MSc Distributed Computing Systems Engineering

Department of
Electronic & Computer
Engineering

Brunel University

Performance Measurements of a Partitioned Database

Simone Ludwig

September 2000

A Dissertation submitted in partial fulfilment of the requirements for the degree of Master of Science

MSc Distributed Computing Systems Engineering

Department of
Electronic & Computer
Engineering

Brunel University

Performance Measurements of a Partitioned Database

Author: Simone Ludwig

Supervisor: Peter van Santen

September 2000

A Dissertation submitted in partial fulfilment of the requirements for the degree of Master of Science

Acknowledgements

First of all, I have to thank my supervisor Mr. Peter van Santen for his advise and support during this dissertation.

Then I would like to thank Michael Höfer for his useful recommendations during the practical work and furthermore Ulrich Becker and David Cook for reading through this dissertation.

Abstract

This dissertation is about measuring the performance of a distributed and partitioned database compared to a non-partitioned database located on a single server machine.

A client application was designed and implemented in Java to access the partitioned database remotely via an application server. As a communication technique between the client and the application server, Remote Method Invocation (RMI) was used. The database, which was the DB2 Extended Enterprise Edition (EEE) from IBM running on Windows NT Server, was filled with different kinds of data, e.g. binary mp3 files or common ASCII files, which could be accessed by the client application in several ways, for example by searching for a key word or listening to an mp3 music file.

After the installation and configuration of the database, including creating tables and storing data, performance measurements are done. Two performance measurement tools are taken, which mainly measure the access time for each SQL statement used in the written Java application. Additionally, a few other performance parameters were measured and displayed with a Performance Monitor.

Table of Contents

Acknowledgements **3**

Abstract **4**

Table of Contents **5**

Table of Abbreviations **9**

1 Introduction **10**

1.1 Distributed Systems 10

1.2 Outline Design 12

1.3 Proposal 13

1.4 Structure of the Dissertation 14

2 Software Development **15**

2.1 Architecture 15

2.1.1 Advantages of the 3-tier architecture 16

2.1.2 3-tier Implementation in DB2 17

2.2 Client 18

2.2.1 List of Requirements 18

2.2.2 Use Cases 18

2.2.3 Design 19

2.2.3.1 Class Diagram 20

2.2.3.2 Class Description 21

2.2.3.3 Client Description 22

2.2.4 Implementation 24

2.3 Application Server 26

2.3.1 Introduction to Remote Method Invocation 26

2.3.1.1 Implementation 27

2.4 Database Server 31

2.4.1 Database Design 31

2.4.2 Table Descriptions 32

2.4.3 Data for the database 33

2.4.3.1 Data for table STUDENTS 34

2.4.3.2 Data for table FILES 35

2.4.3.3 Data for table COURSES 36

2.4.3.4 Technical Data 36

2.5 Testing 37

2.5.1 Introduction to Testing 37

2.5.1.1 Test Tool JUnit 37

2.5.1.2 Implementation steps for testing the database accesses 38

2.5.2 Tests carried out 40

3 Installation and Configuration of the DB2 database 41

3.1 Description of the DB2 Universal Database 41

3.2 Partitioned Database 41

3.3 Nodegroups and Data Partitioning 42

3.4 Instances 44

3.5 Fast Communication Manager 45

3.6 Installation of the DB2 Database 46

3.7 Configuration of the DB2 Database 47

4 Performance Measurements 48

4.1 Introduction into Processing of a Partitioned Database 48

4.2 Defining DB2 Performance 49

4.3 Measurements 50

4.3.1 Benchmark Tool db2batch 50

4.3.1.1 Creating a Benchmark Program 52

4.3.2 DB2 Performance Monitor 54

4.3.3 Measurements Methods 55

4.3.3.1 SQL Transaction Measurements 56

4.3.3.2 Application Transaction Measurements 58

4.3.3.2.1 Measurements with db2batch tool 58

4.3.3.2.2 Measurements within the File Administration Application 59

4.3.3.2.3 Search within the CLOB column of the database 60

4.3.3.2.4 Measurements with Performance Monitor 62

4.3.4 Summary of the Measurements 63

5 Conclusion 64

6 Management of the Project 66

References 68

Bibliography **69**

Appendix A **70**

Used Software 70

Appendix B **71**

Contents of the CD 71

Source Code GUI 71

Source Code for storing data into the database 72

CD 72

Appendix C **73**

Performance Measurements 73

Partitioned database TEST 73

Non-partitioned database TEST2 74

Table of Abbreviations

CAE	Client Application Enabler
CLI	Client Link Interface
DB	Database
DB2 EEE	Database from IBM (Extended Enterprise Edition)
DBA	Database Administrator
DBMS	Database Management System
DDB	Distributed Database
DDBMS	Distributed Database Management System
DSS	Decision-Support
FCM	Fast Communication Manager
GUI	Graphical User Interface
JDBC	Java Database Connectivity
MMP	Massively Parallel Processing
OLTP	Online Transaction Processing
OOA	Object Oriented Analysis
OOD	Object Oriented Design
PM	Performance Monitor
RMI	Remote Method Invocation
SDK	Software Developer Kit
SMP	Symmetric Multi-Processing
SQL	Structured Query Language
WWW	World Wide Web

1 Introduction

1.1 Distributed Systems

At no previous time in the history of computing have there been so many challenging innovations vying for the attention of information systems engineers as there are at present. Technologically, advances are continually being made in hardware, software and “methodologies”. Improving speeds of, possibly parallel, action and the increased size of rapid access stores available, along with new announcements of products and new ideas from researchers bring with them demands from the user population for their exploitation. For applications, even if users are not aware of particular developments relevant to their operations, there is a perpetual demand for more functionality, service, flexibility and performance. As such, when designing a new information system, or prolonging the “life” of an old one, the information engineer must seek ways of linking solutions offered by the technologists to the needs of users’ applications.

One area in which solutions are becoming increasingly viable is in distributed information systems. These are concerned with managing data stored in computing facilities at many nodes linked by communications networks. An important role in distributed information systems plays the distributed database technology which involves the merging of two divergent concepts, namely integration through the database element and distribution through the networking element.

A typical distributed database system is shown in Figure 1.

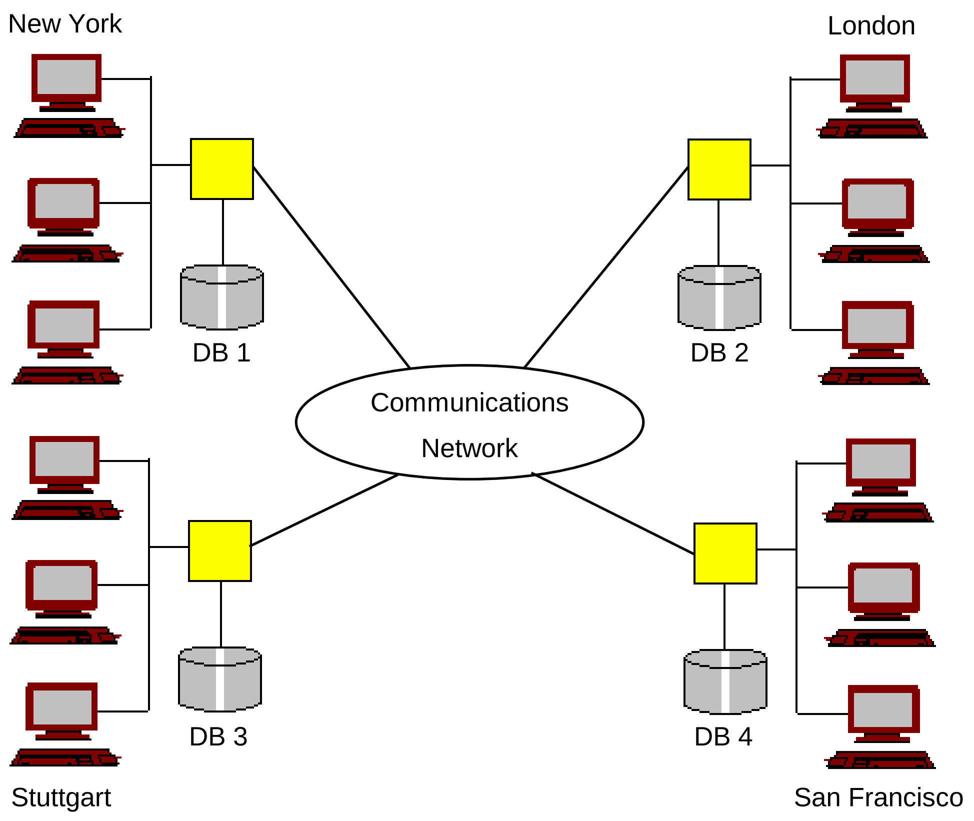


Figure 1: Typical distributed database system

The distribution aspect is provided by distributing the data across the sites in the network, while the integration aspect is provided by logically integrating the distributed data so that it appears to the users as a single, homogeneous database (DB). Centralised database management systems (DBMSs), by contrast, require both logical and physical integration. A distributed database (DDB) can be defined as a logically integrated collection of shared data, which is physically distributed across the nodes of a computer network. A distributed database management system (DDBMS) is therefore the software to manage a distributed database in such a way that the distribution aspects are transparent to the user. A centralised DBMS is a system which manages a single DB, whereas a DDBMS is a single DBMS which manages multiple DBs. The terms global and local are often used when discussing

DDBMSs (and indeed distributed systems in general) in order to distinguish between aspects which refer to a single site (local) and those which refer to the system as a whole (global). For example, the local DB refers to the DB stored at one site in the network, whereas the global DB refers to the logical integration of all the local DBs. Note that the global DB is a virtual concept since it does not exist physically anywhere.

1.2 Outline Design

Figure 2 illustrates the outline design with multi-clients and a distributed database, shown as database server one and two, connected via a communication network. One of the database server machines is responsible for the management of the partitioned data. Therefore, an inter-communication between those two machines must be established.

Here the technical description of the outline:

- Java application is running on client 1.
- Win NT server includes the distributed and partitioned database, whereby database server 1 is acting as a master and database server 2 as a additional node and the application server which is also running on the database server. The database server is responsible for the data exchange between application and database.

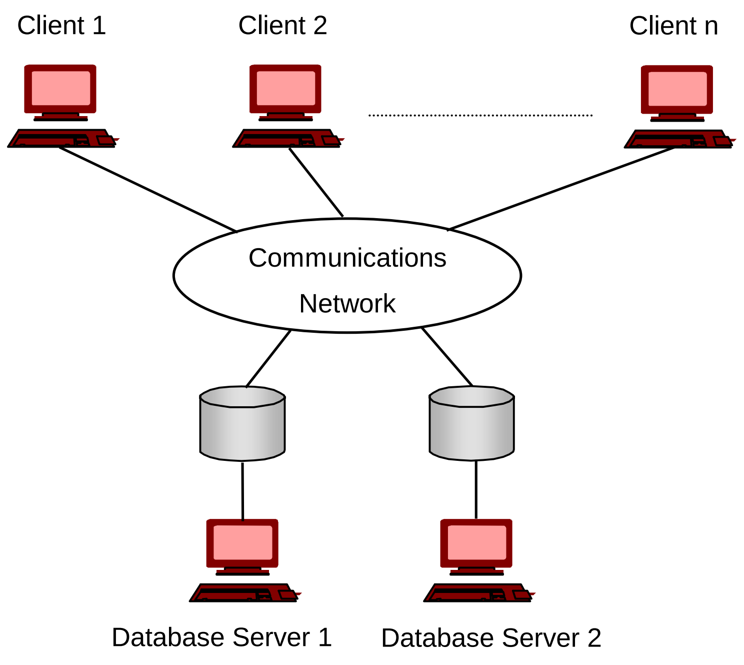


Figure 2: Outline design

1.3 Proposal

Like seen in the outline design, the multi-clients / multi-server solution will be realised.

Firstly, the application will be programmed in order to enable access to the database. A Java application will be chosen as the File Administration Program. Students can store files (text, html and mp3 files) for each module taken into the database and all necessary information of the students will be provided in the database.

Secondly, the database will be installed and configured such that the partitioning of the data can be achieved. The Universal database DB2 EEE (extended enterprise edition) from IBM will be used as this is a scalable, multimedia, web-enabled database.

Finally, performance measurements of the partitioned database will be completed. The performance, mainly the access time of the partitioned database will be compared with the access time of the non-partitioned database running on a single machine. The performance should be improved by a factor of two when using two machines as a partitioned database server.

1.4 Structure of the Dissertation

This dissertation is organised as follows.

The second chapter describes the software development process for the File Administration Application. In this chapter the complete software life cycle is introduced from requirements, use cases, design, implementation and testing.

The third chapter describes the installation and configuration of the partitioned database.

The fourth chapter introduces the performance measurements which are then concluded in the subsequent chapter, highlighting possible improvements.

The final chapter describes the management of this dissertation and includes the milestone plan.

The appendices contain practical work including coding and information for possible future research. Everything is stored on a CD.

2 Software Development

The software was developed by applying the software life cycle, which describes the steps from requirements, use cases, design, implementation and testing.

Note that the chapter Client, Application Server and Database Server was structured by the theoretical order and not by the implementation order.

2.1 Architecture

A 3-tier client-server architecture was adopted for this dissertation (see Figure 3).

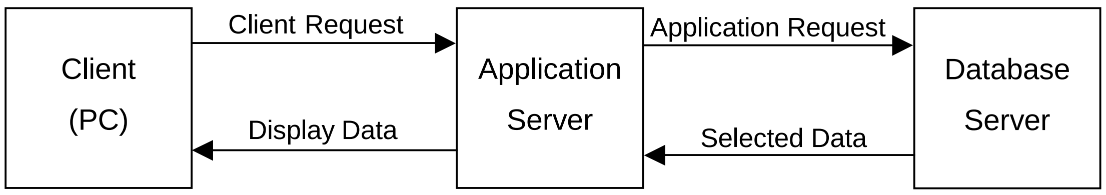


Figure 3: 3-tier client-server architecture

The client-tier is responsible for the presentation of data, receiving user events and controlling the user interface. The actual business logic has been moved to an application-server.

The application-server-tier is new, i.e. it is not present in 2-tier architecture in this explicit form. Business-objects that implement the business rules are implemented here, and are available to the client-tier. This level forms the central key to solve 2-tier problems. This tier protects the data from direct access by the clients. The object oriented analysis (OOA) aims to record and abstract business processes in business-objects.

The database-tier, or in general the server-tier, is responsible for data storage. Besides the widespread relational database systems, existing legacy systems databases are often re-used here. It is important to note that boundaries between tiers are logical. It is possible to run all three tiers on a single (physical) machine, however it is important that the system is neatly structured, and that there is a well planned definition of the software boundaries between the different tiers.

To relate it to the File Administration Application, the client-tier is the GUI (Graphical User Interface) which allows client requests. The application server provides the methods for accessing the database server and the database server is the database itself, where the data is stored in tables.

2.1.1 Advantages of the 3-tier architecture

The differences between the 3-tier and the 2-tier architecture (where the application server is also included in the client tier) are listed below:

- Clear separation of user-interface-control and data presentation from application-logic. Through this separation more clients are able to have access to a wide variety of server applications. The two main advantages for client-applications are firstly, quicker development through the re-use of pre-built business-logic components and secondly, a shorter test phase, as the server-components have already been tested.
- Re-definition of the storage strategy does not influence the clients.
- Business-objects and data storage are brought as close together as possible, ideally they should be together physically on the same server.

- In contrast to the 2-tier model, where only data is accessible to the public, business-objects can place applications-logic or "services" on the net.
- The authorisation of the system is simpler than that of thousands of "untrusted" client-PCs. Data protection and security is simpler to obtain. Therefore, it makes sense to run critical business processes, that work with security sensitive data, on the server.
- Dynamic load balancing: if bottlenecks in terms of performance occur, the server process can be moved to other servers at runtime.
- Change management is easier and faster, e.g. to exchange a component on the server than to furnish numerous PCs with new program versions. It is, however, compulsory that interfaces remain stable and that old client versions are still compatible.

The 3-tier architecture was chosen in order to have a clear separation of user-interface-control and data presentation from the application-logic.

2.1.2 3-tier Implementation in DB2

The 3-tier architecture is implemented in DB2 (see Figure 4). Calls to JDBC (Java Database Connectivity) are translated to calls to DB2 CLI, through Java native methods. This dependency requires that the DB2 CAE (Client Application Enabler) component is installed at the client. A JDBC request flows through DB2 CLI to the DB2 server through the normal CAE communication flow.

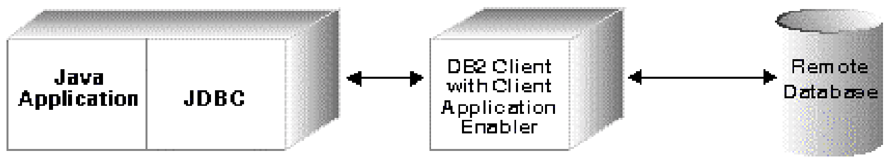


Figure 4: DB2's JDBC Application Implementation

The other possibility would be the JDBC Applet Implementation. The difference between the two means that applets need to be treated differently than applications, as applets are delivered through the World Wide Web (WWW). However the JDBC Application Implementation was selected due to security facts which are present in the JDBC Applet Implementation.

2.2 Client

2.2.1 List of Requirements

The list of requirements describe the users' needs of the system. The following list describe them:

100 All attributes stored in the database should be displayed on the screen.

200 It should be possible to search for the last name, first name and a file.

300 Viewing of a html file or text file should be possible.

400 It should be possible to play a mp3 file.

500 Downloading of files should be provided.

2.2.2 Use Cases

The use case modelling represent the system as a set of different major task areas (the use cases). It identifies the main tasks (scenarios) within each use case - in normal operation and under exceptional conditions. This defines the major functions performed by the system.

There are five use cases in the program of the File Administration Application, these are:

- List all attributes (last name, first name, e-mail address, course, text file, html file and mp3 file)
- Search for (last name, first name, file or a defined string within a file)
- View file (text file or html file)
- Play file (mp3 file)
- Download file (text file, html file or mp3 file)

The use cases are executed by the user. “List all attributes” is executed when the application is started and the other use cases can be selected by the user.

The use case model is shown below.

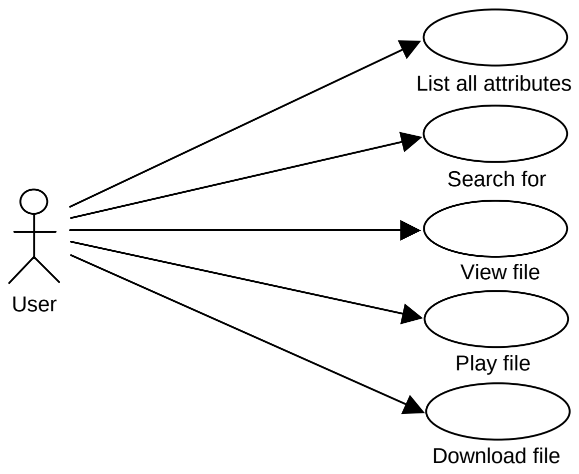


Figure 5: Use case Model

2.2.3 Design

The object-oriented design (OOD) is about designing classes and objects to implement the solution. It describes the design in more detail and takes the implementation problems into account.

The class modelling determine the classes and their attributes (variables) which are necessary to implement the scenarios. It shows classes and their inter-relationships.

2.2.3.1 Class Diagram

The class diagram represented in Figure 6, shows the static class dependencies of the File Administration Application and the classes of the application server. All classes are shown in this figure with exception of standard Java classes. The application uses RMI (Remote Method Invocation) to communicate between the server and the client(s) machine(s). For the application the classes `GUI`, `ViewFile` and `ClosableFrame` were implemented. For the database server the classes `RmiIntServer`, `RmiIntInterface` and `ResultValues` were implemented. The class `DatabaseTestCase` was necessary for testing.

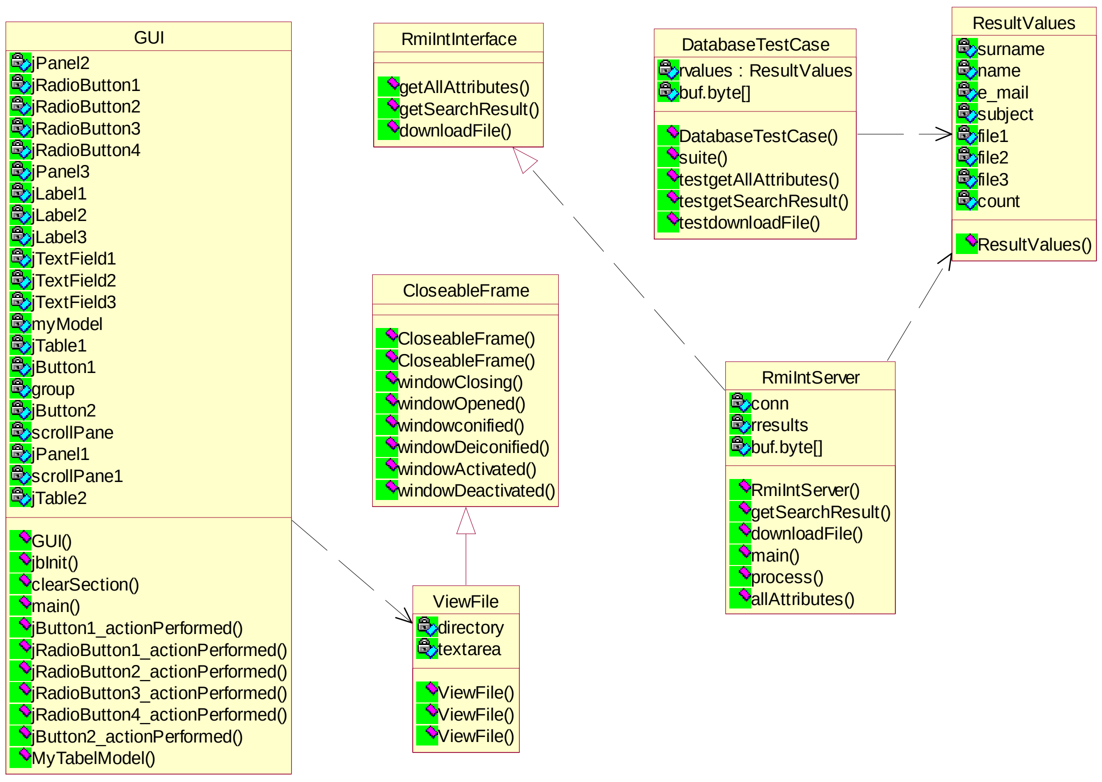


Figure 6: Design

2.2.3.2 Class Description

The following description of each implemented class describe these in more detail and shows its functionality.

`GUI`	is the main class of the design. The `GUI` is the interface for the client which enables processes of accessing the database to be controlled.
`CloseableFrame`	contains the necessary attributes and methods for the class `ViewFile` in order to input the settings for the frame.
`ViewFile`	displays the chosen file in a scrollable text area.

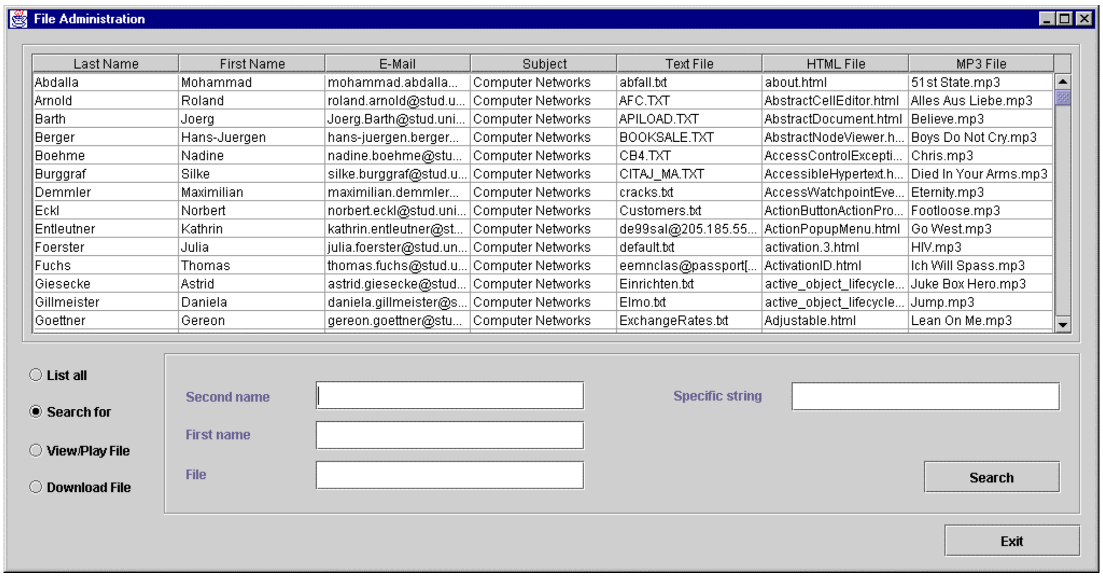
`RmiIntServer`	is the main class on the server side. It provides all necessary methods for the `GUI` to access the database.
`RmiIntInterface`	declares all methods for the RMI (Remote Method Invocation) communication between the client and the server machine.
`ResultValues`	contains all return values. The values requested from the client side are provided on the server side.
`DatabaseTestCase`	is a test class for testing the database methods by using the `JUnit` tool. All three used methods within the written Java application are tested here.

2.2.3.3 Client Description

The File Administration Application, which is illustrated in Figure 7, has the following features:

- List all: All rows of the database regarding the tables with student information, course information and the stored files are listed in a spreadsheet table.
- Search for: allows the user to search for the second name, first name, a file or a specific string in the text files or html files.
- View/Play File: allows the user to display a html file or text file on the screen (see Figure 8) or starts an MPEG player loaded with the chosen mp3 file.

- Download File: enables files to be downloaded from the database in order to store them on the hard drive.



File Administration

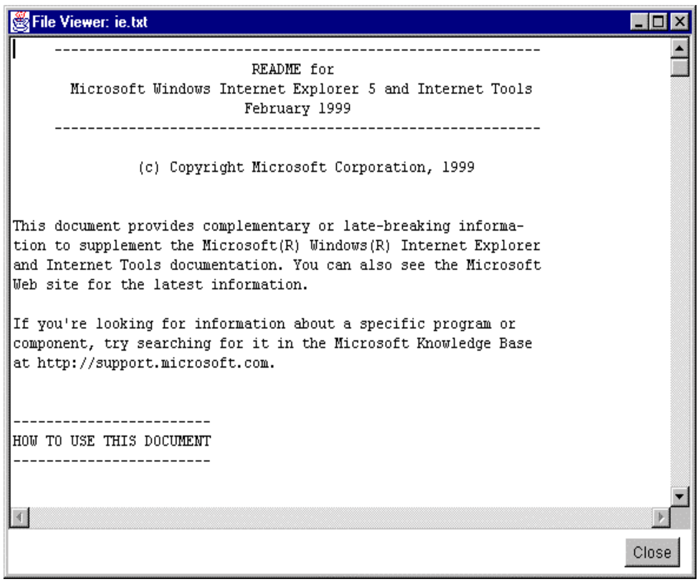
Last Name	First Name	E-Mail	Subject	Text File	HTML File	MP3 File
Abdalla	Mohammad	mohammad.abdalla...	Computer Networks	abfall.txt	about.html	51st State.mp3
Arnold	Roland	roland.arnold@stud.u...	Computer Networks	AFC.TXT	AbstractCellEditor.html	Alles Aus Liebe.mp3
Barth	Joerg	Joerg.Barth@stud.uni...	Computer Networks	APILOAD.TXT	AbstractDocument.html	Believe.mp3
Berger	Hans-Juergen	hans-juergen.berger...	Computer Networks	BOOKSALE.TXT	AbstractNodeViewer.h...	Boys Do Not Cry.mp3
Boehme	Nadine	nadine.boehme@stu...	Computer Networks	CB4.TXT	AccessControlExcepti...	Chris.mp3
Burggraf	Silke	silke.burggraf@stud.u...	Computer Networks	CITAJ_MA.TXT	AccessibleHypertext.h...	Died In Your Arms.mp3
Demmler	Maximilian	maximilian.demmler...	Computer Networks	cracks.txt	AccessWatchpointEve...	Eternity.mp3
Eckl	Norbert	norbert.eckl@stud.uni...	Computer Networks	Customers.txt	ActionButtonActionPro...	Footloose.mp3
Entleutner	Kathrin	kathrin.entleutner@st...	Computer Networks	de99sal@205.185.55...	ActionPopupMenu.html	Go West.mp3
Foerster	Julia	julia.foerster@stud.un...	Computer Networks	default.txt	activation.3.html	HIV.mp3
Fuchs	Thomas	thomas.fuchs@stud.u...	Computer Networks	eemnclas@passport[...	ActivationID.html	Ich Will Spass.mp3
Giesecke	Astrid	astrid.giesecke@stud...	Computer Networks	Einrichten.txt	active_object_lifecycle...	Juke Box Hero.mp3
Gillmeister	Daniela	daniela.gillmeister@s...	Computer Networks	Elmo.txt	active_object_lifecycle...	Jump.mp3
Goettner	Gereon	gereon.goettner@stu...	Computer Networks	ExchangeRates.txt	Adjustable.html	Lean On Me.mp3

○ List all
● Search for
○ View/Play File
○ Download File

Second name:
First name:
File:
Specific string:

Figure 7: File Administration Application

The View File Menu (Figure 8) opens when the chosen file is found in the database, otherwise a message window appears on the screen.



File Viewer: ie.txt

```
     ------------------------------------------------------------
                          README for
        Microsoft Windows Internet Explorer 5 and Internet Tools
                         February 1999
     ------------------------------------------------------------

               (c) Copyright Microsoft Corporation, 1999


This document provides complementary or late-breaking informa-
tion to supplement the Microsoft(R) Windows(R) Internet Explorer
and Internet Tools documentation. You can also see the Microsoft
Web site for the latest information.

If you're looking for information about a specific program or
component, try searching for it in the Microsoft Knowledge Base
at http://support.microsoft.com.


------------------------
HOW TO USE THIS DOCUMENT
------------------------
```

Figure 8: View File Menu

2.2.4 Implementation

For the Graphical User Interface (GUI) the tool `JBuilder` from Inprise was used. The development of new Java applications can be done quickly and easily as it provides a fully modular environment that delivers integrated visual design, editing, compilation and debugging capabilities. The user interface (UI) is the collection of visual windows, menus, and controls user interacts. All components that were necessary for the application to provide the functionality were implemented and arranged in the application window. The code skeleton for setting up the components into this application window is generated automatically. After finishing the design all the logic has been implemented in order to provide the user access to the necessary executions.

All calling methods are activated by choosing the defined radio button (left hand side) and pressing the button (right hand side) accordingly (see Figure 7). Therefore, an action listener was implemented. Every accessible function is explained below.

<u>Radio button *List all* pushed:</u>

The code fragment below was written to call the method `getAllAttributes()`, in order to get all data rows stored in the database.

```
if (jRadioButton1.isSelected())
{
     // Method getAllAttributes()
     rvalues = server.getAllAttributes();
     for (int i=0; i<rvalues.count-1; i++)
     {
          // Filling table with values
          jTable1.setValueAt(((String)rvalues.surname.get
          (i)), i, 0);
          ...
     }
}
```

The method `getAllAttributes()` is called first, then the returned values from this method are displayed in the spreadsheet of the application, achieved by the `jTable1.setValueAt()` statement.

Radio button *Search for* pushed:

Method `getSearchResult()` returns the values selected by the `SELECT` statement and those get then displayed in the table of the application.

Radio button *View/Play File* pushed:

Within this method the differentiation between viewing a text file or html file and playing an mp3 file is achieved. Both require the selected file to be downloaded from the database in order to store it in the working directory.

For viewing a file the following code was implemented:

```
Frame f = new ViewFile(jTextField3.getText());
f.show();
```

An object f from class `ViewFile` is created and the file is displayed in this new created window.

For playing a file the following code was implemented:

```
Runtime runtime = Runtime.getRuntime();
Process process = null;
try
{
     process = runtime.exec("c:\\Programme\\" +
     "MPFree\\MPFree.exe c:\\ablage\\methodsgui\\" +
     jTextField3.getText());
     ...
}
```

The class `Runtime` was implemented in order to be able to start the mp3 player with the file which was downloaded from the database. This is done with the `runtime.exec()` command.

The return value is a process which was defined a line above the statement.

Radio button *Download file* pushed:

A file gets downloaded from the database and stored on the hard disk with this method:

```
buf = server.downloadFile(value, jTextField3.getText());
FileOutputStream ostream = new
FileOutputStream(jTextField3.getText());
BufferedInputStream bis = new BufferedInputStream(new
ByteArrayInputStream(buf));
int c = bis.read();
while (c != -1)
{
     ostream.write(c);
     c = bis.read();
}
ostream.write(c);
ostream.close();
...
```

An object from type `BufferedInputStream` is created to read the binary stream selected by the database. The object `FileOutputStream` is created to store each single byte read (done by `read(buf)`) and put it into a file, which is done by the `write(c)` statement.

The return value of the `downloadFile()` method is a byte array. Each character of this byte array gets read and serialised into a file stream and then stored as a file.

2.3 Application Server

2.3.1 Introduction to Remote Method Invocation

Remote Method Invocation (RMI) is used to create Java applications that can communicate with other Java applications over a network using the TCP/IP protocol. To be more specific, RMI allows an application to call methods and access variables inside another application, which may be running in different Java environments or different systems altogether. In addition, it has the capacity to pass objects back and forth over a network connection. RMI is a more sophisticated mechanism for communicating between distributed Java objects than a simple socket connection would be, due to the mechanisms and protocols, by which communication between objects is made possible, being defined and standardised. RMI can be used to contact another Java program without prior knowledge of its protocols or instructions of use.

2.3.1.1 Implementation

To create an application that uses RMI, the classes and interfaces defined by the `java.rmi` packages are used, these include the following:

• `java.activation`	For remote object activation.
• `java.rmi.server`	For server-side classes.
• `java.rmi.registry`	Contains the classes for locating and registering RMI servers on a local system.

The `java.rmi` package itself contains the general RMI interfaces, classes, and exceptions. To implement an RMI-based client-server application, first an interface that contains all the methods the remote object will support, should be defined. The methods in that interface must all include a throws `RemoteException` statement, which will handle potential network problems; this may prevent the client and server from communicating.

The next step is to implement the remote interface in a server-side application, which usually extends the `UnicastRemoteObject` class. Inside that class, the methods in the remote interface are implemented, and it is also possible to create and install a security manager for that server (to prevent random clients from connecting and making unauthorised method calls). It is possible, to configure the security manager to allow or disallow various operations. In the server application the remote application is also “registered”, which binds it to a host and port.

On the client side, a simple application is implemented that uses the remote interface and calls methods in that interface. A naming class `java.rmi` allows the client to transparently connect to the server.

With the code written, it can be compiled by using the standard Java compiler but there is one other step: The `rmic` program generates the stub and skeleton layers so that RMI can actually work between the two sides of the process.

Finally, the `rmiregistry` program is used to connect the server application to the network itself and bind it to a port, so that remote connections can be made.

Furthermore, the following methods to achieve the requirements are implemented:

- `getAllAttributes()`
- `getSearchResult()`
- `downloadFile()`

Since every method interact with the database the building up of the connection is explained only once as it is the same for all three methods.

1. The first thing to be completed is the loading of the SQL JDBC driver: `Class.forName("COM.ibm.db2.jdbc.app.DB2Driver");`
2. After that, the connection to the database is established with the following statement:
   `Connection conn =`

```
DriverManager.getConnection("jdbc:db2:test","l","p");
```

3. The next step is to create a statement with the following command:

```
Statement stat = conn.createStatement();
```

Now each method is described in more detail.

Method *getAllAttributes()*:

```
public ResultValues getAllAttributes() throws
SQLException, RemoteException
```

returns all stored parameters from the three tables (`STUDENTS`, `FILES` and `COURSES`) and displays them in the spreadsheet. Therefore, the following SQL command was implemented:

```
ResultSet result = stat.executeQuery("SELECT s.Surname,
s.Name, s.E_Mail, c.Subject, f.Name_File1, f.Name_File2,
f.Name_File3 FROM STUDENTS s, FILES f, COURSES c WHERE
s.ID = f.ID AND f.COURSE_ID = c.COURSE_ID AND s.ID > 0
AND s.ID < 201");
```

The return values get stored in the `ResultSet` and a cursor moves from one record to the next record in the database within a loop.

```
rresults = new ResultValues();
while(result.next())
{
     rresults.surname.add(new
     String(result.getString("Surname")));
     ...
}
result.close();
```

All selected values get stored in a string array in order to return this string array to the client as a return value.

Method *getSearchResult()*:

```
public ResultValues getSearchResult(int value, String
surname, String name, String file, String searchstring)
throws SQLException, RemoteException
```

returns all parameters searched by surname or name or file or a specified search string within the text file or html file.

The SQL command is therefore different for each searching parameter. The following command is for the search after a defined surname:

```
ResultSet result=stat.executeQuery("SELECT s.Surname,
s.Name, s.E_Mail, c.Subject, f.Name_File1, f.Name_File2,
f.Name_File3 FROM STUDENTS s, FILES f, COURSES c WHERE
s.ID = f.ID AND f.COURSE_ID = c.COURSE_ID AND s.ID > 0
AND s.ID < 201 AND s.SURNAME = '"+surname+"' ");
```

The following code is similar to the code method `getAllAttributes()`. The return values get stored into a string array and transfered to the client.

Method *downloadFile()*:

```
public byte[] downloadFile(int value, String file) throws
SQLException, RemoteException
```

returns a byte array according to the selected file given in order to search in the database. The following SQL statement was used:

```
ResultSet result=stat.executeQuery("SELECT Cont_File1
FROM FILES WHERE Name_File1 = '"+file+"' ");
// Moves to the next record
while(result.next())
{
    FileOutputStream ostream = new
    FileOutputStream("output.txt");
    BufferedInputStream bis = new
    BufferedInputStream(result.getBinaryStream(
    "CONT_FILE1"));
    int count = bis.available();
    buf = new byte[count];
    int c= bis.read(buf);
    while(c!=-1)
    {
        ostream.write(c);
        c=bis.read(buf);
    }
    ostream.write(c);
    ostream.close();
}
result.close();
```

An object from type `BufferedInputStream` is created to read the binary stream selected by the database. The object `FileOutputStream` is created to store each single byte read (done by `read(buf)`) and put it into a file, which is done by the `write(c)` statement.

Before the implementation took place each method was written as a prototype in order to test the functionality of accessing the DB2 database.

2.4 Database Server

2.4.1 Database Design

A database is an organised collection of related objects. In order to be able to design and implement a database, it is important to understand the roles of the different objects and how they relate to each other.

There are three tables for students, files and courses data. The `STUDENTS` table contains the last and first names and the e-mail addresses of each student. Each student can store 3 files for each course module taken from Table `FILES`. The course modules are listed in the table `COURSES`.

The relationship between the tables is shown in Figure 9. The `STUDENTS` table has a one to zero or more relationship with the table `FILES` and the table `FILES` has a one to one relationship with the table `COURSES`, which means that there is one student taking more than one module and according to the modules taken, the files can be stored.

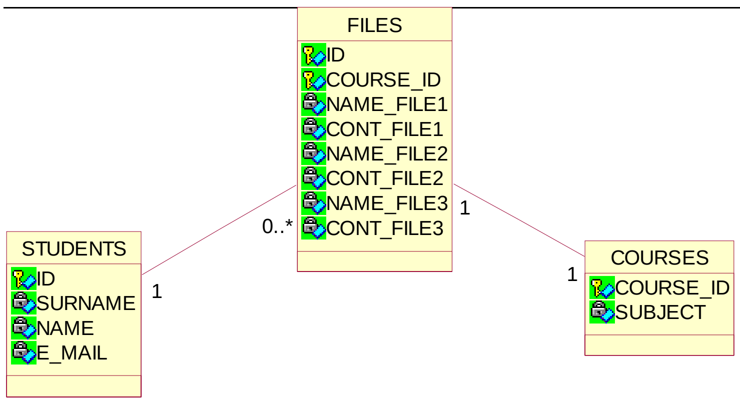


Figure 9: Relationship

2.4.2 Table Descriptions

The following tables describe the tables in detail relating to the data types.

Column name	*Data type*	*Description*
ID	integer	Primary key to connect to table FILES
SURNAME	varchar(20)	Students surname
NAME	varchar(20)	Students name
E_MAIL	varchar(60)	Students e-mail address

Table 1: STUDENTS

Column name	*Data type*	*Description*
ID	integer	Primary key to connect to table STUDENTS
COURSE_ID	integer	Primary key to connect to table COURSES
NAME_FILE1	varchar(40)	Name of File1
CONT_FILE1	clob(1M)	Content of File1
NAME_FILE2	varchar(40)	Name of File2
CONT_FILE2	clob(1M)	Content of File2
NAME_FILE3	varchar(40)	Name of File3
CONT_FILE3	clob(10M)	Content of File3

Table 2: `FILES`

Column name	*Data type*	*Description*
COURSE_ID	integer	Primary key to connect to table FILES
SUBJECT	varchar(50)	Subject of the course

Table 3: `COURSES`

The used data type for storing files is a so called `CLOB` data type. It is a data type for a character large object string. The length of the character string gets defined e.g. by the following declaration: `clob(1M)`

The length is now defined by 1 MB for the column.

2.4.3 Data for the database

The three tables were filled by using three Java programs to store data into the database. The programs are provided on the CD in Appendix B at the end of this document. To explain the implementation, the written methods are described in the following sub-chapters.

Because the communication with the database is always the same it is only explained once. The following implementation steps are necessary:

1. First the SQL JDBC driver should be loaded.

   ```
   Class.forName("COM.ibm.db2.jdbc.app.DB2Driver");
   ```

2. Then a connection is established by calling the statement:

   ```
   Connection conn =
   DriverManager.getConnection("jdbc:db2:test", "login",
   "password");
   ```

3. To be able to set a SQL statement the `createStatement()`method must be called.

4. Every SQL statement can be executed, e.g. for creating a new table.

   ```
   stat.execute("CREATE TABLE STUDENTS(ID integer,
   SURNAME varchar(20), NAME varchar(20), E_MAIL
   varchar(60))");
   ```

5. After that the statement can be closed by calling the

   `stat.close()`method.

2.4.3.1 Data for table STUDENTS

First of all, the table `STUDENTS` has to be created with the following SQL command:

```
stat.execute("CREATE TABLE STUDENTS(ID integer,
SURNAME varchar(20), NAME varchar(20), E_MAIL
varchar(60))");
```

To store data into the table, the following SQL statement fragment was used:

```
stmt.executeUpdate("INSERT INTO students VALUES
("+number+" ,'Abdalla', 'Mohammad',
'mohammad.abdalla@stud.uni-regensburg.de')"); number++;
```

2.4.3.2 Data for table FILES

To be able to store data into the database a table has to be created. This is done with:

```
stat.execute("CREATE TABLE FILES(ID integer, Course_ID
integer, Name_File1 varchar(40), Cont_File1 clob(1M),
Name_File2 varchar(40), Cont_File2 clob(1M), Name_File3
varchar(40), Cont_File3 clob(10M))");
```

Then to find files on a specified hard drive the following method is implemented. It uses this method to get all entries in a directory of a specified file type:

```
private void list_directory(String directory) throws
IOException
{
     File dir = new File(directory);
     ...
     try
     {
          if (!dir.isDirectory()) throw new
          IllegalArgumentException("FindFiles: no such
          directory");
          cwd = dir;
          entries = cwd.list(filter);
          for(int i = 0; i < entries.length; i++)
          {
               FileList.add(new String(entries[i]));
               count1++;
          }
     }
     ...
}
```

After finding all the files with the defined extension, they can be stored into the database with:

```
...
FileInputStream filestream = new
FileInputStream(directory + "\\" +
FileList.get(c-1).toString());
number = filestream.available();
PreparedStatement stmt = conn.prepareStatement("INSERT
INTO files (ID, Course_ID, Name_File1, Cont_File1) VALUES
("+c+", "+var+" ,?, ?)");
col = 1;
stmt.setString(col, FileList.get(c-1).toString());
col = 2;
```

```
stmt.setBinaryStream(col, filestream, number-1);
numUpdated = stmt.executeUpdate();
...
```

Used was the prepared statement, which means that the SQL command `INSERT` can be done dynamically. The two question marks in the `PreparedStatement` are set with the `stmt.setString(...)` and the `stmt.setBinaryStream(...)` command. The statement with the command `stmt.executeUpdate()` is then executed.

2.4.3.3 Data for table COURSES

This is quite similar to the implementation of the students. The table `COURSES` has to be created with the following SQL command:

```
stat.executeUpdate("CREATE TABLE COURSES(Course_ID
integer, Subject varchar(50))");
```

To store data into the table, the following SQL statement was used:

```
stat.executeUpdate("INSERT INTO COURSES VALUES (1,
'Computer Networks')");
```

2.4.3.4 Technical Data

<u>Table `STUDENTS`:</u>

Amount of data rows: 400

Student data were taken from the university of Regensburg (Germany).

<u>Table `FILES`:</u>

Amount of data rows: 400

Kind of Files: HTML-, Text- and MP3-Files

Capacity of those files = 3,5 GB

<u>Table `COURSES`:</u>

Amount of data rows: 9

2.5 Testing

2.5.1 Introduction to Testing

Tests focus on each module individually, ensuring that it functions properly as a unit. Unit testing makes heavy use of white box testing techniques, exercising specific paths in a module's control structure to ensure complete coverage and maximum error detection.

Modules must be assembled or integrated to form the complete software package. Integration testing addresses the issues associated with the dual problems of verification and program construction. Black box test case design techniques are the most prevalent during integration.

After software has been integrated, a set of high order tests are conducted. Validation testing provides the final assurance that the software meets all functional and performance requirements. Black box testing techniques are used exclusively during validation.

Software, once validated, must be combined with other system elements (e.g. hardware, people, databases). System testing verifies that all elements mesh properly and that overall system function/performance is achieved.

2.5.1.1 Test Tool JUnit

`JUnit` is a tool for testing software. It includes a framework for writing test cases appropriate for the software.

Before running the test cases two steps are necessary:

1. Definition of how to run an individual test case
2. Definition of how to run a test suite

`JUnit` supports two ways of running single tests – static and dynamic.

Database methods were tested using the static approach, which results in the `runTest()` method (inherited from class `TestCase`).

The following implementation steps are necessary:

1. Create an instance of `TestCase`
2. Override the method `runTest()`
3. To check a value, call `assert()` and pass a boolean value that is true if the test succeeds

2.5.1.2 Implementation steps for testing the database accesses

To create an instance of `TestCase` the test class `DatabaseTestCase` is extended from `TestCase`. After that a test suite method is written, defining and binding the method into an executable test suite.

Each test method calls the method which should be tested and checks the values from the database with the defined values by using the `assert` statement.

The three test methods for testing the database accesses are:

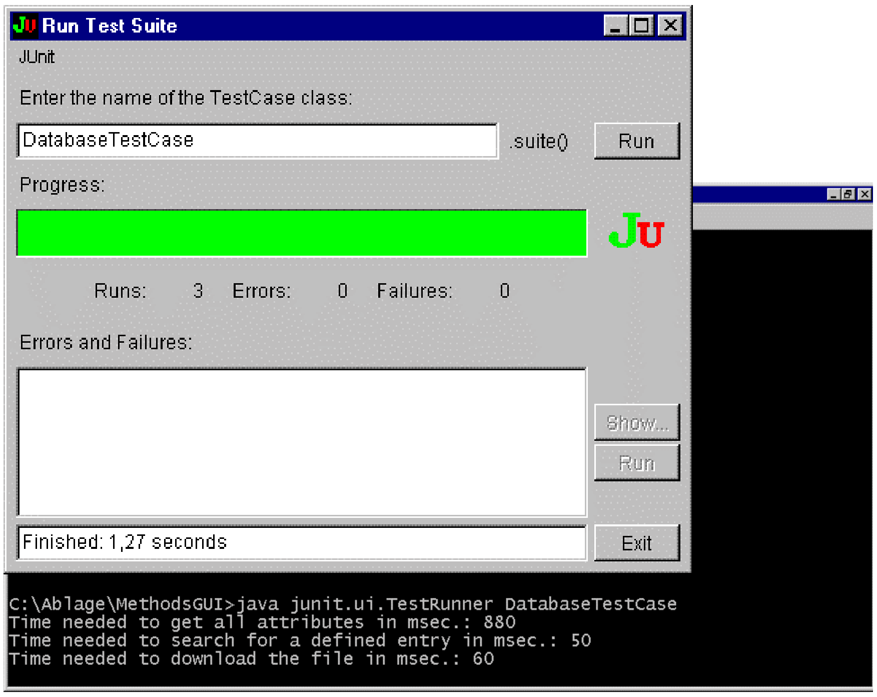
- `testgetAllAttributes()`
- `testgetSearchResult()`
- `testdownloadFile()`

Here is the code fragment of the test method `testdownloadFile()`:

```
public void testdownloadFile()
{
     try
     {
```

```
			RmiIntInterface server =
			(RmiIntInterface)Naming.lookup("rmi://host/D");
			java.util.Date d1 = new Date();
			buf = server.downloadFile(1, "IP.txt");
			java.util.Date d2 = new Date();
			System.out.println("Time needed to download the
			file in msec.: " + Integer.toString
			((int)(d2.getTime()-d1.getTime())));
		}
		catch (Exception ex)
		{
			assert (ex.getMessage(), false);
		}
}
```

Then the following window appears:



Run Test Suite

JUnit

Enter the name of the TestCase class:

DatabaseTestCase .suite() Run

Progress:

Runs: 3 Errors: 0 Failures: 0

Errors and Failures:

Finished: 1,27 seconds

```
C:\Ablage\MethodsGUI>java junit.ui.TestRunner DatabaseTestCase
Time needed to get all attributes in msec.: 880
Time needed to search for a defined entry in msec.: 50
Time needed to download the file in msec.: 60
```

Figure 10: Test Case `DatabaseTestCase`

Figure 10 shows the `TestRunner` application, which is included in the `JUnit` package. The first text field contains the `TestCase` which is `DatabaseTestCase`. The progress bar visualises the errors, if any errors are identified the bar changes from its neutral green to an alerting red. In addition, runs, errors and failures are listed in the text area. The last text field contains the time required for each execution. Furthermore, the time for loading the values from the database was measured and is displayed in the console.

2.5.2 Tests carried out

To validate a system against its specifications it is necessary for test programs to be planned and executed very carefully in order to prevent a system crash. The following final tests were carried out:

- GUI itself

  The following points were tested:

  1. Does the program respond to a new event (e.g. button pressed “List all”)?
  2. How does the software react to faulty user input?
  3. Are possible error messages and warnings expressive and understandable?
  4. Does the software response to events reliable and repeatable?

- Database itself

  Tested with `JUnit` (see previous two chapters)

- RMI connection between GUI and database

  This was tested whether the connection was correctly built up.

On the completion of these tests and assessments an objective decision pertaining to the systems reliability and validity can be made.

3 Installation and Configuration of the DB2 database

3.1 Description of the DB2 Universal Database

The DB2 Universal Database from IBM is used as a partitioned database. This database is a scalable, multimedia web-enabled database, running on platforms ranging from Intel to Unix, from uniprocessor to Symmetric Multi-Processing (SMP) and Massively Parallel Processing (MPP) environments. NetData, Java and Java Database Connectivity (JDBC) support enables secure web connectivity to facilitate real-time electronic business. The DB2 Universal Database can handle tasks as diverse as distributed multimedia data warehousing, powerful complex querying, MPP scalability and transaction processing. Included as a part of DB2 Universal Database are database tools, world wide web connectivity and multimedia object-relational support. It also includes a rich function set, integrated features, ease of use, quality and reliability of the DB2 Universal Database.

3.2 Partitioned Database

A database is simply a collection of data. A database manager is the software that allows users to store and access data in a database. It achieves this function by using system resources, including CPU, memory, disk, and communications. In a partitioned database system, a single database manager and the collection of data and system resources that it manages are referred to collectively as a database partition server (node). A partitioned database system is the collection of all the database partition servers that are created to handle data requests.

In a partitioned database system, multiple database partition servers can be assigned to a machine (or to multiple machines), and the database manager at each machine is responsible for a portion of a database's total data (each database partition server houses a portion of the entire database). This portion of the database is known as a database partition (node). The fact that databases are partitioned across database partition servers is transparent to users and applications.

A partitioned database system can maintain very large databases and open opportunities for new applications. DB2 provides fast response time for both decision-support (DSS) and online transaction processing (OLTP) applications. The database can be configured to execute on a shared-nothing hardware architecture, in which machines do not compete for resources. Each machine has exclusive access to its own disks and memory, and the database partition servers that run on the machines communicate with each other through the use of messages. For a database system to exploit shared-nothing architecture, typically one database partition server is assigned to each machine. Another possible configuration is running multiple logical nodes, in which more than one database partition server runs on a machine.

3.3 Nodegroups and Data Partitioning

Named subsets of one or more database partitions in a database can be defined. Each subset defined is known as a nodegroup. Each subset that contains more than one database partition is known as a multi-partition nodegroup. Multi-partition nodegroups can only be defined within database partitions that belong to the same database.

Figure 11 shows an example of a database in which there are three nodegroups. Nodegroup 1 is a multi-partition nodegroup made of four database partitions, and nodegroups 2 and 3 are both single-partition nodegroups.

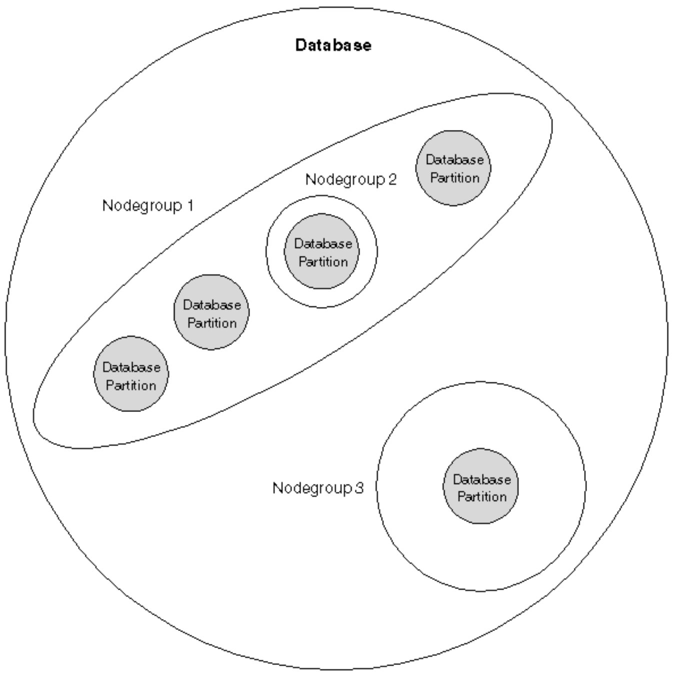


Figure 11: Nodegroups in a Database

As the database increases in size it is possible to add database partition servers to the database system for improved performance. This is known as scaling the database system. By adding a database partition server, a database partition is created for each database that already exists in the database system. After that the new database partition is added to an existing nodegroup that belongs to that database. Finally, the data is redistributed in that nodegroup to utilise the new database partition.

The use of nodegroups means that:

- Data can be distributed across multiple database partitions to reduce I/O and processing bottlenecks
- Data can be redistributed when large volumes of system activity or an increase in table size require the addition of more machines.

3.4 Instances

A database manager instance is a logical database manager environment that is an image of the actual database manager environment. There can be several instances of a database manager on the same workstation. An instance has its own databases and instance directory. The instance directory contains the database manager configuration file, system database directories, node directories, and the node configuration file. In DB2, an instance contains all the database partition servers (nodes) that were defined to take part in a given partitioned database system. The instance-owning machine (known as node 0) owns the shared directory where this information is stored. Other database partition servers that are added to an instance are said to be participating in the instance.

Each instance has different security from other instances on the same machine. This is shown in Figure 12, which shows two separate instances. Instance 1 contains six database partition servers and Instance 2 contains eight database partition servers. (Multiple database partition servers are indicated when more than one line is shown between a database partition server and the instance directory.) The two instances appear to overlap, but this is due to the assignment of two database partition servers to each of the three machines in the middle of the figure.

Database partition servers only belong to one instance. For example, Instance 1 does not contain database partition servers that belong to Instance 2.

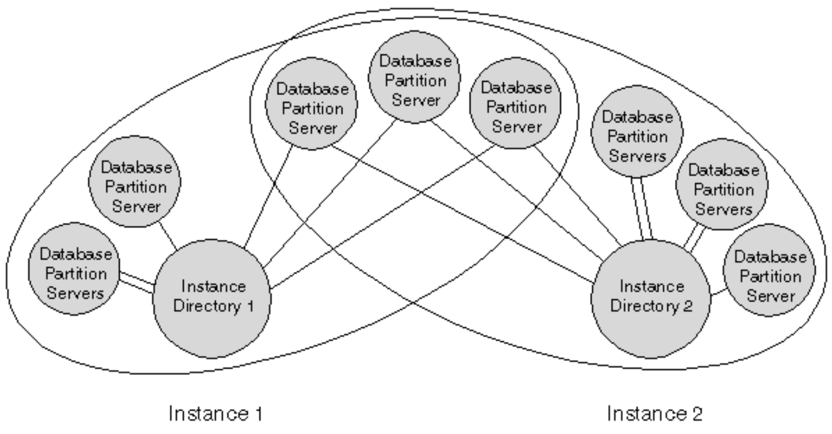

Database Partition Server
Database Partition Server
Database Partition Server
Database Partition Servers
Database Partition Servers
Database Partition Server
Database Partition Servers
Instance Directory 1
Instance Directory 2
Database Partition Server
Instance 1
Instance 2

Figure 12: Two Instances

3.5 Fast Communication Manager

The fast communication manager (FCM) provides communication support for DB2. Each database partition server has one FCM thread to provide communications between database partition servers to handle agent requests, and to deliver message buffers. The FCM thread is started after starting the instance.

If communications fail between database partition servers or if they re-establish communications, the FCM thread updates information and causes the appropriate action (such as the rollback of an affected transaction) to be performed.

3.6 Installation of the DB2 Database

The installation of the database is done by using the wizard. One of the software requirements of the database installation is to have Windows NT Server installed on all partition servers.

In the case of this dissertation two server machines were used as the partitioned database server.

1. The first machine was installed as the instance-owning database partition server (node0) and the second machine was installed as a new node (node1) on an existing partitioned database system.
2. The Fast Communications Manager (FCM) needs to set a port range in order to be able to handle the communication between partition servers.

The installation program has:

- Created DB2 program groups and items
- Registered a security service
- Reserved the port specified for the DB2 Performance Monitor in the `services` file with the service name `db2ccmsv`
- Reserved the port range specified for FCM in the `service` file with the service names `DB2_DB2MPP` for the first port number and `DB2_DB2MPP_END` for the last port number
- Updated the Windows registry
- Created a default instance named `DB2MPP` and added it as a service. The instance directory is located in the `\sqllib` directory. During the installation, the DB2 setup program gives the instance directory the shared name `DB2-instance_name`. Read and write permissions are automatically granted to everyone in the domain. After completing the

installation, the permissions to restrict access to the directory can be changed.

- Created the DB2 Administration Server

3.7 Configuration of the DB2 Database

Before data could be stored into the database the following steps were essential. As it was necessary for the comparison of the performance, one time having the data partitioned on two servers and the other time having the data stored just on one machine.

1. Create a new nodegroup which defines node 0 and node 1 as a nodegroup (or for the single version: create a new nodegroup just for node 0).
2. Create a table space within this created nodegroup.
3. Create the tables in that table space.

After that, the tables could be filled according chapter 2.4.3.

4 Performance Measurements

4.1 Introduction into Processing of a Partitioned Database

A database is distributed across multiple machines, and database partition servers are installed on a set of machines. Because the database is partitioned across multiple machines, multiple CPUs across multiple machines are used to satisfy requests for information. The retrieval and update requests are decomposed automatically into sub-requests and executed in parallel on the database partition servers on each machine.

As an illustration of the power of processing in a partitioned database system, assuming 100000000 records to scan in a single-partition database. This scan would require that a single database manager search 100000000 records. Supposing that these records are spread evenly over 20 database partition servers; each database manager only has to scan 5000000 records. If each database partition server scans at the same time and with the same speed, the time required to do the scan should be approximately 5% of that of a single-partition system handling this task.

User interaction is handled through one of the database partition servers (node). This database partition server is known as the coordinator node for the partitioned database system. The instance owning machine, by default, is the coordinator node for a partitioned database system. Any database partition server can act as a coordinator node. The database partition server that a client or application connects to becomes the coordinator node. Spreading out users across database partitions servers to distribute the coordinator function should be considered.

DB2 keeps communication overhead as low as possible. For example, if a row is being added to a table, the database partition server checks a partitioning map, which specifies the database partition server where the row is stored. The row is only sent to that database partition server, with the result that only the interested database partition servers take part in the insert.

4.2 Defining DB2 Performance

There are five factors which influences DB2's performance:

- Workload
- Throughput
- Resources
- Optimisation
- Contention

The **workload** that is requested of DB2 defines the demand. It is a combination of online transactions, batch jobs, and system commands directed through the system at any given time. Workload can fluctuate drastically from day to day, hour to hour, and even minute to minute. Sometimes workload can be predicted, but at other times it is unpredictable. The overall workload has a major impact on DB2 performance.

Throughput defines the overall capability of the computer to process data. It is a composite of I/O speed, CPU speed, and the efficiency of the operating system.

The hardware and software tools at the disposal of the system are known as the **resources** of the system. Examples include memory, cache controllers, and microcode.

The fourth defining element of DB2 performance is **optimisation**. All types of systems can be optimised, but DB2 is unique in that optimisation is primarily accomplished internal to DB2.

When the demand (workload) for a particular resource is high, **contention** can result. Contention is the condition in which two or more components of the workload are attempting to use a single resource in a conflicting way. As contention increases, throughput decreases.

DB2 performance can be defined as the optimisation of resource use to increase throughput and minimise contention, enabling the largest possible workload to be processed.

For the following measurements only the throughput was looked at as this is the most interesting point.

4.3 Measurements

4.3.1 Benchmark Tool db2batch

The `db2batch` is a benchmark tool included in the DB2 package. Benchmarking is a normal part of the application development life cycle. It is a team effort involving both application developers and database administrators (DBAs), and should be performed against the application in order to determine and improve performance. Assuming that the application code has been written as efficiently as possible, additional performance gains can be realised from tuning the database and database manager configuration parameters, and even the application parameters to meet the requirements of the application.

There are several different types of benchmarking. A transaction per second benchmark would determine the throughput capabilities of the database manager under certain limited laboratory conditions. An application benchmark would test the same throughput capabilities, but under conditions that are closer to those under which the application will run when it is implemented. Benchmarking for the purpose of tuning configuration parameters is based upon these "real-world" conditions, and involves repeatedly running SQL taken from the application with varying parameter values until the application runs as efficiently as possible.

The benchmarking methods described in this section are oriented towards the configuration parameters.

However, the same basic technique can be used for tuning other factors that affect performance, such as:

- SQL statements
- Indexes
- Table space configuration
- Application code
- Hardware configuration.

Benchmarking is helpful in understanding how the database manager responds under varying conditions.

This benchmarking technique is based on the scientific method. A repeatable environment is created in which the same test, run under the same conditions, will yield comparable results.

Benchmarking can also begin by running the test application in a normal environment. As a performance problem is narrowed down, specialised test cases can be developed to limit the scope of the function that is being tested and observed. The specialised test cases need not emulate an entire application in order to obtain valuable information. Start with simple measurements, and increase the complexity only when warranted.

Characteristics of good benchmarks (or measurements) include:

- Each test is repeatable.
- Each iteration of a test is started in the same system state.
- There are no functions or applications active in the system other than those being measured (unless the scenario includes some amount of other activity going on in the system).
- The hardware and software used for benchmarking matches the production environment.

As with any benchmarking, a scenario must be devised and then executed.

4.3.1.1 Creating a Benchmark Program

There are a variety of factors to consider when designing and implementing a benchmark program. Since the main purpose of the program is to simulate a user application, the overall structure of the program can vary. The entire application can be used as the benchmark and simply introduce a means for timing the SQL statements to be analysed. For large or complex applications, it may be more practical to just include blocks containing the important statements.

To test the performance of specific SQL statements, another approach would be to include these statements alone in the benchmark program along with the necessary `CONNECT`, `PREPARE`, `OPEN`, and other statements and a timing mechanism.

Another factor to consider is the type of benchmark to use. One option is to run a set of SQL statements repeatedly over a time interval. The ratio of the number of statements executed and this time interval would give the throughput for the application. Another option would be to simply determine the time required to execute individual SQL statements.

Regardless of the type of benchmark program, an efficient timing system is necessary to calculate the elapsed time, whether for individual SQL statements or the application as a whole. For simulating applications in which individual SQL statements would be executed in isolation, it may be important to consider times for `CONNECT`, `PREPARE`, and `COMMIT` statements. However, for programs processing many different statements, perhaps only a single `CONNECT` or `COMMIT` is necessary, so focusing on just the execution time for an individual statement may be the priority.

While the elapsed time for each query is an important factor in performance analysis, it may not necessarily reveal bottlenecks. For example, information on CPU usage, locking, and buffer pool I/O could show that the application is I/O bound instead of using the CPU to its full capacity. A benchmark program should allow the user to obtain this kind of data for a more detailed analysis if needed.

Not all applications will need to send the entire set of rows retrieved from a query to some output device. For example, some may use the whole answer set as input for another program (that is, none of the rows are sent to output). Formatting data for screen output usually has high CPU cost and may not reflect user need. In order to provide an accurate simulation, a benchmark program should reflect the row handling of the specific application. If rows do get sent to an output device, inefficient formatting could consume the majority of CPU processing time and misrepresent the actual performance of the SQL statement itself.

The following `Performance.sql` file was written according to the SQL statements used within the File Administration Application:

```
SELECT    s.Surname,    s.Name,    s.E_Mail,    c.Subject,
f.Name_File1, f.Name_File2, f.Name_File3 FROM STUDENTS s,
FILES f, COURSES c WHERE s.ID = f.ID AND f.COURSE_ID =
c.COURSE_ID AND s.ID > 0 AND s.ID < 201;

SELECT    s.Surname,    s.Name,    s.E_Mail,    c.Subject,
f.Name_File1, f.Name_File2, f.Name_File3 FROM STUDENTS s,
FILES f, COURSES c WHERE s.ID = f.ID AND f.COURSE_ID =
c.COURSE_ID AND s.ID > 0 AND s.ID < 201 AND s.SURNAME =
'Henkel';

SELECT   Name_File1   FROM   FILES   WHERE   Name_File1   =
'Jump.mp3';
```

Figure 13: File Performance.sql

The benchmark tool gets called by using the following command:

```
db2batch -d test -f performance.sql
```

4.3.2 DB2 Performance Monitor

DB2 Performance Monitor (DB2 PM) is a tool for analysing, controlling, and tuning the performance of DB2 and DB2 applications. It includes a history facility to view recent events, a wide variety of reports for in-depth analysis, and an `EXPLAIN` feature to analyse and optimise SQL statements.

The statistics windows show the behaviour of a DB2 subsystem as a whole and some of the main statistics are displayed. It shows vital statistics of the DB2 subsystem in order to give an overview as a series of snapshots in graphical format.

The Performance Monitor was called from a client machine and the performance results of the database were monitored during a set of performance measurements. The output is shown in Figure 23.

4.3.3 Measurements Methods

A series of measurements are done in order to compare the performance of transactions between each other. First transaction measurements with the common SQL statements like `SELECT`, `INSERT`, `UPDATE` and `DELETE` are performed, after that transaction measurements which are used within the File Administration are performed.

The measurements with the `db2batch` tool are executed on the instance-owning machine of the two database servers. The measurements within the application are executed on the client machine, so additional time is necessary to transfer the data from the database to the client via the network.

The measurements are done as follows:

- Runs of measurements for the non-partitioned database running on one machine
- Runs of measurements for the partitioned database running on two machines
- Every run has been done 10 times to achieve a better accuracy.

4.3.3.1 SQL Transaction Measurements

All SQL commands were tested here, which are `SELECT`, `INSERT`, `UPDATE` and `DELETE`.

The following transactions were performed:

T1:

```
SELECT s.Surname, s.Name, s.E_Mail, c.Subject,
f.Name_File1, f.Name_File2, f.Name_File3 FROM STUDENTS s,
FILES f, COURSES c WHERE s.ID = f.ID AND f.COURSE_ID =
c.COURSE_ID AND s.ID > 0 AND s.ID < 201
```

T2:

```
INSERT INTO COURSES VALUES (11, 'TESTMODULE');
```

T3:

```
UPDATE COURSES SET SUBJECT = 'NEW MODULE' WHERE
COURSE_ID = 11
```

T4:

```
DELETE FROM COURSES WHERE COURSE_ID = 11
```

Figure 14 provides an example of an output fragment of an output file.

```
Number of rows retrieved is:       200
Number of rows sent to output is:   200

Elapsed Time is:  1.723 seconds
---------------------------------------------
Statement number: 2

SELECT    s.Surname,    s.Name,    s.E_Mail,    c.Subject,
f.Name_File1, f.Name_File2, f.Name_File3 FROM STUDENTS s,
FILES f, COURSES c WHERE s.ID = f.ID AND f.COURSE_ID =
c.COURSE_ID AND s.ID > 0 AND s.ID < 201 AND s.SURNAME =
'Henkel'

SURNAME  NAME  E_MAIL  SUBJECT  NAME_FILE1  NAME_FILE2
NAME_FILE3
-------------------------------------------------------
Henkel    Tobias    tobias.henkel@stud.uni-regensburg.de
Dissertation Java_DB2.txt Applet.html Take A Ride.mp3
...
Number of rows retrieved is:       1
Number of rows sent to output is:   1

Elapsed Time is:           0.330      seconds
---------------------------------------------
Statement number: 3
SELECT   Name_File1   FROM   FILES   WHERE   Name_File1   =
'Jump.mp3'
NAME_FILE1
------------------------------------------

Number of rows retrieved is:        0
Number of rows sent to output is:   0

Elapsed Time is:           0.321      seconds

Summary of Results
==================
Elapsed Agent CPU Rows Rows
Statement # Time(s) Time(s) Fetched Printed
1     1.723     Not Collected     200     200
2     0.330     Not Collected       1       1
3     0.321     Not Collected       0       0

Arith. mean     0.791
Geom.  mean     0.567
```

Figure 14: File Output.txt

The necessary execution time is displayed along with the results of the SQL statements in this output. With the benchmark tool the emphasis lies on the execution times and therefore these parameters were monitored.

The following measurements were recorded:

Transaction	**Execution Time in sec (DB2 on one machine)**	**Execution Time in sec (DB2 on two machines)**
T1	0,019	0,235
T2	0,010	0,010
T3	0,010	0,010
T4	0,010	0,010

Table 4: SQL Transaction Measurements

These measurements were recorded on the instance-owning machine of the partitioned database. It can be seen that the execution time for transaction T1 is much faster on the non-partitioned database than on the partitioned one. In the case of the partitioned database, the instance-owning machine must communicate with the other node in order to execute the command, which in turn takes time. Transactions T2, T3 and T4 take only 10 ms compared to T1. This is because for transaction T1 the printing of the results are included in the measurement as formatting data for screen output has a high CPU cost. Ten milliseconds is the smallest measurable value and therefore the measurements could only be done with this accuracy.

4.3.3.2 Application Transaction Measurements

4.3.3.2.1 Measurements with db2batch tool

The following `SELECT` statements were tested which are those from the application of the File Administration Program.

SELECT 1 - Select of all rows:

```
SELECT s.Surname, s.Name, s.E_Mail, c.Subject,
f.Name_File1, f.Name_File2, f.Name_File3 FROM STUDENTS s,
FILES f, COURSES c WHERE s.ID = f.ID AND f.COURSE_ID =
c.COURSE_ID AND s.ID > 0 AND s.ID < 201;
```

SELECT 2 - Select of a chosen surname:

```
SELECT s.Surname, s.Name, s.E_Mail, c.Subject,
f.Name_File1, f.Name_File2, f.Name_File3 FROM STUDENTS s,
FILES f, COURSES c WHERE s.ID = f.ID AND f.COURSE_ID =
c.COURSE_ID AND s.ID > 0 AND s.ID < 201 AND s.SURNAME =
'Henkel';
```

SELECT 3 - Select of a chosen file:

```
SELECT Name_File1 FROM FILES WHERE Name_File1 =
'Jump.mp3';
```

Transaction	**Execution Time in sec (DB2 on one machine)**	**Execution Time in sec (DB2 on two machines)**
SELECT 1	0,020	0,313
SELECT 2	0,010	0,296
SELECT 3	0,010	0,233

Table 5: Application Measurements with `db2batch`

A comparison between the execution times recorded for the partitioned database and those recorded for the non-partitioned database indicate that it takes around 20 times longer to perform the instructions on the partitioned database due to the inter-communication of both machines.

4.3.3.2.2 Measurements within the File Administration Application

As was mentioned previously, the same `SELECT` statements, were executed and the times were measured within the application:

Transaction	Execution Time in sec (DB2 on one machine)	Execution Time in sec (DB2 on two machines)
SELECT 1	0,011	0,109
SELECT 2	0,035	0,053
SELECT 3	0,003	0,037

Table 6: Measurements within the File Administration Application

The results of the measurements are wide-ranging (tabulated in Appendix C). This is due to the fact that accessing the database is done remotely. The reasons for the different results of the execution times are:

- Different network traffic
- Signal propagation delays
- Different server load

Both measurements are not comparable as the `db2batch` tool has an overhead each time performing a statement and the formatting of the data for displaying it on the screen takes also time. Without that it would have been expected that the measurements done within the application take longer than the measurements with the `db2batch` as the accesses to the database are done remotely and therefore additional time is necessary.

4.3.3.2.3 Search within the CLOB column of the database

In order to measure the execution time of an exhaustive search on the database, the following search statement was performed on the `CLOB` column `CONT_FILE`. This column contains a character large object string.

```
SELECT Name_File3 FROM FILES WHERE Cont_File3 like 'AND';
```

The first measurements are performed with the `db2batch` tool, the second with the application.

Transaction	Execution Time in sec (DB2 on one machine)	Execution Time in sec (DB2 on two machines)
SEARCH	0,010	0,262

Table 7: Measurements using the `db2batch` tool

Transaction	Execution Time in sec (DB2 on one machine)	Execution Time in sec (DB2 on two machines)
SEARCH	0,043	0,313

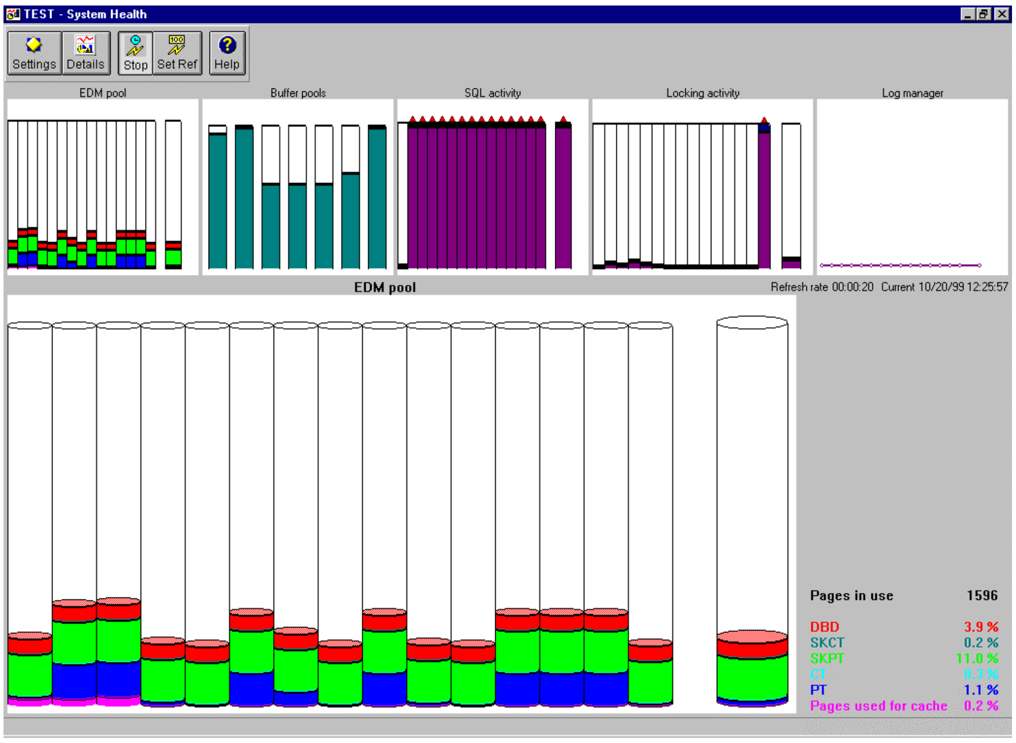
Table 8: Measurements using the application

The values measured with the `db2batch` tool are smaller than the values with the application. They must be smaller because additional time is necessary for the remote access and communication from the client to the server machines.

Unfortunately, the expected result by accessing the partitioned database can not be shown here; the accesses should be around two times faster. The main reason may be the small number of rows, as normally more than only 400 rows are stored in a database.

4.3.3.2.4 Measurements with Performance Monitor

All the measurements were monitored with the Performance Monitor. The result can be seen in the figure below.



Refresh rate 00:00:20 Current 10/20/99 12:25:57

Pages in use	1596
DBD	3.9 %
SKCT	0.2 %
SKPT	11.0 %
CT	0.3 %
PT	1.1 %
Pages used for cache	0.2 %

Figure 15: Performance Monitor monitoring the Measurements

The statistic window shows:

- The state of the EDM Pool
- Buffer pool hit ratios
- SQL Activity
- Locking Activity
- Log Manager Activity.

The upper portion of the window shows thumbnail graphs of each of the counter sets. The lower portion of the window shows one of these graphs in greater detail, with the values shown for the current snapshot in the right-hand portion.

4.3.4 Summary of the Measurements

To summarise the measurements, it can be said that the expected results, the performance improvement could not be achieved. The `db2batch` tool can measure the execution times only with an accuracy of 10 ms. The values of the repeated measurements were in a small range which means that they are quite precise. Whereas the measurements performed with the written Java application give results which are wide-ranging because of the remote communication via a network and the signal propagation delays.

The expected two times faster execution time could not be achieved. This may be explained by the fact that only 400 rows were stored on the database for each table which is small for a typical database as normally a huge amount of rows are stored. As a result, all SQL statements were only performed on those 400 rows, e.g. a `SELECT` can be done faster on 400 rows than e.g. on 4 millions. Although mp3 files which are around 5 MB large were stored on the database, only 4 GB of data could be stored. Another reason is that just one application was accessing the database at one time, so parallel accesses to the database were not performed and no measurements were done to examine the performance in this case.

5 Conclusion

This dissertation described the software development of the File Administration Application, the installation and configuration of the partitioned database and the realised performance measurements.

The performance measurements did not show the time improvement by accessing a partitioned database running on two machines compared by accessing a non-partitioned database only running on one machine. The reasons were the following. Firstly, only 400 rows were stored in the database, which is small for a database. That implies that the execution times for the database statements are too short to be meaningful. Secondly, there was only 4 GB of space available on the hard drive where the database was installed, and consequently no more data could be stored. Thirdly, only one application was accessing the database at one time and therefore no different workload for the database was given.

Achieving maximum performance from DB2 requires a different approach and stricter methodologies than simply getting a DB2 application to perform well. To achieve high performance specific issues of system design, physical object design, tuning methods, and application methods are required. At least 70 % of DB2 performance (and resource consumption) lies within the application design, and SQL coding. The application of all these suggested achievements for getting the maximum performance requires an understanding and knowledge about the management of data of the DB2 database. There are so many tuning parameters that an improvement of one can increase e.g. the system paging rate but the overall performance might degrade rather than improve.

It would have been beneficial to perform another set of measurements with the following improvements. More data is stored in the database in order to perform exhaustive searches, as the contents of databases normally exceeds 400 rows of data. Subsequently, increasing the number of clients accessing the database in order to look at the performance and the expected higher workload level for each further attached client. It would also be interesting to note the results of adding table indexes and tuning table space configurations.

The topic of a partitioned database is an interesting area to highlight the need for a thorough working knowledge of a database and the interactions within the application in order to achieve maximum performance of the overall system.

6 Management of the Project

The milestone plan had to be changed due to a technical redesign of the application, lack of machines, a stolen data server and wrong database software. But the final milestone (M4) is achievable in time. This is shown below.

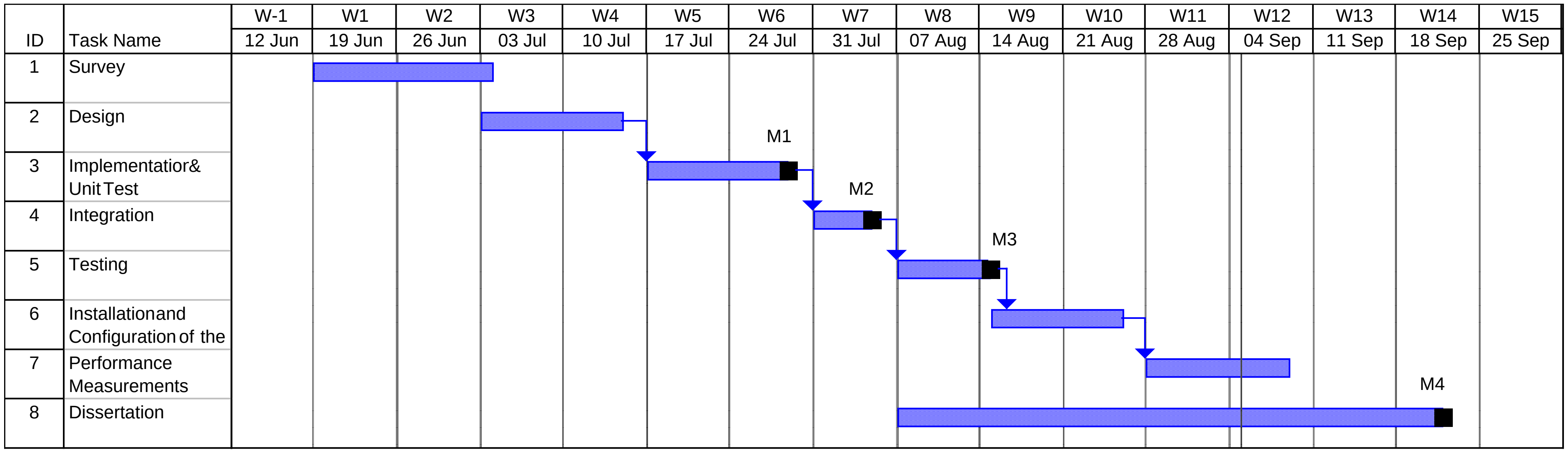


Figure 16: Milestone Plan

The following four deliverables, which are also known as milestones were defined:

M1	Implementation & Unit Test	27/07/2000
M2	Integration	31/07/2000
M3	Testing	14/08/2000
M4	Dissertation Report	22/09/2000

The software development was done quicker than planned, although there was a time loss due to a redesign, which was necessary because of a technical impossibility. The planned CD Administration Application could not be realised, as it was not possible to get a huge amount of data (which were necessary for the performance measurements of the database) of the CD database in California called CDDB. It was possible to register there and to get their Software Developer Kit (SDK). But with this SDK, it was only possible to retrieve data by putting a music CD into the CD-ROM drive. Read access to the database was not possible and therefore the application got redesigned from a CD Administration to a File Administration program.

The task “Installation and Configuration of DB2” was set additionally and it took almost 2 weeks to get the partitioned database running.

References

[1] The handbook of project based management, J. Rodney Turner, McGrawHill

[2] Classical and Object-Oriented Software Engineering, Stephen R. Schach, McGrawHill

[3] Distributed Systems, Coulouris, Dollimore, Kindberg, Addison-Wesley

[4] An Introduction to Database Systems, C. J. Date, Addison-Wesley

[5] Distributed Database Systems, David Bell, Jane Grimson, Addison-Wesley

[6] The DB2 Cluster Certification Guide, IBM

Bibliography

http://www-4.ibm.com/software/data/db2/udb/udb-nt/

ftp://ftp.software.ibm.com/ps/products/db2/info/vr6/htm/index.htm

http://www-4.ibm.com/software/data/dbtools/download-nt4.html

http://www.transarc.com/Library/documentation/websphere/WAS-EE/en_US/html/erzhab/erzhab23.htm

Appendix A

Used Software

The following software was necessary for the development of the dissertation work:

- Rational Rose from Rational for the software design
- JBuilder from Inprise for the GUI programming
- Visual J++ from Microsoft for programming the methods
- JDK1.2.2 from SUN as Java compiler
- JUnit from Beck and Gamma for testing the written software
- DB2 EEE from IBM as the partitioned database
- Performance Monitor from IBM for measuring the database performance

Appendix B

Contents of the CD

The following folders are on this CD:

1. Code:	Application, Data (to fill the database) and Testing
2. Docu:	A few documents written during the dissertation work
3. Measurements:	Folder Test contains all measurements for the partitioned database and folder Test2 contains all measurements for the non-partitioned database
4. Tools:	JUnit and Performance Monitor

Source Code GUI

The application source code is available on the CD in path \\code\application on the last page of this dissertation. To run the Application the following steps are necessary (assuming that the database is set up and data is stored in):

1. Open a first and second console on the machine where the database is stored on.
2. Open a third console on the client machine.

Type the following into the consoles:

First console:	rmiregistry
Second console:	java Server rmi://<hostname>/<application>
Third console:	java GUI

Source Code for storing data into the database

Table STUDENTS:

\\code\data\students

Table FILES:

\\code\data\files

Table COURSES:

\\code\data\courses

CD

Appendix C

Performance Measurements

Partitioned database TEST

All measurements are measured in seconds.

File 1: performanceSQL.sql

Call: db2batch –d test –f e:\performanceSQL.sql > outputSQL.txt

	1	2	3	4	5	6	7	8	9	10	Av.
1	0,230	0,231	0,260	0,231	0,230	0,240	0,230	0,241	0,231	0,230	0,235
2	0,010	0,010	0,010	0,010	0,010	0,010	0,010	0,010	0,010	0,010	0,010
3	0,010	0,010	0,010	0,010	0,010	0,010	0,010	0,011	0,010	0,010	0,010
4	0,010	0,010	0,010	0,010	0,010	0,010	0,010	0,010	0,010	0,010	0,010

File 2: performance.sql

Call: db2batch –d test –f e:\performance.sql > output.txt

	1	2	3	4	5	6	7	8	9	10	Av.
1	0,291	0,310	0,321	0,300	0,301	0,300	0,331	0,320	0,321	0,330	0,313
2	0,291	0,300	0,301	0,291	0,300	0,300	0,300	0,300	0,281	0,291	0,296
3	0,250	0,220	0,271	0,231	0,231	0,241	0,240	0,220	0,220	0,210	0,233

Application:

	1	2	3	4	5	6	7	8	9	10	Av.
1	0,300	0,050	0,220	0,040	0,060	0,060	0,230	0,040	0,051	0,040	0,109
2	0,080	0,050	0,070	0,050	0,050	0,040	0,070	0,040	0,030	0,030	0,053
3	0,050	0,030	0,060	0,030	0,030	0,030	0,050	0,020	0,030	0,040	0,037

File 3: performanceSEARCH.sql

Call: db2batch –d test –f e:\performanceSEARCH.sql > outputsearch.txt

	1	2	3	4	5	6	7	8	9	10	Av.
1	0,270	0,291	0,250	0,280	0,171	0,250	0,301	0,260	0,280	0,271	0,262

Application:

	1	2	3	4	5	6	7	8	9	10	Av.
1	0,941	0,371	0,270	0,280	0,040	0,190	0,350	0,290	0,040	0,361	0,313

Non-partitioned database TEST2

All measurements are measured in seconds.

File 1: performanceSQL.sql

Call: db2batch –d test2 –f e:\performanceSQL.sql > outputSQL.txt

	1	2	3	4	5	6	7	8	9	10	Av.
1	0,020	0,020	0,020	0,020	0,020	0,020	0,020	0,020	0,010	0,020	0,019
2	0,010	0,010	0,010	0,010	0,010	0,010	0,010	0,010	0,010	0,010	0,010
3	0,010	0,010	0,010	0,010	0,010	0,010	0,010	0,010	0,010	0,010	0,010
4	0,010	0,010	0,010	0,010	0,010	0,010	0,010	0,011	0,010	0,010	0,010

File 2: performance.sql

Call: db2batch –d test2 –f e:\performance.sql > output.txt

	1	2	3	4	5	6	7	8	9	10	Av.
1	0,020	0,020	0,020	0,020	0,020	0,020	0,020	0,020	0,020	0,020	0,020
2	0,010	0,010	0,010	0,010	0,010	0,010	0,010	0,010	0,010	0,010	0,010
3	0,010	0,010	0,010	0,010	0,010	0,010	0,010	0,010	0,010	0,010	0,010

Application:

	1	2	3	4	5	6	7	8	9	10	Av.
1	0,020	0,010	0,010	0,010	0,010	0,030	0	0,010	0	0,010	0,011
2	0,020	0,010	0	0,010	0,010	0,010	0	0	0	0,020	0,035
3	0	0	0,010	0	0	0	0	0,010	0,010	0	0,003

File 3: performanceSEARCH.sql

Call: db2batch –d test2 –f e:\performanceSEARCH.sql > outputsearch.txt

	1	2	3	4	5	6	7	8	9	10	Av.
1	0,010	0,010	0,010	0,010	0,010	0,010	0,010	0,010	0,010	0,010	0,010

Application:

	1	2	3	4	5	6	7	8	9	10	Av.
1	0,340	0,010	0,010	0,010	0,010	0,010	0,020	0	0,010	0,010	0,043

NOTE: Performance Monitor must run during the measurements