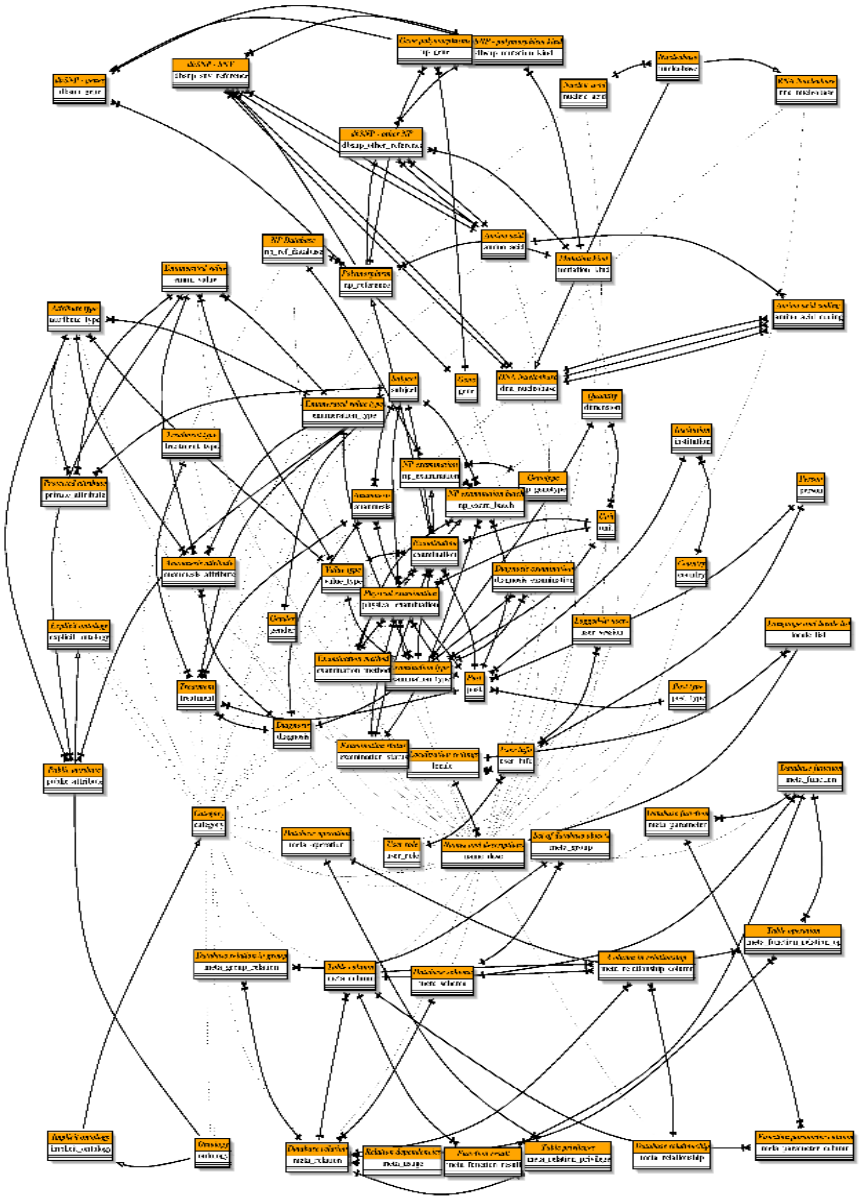

Fig. 1 Overview of the database model

III. The build system

The database source script is generated by our custom bash-based build system (called basql), which translates source files written in a domain-specific SQL-like language into a valid SQL script for a particular database system.

This build system has several purposes:

- it allows to keep the naming of database objects, like table columns, functions, etc. consistent, and makes changing them, if necessary, much simpler,
- it translates symbolic data types like `UID`, `STRING`, `BOOLEAN`, `COUNTRY_CODE` and many more, into concrete data types supported by the target database system,
- it assembles metadata describing the database model and creates SQL statements for storing them in the metamodel,
- it implements the auto-assignment of record identifiers,
- it automates the implementation of helper functions and triggers used to support localization, temporal history, classification of records into the ontology, etc.
- it automatically generates some of the data manipulation functions,
- it implements various types of relationships in the database, since for performance reasons and to simplify maintenance, some of them are not implemented by using foreign keys, and it imports initial data from external formats.
- `basql` allows to keep the implementation of these patterns consistent throughout the whole model, and keeps it portable.

The following code snippet shows an example of a schema source script in the basql language defining the physical_examination table.

```
CREATE  TABLE  physical_examination
WITH REFERENCE_TO subject
WITH REFERENCE_TO examination_type
WITH REFERENCE_TO examination_status
WITH WEAK_REF_TO examination_method
WITH REFERENCE_TO value_type
WITH WEAK_REF_TO unit
WITH ROLE_REF_TO post examiner
WITH HISTORY examination;

ALTER TABLE physical_examination
ADD COLUMN num_value $(FLOAT) NULL;

ALTER  TABLE  physical_examination
ADD  COLUMN  num_value2  $(FLOAT)  NULL;

ALTER  TABLE  physical_examination
ADD  COLUMN  notes $(STRING  500) NULL;

ALTER    TABLE    physical_examination
ADD PRIMARY KEY
  $(TABLE_PK_NAME subject),
  $(TABLE_PK_NAME examination_type),
  $(TABLE_HISTORY_COLNAME    physical_examination);

ALTER  TABLE  physical_examination
WITH SUBJECT_HISTORY examination_type;

CREATE   INSERTER   FOR   physical_examination;
CREATE  UPDATERS  FOR  physical_examination;

GRANT SELECT ON physical_examination TO $(EDITOR_ROLE);
GRANT INSERT ON physical_examination TO $(EDITOR_ROLE);
GRANT UPDATE ON physical_examination TO $(EDITOR_ROLE);
```

This example shows, that relationships are defined by highlevel statements and the details, like determining the names of the primary and foreign key columns and the formatting of the low-level SQL commands to the syntax of a concrete database system is left to the build system.

Also note that `basql` allows to query the real names of primary key and other special columns and substitutes symbolic type name statements like `$(FLOAT)` and `$(STRING 500)` or database role names like `BROWSER_ROLE` with their lowlevel equivalents.

The `WITH HISTORY` and `WITH SUBJECT_HISTORY` statements create additional columns, indexes and functions which simplify querying the records in this table in relation to their date and time. The build system also automatically prefixes the identifiers with the database schema name (`gen_seq.`) where necessary.

The next snippet shows several examples of data source scripts:

```
INSERT  TYPE  attribute_type
WITH ATTRIBUTE str_code 'PERSON_NAME'
WITH ATTRIBUTE $(TABLE_PK_NAME value_type) \
  $(PREFIXED   get_value_type('STRING'))
WITH ATTRIBUTE is_private $(TRUE)
WITH ATTRIBUTE exclusive_enum $(FALSE)
WITH ATTRIBUTE mandatory_enum $(FALSE)
WITH ATTRIBUTE mandatory_string  $(TRUE)
WITH NAME_DESC 'sk_SK' 'Meno' 'Meno osoby'
WITH NAME_DESC 'en_US' 'Name' 'Person''s first name'
;

INSERT COUNTRY 'SVK'
WITH  NAME_DESC  'sk_SK' 'Slovensko' 'Slovenska republika'
WITH NAME_DESC 'en_US' 'Slovakia' 'Slovak Republic'
;

INSERT INTO config ($(TABLE_PK_NAME config), value)
VALUES('GUEST_LOCALE_CODE',   'sk_SK');
```

IV. The Metamodel

The metamodel consists of 15 tables (shown on Figure 2). Some of the metadata is automatically generated by the build system during the generation of the source SQL script, and the rest (mostly the localized names and descriptions) has been filled-in manually.

- `meta_schema` - localized names and descriptions of relevant database schemata.
- `meta_relation` - localized names and descriptions of a database tables or views.
- `meta_column` - localized names and descriptions, ordinal positions, data types and other information about relation columns.
- `meta_group` - localized names and descriptions of logical group of relations.
- `meta_group_relation` - associations between groups and a relations, one relation can belong to several groups.
- `meta_operation` - localized names and descriptions of basic data manipulation operations like INSERT, UPDATE, DELETE and SELECT.
- `meta_relation_privilege` - permissions granted to users to perform data manipulation operations on relations.
- `meta_usage` - associations between views and relations on top of which the views are defined.
- `meta_relationship` - localized names and descriptions of relationships between views and tables.

- `meta_relationship_column` - lists of relation columns forming a particular relationship.
- `meta_function` - localized names and descriptions of database functions.
- `meta_function_result` - the return types of functions and possible relationships with particular relation columns; it can for example indicate that a function returns a value to be stored in a table column.
- `meta_parameter` - localized names and descriptions of function parameters.
- `meta_parameter_column` - relationships between function parameters and table or view columns; it can for example indicate that the parameter only accepts values which are present in a particular relation column, etc.
- `meta_function_relation_op` - indicates which basic data manipulation operations does a function perform on a particular table or view; this information can be used for example to determine, which functions can be invoked by a particular database user, depending on their privileges.

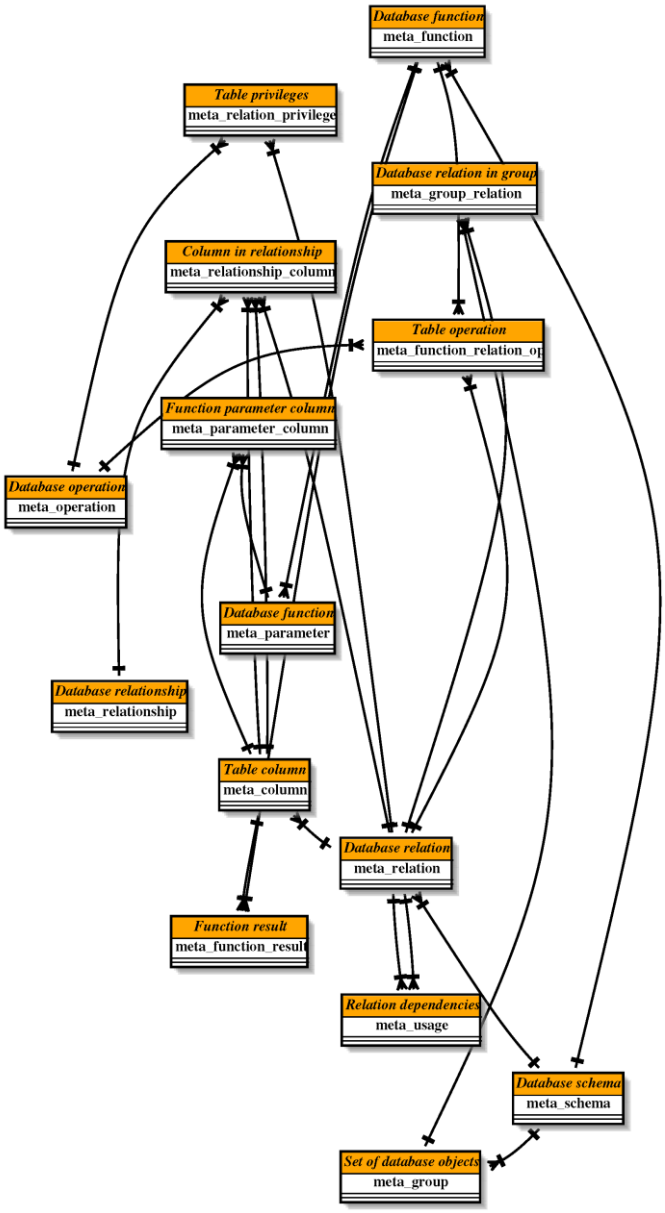

Fig. 2 Overview of the database metamodel

The metadata can be used for several different purposes:

- Automated generation of the documentation and of visual diagrams of the database model.
- Dynamic generation of application graphical user interfaces. For example an application serving as a front-end for PL/SQL functions implementing the business logic can query the metadata describing a particular function, including its localized name, the return value, the list of parameters and their names, types, ordinal positions, etc. and possibly their relationships to table columns and dynamically generate appropriate visual components and in case of parameters with enumerated types populate them with relevant values from which the user can select.
- Visual exploration and visualization of the database, where the user is presented with a visual representation of the database and can execute functions, query or search the data stored in the individual relations or even join them through the relationships described in the metamodel.
- Dynamic generation of SQL which is described in greater detail below. Using the metadata instead of hard-coding everything into the applications has the advantage that the applications handle changes in the database much more gracefully and the whole system is easier to extend and maintain.

V. Automated generation of SQL statements

In order to avoid hard-coding complex SQL queries and to allow ad-hoc exploration of the stored data, applications can generate SQL queries containing `JOIN` operations dynamically, with the help of the metadata stored in the metamodel. The database even provides several functions which simplify the generation of the `JOIN` statements.

The `get_relationship_join_predicate` function can for example return the `USING` or `ON` clause which should be used to join two particular relations through a selected relationship. For example in order to find out how to join the `anamnesis` and `subject` relations the following invocation of `get_relationship_join_predicate` can be used:

```
SELECT        gen_seq.get_relationship_join_predicate(
'gen_seq',
'anamnesis',
'subject', NULL, NULL
)
```

which returns the following clause that can be appended to the query string:

```
USING(subject_id)
```

The `get_relationship_join_expr` function is similar, but returns the whole join statement (joining the child table or view in the relationship). Also note, that relation aliases can be specified, which allows to join the same relation multiple times in case of a more complex query:

```
SELECT    gen_seq.get_relationship_join_expr(
'gen_seq',
'anamnesis',
'examiner',
'a',  'e'
)
```

which returns this join clause:

```
JOIN  gen_seq.anamnesis  a
ON((a.examiner_id=e.post_id))
```

which can be appended for example after

```
FROM  gen_seq.post e
```

where `post` is the master table in this relationship.

Multi-column relationships are also supported as can be seen in the next example (that the metamodel also describes itself.).

```
SELECT gen_seq.get_relationship_join_expr(
       'gen_seq',
       'meta_parameter_column',
       'meta_parameter',
       NULL, NULL
)
```

for which the following SQL fragment is generated:

```
JOIN gen_seq.meta_parameter_column USING(
    function_catalog,
    function_schema,
    function_name,
    parameter_name
)
```

VI. Conclusion

In this paper we described the database model used by a software that will be used for the collection and evaluation of data from genetic analyzer. The resulting data can be analyzed and retroactively classified to the data required for the determination of the final evaluation. When collecting sufficient number of reference samples, will be these genetic polymorphisms analyzed, correlated and used in research of association of these polymorphisms to tumor formation [7].

Part of this database is a metamodel, stored in the database itself, which provides useful metadata and conceptual data helping with front-end application development.

Acknowledgment

This work was supported by project "Competence Center for research and development in the field of diagnostics and therapy of oncological diseases", ITMS: 26220220153, co-financed from EU sources and European Regional Development Fund.

"Podporujeme výskumné aktivity na Slovensku/Projekt je spolufinancovaný zo zdrojov EU "

References

[1] D. G. Brown, B. Brejova, M. Li and T. Vinar, ExonHunter: A comprehensive approach to gene finding, Bioinformatics, 21(Suppl 1): i57–i65. Proceedings of the 15th International Conference on Inteligent Systems for Molecular Biology (ISMB 2005).
[2] M. Chochlık, Distribuovane spracovanie dat v heterogennom prostredi, PhD. thesis., University of Zilina, 2008, (in Slovak).
[3] M. Chochlik, Implementing the Factory pattern with the help of reflection, Computing and Informatics, 2015, (in print).
[4] Chochlik M., Portable reflection for C++ with Mirror, Journal of information and organizational sciences. ISSN 1846-3312. Vol.36, No.1, 2012.
[5] M. Chochl´ık, Storing dimensions and units of physical quantities in a relational database, Acta Electrotechnica et Informatica. ISSN 1335-8243. Vol.8, No.3, 2008.
[6] dbSNP - a database of short genetic variations, http://www.ncbi.nlm.nih.gov/SNP/.
[7] T. Matakova et al., Polymorphisms of biotransforming enzymes (GSTs) and their association with colorectal cancer in the Slovak population, In: Neoplasma. - vol. 56, c. 5, 2009, pp. 422-427.