Teaching implementational aspects of distributed data management in a practical way

Conference Paper · July 2011

CITATION
1

READS
74

2 authors:

Christian Pape
University of Applied Sciences Fulda
11 PUBLICATIONS **15** CITATIONS
SEE PROFILE

Peter Peinl
University of Applied Sciences Fulda
46 PUBLICATIONS **128** CITATIONS
SEE PROFILE

CSECS 2011, pp. 000 - 000

The 7 Annual International Conference on
Computer Science and Education in Computer Science,

July 06-10 2011, Sofia, Bulgaria

TEACHING IMPLEMENTATIONAL ASPECTS OF DISTRIBUTED DATA MANAGEMENT IN A PRACTICAL WAY

Christian PAPE, Peter PEINL

***Abstract**:* One objective of a database management course is to enable students to participate in real-world database development processes. Very often deep knowledge of the underlying technology of database management systems is required to achieve practical tasks. Actual computer information systems are multi-tiered and distributed. Additional complexity is introduced by the fact that almost all of them are extremely heterogeneous. Such systems can only be designed, analyzed and correctly implemented with a thorough conceptual knowledge and hands on experience of distributed data management. This paper describes a comprehensive lab experiment to teach the above mentioned skills by the means of a complex programming assignment which was given to groups of 4 students. To accomplish the assignment it was necessary to at first understand the principal problems of the management of federated database systems and after that, implement a limited scope of functionality commensurate with the available time and resources.

***Keywords**: heterogeneous databases, distributed databases, education*

***ACM Classification Keywords**: H.2.5 Heterogeneous Databases; H.2.4 Distributed databases; K.3.2 Computer and Information Science*

1. Introduction

Database systems (DBS) have established themselves as a basic building block of almost all computer information systems (CIS). CIS all employ a commercial DBS to store and manage the underlying information. However, the visibility of DBS from the point of view of the end user of a CIS, such as an airline reservation system, an electronic commerce system, a stock trading system or a generic enterprise resource planning (ERP) system has been gradually reduced in the course of the past 40 to 50 years. DBS have moved in many instances from the "prime stage" of the primary data management tool as seen and manipulated by a systems developer or programmer to a more or less distant role of an indispensable, but less visible empowering technology into the "background" [15].

Moreover, CIS are steadily becoming more complex. Real world CIS are distributed and heterogeneous. Both heterogeneity and distribution have many important aspects to be dealt with in a computer science curriculum. Among those we will only mention the heterogeneity of data sources and the heterogeneity of software technologies and frameworks to bridge the gap between. The solution to this problem is often referred to as information integration technology, which includes Service Oriented Architectures (SOA) [3], J2EE [9] , Federated and Distributed DBS and many others. The creation of a data warehouse [8] from a set of heterogeneous data sources in a large and truly distributed environment is a prototypical and widely cited example of an information integration project.

The typical computer science curriculum comprises one or more courses on data(base) management and on software technologies dealing with distributed systems (middleware). There is a growing interest in the appropriate methods and content to be taught in such courses [1, 2], especially on the design and execution of practical lab assignments.

Because of the lack of time, the focus generally is on interfaces and use of the respective technologies, not on the in-depth understanding and implications of their architecture and functioning. Especially for young and inexperienced computer science students with a technical leaning this is a serious drawback, because they will have to understand and practically solve the problems originating from a multi-tiered heterogeneous information system architecture.

The authors of this paper share the opinions of [18] on the distinctive role of lab practical classes in computing education. Therefore, they strive to provide some hands-on experience and insight as part of their courses in database management, within the given limitations of time and resources. In [15] we reported on the design, execution and results of several small experiments aiming at making students performance aware, i.e. to drive home the realization that programs performing reasonably well on small amounts of data need not necessarily work equally well on large and huge amounts of data. In this case the students played the role as an application programmer, i.e. they performed tasks outside the database engine. In [13] we changed sides and made the students understand and perform some operations on the heart of a DBS, i.e. transaction management, in particular the functioning and implications of logging and recovery algorithms.

All our previous experiments delivered valuable insights to the students, but were conducted in a centralized environment. Given the overwhelming importance of the distributed nature of state of the art CIS, it was natural to devise an in kind exercise in distributed data management. In this paper we present our observations and results on the implementation/emulation of some core elements of a Federated Database System (FDBS) [7, 6] . The task was given to several groups of students (group size 4) as a programming assignment. It concluded a master course on distributed database systems (DDBS).

The remainder of the paper is organized as follows. Section 2 identifies some problems in distributed data management worthwhile to consider for teaching. It then explains the rationale for narrowing them down to the concrete assignment, Federated Database Systems (FDBS) and finally sketches the tasks given to the students. In section 3 the architecture of our FDBS and the framework given to the students is explained in greater detail. Section 4 then covers the design and execution of scenarios to check for the correctness and completeness of the students' implementations. Section 5 summarizes the results of our experiment. Finally, we discuss the results and relate our conclusions and possibilities for additional work in section 6.

2. Teaching data distribution

The general subject of the course being DDBS it was appropriate to concentrate on generic solutions to the information integration problem by DDBS, in particular because they had already been addressed in the conceptual part of the course. Additional justification arises from the fact that database systems in their distributed version actually were the first large scale distributed systems. Therefore a good part of the research into the general problems of distribution was directed to the original DDBS [12]. Many of the problems identified, the solutions found and the techniques and algorithms developed in the realm of distributed systems have been pioneered in the first research and commercial DDBS. Later on these techniques have been adopted, modified and refined to create generic and flexible software frameworks (middleware) that allow for the integration of other data sources that manage less structured data than normally found in databases. The same frameworks allow for the integration of applications and any kind of data source, be it a database or not. As a consequence a great number of problems of data distribution can already be studied and explained by taking a good look at the different types of DDBS.

Taking into account the limited amount of time and resources it was essential to define a *practical task* that built on as much of the experience and practical skills of the participating students, made use of the theoretical concepts already covered by the course and was feasible with the resources of the university, i.e. mainly the hardware and software infrastructure. Ideally one should do with a good working knowledge in one programming language (Java in our case) and the SQL database language. The aim of the task was to offer the students the opportunity to *practically discover* these techniques, get a deep understanding and good grasp of them and implement a selected subset of the solutions that are published in literature and may be refine them.

2.1 Federated database systems

DDBS may be classified in several ways depending on the degree of autonomy and heterogeneity of the participating centralized DBS. There exists ample literature describing in detail the state of the art and the evolutionary steps from centralized DBS (CDBS) to fully homogenous DDBS to completely heterogeneous DDBS and further on. Therefore we will refrain from lengthy repetitions and just refer the interested reader to one of the standard textbooks on (D)DBS [19].

The so-called Federated Database Systems (FDBS) are an important class of DDBS [17]. Their specific properties render them rather appealing in a general technical perspective. In particular they very well serve our goal, i.e. defining a lab exercise in distribution technology with limited time and resources. The term federation is borrowed from the political realm where it designates a group of entities (states) that have some common goals and principles and are loosely connected by a small set of mandatory rules, common institutions and regulations. Otherwise the states are almost completely independent and autonomous in their dealings and decisions.

In the database world a federation designates a certain number of full-fledged CDBS, which cooperate in an analogous way. FDBS do not intrude into or require any modification of the intrinsic functionality of any participating centralized DBS. in other words

"a federated database architecture is described in which a collection of independent database systems are united into a loosely coupled federation in order to share and exchange information." [5]

Because the centralized DBS will primarily want to preserve their autonomy, all the functionality enabling them to act as federation member will have to be provided *on top* of the CDBS. Whatever functionality is generally available from the CDBS will have to be employed to implement the rules and regulations of the federation. Though the following statement probably sounds a little bit superficial, it might be said that an FDBS can be simply regarded as a software layer on top of the participating CDBS that provides the programmers with a unified interface to access whatever data the federation makes available. This interface tries to hide as much as possible that the CDBS forming the federation may logically organize their data according to different data models (e.g. relational, object-oriented, semi-structured), use different data manipulation languages and are offered by different vendors. By this means application development is isolated from the underlying data models and data manipulation languages. The totality of the necessary adaptations and transformations are effected in the intermediate layer, the FDBS. Furthermore, the FDBS renders the physical distribution of the data totally transparent.

2.2 Task definition and restriction

After identifying FDBS as a suitable object of studying data distribution aspects, the task to be assigned to the students had to be properly limited to match available time and resources. Those given, it was

beyond question to demand anything that comes close to the full scale implementation of a federation layer of actual CDBS. Our first decision related to the various aspects of heterogeneity normally covered by the FDBS layer. Heterogeneity of the data models of the federated CDBS was eliminated by opting for the relational data model in all cases, which by all practical means also rid us of the problem of widely differing data manipulation languages, SQL being the lingua franca of relational database world. It might have been tempting to group relational DBS of various vendors into a federation. Nevertheless, we decided against that option. Though SQL is an internationally accepted standard, it leaves (too) much leeway to the implementors and hence there exist many small differences in SQL dialects. This would have put too much emphasis on language, i.e. syntax, processing in our assignment, potentially making it an exercise in compiler construction. As a consequence, we chose a single commercial DBS (*Oracle*). That choice was pure pragmatical because our students are familiar with the product and its intricacies from the basic database courses.

By totally eliminating heterogeneity we considerably simplified the task and made it possible to focus on distribution issues. Even in a totally homogeneous environment the transparent distribution of tables and its management poses enough technical problems worthwhile to study in such a course. Furthermore, even in the limited share of partitioning and fragment allocation of relations several options exist, i.e. vertical, horizontal and hybrid partitioning on the one hand and disjoint versus non disjoint partitioning (up to and including complete replication on each site) on the other hand. To again restrict the scope of the task, we demanded only horizontal partitioning to be implemented and ruled out any non disjoint scheme. The latter is also consistent with the view on a federation adopted by our exercise, i.e. as a set of independent and non redundant data sources. All but one partitioning scheme thus being

eliminated, the students were able to concentrate on the various aspects of distributed query processing in an FDBS [10], i.e. decomposing given SQL statements, allocating them to the appropriate subset of the federation and processing them on the chosen and independently acting DBS. Further details will be explained in the following subsection 2.3.

2.3 Architecture of our FDBS

The architecture of an FDBS has so far been sketched in a rather general fashion in subsection 2.1. In order to define a technical basis for a concrete program assignment proper interfaces have to be specified or chosen. Fortunately, the Java language comes with several interfaces and abstractions that almost ideally serve our purpose. Java comprises a standard interface (JDBC) that defines and gives access to a single relational DBS in a transparent fashion and allows the execution of SQL statements. For technical details see [16]. Given the existence of JBDC and its prevalence and ease of use in programming, it seemed obvious to try to profit from the familiarity of the students with the JDBC classes. In the end we came up with a concrete architecture for our limited scope FDBS layer that heavily relies on JDBC as the interface to the members of the federation and a slightly modified version of JDBC, which we defined and called Federated JDBC or FJDBC. FJDBC acts as the interface to application programs written in the Java language and delivers exactly the same functionality as JDBC, but it hides the distribution of the data.

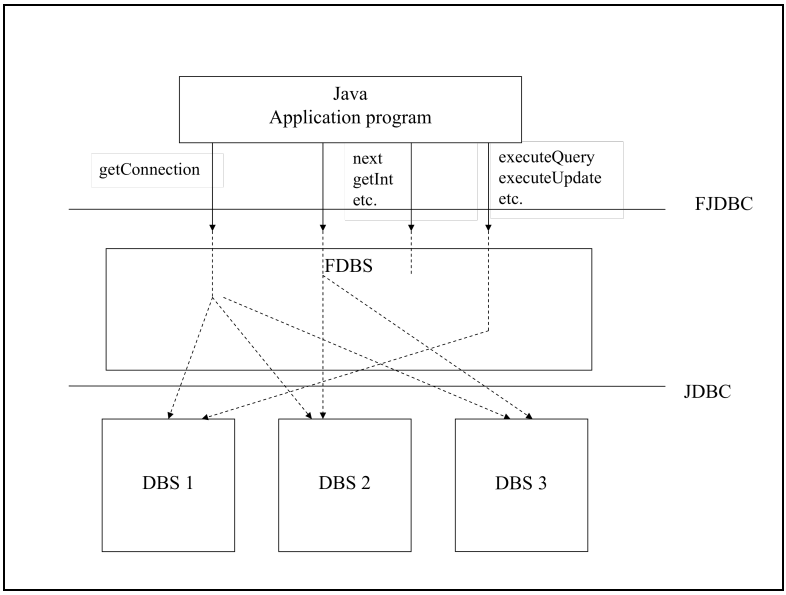

Figure 1 Federated database architecture

Figure 1 depicts the basic elements our FDBS layer. It explains the principal interaction between an application program written in Java at the top of figure 1 invoking the FJDBC interface and the FDBS layer that implements the FJDBC interface in the center of the figure. Underneath can be seen the JDBC interface that is in turn invoked by the FDBS layer to access three independent members of the federation at the bottom of figure 1. It is the task of the FDBS layer to map invocations of the methods of the FJDBC interface to zero or more invocations of appropriate JDBC methods accessing the federation members. Figure 1 also shows the names some of the methods common to our FJDBC and JDBC.

Imagine, for instance, an INSERT statement that adds a new tuple to a global relation which is transparently distributed over the three member DBS of the federation. In this case the FDBS layer has to determine the

correct partition and transform the FJDBC INSERT into a single JDBC INSERT into the proper DBS. UPDATE and DELETE statements are treated likewise. In order to make that decision the FDBS layer, among others, at least has to keep some meta data specifying the partitioning criteria.

Processing an SQL SELECT query is considerably more difficult than the requests mentioned above. This holds even if the SELECT refers only to a single global relation distributed over several federation members. Matters may get really complicated when a SELECT query dynamically links several global distributed relations via one or more relational join operators.

2.4 The concrete assignment

The assignment consisted of the implementation of the specific FDBS layer just described. The task includes, among others, syntax analysis of the SQL statements received by the FDBS layer. In order to limit the amount of work going into that part of the assignment, we specified a strictly limited set of SQL syntax to be processed. Among others we restricted the number of tables involved to two, i.e. we limited the number of joins to one. Similarly, the complexity of the logical expressions to be dealt with in the WHERE and HAVING clause were restricted. To give the students a chance to complete the assignment within the available time frame the extent of the functionality of SQL to be supported was limited to a very small subset of the language. Numerous other measures were taken to achieve that goal. They cannot be explained here for the sake of conciseness and the lack of space. It might only be mentioned that the students were given several sets of queries and updates for testing purposes.

Some of these test sets contained mandatory functionality, others optional one. The mandatory general tasks to be performed by our FDBS have been defined as:

- Federated database catalog to manage the distribution schemata
- Global partitioning of relations
- Syntax analysis of SQL commands
- Query analysis, query partitioning and query distribution
- Query optimization
- Implementation of a distributed join processing strategy
- Merging of multiple results and their management

Among the optional tasks to be performed by our FDBS were:

- Distributed transaction processing
- Implementation of an open two phase commit protocol
- Control and enforcement of database integrity constraints
- Implementation of SQL aggregate functions

3. Implementation

To make it as easy as possible for the students to start, we used our existing infrastructure of three *Oracle* 10g DBS servers[1]. There are depicted in figure 2. All servers run the same version of the DBMS software and are all accessible by the students on the campus network and also remotely from their home.

[1] Each DBS is named after a volcano that made history due to a horrific eruption

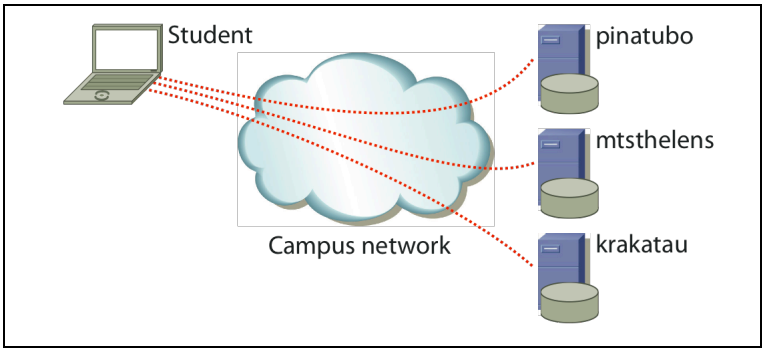

Figure 2 Database topology

To establish a common starting point for the students from which they could implement their solution we created a small framework of interfaces and classes following the style of the Java JDBC framework:

FedConnectionInterface: This interface is the foundation of the framework. It defines the methods used for transaction processing like *commit()* and *rollback()* and also allows to enable or disable the auto-commit feature.

FedPseudoDriverInterface: The pseudo driver interface creates an instance of class implementing the FedConnectionInterface interface.

FedResultSetInterface: This interface represents a result-set of a database query. Methods like *next()*, *getString()*, *getInt()* and *getColumnCount()* are defined by this interface.

FedStatementInterface: Database statements are specified using this interface. Familiar methods like *executeQuery()* and *executeUpdate()* are declared.

FedException: Specialization of *java.lang.Exception* used for exception-management inside the framework.

FedTestEnvironment: The main testing environment for a student's implementation. This class uses an instance of a class implementing the FedConnectionInterface interface and runs a number of testing and validation SQL-scripts on this connection. Status and timing information are returned on standard output. The class also ensures, that SQL-commands like *COMMIT* and *ROLLBACK* are correctly delegated to the corresponding methods. Figure 3 summarizes the interfaces and classes we have defined in the form of an UML diagram.

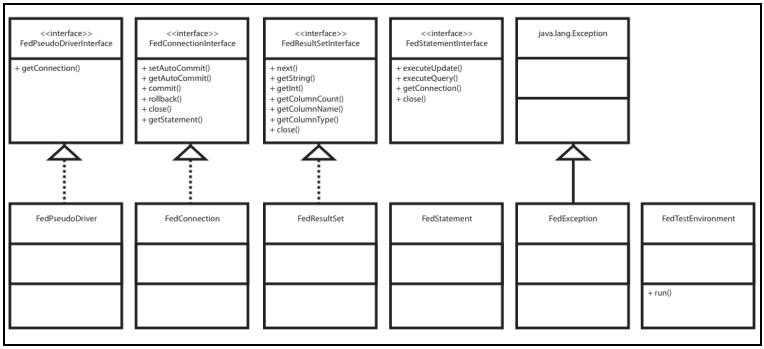

Figure 3 UML diagram of the used interfaces and classes

The student objective is to implement these interfaces in a way that the data is correctly partitioned over the three database management systems. This includes to correctly identify DDL-queries (data definition language) like *CREATE TABLE* or *DROP TABLE* to be executed on all three database servers. Of course DML-queries (data manipulation language) must also be parsed and modified: insertion of data must be made on the correct database partition and updates must be issued on all affected partitions. Tuples can move from one database partition to another, if altered attributes are included in the partitioning criteria. Last but not least TCL-commands (transaction control) to commit and rollback a transaction must be implemented by the students. Students have

always to ensure reliability and consistency of their implementation and also performance is important.

4. Validation

In order to compare the quality of the students' work as well as to check the completeness and correctness of the various implementations we designed a set of test suites each consisting of a set of function wise related SQL statements.

Those test suites were sequentially executed by feeding them into the students' implementation of the FDBS layer by a driver program provided by us. Some details have already been explained in section 3. Among other things the driver program records any errors or exceptions that have been raised by the FDBS layer or otherwise, the name of the test suite, the processing time of the whole suite and the individual SQL statements. To be able to check the correctness of the results they are listed as well[2]. An excerpt of the recordings of the program driving the tests is shown in figure 4.

[2] This of course requires a careful design of the SQL queries with respect to the actual database contents in order to minimize the amount of the output

```
************************************************************************
Executing script file 'Test/INSERTPASSENGERS.SQL'...
File 'Test/INSERTPASSENGERS.SQL', 78 operations, 1864 milliseconds
************************************************************************

************************************************************************
Executing script file 'Test/INSERTFLIGHTS.SQL'...
File 'Test/INSERTFLIGHTS.SQL', 100 operations, 2876 milliseconds
************************************************************************

************************************************************************
Executing script file 'Test/INSERTBOOKINGS.SQL'...
File 'Test/INSERTBOOKINGS.SQL', 188 operations, 5695 milliseconds
************************************************************************

************************************************************************
Executing script file 'Test/PARSELS1T.SQL'...
--> /* SELECT ONE TUPLE */ <--
Executing "SELECT * FROM FLUGLINIE WHERE FLC = 'BA'"...

FLC             LAND            HUB             NAME            ALLIANZ
--------------- --------------- --------------- --------------- ---------------
BA              GB              null            British AirwaysOneWorld

--> /* SELECT FROM ONE PARTITION */ <--
Executing "SELECT FLC, NAME, LAND FROM FLUGLINIE WHERE NAME = 'United Airlines'"...

FLC             NAME            LAND
--------------- --------------- ---------------
UA              United AirlinesUSA
```

Figure 4 Sample output of a student's program

5. Results

The test suites mentioned in section 4 served three main purposes, namely to

1. check for the correctness of the implementation,
2. check for the completeness of the implementation,
3. evaluate the performance of the implementation.

In a first step correctness and completeness of the implementation were validated, performance came later. To achieve this two sets of data were loaded into the federated database. The same database schema, however, was kept for all tests. The first set of data was hand crafted and consisted of just a few dozen tuples. The second set of data, employed in the performance tests, was considerably larger, i.e. about 60000 tuples, the majority of which were stored in two horizontally partitioned relations. The correctness tests addressed

- maintenance of the partitioning under insert, update and delete statements,
- enforcement of integrity constraints defined by SQL, such as primary key, foreign key, etc.,
- results of queries, in particular
    - joins,
    - logical expressions in the where clause,
    - aggregate functions in the select and having clause,
    - merged result set,
    - order within the result set.

Completeness was tackled by hand crafting a large set of queries. Two performance related aspects of the students' implementations were evaluated, i.e.

- the data structures and algorithms employed to implement distributed joins,
- bulk or mass inserts.

6. Discussion and conclusions

Though a great many quantitative results have been assembled and might have been presented, it is more appropriate to discuss our experiences and findings in a rather qualitative way. We had a total of seven groups partaking in the exercise. The range of theoretical knowledge, previous practical experience, degree of computer proficiency and general motivation was huge. This was the case when the students enrolled in the course, and the impressions lingers with the authors of the paper that the gap has not been narrowed in any way in the meantime. On the contrary, the good students used the opportunity given to them by the assignment to gain some valuable insight. They implemented a lot of the optional features and even more than those.

One group even implemented a transaction manager to guide and control an open two-phase commit protocol using the respective *Oracle* interface. However, the average students just went for the minimal effort strategy, i.e. just implement the mandatory features and wasted a chance for hands on learning about distributed systems. The same gap became apparent when comparing the execution time measured on the large database. To give just an example, the duration to mass load 60000 tuples into the database ranged between 15 and 80 minutes, and one group did not succeed at all. The spread of the numerical results with respect to join processing were similar, but not as drastic as in the bulk load situation.

In all, the assignment has shown to be well suited for highly motivated student and should be continued and refined in the future.

Bibliography

[1] Sikha Bagui and Leo Ter Haar. Database education in the new millenium. J. Comput. Small Coll., 24:80–87, April 2009.

[2] Joobin Choobineh and Sudha Ram. Practical aspects of teaching an applied database course. SIGMIS Database, 27:70–82, June 1996.

[3] Thomas Erl. Service-Oriented Architecture: Concepts, Technology, and Design. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2005.

[4] T. Haerder and E. Rahm. Datenbanksysteme: Konzepte und Techniken der Implementierung. Springer Berlin, 2001.

[5] Dennis Heimbigner and Dennis McLeod. A federated architecture for information manage- ment. ACM Trans. Inf. Syst., 3:253–278, July 1985.

[6] David K. Hsiao. Federated databases and systems: part i — a tutorial on their data sharing. The VLDB Journal, 1:127–180, July 1992.

[7] David K. Hsiao. Federated databases and systems: part ii — a tutorial on their resource consolidation. The VLDB Journal, 1:285–310, October 1992.

[8] W. H. Inmon. The data warehouse and data mining. Commun. ACM, 39:49–50, November 1996.

[9] B.V. Kumar, P. Narayan, and T. Ng. Implementing SOA using Java EE. Java series. Prentice Hall, 2009.

[10] Ee-Peng Lim and Jaideep Srivastava. Query optimization and processing in federated database systems. In Proceedings of the second international conference on Information and knowledge management, CIKM '93, pages 720–722, New York, NY, USA, 1993. ACM.

[11] Dennis McLeod and Dennis Heimbigner. A federated architecture for database systems. In Proceedings of the May 19-22, 1980, national computer conference, AFIPS '80, pages 283–289, New York, NY, USA, 1980. ACM.

[12] M. Tamer Ozsu and P. Valduriez. Principles of distributed database systems. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1991.

[13] Christian Pape and Peter Peinl. Improving learning experience on conceptual topics of database management systems. In Proceedings 6th International Conference on Computer Science and Education in Computer Science, Fulda/Munich, Germany, 2010.

[14] Roy P. Pargas. Reducing lecture and increasing student activity in large computer science courses. SIGCSE Bull., 38:3–7, June 2006.

[15] Peter Peinl. Teaching database technology and the importance of performance issues. In

Proceedings 5th International Conference on Computer Science and Education in Computer Science, Boston, USA, 2009.

[16] George Reese. Database Programming with JDBC and Java, Second Edition. O'Reilly & Associates, Inc., Sebastopol, CA, USA, 2nd edition, 2000.

[17] Amit P. Sheth and James A. Larson. Federated database systems for managing distributed, heterogeneous, and autonomous databases. ACM Comput. Surv., 22:183–236, September 1990.

[18] Simon, Michael de Raadt, Ken Sutton, and Anne Venables. The distinctive role of lab prac- tical classes in computing education. In Proceedings of the

6th Baltic Sea conference on Computing education research: Koli Calling 2006, Baltic Sea '06, pages 54–60, New York, NY, USA, 2006. ACM.

[19] Jeffrey D. Ullman, Hector Garcia-Molina, and Jennifer Widom. Database Systems: The Complete Book. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1st edition, 2001.

Authors' Information

Christian PAPE, Diplom-Informatiker, University Of Applied Sciences Fulda, Marquardstraße 35, 36039 Fulda, Germany, Christian.Pape@informatik.hs-fulda.de

Major Fields of Scientific Research: *Programming, Network Communications, Database Systems, Information Systems*

Peter PEINL, Prof. Dr.-Ing. Prof. h.c., University Of Applied Sciences Fulda, Marquardstraße 35, 36039 Fulda, Germany, Peter.Peinl@informatik.hs-fulda.de

Major Fields of Scientific Research: *Database Systems, Transaction Management, Information Retrieval*