# From Questions to Queries

Jozef Hvorecký, Martin Drlík

# From Questions to Queries

*Jozef Hvorecký*[1], *Martin Drlík*[2]

[1]Vysoká škola manažmentu v Trenčíne, [2]Univerzita Konštantína Filozofa v Nitre

**Key words:** *Tele-learning, teaching Database Management, simplified creation of database objects*

## Abstract:

*The extension of (Internet) databases forces everyone to become more familiar with techniques of data storage and retrieval because users' success often depends on their ability to pose right questions and to be able to interpret their answers. University programs pay more attention to developing database programming skills than to data exploitation skills. To educate our students to become "database users", the authors intensively exploit supportive tools simplifying the production of database elements as tables, queries, forms, reports, web pages, and macros. Videosequences demonstrating "standard operations" for completing them have been prepared to enhance out-of-classroom learning. The use of SQL and other professional tools is reduced to the cases when the wizards are unable to generate the intended construct.*

# 1 Introduction

The extension of databases forces everyone to become familiar with techniques of data storage and retrieval. For example, a query forms a bridge between posing a question formulated by a human and getting an answer to it from a database engine. As all potential questions of future users can never be known in advance – and therefore cannot be programmed ahead – the success of the database application often depends on users' readiness to see the interrelations between "two banks of the bridge" and their ability to build them. We often witness inefficient exploitations of the data resources because their users are:

(a) Unable to pose complex questions;
(b) Incompetent to interpret the response.

A part of the problem is caused by inadequate education. Universities concentrate on preparation of developers, but not of users. Their study programs pay more attention to developing programming skills than to data exploitation skills. Authors lack Database Management courses asking "meta-questions" like:

- *Can the question X be solved based on the existing content of the database?*
- *What has to be added to the database in order to make it capable of gaining a solution to the problem X?*
- *Can you formulate another problem which cannot be solved based on present data?*

The authors believe that the ability to react to them would substantially help in education of future users as they would become aware of risks of data incompleteness, incompatibility and inconsistency. Relevant courses do not need Relational Algebra or Object-Oriented Database Design. They should rather demonstrate practicality of databases, the importance of data structuring, principles of SQL-like searching mechanisms, and the importance of user-friendly communication.

The authors designed a *Database Management* course that incorporates the above intentions. The course is based on Access and intensively exploits its "wizards" – supportive tools

simplifying the production of database elements as tables, queries, forms, reports, web pages, and macros. In addition to that, wizards in combination with another set of tools – design views – visualize the structure of the elements and facilitate students' comprehension. The time saved due to faster production can be efficiently used for explaining the role of primary and foreign keys, indexes, and principles of data integrity.
The use of SQL and other professional tools is reduced to the cases when the wizards are unable to generate our intended construct. Similar (counter) examples have a high pedagogical value as they show that not all problems can be solved using "amateur approaches" and make a clear distinction between user-oriented courses and professional ones. They also indirectly show to the future users they role in the database development: *The users have to know what they want from the databases whilst professionals have to know how to implement their desires*.
The database developers can speak long about low-qualified and ignorant (future) users unable to express their desires properly and blaming them later for not implementing their unspoken or wrongly expressed requests. For that reason, the course ends with student projects – complete, simple and user-friendly database applications. The students learn about the complexity of the process and the necessity to specify all of their expectations in its early stages.
In our paper we exemplify our teaching methodology using typical problems solved by our students. Its aim is to point to the fact that education of qualified database users – potential "translators" between business community and software developers – has its specifics. It requires a combined background; partially from business and management and partially from computer science. The application of wizards helps us to reduce the number of prerequisites and theoretical concepts not-substantial for our students and still to guarantee a relatively high students' competence in databases.
In a separate section we discuss specifics of the distance-learning methodology. To support it, the teaching manuals and the sets of solved and unsolved problems originally designed and created for our on-ground classes have been enriched by sample databases and short videosequences demonstrating "how to perform standard operations". All of them are now successfully applied in our online classes, too. Step by step, the videos show what must be performed for completing the problem solution. Their aim is to build the skills that are critical for transforming questions posed by humans to the notation accepted by the machine. The students can consult them in addition to their regular communication with instructors.

# 2 User-friendly Database Design and Development

## *2.1 Tables*

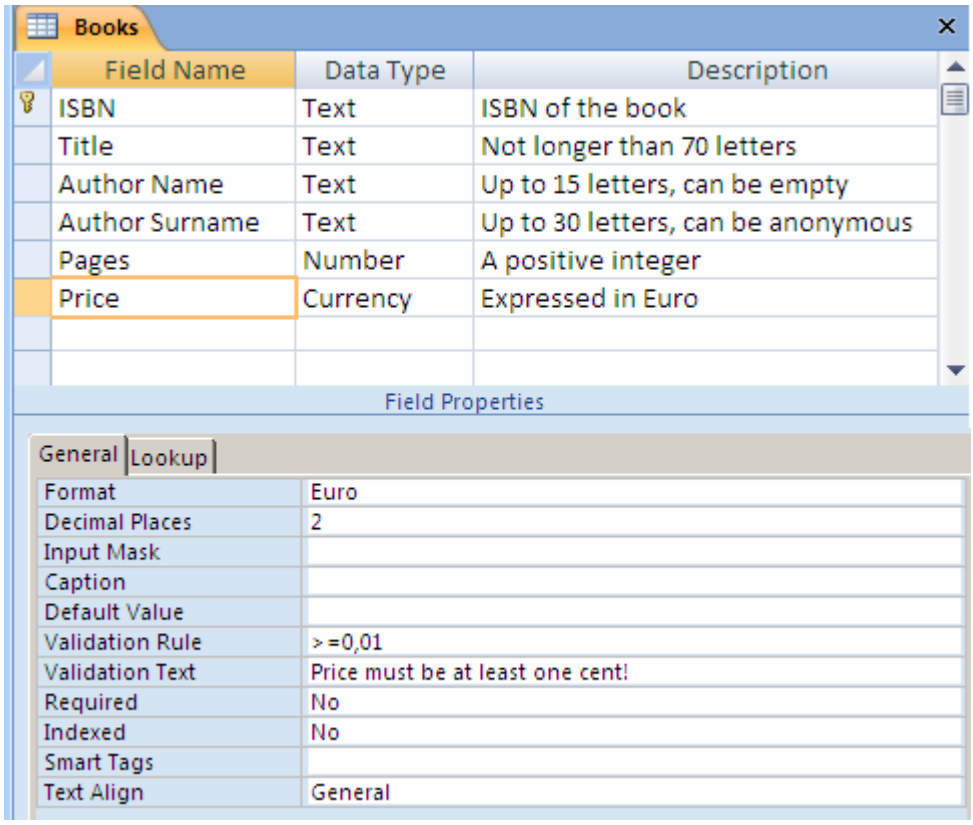
The fundaments of databases are formed of entities and relationships. Separate entities are stored in separate tables. As the relationships between them are usually complex and not always correspond to our intuitive concepts, forming entity-relationship diagrams (E-R diagrams) often precedes table design. In our opinion, this approach is not a fortunate one as the novices have no idea about the role of tables and of the reasons why they must be interconnected by relationships. The proper design requires a high level of abstraction which can hardly be achieved without seeing a sufficient number of concrete examples. For that reasons our course starts with forming one-table databases. Databases with several interrelated tables are introduced a few weeks later.
Access tables can be created using several simplified methods.

- The simplest one uses the *databases templates*. We do not use it as we expect our students to understand how the entity is structured and why.

- The next method is based on typing values into *Datasheet view*. We do not use it neither as we want our students to learn as much as possible about metadata. Our experience showed that once concrete values of attributes are entered into the table, the students prefer them to a general discussion about *all* potential values. Often, the data types of database objects do not correspond to our naïve types. For example, telephone numbers are not of any numerical data type – they are texts. If a telephone number beginning with leading zeroes or a "+" symbol is typed, the numerical data types automatically delete them. As country codes begin with them, crucial information is lost.
- Our experience resulted in preferring *Design view*. This form of wizard has two advantages. First, the student starts with specifying attributes and their metadata, not their values. Secondly, their typing takes much shorted time than using SQL Data Definition Language.

Let us specify the table *Books* with six attributes *ISBN*, *Title*, *Author Name and Surname*, *Pages*, and *Price*. Compare two following versions of the same definition – one in SQL, the other in Access *Design View*:

```
CREATE TABLE  Books (
        ISBN                CHAR (15)           NOT NULL            UNIQUE,
        TITLE               CHAR (60)           NOT NULL,
        AUTHOR NAME         CHAR (15),
        AUTHOR SURNAME      CHAR (30),
        PAGES               INTEGER             NOT NULL,
        PRICE               NUMBER (8,2)        NOT NULL,
        PRIMARY KEY (ISBN)
        );
```




Advantages of *Design View* are obvious: Its parameters are better organized and can be accompanied by comments. The restrictions to each attribute can be specified to a high level of detail without knowing the DDL keywords. Our discussion on them can concentrate on

their meta-properties like: *Is the attribute always required? What are its presumed values? What data type is therefore the most appropriate and why?* Such discussions are crucial elements of our teaching methodology. During them, our students discover the importance of data types and differences between them by themselves. For example: *Why has ISBN to be a Text, not a Number?*
Their gained experience helps them later in understanding their role in cooperative database design and development. For that reason, many assignments have the open-end form like: *You are a police officer. Your superior asks you to prepare a database of wanted persons. What data should the database contain and why? Prepare its table(s) in Access.*
The outputs differ and can therefore be a subject of discussions among the authors of different versions during next classes.
As the data types can be selected from a menu, the most active students investigate the unknown ones by themselves. For that reason, one can often find photographs or internet links among their data. This is another advantage of wizards: they offer more options than teachers can usually introduce during their limited lecture time. As such, they also offer room for investigations, experiments and self-education.
Our students are also invited to discuss problems of data quality. They should understand why data must be *complete*, *accurate* and *up-to-date* – and learn how to guarantee it. The most problems solved here belong to *domain integrity* and *entity integrity*. The latter is achieved by introducing restrictions on several attributes simultaneously:

- *Your company sells airline tickets. Each record about a purchase must contain the port of departure, port of arrival, date of flight, the flight number and the number of adults and children. Each child must be accompanied by at least one adult. Include this restriction into your database.*
- *A transportation company asks you to prepare its database of truck routes. Each route record will contain the driver's name, destination, date of departure and presumed date of return, as well as information on the weight of the load – separately in two columns for the way there and back. As you can guess, the date of return is always greater or equals to the date of departure. The truck cannot travel to the both directions empty. Protect your data against wrong inputs of this sort.*

Similar restrictions lead to rather complex logical expressions with several equalities and inequalities. Using Expression Builder is a method of speeding up the process. As it contains the list of attributes of the given table, the process goes faster and the probability of typing errors decreases. This is one of its main advantages and makes its usage very popular among our students.

## *2.2 Relationships*

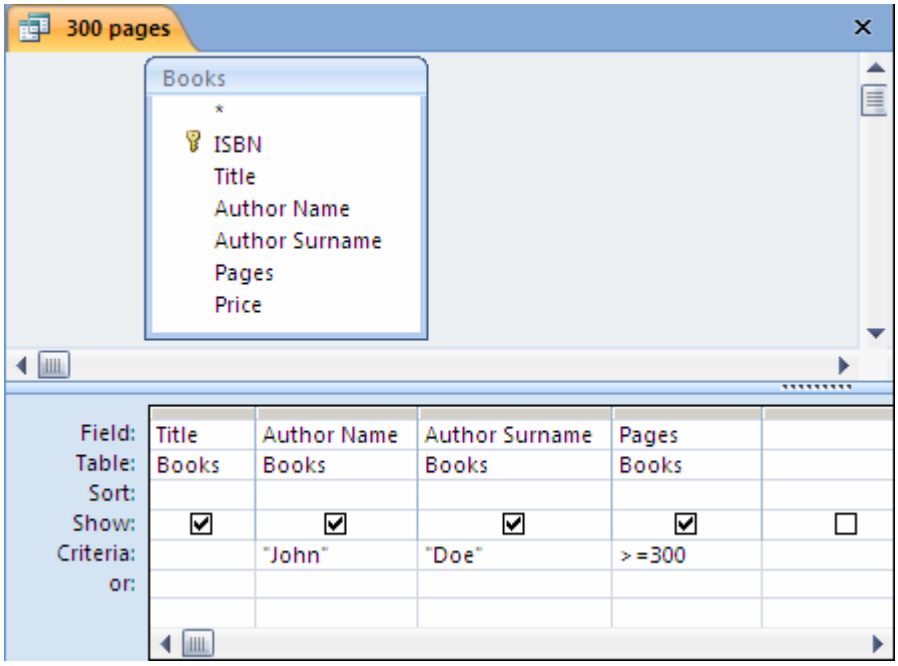
Basics of relationships between tables can also be explained using a wizard. Let us presume that we decide including *Genre* of the book into our above table. The number of its options is limited: *History, Fiction, Non-fiction,* etc. The students quickly grasp that entering them is boring and not a reliable method due to typing errors. They welcome the opportunity to choose them from a menu.
*Lookup Wizard* offers a way. First, a simple, one-column table has to be created. As the new table will contain nothing more than the list of all genres - and they are different – it is the only candidate for the primary key. Later, this table can be interconnected to an attribute as a set of its proposed values using *Lookup Wizard*. The wizard creates a weak relationship between the specified attributes of these tables.
It is very important to point the students' attention to the fact that weak connections do not propose a sufficient protection against typing wrong values. The menu created by the wizard allows introducing the input values by selecting one from the menu but does not prohibit

typing other (incorrect) ones. To achieve this, the relationship must be changed into a strong one. Since it is established, referential integrity is guaranteed. In this stage we do not talk about normalization yet. We only discuss the relationship between the values of a pair of attributes in two relating tables. The relationship locks together the primary key with its related foreign key. Nevertheless, our explanation is an intentional introduction to stronger partnerships required within sets of normalized tables.

### *2.3 Queries*

Our main aim in this section is training our students to become capable translating questions in their minds into their equivalents in a query language. So, our assignments contain only questions in a natural language and never mention "tables, attributes or relationships" in their database-oriented meaning. Our students are supposed to build the relationships between them and the natural-language questions on their own. *Query Wizard* and *Query Design View* are very helpful tools for similar "translations" and simplify the query creation. The students do not need to concentrate too much on its syntax and can focus on semantics. As everyone likely agrees, semantics must be the leading factor in the process.
*Query Wizard* speeds us building the basic structure of SELECT queries by picking up the needed attributes from a two-level menu. At the first level, the user specifies the table or query; at the next level, he/she selects the attributes. The process allows combining data from all tables and (earlier formed) queries. If the tables/queries are connected by a relationship, a JOIN query is automatically generated. The students are no supposed to type these elements. The probability of making typing errors is negligible. The student can better concentrate on the proper choice of the data sets.
We also stress that the outputs of *Query Wizard* are not always identical with user's intention. If we for example select data from two tables not connected by a relationship, Query Wizard generates their Cartesian product. Even if they are connected, the records are combined by using the inner join. Naturally, the user might have a different combination in his/her mind. We introduce specific problems that visualize these (and similar problems). Their main purpose is to visualize a wide variety of combinations of database elements and risks of misunderstanding caused by them. Our main aim is to indicate that database specialists should be invited for creation of very complex queries.
*Query Wizard* outputs are simple SELECT – FROM or aggregate SELECT – FROM – GROUP BY queries. Their SELECT part consists of all selected attributes; their FROM part contains the chosen table(s), possibly joined on identical values of their relating attributes. Unfortunately, the branch for creating aggregate queries opens only when at least one of the chosen attributes have a numerical data type. This complicates creation of queries that count the number of non-numerical elements (*How many books are in the list?*) Also, it does not allow introducing the DISTINCT keyword impeding other set of frequent questions (*How many different books are in the list?*) For that reason, *Query Wizard* must be used for very simple question only. All others require its combination with *Design View* and/or *SQL*.
In the language of laymen this results into two-step processes: *Using Query Wizard we select needed columns, using Query Design View (or SQL) we select requested rows*.
This principle fits well with a picture of SQL as a language operating separately over the table's columns and separately over its rows.
Let us presume that we ask: *Did John Doe publish a book with more than three hundred pages?* In the first step we have to decide which attributes (columns) play the role in our decision making. The complete response only requires the attributes *Author* and *Pages*.
On the other hand, we expect our students to generate responses having the "full logic for a common-sense database user". Taking this into our account, *Title* is another reasonable

candidate. Our experience shows that the majority of our students makes such identification. In fact, *Title* must only participate in this question: *Which books of John Doe have more than three hundred pages?* We stress to our students that these two questions are different and the interpretation of the first one depends on the content. When we are interested in the sheer fact "Yes or No", two attributes are sufficient. If we are interested in the books themselves, three are needed.
Notice that the interpretation of the meaning of the question: *Did John Doe publish a book with more than three hundred pages?* is individual. Some people understand it in its straight meaning, others in the extended one. Different interpretations of the same sentence are one of the most common sources of misunderstanding between users and developers. We point to the problem from very first days of our classes.
Similar problems appear with interpretations of results. When the query is formulated (using *Query Design View*) as below, it can generate an empty set of data. Its meaning is "No".




The teacher must explain that the result can be displayed in a more legible form e.g. by counting the number of records in the query (named *300 Pages*):

```
SELECT COUNT(*) AS [Number of Books with at least 300 pages]
FROM [300 pages];
```

Now the result will be 0, 1, 2, etc. depending on the factual number of the books with expected property. Interpreting *zero* as "*no book of the expected size*" is much more probable than giving the same meaning to the empty data set.
There are many opportunities for similar misunderstandings. For example, there is a difference between interpreting "AND" and "OR" in our common life and in computer languages. In *Query Design View*, the conditions connected by AND are in the same row, whilst those connected by OR are in different rows.
Again, the best method of pointing to this problem is using formulations of problems in a natural language. We train our students to distinguish among these subtle differences by series of questions like:
*Whose first name starts with "J"?*
*Whose surname starts with "J"?*
*Whose first name and surname starts with "J"?*

*Whose first name or surname starts with "J"?*
It is rather surprising for our students that the questions produce very different solutions. As forming their solutions is quite simple, the following question often appears: *How to formulate a query which produces all authors with the same first letter in their name and surname?* They are rather amazed that the problem is much more difficult and requires applying the LEFT function to the both attributes. Naturally, when they express their desire to learn it, we show them the way.
Similar students' demands are good entrances to SQL. Solving the problem in *Query Design View* is easy – but it strongly depends on Access specifics. As our desire is teaching concepts not tools, we always prefer more general ones. (It can be easily shown that not all problems are solvable using *Query Wizard*.)
Students are often surprised how small differences in the text formulation lead to substantially different solutions. We try to raise their curiosity by using appropriate problems. For example, the question: *How many pages does the longest book have?* results in the following simple SQL query:

```
SELECT MAX(Pages)
FROM Books;
```

Its extended interpretation (also asking: *Which book is it?*) uses the above statement as its subquery:

```
SELECT Author, Title, Pages
FROM Books
WHERE Pages = (SELECT MAX(Pages) FROM Books);
```

The above examples indicate that we also introduce SQL in our courses. Without it, the students might face problems to build up more complex queries. We advise them to solve the task using its "narrowed" interpretation (only producing the number of pages). The problem is easily solvable by *Query Wizard* and *Design View*. Let us presume that its result is 746. In the next step, the problem is transformed into: *Which book has 746 pages?* Again, *Query Wizard* and *Design View* suffice for solving it. Its result is:

```
SELECT Author, Title, Pages
FROM Books
WHERE Pages = 746;
```

Then, we ask the students what will happen when a book with 820 pages appears: *Will the query generate the new book?* Evidently, not. It will again generate the one with 746 pages. So, we need a query which does not depend on any combination of the books in our database. The best students easily conclude that replacing the number by the query that calculates it produces the right answer under any circumstances. We exploit their discovery and take a broad view to it as to a general method for creating subqueries.
Students frequently ask about differences between available tools. First, we conclude that there is no response covering all potential cases. Nevertheless, the guideline can look as follows:

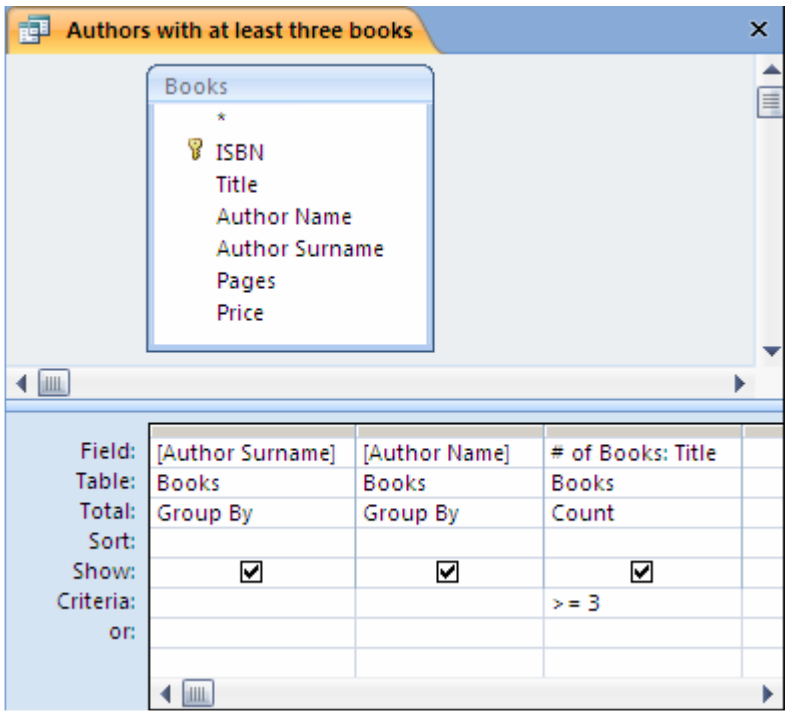
1. Simple conditions (comparisons to an attribute value) can be written directly into the particular column in *Design View*.
2. SQL is more appropriate for forming more complex conditions e.g. those combining several attributes.
3. In most cases, the SQL text is the final one. Nevertheless, switching between SQL and *Design View* sometimes leads to a faster query development. So, even if the text is

already in SQL but the developer sees an opportunity to enter some conditions in Design View, he/she should not hesitate doing so.

The situation can be exemplified by the problem: *Name the authors publishing at least three books.* As we teach our students to solve the problem in a stepwise manner, they often come to the following query containing GROUP BY but not HAVING yet:

```
SELECT [Author Surname], [Author Name], COUNT(Title) AS [# of Books]
FROM Books
GROUP BY [Author Surname], [Author Name];
```

This command is a partial solution of the problem as it shows all authors with their corresponding numbers of the books. The restriction has to bind the last column – the number of books. Switching to *Design View* and typing the condition solves the problem entirely.




Switching back to SQL reveals its equivalent:

```
SELECT [Author Surname], [Author Name], COUNT(Title) AS [# of Books]
FROM Books
GROUP BY [Author Surname], [Author Name]
HAVING COUNT(Title)>=3;
```

There are many interesting problems connected to the semi-automated creation of queries by end-users and to the relationships between questions (in a natural language) and their equivalents (queries in SQL-like language). Some research has been done on relationship between Query-By-Example languages and SQL (e.g. [4] and [5]) but we could not find any that applies the principle for teaching introductory database courses. One of the authors prepares his doctoral thesis aimed to enhance high-school and university database courses in the particular direction.

### *2.4 Forms*

Wizards are tools for common users, not for professionals. Even if they certainly use them as well, they are still capable to achieve their aims by standard tools. For that reason, one can claim that database developers spend their time and money for the wizard development because they believe that with their number and quality will also bring greater numbers of layman- or low-qualified-users.
In general, their conclusion is correct. On the other hand, this positive experience leads to creating more sophisticated wizard tools – so complex that the opposite effect appears: "*Too much cooks spoil the cake*". One can witness such effects in the newest version of *Access* – version *2007*. In the previous one, *Form Wizard* began with offering options: its user could decide whether he/she would start generating a form starting with an isolated table or with several ones. If the tables were related, their relationships were automatically included and resulted into forms with subforms.
The newest version does not propose any choice. When a table relates to other ones, all of them are automatically included. For a layman, creating a form without a subform becomes practically impossible. One has to create the more complex one – and then delete the unwanted parts. This can hardly be presumed as a user-friendly approach.
In addition to that, all controls in the wizard-generated form are interconnected by hidden links that prohibit moving a single control. The move of one box results in moving all so their relative positions remain same. The users can personalize their forms much harder than before.
As a result, instead of our intensive use of *Form Wizard* in the past, we now recommend our students to build their forms manually.

# 3 Supporting Distance Learning

A big portion of our students are studying at distance. For them, having additional support is a precondition for their success. It consists of three compatible sets of materials:

- **Textbook:** Our approach has been implemented in the textbook [1]. It not only covers the content of the course but also offers various strategies for building the database components as input masks, validation rules, complex expressions etc. It also explains differences between various types of queries, approaches to normalization, and others. The book is trying to keep the balance between theory and practice. Its writing style is quite simple ("newspaper-like") but sticks as much as possible to the level of exactness necessary for creating functioning database application. We believe that our text has to be readable and fun – one of our unwritten objectives is to make our readers to *enjoy* databases. So, only the most important details are in the book. We expect our students to become able to consult their application manuals and to use on-line help service effectively.
- **Sample databases:** The book is accompanied by a CD. To simplify the readers' orientation, its folders are organized by chapters. Text of each chapter is accompanied by dozens of sample databases serving for two main purposes. The first set exemplifies the concepts explained in the particular chapter. By opening it, the user can confront the idea and its implementation in a very detailed manner. The other group relate to assignments. Every of these databases appear in pairs: "empty" and "solved". The "empty" database contains all data necessary for commencing the solution in accordance to the assignment. The "solved" one shows a solution – not necessarily identical with that found by the student. When the student is trapped, he/she can confront his/her approach and find a way out.

Several of the databases appear recurrently throughout the book. In a stepwise manner, they demonstrate how the idea can be adapted, extended etc. It takes the reader forward to comprehend that the process can be performed at various levels of precision and complexity. The CD contains updated versions at each stage that we re-use them so that the whole class can start from the same point.

- **Videosequences:** Many operations necessary for design and development require manual skills. In classes, teachers can demonstrate them in order to help their students to perform them in the simplest and fastest manner. This function of teacher is replaced by videosequences in Flash [2]. By watching them, our distance-learning students can easily comprehend what should be done to complete a particular activity and how. As they can stop the video at any moment, they can combine their own hands-on activity with the presentation. Macromedia Captivate [3] is used for capturing the activity shown in the video. In the next step, the captured sequence is edited and a enriched by comments. The completed sequences are converted into the swf format. Due to that, they can be downloaded by any browser without a necessity to download other program(s).

Regular and high-quality communication between instructors and their students as well as among the students themselves is another *sine qua non* condition necessary for achieving adequate educational results. Its large part simulates real-life situations appearing during database development. The students review their partners' projects, express their opinion and expect the authors to defend their proposals – or to modify them in accordance to their suggestions. During the discussion, they are not requested to use exact database terminology. Expressing the ideas in the form and style that is comprehensible by the others is sufficient. This also simulates the situation our students may face in their future. They are supposed to be highly qualified users – not programmers or developers. As such they should be capable to evaluate proposed half-finished products, assess the degree to which it satisfies their expectations and propose their improvements. The ability to express one's opinion in a legible manner is for such individuals much more important qualification than terminology itself.

# 4 Conclusions

The students taking our courses are not computer scientist. They are future managers or teachers i.e. typical database users from application fields distant from Computer Science. *Database Management* does not belong among their basic subjects. The course has therefore to be short and efficient.

As you could see above, we have given careful thought to our struggling learner and not allowed the logic of the subject to dictate the order and approach. We have avoided introducing theory first. If anything, we have taken an inductive approach, using hands-on examples before explaining underlying theory. One can call it the "concrete-to-abstract" method. As the result, our course has a good balance between the needs of the learner and the nature of the subject. In many respects it is unique because of its mix of theory and hands-on, its easy style and its rigor, its "real world" examples to illustrate abstract theory.

We have deliberately adopted a lighter style and hope that we have not allowed ourselves to wander too far from the subject too often. Some people may feel that the approach is inappropriate for such a serious topic. It is true – one should not let an easy read lull the student into believing that databases are easy stuff! We opted for such a style because our unwritten aim is to get readers to enjoy databases and convince them that quality databases cannot be built without cooperation with specialists, good judgment for details and a sense for other partner's need.

As a result, our speeches during the course, textbook and accompanying materials balance between the needs of the learner and the nature of the subject. We have made an attempt to

make our course interesting and accessible. *We are not out to make experts, we want to make converts!* (People who can immediately create good applications, but who also understand that their future study can take them even farther.) In some cases, we evidently succeeded. Some students were so fascinated by the subject that they change their professional orientation and become database developers.
The course brings benefits also to those who are not so widely involved. As new and new database-oriented applications appear, the need for users familiar with database structure and design grows. Those who are capable of serving as "translators" between customers and developers can easily find their job. There is a huge demand for people with similar qualities. So we believe that the concept can be applicable not only in our local courses but in a much larger circle.

## References:

[1] Hvorecký, J.; Drlík, M.: **Pochopiť databázy** (Understanding Databases – in Slovak). Vysoká škola manažmentu, Bratislava (manuscript)
[2] Rebenschied, S: **Macromedia Flash 8 Professional**. 352 pp. ISBN: 978-80-251-1696-8
[3] Huettner, B.: **Macromedia Captivate: The Definitive Guide** (Wordware Applications Library). Wordware Publishing, Inc., 2005, 350 pp. ISBN-10: 1556224222.
[4] Özsoyoglu, G.; Wang, H.: **Example-Based Graphical Database Query Languages**. *Computer* 26, 5 (May. 1993), pp. 25-38.
[5] Thomas, J.; Gould, J.: **A Psychological Study of Query by Example**. *National Computer Conference*. AFIPS Anaheim, CA, 44: 1975, 439-445 pp.

## Author(s):

Jozef Hvorecký, Prof. Dr.
Vysoká škola manažmentu, Department of Information Sciences
Panónska cesta 17, 851 04 Bratislava, Slovakia
jhvorecky@vsm.sk

Martin Drlík, Dr.
Univerzita Konštantína Filozofa, Department of Informatics
Trieda A. Hlinku 1, 949 01 Nitra, Slovakia
mdrlik@ukf.sk