

## Prototype for Traversing and Browsing Related Data in a Relation Database

**Guntis Arnicans, Girts Karnitis**

University of Latvia, Raiņa bulv. 19, Rīga, Latvia

{Guntis.Arnicans, Girts.Karnitis}@lu.lv

People who develop, test and maintain information systems often have to inspect the content of databases to make sure that data have been stored correctly or to find errors in the data. The most popular RDBMS and specialised database management tools usually offer single-table browsing. Sometimes SQL requests will be required before the necessary view of multiple tables can be ensured. This paper offers simple principles for inspection and traversing of databases, which may serve as a framework for the establishment of more effective tools for the visual inspection of database content. Browsing and traversing are based on the ER model of a database. Relationships among entities make possible definitions of various specialised views for each record from a table as well as displaying linked information from records in other tables. The relationships also define transitions for traversing the database so that the user can move from one data object to another. A prototype for data traversing and browsing based on these principles is developed.

**Keywords:** data browsing, database inspection, database traversing, transition graph.

### 1 Introduction

There are countless applications today which use a relation database management system (RDBMS). When information systems (IS) and databases are developed, that involves inspection of the relevant database. IS developers need to see real data that are stored in the relevant database – a process which is usually described as data browsing. We believe that the inspection of the contents of a database is a complicated process. That can be attributed to a lack of appropriate tools, and this both encumbers and delays the process of development. In this paper, we will review ways in which the data browsing process can be improved so as to make the work of developers easier and to reduce the possibility for errors.

We can divide all applications that work with database in two groups: *Business-specific applications* and *IS development-oriented applications* (Fig. 1). Main features of both groups are shown in Table 1.

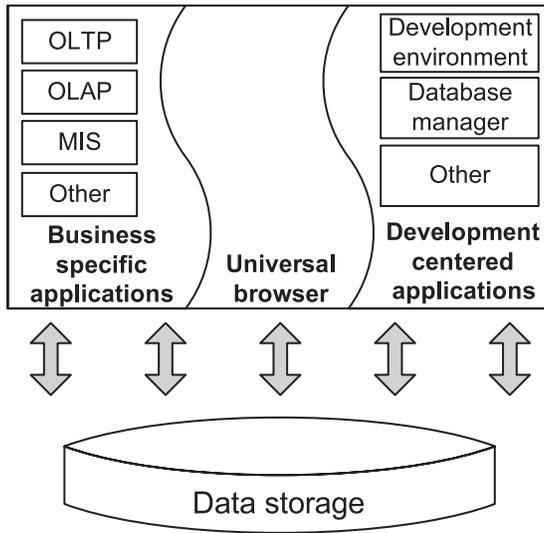


Fig. 1. Applications that work with database

Table 1

**Features of the application groups**

Feature	Business-specific application	Development-centered application
Target	Business requirements	Software engineering requirements
Data granularity	Low	High
Data view	Logical	Physical
Requires specific knowledge	Good knowledge on business, poor knowledge on IT	Poor knowledge on business, good knowledge on IT

There are situations when none of the applications mentioned above are suitable for use in practice. For instance, if we need to solve a problem with some IS and to ensure that complex object data is correctly stored in number of related tables.

Business specific information system represents data according to the business rules. It is hard to collect precise information what data is stored in which table in which format with these kinds of IS. Additionally, if we have a problem, there is a high possibility that IS is not working properly. It is impossible to ensure correct information retrieval from database tables if IS is working incorrectly.

Problems outlined above signify that we must inspect data by not using a business application. Most RDBMS have tools with good functionality for the management of databases in accordance with the ER model and with contemporary requirements. Unfortunately, RDBMS management tools also have certain shortcomings because they do not have all of the functionality that is necessary for data inspection.

RDBMS and the many tools that are used when working with a database will ensure data browsing at the entity level. The developer can review data from the table or view, as obtained after an SQL request. We believe the characteristics listed below are the most important that make data inspection more difficult than the user expects.

- Browsing of data from a table is a local process, one that is limited to only one table. Data values cannot be very expressive (e.g., numbers which are identifiers or keys to real world values stored in the other tables).
- Writing view can require effort and appropriate IT skills.
- View can be erroneous in the sense of not satisfying user requirements and creating a false impression of the true situation inside the database.
- View may not display those records which, at the logical level, are near to, but not fully in line with the selected data criteria (i.e., the low-quality data which we are seeking may not appear in the results).

Problems mentioned above indicate that we need supporting tools that ensure database inspection in a fast, easy and safe way during IS development and maintenance. In this paper we consider a simple tool which reduces the aforementioned problems in the inspection of data.

Our approach is based on the Entity-Relationship model that was introduced by Chen [1]. At first, tool obtains meta-information about database structure (tables and their attributes, relationships between tables) and allows for data browsing and traversing from one record to related records based on the data model. The user can select any table as source table and get records on the basis of criteria entered for this table. The user chooses a single record and the tool displays related records from other related tables. By selecting any relation, the related table becomes a new source table filled with related records.

## **2 The Problem of Database Inspection**

When information systems which preserve their data in an RDBMS are developed and maintained, various specialists have to study their content – system analysts, software designers, coders, testers, database administrators, system maintainers, system users (in specific situations), etc.

Let us examine a few typical cases in which database inspection and debugging are necessary.

For example, the user of an information system reports that the system is returning incorrect results. System maintainers must find the cause. First of all, they must understand whether the data in the database is correct. This leads them to look for the data which is involved in the preparation of the result. This kind of data is usually found in various tables linked by relations.

System maintainers try to follow the algorithm for the extraction of results in a step-by-step way, writing individual SQL requests and testing the intermediate results at each step. If the intermediate results seem valid, then the next step can be taken. It might be necessary to make use of values from the fields of earlier intermediate results, and these have to be part of the next SQL request. The person who is doing the work must simultaneously record and store these values, either electronically (file, clipboard) or on paper. This is a process which takes a lot of time, and human errors are quite possible.

When an SQL request is based on multiple tables, the intermediate result can be erroneous. That can be the result of incorrect data in a table, or an incorrect SQL request.

For that reason, the system maintainer should use SQL requests which do not require the joining of multiple tables.

The work is similar for software developers who believe that new software is not returning the correct results.

When a system developer wishes to get a better sense of the design of relevant systems and particularly databases, the study of the data model must be supplemented by a survey of data in the actual database. Before relationships among data can be determined, the developer must write SQL requests, as described above.

System testers who use test cases will sometimes find that they need to take a look inside the database to make sure that the data have been stored correctly. If data are stored in multiple tables or are related to data in other tables, then the tester must write SQL requests, as described above.

All of these activities involve a user-accessible tool which usually makes it possible to review the structure of a database – tables and their fields, as well as primary and foreign keys. SQL requests for the database can be executed, and the results of such requests become visible. With such tools, the data browsing process is not always convenient or fast, and mistakes are possible.

The scenario for data examination (i.e., the series of SQL requests) and the data studied as part of that research will be different each time. That makes it more difficult to automate data examination processes on the basis of existing tools. At best we can save the series of SQL requests from the most typical incidents so that it can be reused with minor modifications.

If the software is being developed by a non-IT company, fundamentally important aspects of the testing duties are clearly handled by employees who have no specific IT-related knowledge [2]. The inability to obtain and analyse data from the database clearly reduces the quality of the testing process in such situations.

Let us now review an example that will be discussed in detail in this paper. There is a small university-based information system which is used to record whether students attend lectures and the grades that they receive on tests. The diagram of this entity-relationship system is presented in Fig. 2. The letters PK is used to show the primary key of the table and the letters FK is used to show the foreign key.

Let us look at a specific record in the table Grade, which contains the field values

```
<Examination_ID=5000001,  
Student_ID=100002,  
Grading_Teacher_ID=2001,  
Grade=10>.
```

Let us call this the viewpoint for the database. The first three numbers are internal identifiers, and they offer us no semantic information. When it comes to the grade of 10, the student who received that grade, the examination to which the grade applies, the instructor who prepared and graded the examination, and the subject area in which the examination was taken – all those are of interest to us. In a graphical form it could look like Fig. 3.

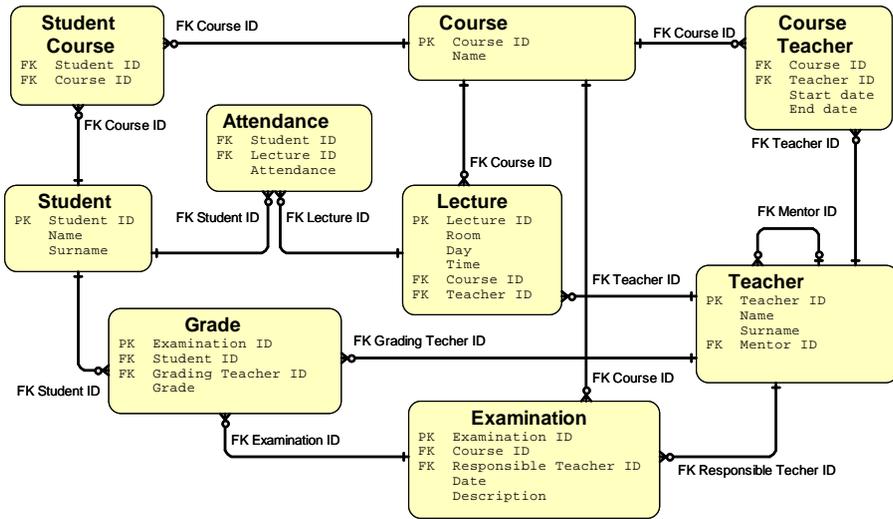


Fig. 2. A sample diagram of an entity-relationship database

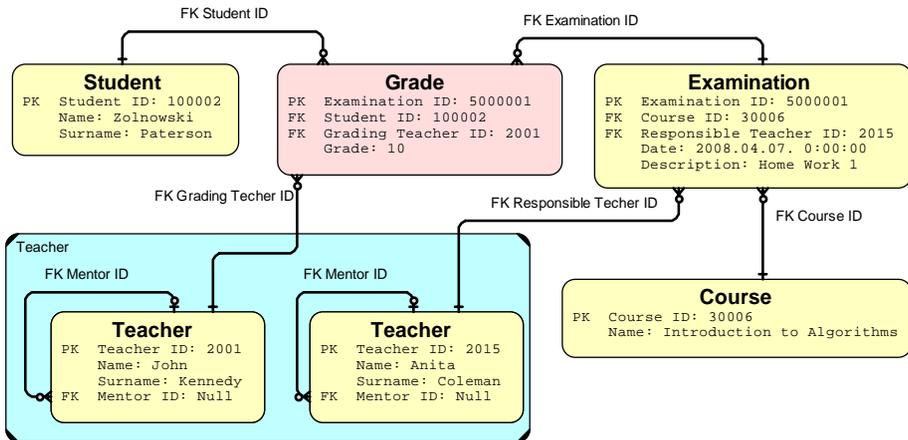


Fig. 3. A graphical form of related information for the fixed Grade record

Our first option here is to present an SQL request which joins five tables – Grade, Student, Examination, Course and Teacher twice. In this SELECT request, we can immediately spot two areas in which there might be problems. If the tables are joined with the ordinary JOIN function, as opposed to the appropriate LEFT JOIN or RIGHT JOIN function, then some resulting rows can be lost. Secondly, the user may fail to notice that the Teacher table is used two times in the SQL request – each case being independent of the other. This means that two different local names must be applied. A proper SQL request can be made only by someone who has knowledge exceeding that of the average programmer or tester. There is the risk that the incorrect request will lead to an incorrect result.

```

SELECT * FROM Grade G
LEFT JOIN Student S ON G.Student_ID=S.Student_ID
LEFT JOIN Examination E ON G.Examination_ID = E.Examination_ID
LEFT JOIN Course C ON E.Course_ID=C.Course_ID
LEFT JOIN Teacher T1 ON G.Grading_teacher_ID=T1.Teacher_ID
LEFT JOIN Teacher T2 ON E.Responsible_teacher_ID=T2.Teacher_ID
WHERE G.Examination_ID=5000001 AND G.Student_ID=100002

```

The second option is to prepare separate requests for data selection from specific tables, presenting the values of fields related to the selected parameters for this purpose. These requests would be as follows:

```

SELECT * FROM Student WHERE Student_ID=100002;
SELECT * FROM Teacher WHERE Teacher_ID=2001;
SELECT * FROM Examination WHERE Examination_ID= 50000001.

```

Here we assume that the response will be:

```

<Examination_ID=50000001,
Course_ID=30006,
Responsible_Teacher_ID=2015,
Date='2008.04.07',
Description=Home Work 1>.

```

Now we can select information about Course and Teacher, prepared Examination:

```

SELECT * FROM Teacher WHERE Teacher_ID=2105;
SELECT * FROM Course WHERE Course_ID=30006.

```

For an ordinary programmer, this is a scenario which should lead to fewer mistakes than in the first SQL request that was described. This is also a scenario which makes it easier to spot incorrect data.

Someone who is not familiar with SQL requests as such, however, would not be able to do this work at all.

Ideally we would want to obtain this information without preparing SQL requests at all. A change in the viewpoint might be another desired element. This means that we choose a linked record from a different table and imagine that we “transfer ourselves” to that record. In that case we can find information that relates to the new viewpoint.

### 3 Traversing and Browsing Databases

In RDBMS or tools that are used to work with databases, the concept of database browsing usually refers to an exploration of the structure of the relevant database so as to present the various elements therein – tables, views, stored procedures, indexes, relationships, etc. Browsing also identifies the relationship among the various elements, and it makes it possible to view data from a specific table. The elements in the structure of the database are usually presented as a tree. The content of the table is displayed in grids, which ensure the filtration and arrangement of the data, changes in the order in which columns are presented, the hiding of columns, and other activities that make it easier to review the data in the table. One can also review the content of several tables in individual grids which are mutually independent of one another.

What we need to see is not only the records that are in our selected table, but also related records from other tables. That would give us more information about the records in the selected table, and we could also assess the correctness of the data in our table and in the related tables. Where necessary, we can shift our viewpoint to a linked table, where we can find related information from a new position.

For the database which we are reviewing here, let us assume that we have opened the Student table and chosen the records of student X. We see that there are related records in the tables Student\_Course, Attendance and Grade. All of the records that are related to student X are automatically filtered and made available for review in these tables. If we look at the Student\_Course table, we can see only the records that apply to student X. Now we have a new viewpoint from the Student\_Course table, and we also see the related tables – Student and Course. When we move from one record to another, we should see the records of the initially chosen student X in the Student table, as well as the courses which student X has chosen in the Course table.

The selected solution of this task should be sufficiently universal to be utilised in various relation databases. The one thing which relation databases have in common is that they are based on the ER model. Essentially this is a graph in which the vertex is in line with the entity, and the edge is in line with the relationship. In our example, the ER diagram can be presented in a simplified way in the graph that is seen in Fig. 4. The vertexes of the graph are marked with the first letters of words that are used in the name of the entity.

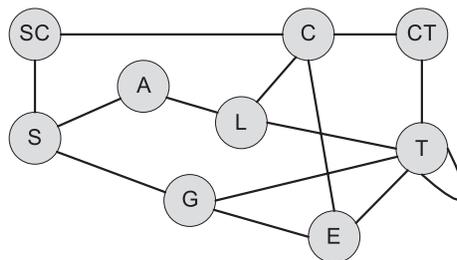


Fig. 4. The graph of entities and relations for the example

Traditional database management tools allow us to present the concepts of this simplified data model, and that means that we can review:

- the full list of vertexes (list of tables);
- the full list of edges (list of relationships);
- the list of edges for each vertex (the list of relationships for each table);
- the structure and properties of the vertex (a description of the table);
- the properties of the edge (a description of the relationship);
- the content of each vertex (viewing the table's record via filtration and various opportunities for visualisation);
- the content of several related vertexes via a special SQL request.

We offer new and standardised means to extract information from such a graph so as to reduce the shortcomings of the last point that we made. SQL requests require

IT skills. Time must be invested in order to understand a data model precisely and to write up an SQL request. An SQL request can be erroneous, and that will lead to an incorrect data view. These problems are reduced, but by no means eliminated by the use of predefined views or stored procedures or scripts.

From here on in, we will call the vertex of our graph an entity so as to use concepts that are more appropriate when discussing data models.

Our proposal on the design of new tools is based on a simple principle. We choose an entity on the graph which we can call the source entity. We can then make note of those records in the entity that are of interest to us. All records that can be reached from fixed records via the use of edges (the relations in database) can be called achievable records. Records that can be reached from an achievable record also are achievable records. Any entities that contain achievable records can be called achievable entities. We can set up an achievable graph which includes the selected source entity with fixed records, as well as all of the achievable entities with their achievable records.

Because this makes it possible to select all of the records in a database in a general case, it is necessary to set several sensible limitations on the selection of related records so that the set of records which is obtained is manageable. The limitations dictate those achievable entities from which we do not need to achieve following achievable entities. Let us call these destination entities. One of the simplest limitations in this kind will be this: we will define neighbouring entities as destination entities. We make use of only one record in the source entity and make it possible to review destination entities, automatically showing only the achievable records from this fixed record.

The graph in Fig. 4 is a very simplified model of the relevant ER diagram (Fig. 2) because the relations are depicted with a simple edge. Let us make use of the fact that each relationship between two entities in the ER diagram determines the information that is needed from a single entity, as well as the related information that can be obtained from the other entity. The cardinality of the relationship at both ends of the relation determines how many records from the other entity are or can be associated with or linked to the record in the first entity. Not to make the model which we are reviewing here too complex, let us assume that it is possible to form links from one field in one entity to one field in the second entity.

We can also imagine each relation as a transition from the specific record in one table to records in another table in accordance with the type of cardinality. We can use the ER diagram (Fig. 2) and the simple graph (Fig. 4) to prepare a new graph – the Browsing Transition Graph (BTG). Each edge or relation is changed into two oriented edges or transitions. If X is the source entity, and Y is the achievable entity, then the transition determines that if the record from entity X (the beginning of the transition) has been identified, then it is possible to find linked records in entity Y (the end of the transition).

This example offers us the image that is seen in Fig. 5.

The transition can also have properties that can be depicted visually and are obtained in accordance with the relation's cardinality, as well as the properties of the specific database:

- 1) a single-arrowhead edge – the record from entity X is linked to no more than one record from entity Y;

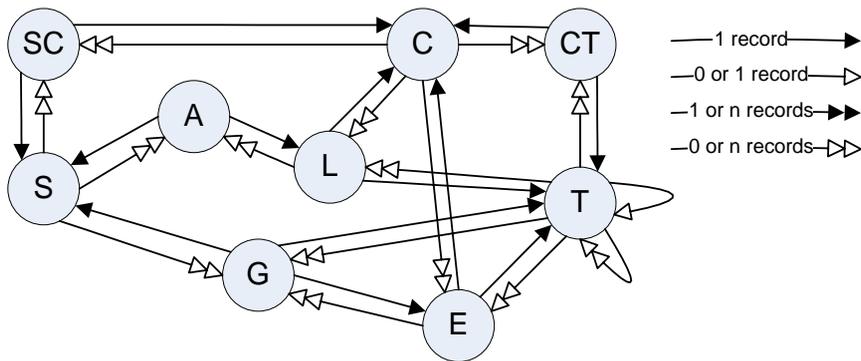


Fig. 5. A Browsing Transition Graph (BTG)

- 2) a double-arrowhead edge – the record from entity X is linked to an unlimited number of records from entity Y;
- 3) the full arrowhead – the record from entity X is not linked to fewer than one record from entity Y;
- 4) the empty arrowhead – the record from entity X might not be linked to any record from entity Y.

In our sample database (Fig. 2), each record in the Lecture entity describes a specific lecture and is linked to one specific Teacher entity. That, in turn, defines the one teacher who will teach the relevant class (in the BTG, this is a transition from entity L to entity T). In the opposite direction, the transition from entity T to entity L specifies that there might be many classes which are taught by the same teacher, while there can also be teacher who teaches no classes at all.

A double-arrowhead edge should make us cautious, because the number of linked records could be enormous. That would mean extra time to select the records and to deliver them to the end user. There would also have to be additional memory space to store and transport the relevant records.

Usually the records from the linked entity can be obtained relatively fast. This means that the necessary records from entity Y can be found on the basis of existing indexes in a period of time that is no longer than  $\log(n)$ , where  $n$  is the number of records in entity Y. Alternatively  $n$  might be so small that we can check every record from entity Y in order to select the right ones.

Sometimes there might be a need for a considerable amount of time to find the necessary records in entity Y. This means that we do not have the necessary indexes, and we will have to review all records from entity Y to select the ones that are necessary. If, for instance, we have information about attendance at lectures over the course of several years, and if there are many students to whom such information apply, then that means that we would have to inspect more than one million Attendance records (transition (S, A) in graph BTG) in order to specify the attendance of a specific student (a record from the Student entity).

The term database traversing refers to movement in the BTG. This involves specifying that the new source entity is one of the achievable entities, and then only the

records which can be accessed before that are depicted in this new source. As soon as a new source entity is defined, it identifies achievable entities, and records therein come from the records which are identified by the source of the new entity.

The term browsing view refers the view of the source entity, the transitions and the linked records in the achievable entities. The definition of the view defines the entities, transitions and records that are to be displayed and the way in which they are to be visualised for the end users. The tool used for browsing and traversing a database should ensure several pre-defined views so that the user does not have to do any additional work to receive the necessary information from the related entities.

Databases do not always contain definitions of all possible relations among entities. This means that we must allow the person who is configuring the tool to establish additional transitions in the BTG, indicating linked entities and linked fields.

When we define various browsing views, we must certainly think about protection against cycles which exist in the BTG. Simple protection involves defining the maximum visibility or distance of achievability (i.e., the number of transitions) from the source entity to the achievable entity. A slightly more intelligent protection would take into account the number of cycles in the BTG, as well as the total number of achievable records.

## 4 The Tool's Prototype

The authors of this paper have years of experience with various information systems. Many of those systems were based on database meta-models. They were used to integrate information in single systems and under the framework of several heterogeneous systems. Closest to the principle that is described in the previous chapter was a system which had the essential goal of integrating information [3].

The tools that have been developed in the past were information systems that were based on a logical data models. There was no tool, however, to survey a physical database. Principles of such tool were given in [4]. As the complexity of information systems increased, developers complained more frequently that traditional database administration tools did not ensure sufficiently convenient review of data. They asked for a simple and universal database browsing tool which would display tables of information related to the record without having to write up an SQL request.

### 4.1 Requirements for the Tool's Prototype

We developed a prototype for database browsing and traversing that was based on two simple browsing views. The views were chosen with the basic principle that the critical needs of users had to be satisfied as much as possible with as little developmental effort as possible.

These were the requirements for the tool's prototype:

- the tool must operate on the basis of the Browse Transition Graph (BTG) that was established for the database;
- the tool has to be able to read database metainformation from various popular RDBMS;

- the BTG has to be established automatically after a reading of meta information from the selected database;
- the system must provide means to choose a table that serves as the initial source entity for the BTG and makes it possible to select records on the basis of criteria entered for this table;
- the system must represent selected records and the related information on the basis of the browsing view;
- the system must ensure traversing to another entity in accordance with the browsing views.

The first browsing view provided for the display of the following data, as well as the following traversing opportunities:

- to create the selected source entity records on the basis of grid view control (grid view control ensures the filtration and arrangement of selected records, changes in the succession of columns, and other opportunities for visualisation);
- to display all transitions from the source entity to linked entities on the basis of grid view control;
- when user chooses a single record from the source entity and one transition, the system must display other relevant achievable records on the basis of grid view control;
- the determined transition must enable database traversing to the relevant achievable entity.

The second browsing view ensures the following data display and traversing possibilities:

- displaying fields of records from the identified source entity on the basis of tree view control (tree view control provides for a display of selected records in the form of a tree, with sub-tree collapsing and expanding, as well as other opportunities for visualisation);
- finding all achievable entities and achievable records that can be accessed on the basis of transitions with the single arrowhead transition and displaying them in a hierarchical tree;
- representing as child nodes all transitions to achievable entities in which the relevant field is used;
- enabling database traversing to the relevant achievable entity from any transition that is displayed on the tree.

## **4.2 The Architecture of the Tool**

The architecture of the tool is represented in Fig. 6. The main elements of the tool are the Meta Database, the Main Engine, the Wrapper and the Presentation Engine.

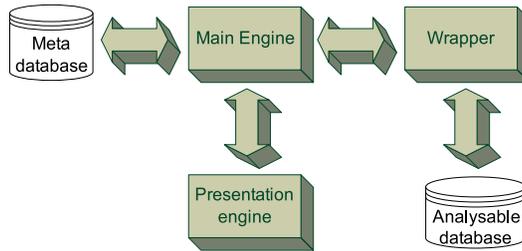


Fig. 6. The architecture of the prototype tool

The Meta Database contains a part of the description of the relevant database's structure, which is necessary to establish a BTG. The ER model for the Meta Database is seen in Fig. 7. Table "Tables" offers information about tables from the database. Table "Fields" offers information about fields in the table. Table "Relations" offers information about relationships among the tables. The Meta Database is filled-in when a connection to the database is established. The SQL requests that are necessary to fill in the Meta Database are specific to each specific RDBMS.

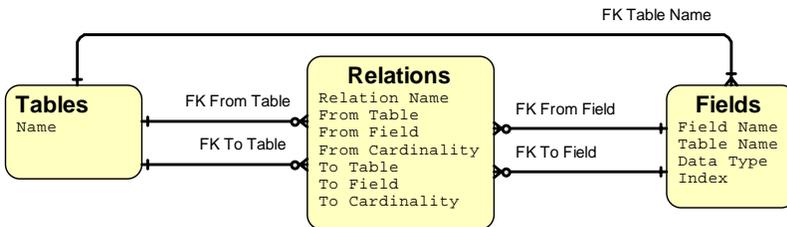


Fig. 7. A Meta Database

A Wrapper is a module which ensures the selection of metadata and data from the database that is being analysed. These are the functions which the wrapper provides in obtaining metadata:

- selecting the list of tables from the database, thus filling data in the "Tables" table;
- selecting the list of fields from the database, thus filling data in the "Fields" table;
- selecting the list of relationships from the database, thus filling data in the "Relations" table.

In order to obtain data, the wrapper provides the following request:

```
SELECT * FROM <Table> WHERE <Filter Expression>.
```

The Presentation engine is a module which displays information and offers navigation opportunities to traverse the database.

Let us explain operation of tool prototype on an example given in Chapter 2. Fig. 8 is a screenshot of the prototype we have developed.

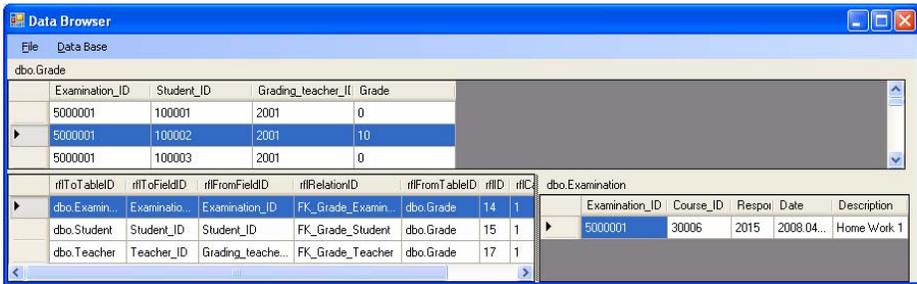


Fig. 8. A view of the Grade and Examination entities

At the top we see the records from the source entity (the Grade table), which are in line with the filter Grading\_Teacher\_ID=2001. One of the Grade records is fixed. At the bottom left, we see a list of transitions (relationships from the Grade table), which notes the transition to Examination, which is an achievable entity. To the lower right we see the achievable records from the Examination entity which is in line with the fixed source entity record and the fixed transition.

If we change the record of the source entity or transition, we also change the achievable records. The change in the fixed record involves a mouse click on the relevant grid view control record. In choosing the second record in the transition grid view control, for instance, we will find that records from the Student table will be displayed as achievable records. These will be in line with the filter Student\_ID=10002 (Fig. 9).

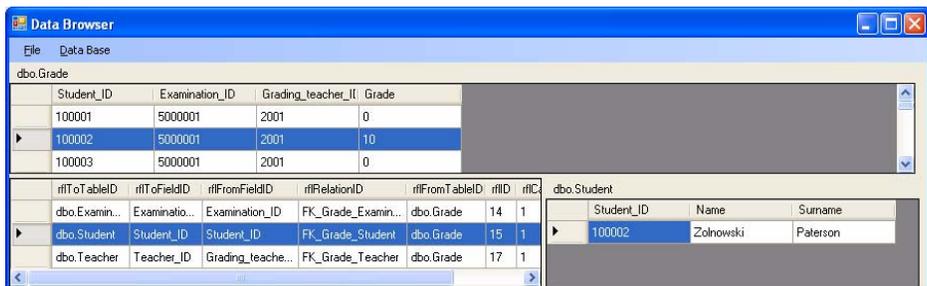


Fig. 9. A view of the Grade and Student entities

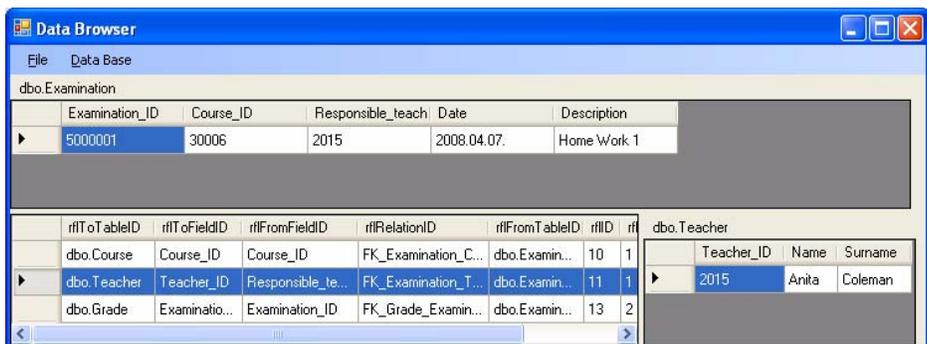


Fig. 10. A view of the Examination and Teacher entities

Database traversing here is ensured with a double mouse click on the relevant record from the transition grid view control. Here we have double-clicked on the first record from the transition grid view control, which means that our source entity is the Examination table. It in turn shows records which are in line with the filter Examination\_ID=5000001 (Fig. 10).

Fig. 11 shows how linked information is depicted in a tree. This is the second additional option for the browsing view.

The first level of the tree represents the source entity (the Grade table), with all relevant fields and values. If a field is involved in a transition, then that is represented as field node children under that field (<To\_Table\_Name> <To\_Field\_Name>). Depending on the properties of the transition, information about achievable records is displayed in different ways.

- If the transition has the single-arrowhead edge, which means that the number of achievable records is 0 or 1, the names and values of achievable record fields are attached as transition node children. For example, the Grade field Grading\_Teacher\_ID is used in the transition to the Teacher\_ID field of the Teacher entity. This means that under the node of this field, the child node is the relevant information about transition (dbo.Teacher Teacher\_ID). It, in turn, has the values of achievable record fields as child nodes.

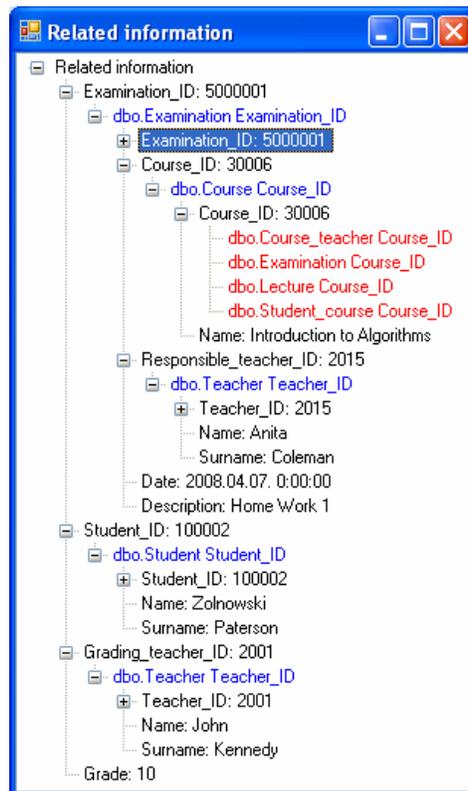


Fig. 11. A view of the Grade entity in the tree version

- If the transition has the double-arrowhead edge, which means that the number of achievable records is unlimited, then there are no transition node children. For instance, the Course\_ID field in the Course achievable entity of the Grade entity is involved in four transitions (dbo.Course\_Teacher Course\_ID, dbo.Examination Course\_ID, dbo.Lecture Course\_ID, and dbo.Student\_Course Course\_ID).

In the example displayed in Fig. 11, information about transitions with the double-arrowhead edge is collapsed and is not visible. In the depiction of information about fields from achievable entities, transitions are arranged on the basis of the same algorithm that is used for the source entity.

The Main Engine provides the operating logic of the tool. At first, the Main Engine makes use of the Wrapper to hook up to the selected database and receive meta-information about its structure so that the BTG can be established. Then management is passed to the Presentation Engine component, and that allows the user to choose the source entity and the criteria for selecting records. The Main Engine then uses the Wrapper to select the relevant records so that they can be depicted by the Presentation Engine. The Main Engine maintains information about the current condition of the source entity and the selected browsing view, and it also supports the traversing mechanism.

In Fig. 11 we see the same record which was used in our example. All of the requested linked information is presented automatically.

The main shortcoming of the prototype from the perspective of the view functionality is that it has two “hard-coded” browsing views. Unfortunately there are situations in which a different browsing view is needed. When selecting a student, for example, we might want to see all the classes which the student is taking, but that’s not possible with the current browsing view. If we want that information from the prototype, we will have to define the student, look at the linked records in the Student\_Course table, and then select the records one by one to see the names of the courses.

## 5 Conclusions and Future Work

Over the course of the last 15 years, we have produced many informative systems that are largely based on a data model, are universal, and are fairly independent from the use to which they are put. We had to seek out general ways of obtaining, integrating and displaying information. The solutions were focused on the end users of information systems, but not on the developers of such systems.

We regularly received complaints from system developers about problems in the maintenance of databases, and we recommended the use of appropriate tools which better display the relevant data and their linkage to other data. Much to our surprise, our search for ready-made tools was unsuccessful. That does not mean that there are no such tools – perhaps we just didn’t have the skills that were needed – but it is certainly a sign to show that many developers have to deal with problems on their own. Others may be as unskilled as we were in searching for an appropriate tool.

We applied the experience of our previous solutions to show how to put together a tool which offers not just the traditional view of physical data in a single table, but also a view of the nearest surroundings at the physical level of the database. We have presented

what we consider to be the necessary minimum to explain the essence of our idea. We demonstrated the two simplest views and traversing principles. This is a simple and even primitive tool, and it does not require vast resources. In a large project, it pays off to establish such a tool if an appropriate one is not available in the software market.

The prototypes which we developed were demonstrated to several developers and system maintainers who need to study data from databases on a daily basis. The positive evaluation of the prototype tool from all of the users was a surprise, and the tool already can be used in real life. Users reported that the tool makes it substantially easier and faster to select and review data. They also recommended a whole series of improvements, which would make the tool more effective. The primary functional improvements which users would like to see implemented are the following:

- the ability to configure the table fields that are viewed and the order in which they are presented, with full ability to store the configurations that are established;
- a back-forward system which would make it possible to return to data that have been viewed recently (in the reviewed prototypes, the transitions between two entities in the BTG are basically asymmetrical, and when one moves back, a different set of records is obtained);
- the ability to edit data in a way that allows maintainers and testers of databases immediately to apply necessary fixes when data mistakes are identified.

These are the requests which indicate the priorities of users who want to prepare such a database viewing and traversing tool. This might be more important than creating new and additional views or the many data integration and traversing opportunities which the transitions in the BTG might potentially offer. Work on a new version of browser with improvements mentioned above is in progress.

### Acknowledgments.

This work has been supported by the European Social Fund Project No. 2009/0216/1DP/1.1.1.2.0/09/APIA/VIAA/044.

We wish to thank Karlis Streips for improving the English of this paper.

### References

1. P. P.-S. Chen. The entity-relationship model: Toward a unified view of data. *ACM Transactions on Database Systems*, 1(1), 1976, pp. 9–36.
2. V. Arnicane. Use of Non-IT Testers in Software Development. In: J. Münch, P. Abrahamsson (eds.), *Product-Focused Software Process Improvement*. Lecture Notes in Computer Science, Vol. 4589. Berlin-Heidelberg: Springer-Verlag, 2007, pp. 175–187.
3. G. Arnicans, G. Karnitis. Heterogeneous Database Browsing in WWW Based on Meta Model of Data Sources. In: J. Barzdins, A. Caplinskas (eds.), *Databases and Information Systems, 4th International Baltic Workshop BalticDB & IS 2000, Selected Papers*. Vilnius, Lithuania, May 1–5, 2000. Kluwer Academic Publishers, 2000, pp. 167–178.
4. G. Arnicans. Application generation for the simple database browser based on the ER diagram. In: Jānis Bārzdiņš (ed.), *Databases and Information Systems, Proceedings of the Third International Baltic Workshop*, Volume 1, Rīga, 1998, pp. 198–209.