Analyzing User Behavior in Auboline
with Web Usage Mining

Master Thesis

Louise Due

DEPARTMENT OF COMPUTER SCIENCE
AALBORG UNIVERSITY

TITLE:	Analyzing User Behavior in Auboline with Web Usage Mining - Master Thesis
THEME:	Database Systems
SEMESTER:	INF8
PROJECT TERM:	September 2004 - January 2005

AUTHOR:
Louise Due
louise@cs.aau.dk

SUPERVISOR:
Torben Bach Pedersen
tbp@cs.aau.dk

Abstract

This report describes the design and implementation of the AUBA tool, which is a web usage mining tool for AUB. The goal of the project is to give AUB a tool to analyze the behavior of the users of Auboline and to be able to establish success rates for the different functions in the system.
A post-processor has been implemented to accommodate the ETL process of extracting the data from the log files, transforming it to fit the data warehouse format and finally loading it into the data warehouse. The data warehouse consists of two star schemas that are designed to enable general as well as domain specific analysis of the data. Audit control and other data quality assurance activities have been integrated in the ETL process to make sure that the data is not flawed.
The analyses made with the AUBA tool give the AUB staff information that they have not previously been able to learn about the use of Auboline. Analysis has been made easy with a simple user interface, and query performance has been optimized by using summary tables.
A user survey has been conducted to find out if the results found with the AUBA tool match the users' own opinions of their behavior in Auboline. The results of the two types of analysis are comparable for the most part.

NUMBER PRINTED:	5
NUMBER OF PAGES:	144
FINISHED:	January 14th, 2005

Preface

The purpose of this report is to communicate the process and results of the design and implementation of the AUBA tool, a web usage mining tool for AUB. The project has been carried out within the field of Database Systems and is part of the tenth semester of the Informatics education (INF8) at the Department of Computer Science at Aalborg University.

I would like to thank the staff at AUB for providing data for the project and being helpful with information whenever possible. I would also like to thank my supervisor, Torben Bach Pedersen, for competent and inspiring supervision throughout the project period.

The AUBA tool will be used by both male and female users when it is implemented at AUB, but to ease the reading of the report, the male pronoun will be used for the users of the AUBA tool throughout the report. The female pronoun will be used for the borrowers at the library and users of Auboline.

The source code for the AUBA tool is included in the appendix along with the data definition for the tables in the database.

Louise Due

Contents

CHAPTER 1

Introduction

This report describes the design and implementation of a prototype for the Auboline User Behavior Analysis (AUBA) tool. AUBA is a web usage mining tool [Kosala and Blockeel, 2000] for the staff at The Library of Aalborg University (Aalborg UniversitetsBibliotek, AUB). It aids the staff in analyzing the user behavior in Auboline, which is an on-line system that the borrowers can use to search for books [Auboline, 2004]. The core of the implementation is a data webhouse [Kimball and Merz, 2000] that contains information from the access log files of the web server that Auboline is running on.

Before the implementation of the AUBA tool, a group of computer science students have implemented a book recommendation system that is going to be integrated with Auboline. It will be difficult for the staff at AUB to find out if the book recommendation service is being used in the intended way and if it realizes its goal of helping the borrowers find books that are of interest. Aiding the analysis of the success of the recommendation service was one of the main motivational factors behind the AUBA tool. The AUBA tool helps the staff at AUB find out how Auboline and the recommendation service are used by the borrowers. For instance, they can make an analysis of which periods of the year, semester, month, week or day Auboline is used most or which functions are the most popular in Auboline. Since searching is a central part of Auboline, the AUBA tool can also be used to find out which types of searches are used most and which are most efficient. The efficiency of the different search types can be compared to the efficiency of the recommendation service. The efficiency can be measured by how likely the different search types and the recommendation service are to lead to reservations.

The AUBA tool is developed specifically for Auboline and can not be used to analyze the user behavior on other web sites without being modified. This enables the AUB staff to do more domain specific analysis than they have been able to so far with regular web statistics tools. They can compare the use of the different types of functions and the different types of searches. They can also look at what search constraints have the highest success rate, meaning that they have a better chance of leading to book reservations than other search constraints. The information that the AUB staff can get from the AUBA tool can be used to understand the users better. The better the staff understand the users, the better they can help them find the books that they want and thereby increase the loan numbers. Increasing loan numbers will help to improve the economy of the library.

The data webhouse that the AUBA tool uses consists of two star schemas. The first one is a page event star schema similar to the ones that are often used in data webhouses for general analysis of web site user behavior. Since searches are a central part of Auboline and most of what the users can do in Auboline relates to searches, a search star schema has also been designed. It is used to record information related to the searches that the users make and information that results from searches.

The main part of the implementation is a post-processor that handles the extract-transformation-load processes. The input of the post-processor is a collection of web server access log files that contain all the collected information about user access to the Auboline system. These text files are recognized and transformed to the format of the data warehouse. In this process the post-processor determines the kind of page that is requested for each log line and groups the page events into sessions. It also determines search information and decides which dimension table keys to link to when adding each page event fact to the page event star schema and possibly also the search star schema if the page event is a search.

As mentioned, the AUBA tool has been developed specifically for AUB. Web usage mining systems can also be developed as general systems that can be used to analyze the user behavior on different web sites [Srivastava et al., 2000], but the analysis can be much more precise and useful with a tool that has been designed or adjusted for a specific web site. Since Auboline is a mass produced system that is used by many public libraries in Denmark, the AUBA tool can potentially be used to analyze user behavior on the web sites of other library that have bought the same system. In order for it to be used for web sites that are not based on the same system as Auboline, the AUBA tool would have to be adjusted for the specific structure of these web sites. The group of students that have developed the recommendation service are working on commercializing their system and selling it to libraries. The AUBA tool can be sold with the recommendation service so the buyers have the opportunity of measuring the success of the recommendation service after they integrate it with their book search engine.

The motivational factors behind the present project are described in the next chapter. Chapter 3 presents the specific case in detail, including the current systems of AUB, the source data for the AUBA tool and the ideas and requirements put forth by AUB. Chapter 4 is a presentation of the different concepts that are involved in implementing a web usage mining system, and the schemas of the designed data warehouse are described in detail in Chapter 5. Chapter 6 presents the implementation of the post-processor that is the bridge between the source data in the log files and the data warehouse. It also discusses the performance of the ETL processes. Chapter 7 describes how audit control and error handling has been integrated with the AUBA tool. The graphical user interface for the AUBA tool is presented in Chapter 8, and Chapter 9 describes what has been done to improve query performance. A survey has been conducted to uncover the borrowers' opinions of their own behavior in Auboline. The results of this survey are described in Chapter 10. Chapter 11 discusses the possibility of including data from other databases in the data warehouse. Finally the conclusion sums up the report in Chapter 12 and discusses ideas for future work. The appendices contain data definition and source code for the system and the questionnaire for the survey.

This master thesis is an extension of my ninth semester project [Due, 2004]. The present report documents the process as a whole, and therefore it describes activities that have taken place on both semesters. The parts of the report that contain new contributions are listed below.

- The project goals in Section 3.5 have been extended to include data quality assurance, error handling, graphical user interface and good query performance.
- In Chapter 5 the two star schemas have been extended with an audit dimension and outriggers from the session dimension to the date, time of day and page dimensions. The outriggers are inclued to improve query performance for certain types of queries.
- The implementation of the post-processor described in Chapter 6 has been altered to improve ETL performance. Two processes that were previously performed separately are now combined and this has cut the time to perform an initial or incremental load in half.
- In order to make sure that data is extracted, transformed and loaded into the database from the web log files in the correct way, quality assurance has been integrated in the ETL process. Checks are made to ensure that the data flows through the process as expected and that abnormalities are handled. Furthermore system crashes are handled so the data is not corrupted. This is discussed in Chapter 7.
- A graphical user interface (GUI) has been implemented to allow users to get results with the AUBA tool without having to query the database directly. The results are displayed in graphical charts. The GUI is described in Chapter 8.
- Different steps have been taken to improve query performance. The most important of these is the use of summary tables. Chapter 9 discusses query performance and how the Greedy algorithm [Harinarayan et al., 1996] has been used to decide which views are most beneficial to materialize.
- A user survey has been conducted to illustrate similarities and differences between the results of analysis of Auboline user behavior with the AUBA tool and the users' own opinions of their use of Auboline. The results of this survey along with a discussion of the types of information that can be collected with different methods can be found in Chapter 10.
- Chapter 11 discusses the possibilities of including data from other databases in the data webhouse. Information about books is an obvious opportunity. A design for such an extension of the AUBA tool is also suggested in Chapter 11. This extension has not been implemented.

CHAPTER 2

Motivation

The Library of Aalborg University is interested in improving the services that they offer their borrowers. As part of this service improvement, a group of students at Aalborg University has developed a book recommendation service, that is going to be implemented on the AUB web site. The development of the recommendation service is described in [Ly et al., 2003] and [Ly et al., 2004]. The book recommendation has been accomplished by finding patterns in the library loans data. These patterns are used to recommend other books to borrowers who seek information about a book on the library web site. The goal of the service improvement is to increase the number of loans, which is a deciding factor in the economy of the library.[1]

2.1 Business Versus Library

In many ways a library can be compared to a business that sells products or provides services. The main service that a library provides is making literary material available for the public to use at the library or take home for a period of time. A public library is funded by the government, which pays for the services that the borrowers receive. A widely used method of improving profit of a business is to examine patterns of customer behavior and use this knowledge to adjust the business to try to affect the customers into behaving in profitable ways. The patterns of behavior can be found by use of data warehousing and data mining.[2] These techniques can be used to find patterns and correlations among large amounts of data and thus are well suited for analyzing the behavior of the customers. The methods can be used in the library world as well, since this is another place where it could be valuable to learn more about customer behavior.

One of the differences between the business world and a library is the currency of the customers. The borrowers can be seen as the library customers, though they do not pay for the service directly, but rather indirectly through their taxes. Therefore the number of books that are borrowed by the library borrowers is not limited by how much money they are willing to spend. Rather the limit is set by how much time the borrowers have to use the books and how many books they can carry home. Therefore, taking money out of the business/customer relationship does not mean that the number of loans can be increased limitlessly.

2.2 Available Information

Unfortunately, only a small part of the decision making processes that the borrowers go through with respect to borrowing books are available. Without directly asking the borrowers it is not possible to know whether the books that they borrow are recommended by friends or why they have chosen to borrow exactly these books instead of some of the thousands of other books in the library. It is not possible to read the minds of the borrowers, nor to follow them around the library. Therefore, it is not known whether the borrowers walk directly to the books and borrow them, or they browse around and read the descriptions on the backs of several books before they make a decision on which books to borrow.

The physical side of the loan process is difficult to explore because the only hard data available is the loans data. The book recommendation system has been developed on the notion that even though borrowers are all different, they often follow similar patterns. When borrowers show interest in a book, the system offers them a list of books that other borrowers have borrowed with this book. This way the system gives the borrowers easy access to books that have a good chance of being of interest

[1]The connection between the number of loans and the economy of the library is explained in Section 3.1.
[2]Data warehousing and data mining are discussed in Chapter 4.

to them. This method is a way to attempt to increase loans using only the loans data as background for the analysis.

2.3 Web Site User Behavior

As mentioned, the only hard data that is available from the physical library is the loans data. It is not possible to follow all of the borrowers around the library and look over their shoulders. But if focus is shifted from the physical library to the library web site, it becomes possible to do just that. The library web site can be viewed as a second library that the borrowers can visit from home. In this library, many of the clicks of the users of the web site are recorded one by one in the web server log files. Therefore it is possible to see what actions lead to reservations. This information can help us reason about the decision making processes that the borrowers go through. It is still not possible to get all the information that could be collected by asking the borrowers, but the method allows for collecting information without bothering the borrowers. In the long run this is a cheap and easy way of analyzing the behavior of thousands of borrowers. The results of this analysis should reflect the behavior and intentions of the web site users. This is discussed in Chapter 10 where the results of the analysis are compared to the results of a user survey.

2.4 Recommendation Service

When the book recommendation service is implemented on the AUB web site it will be possible to use web log analysis to distinguish between loans that are the outcome of an on-line recommendation and loans that are a result of a regular search. This would be an effective and precise way to measure the success of the recommendation service. Without web log analysis, this success could only be measured by looking at increases in loans. This would have many possible sources of error since changes in the loans data could be caused by a number of reasons other than the recommendation service.

2.5 Future Possibilities

As is also true for business customers, library borrowers will be more likely to come back for more, if they are satisfied with what they have received. Therefore the goal is not simply to get the borrowers to borrow more books. For the effects to be permanent, the books that the borrowers take home should be the right ones to fit their needs. To have this as a success criteria, more information than the actions on the web site that lead up to a loan is necessary. For example, successful loans could be distinguished from unsuccessful ones by checking if the loans are prolonged or by asking the users to grade the books that they have borrowed the next time they log on to the web site. Since it is not possible to distinguish between different users, user satisfaction analysis is not included in the AUBA tool.

By having ways of distinguishing a successful loan from an unsuccessful one, it would be possible to investigate if there is a difference between the patterns that lead to successful loans and the ones that lead to unsuccessful ones. For example, this would make it possible to find out if there is a higher success rate with books that have been recommended than with books that have been found using key word search. It could also be used to improve the recommendation service, for example by removing recommendations that tend to lead to unsuccessful loans.

2.6 Limitations

Using data warehousing and data mining to analyze log files from the library web server has many potential applications. Whether these uses are possible or not, depends on the information collected in the log files. This information can make a huge difference in whether it is possible to follow a user through a session, whether a user can be recognized from previous sessions and how much can be known about what the user has seen and done on the web site. For example, if the user name is not recorded in the log file it can be difficult to know for sure when the same user has returned to the web site, because the rest of the information in the log file that could potentially be used to recognize users, such as the IP address, is too unreliable to identify users. The users of Auboline can log in if they are borrowers at AUB, but because of privacy concerns the usernames are not visible in the log files. Therefore it is not possible to distinguish between individual users. This is explained in detail in Section 3.3.

CHAPTER 3

The Library of Aalborg University

AUB is a public research library for the region of North Jutland. The principal task of the library is to support research and teaching at Aalborg University by providing relevant documentation and access to quality information resources [AUB, 2004]. The main branch of AUB is located on Langagervej in Aalborg. In addition to this there are three smaller specialized branches at different parts of the university in Aalborg and one in Esbjerg, where part of Aalborg University is located.

3.1 Facts

AUB has 12,222 borrowers, out of which 8,791 are students, 1,766 are employees at Aalborg University and 690 are AUB employees. The library has 669,783 books and journals and access to 10,298 different electronic journals. In 2003, 116,194 reservations were made at AUB and the library had 173,700 loans. 86,941 of the reservations were carried out through the web site.

The economy of a public library depends mainly on government grants. A few years ago there was a simple connection between the number of loans at AUB and the grants, because AUB received 28.50 DKK for each book or magazine loan. Now the grants are calculated on the basis of a number of parameters, so the connection is not as straight forward as earlier. A new budget model is underway because the number of on-line downloads exceeds physical loans and therefore needs to be taken into account when calculating the grants. The economic benefits that AUB would have from increasing the number of physical loans is not as straight forward as it used to be, but according to Anton L. Nielsen, who is an organization consultant at AUB, it is still a fair assumption that an increase in physical loans will lead to an increase in government grants.

Purchasing licenses to on-line material is a very large post in the library budget. AUB spends 2,981,600 DKK a year on on-line materials. Since there is no exact method of measuring the number of downloads of these materials at the moment, AUB can not always prove the number to the government and therefore can not always get money when somebody uses their on-line materials.[1]

3.2 Web Site and Architecture

The web site of AUB can be seen at http://www.aub.aau.dk. From this web site it is possible for users to search the databases of on-line material of the library, search for books, manage loans, etc.

The system that manages the loans at AUB is an off-the-shelf system that has been adjusted to fit the needs of AUB. The system is called Aleph and is developed in Israel [Aleph, 2004]. The data of this system is located in an Oracle database. Auboline is the name that AUB has given the web interface that can be used to access Aleph through Apache. There are several web servers at AUB, but they are not connected to the Aleph server.[2]

The recommendation service described in Chapter 2 uses the data in the Oracle database as input and outputs XML to Auboline so the borrowers are presented with a list of recommended books with the book descriptions in Auboline. Furthermore the recommendation service includes a librarian service that the AUB employees can use to see statistics about loans [Ly et al., 2004].

The AUBA tool will not access the Aleph system directly. It will take the Apache log files as input and will function as a system on its own. It will not be accessed by the borrowers, but only by technical staff at AUB.

The architecture of the current system and the AUBA tool is depicted in Figure 3.1.

[1]Explained by Kasper Løvschall at our meeting on 16 April 2004.
[2]Explained by the webmaster of AUB on 3 April 2004

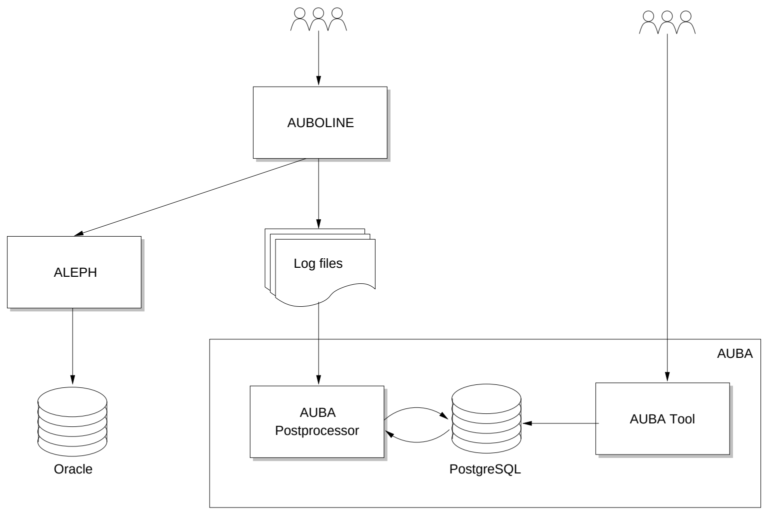


Figure 3.1: The architecture of the current system and the new system.

The borrowers can use Auboline to search for books. They can also log on to the system and get an overview of the books that they have borrowed, see the return date, or prolong the loans.

The Auboline system has its own physical server; a SUN SPARC server with four CPUs running Solaris 8. Therefore the log files of the Auboline system are recorded separately from the access logs of the general AUB web site. Since the Auboline system handles all book searches and other user functions that deal with books on-line, the log files used will be exclusively from the a500 server, which is the Auboline server.

3.3 Log Files

The log files that were received from AUB is a record of all requests to the a500 web server between February 25, 2003 and March 22, 2004. There is a log file for each day in this period of time, and they contain a total of 17.7 million lines. Since only the true requests to the Auboline system are needed for the analysis, the rest of the log lines are ignored by the post-processor (see Chapter 6). The log lines that are ignored typically contain image and style-sheet requests, requests from search bots and web crawlers and requests to administrative systems. After ignoring these kinds of requests there are 3.9 million log lines left to use in the analysis of the behavior of the Auboline users.

The AUB web server runs Apache 1.3.28 [Apache, 2004b]. It uses the Combined Log Format [Apache, 2004a], and has three fields more than the Common Log Format [Apache, 2004a]. The Common Log Format has the seven attributes IP address, RFC 1413 identity, userid, time, request, status code and size. The three extra attributes in the AUB log files are server name, referrer and user agent. The attributes are explained in the following as described in [Apache, 2004a].

IP address: This is the IP address of the remote host that made the request to the server. The IP address is not necessarily the same each time a given borrower accesses the website from the same computer. If the connection is made through a proxy, the IP address in the log file will be the IP address of the proxy instead of the end user computer. Therefore, the same IP address could point to several different users. The IP address could also be dynamically associated with the computer, which means that a user can have different IP addresses at different times. Therefore, an IP address is not enough to identify a computer [Kimball and Merz, 2000]. There are 32,042 distinct IP addresses in the log files which means that there is an average of 122 clicks per IP address. As mentioned this does not mean that 32,042 distinct users have used Auboline. The actual number of users could be either higher or lower depending on how many users use the same proxies and how many users use dynamically associated IP addresses.

RFC 1413 identity (ident): This attribute gives the RFC 1413 identity of the user. The RFC 1413 identity is a way to determine the identity of a user of a particular TCP connection [IETF, 2004]. According to Apache [Apache, 2004a] it is highly unreliable and should never be used for a public server. The attribute needs to be activated to get shown in the log file. It is not activated in the AUB log files and therefore is empty in all the log lines.

Userid (authuser): This is the userid determined by HTTP authentication [Apache, 2004b]. It only has a value if the requested document has password protection. The borrowers can log on to get to the user specific information in Auboline, but Auboline does not use this attribute when users log in so it is empty in all the log lines.

Timestamp: The timestamp gives the date, time, and timezone that make up the time that the server finished processing the request. The timestamp is precise to the nearest second on the clock of the web server, which is accurate enough for the web log analysis to be made in the present project.

Request: The request attribute consists of three sub-attributes. These are the method, the request URL and the protocol. The method describes the method used in the request. The most common methods used are the GET (93.8 %) and POST (6.2 %) methods, which are used to retrieve information from and send information to the server, respectively [W3C, 2004]. Other methods in the log files include the HEAD method that only returns response headers from the document. In these log files it is only used by web crawlers and the HEAD requests are therefore ignored.

The request URL can be used to recognize sessions and in most cases find out which kind of page the user has requested. Therefore it contains a lot of information that is useful to the analysis. The special structure of the request URLs of Auboline is described in Section 3.3.1.

The protocol is the HTTP protocol used for the request. In the log files it has the values HTTP/1.0 and HTTP/1.1.

Status: The status is a three digit integer indication of the success status of the page retrieval. A successful page retrieval has the status number 200. Other numbers indicate what went wrong in the page retrieval. This might be used to explain why a user leaves the web site, for example if the page that she is looking for is not working properly. All the log lines in the log files have status code 200 except for two single log lines. One of these has status 400 (client error - bad request) and the other has status code 500 (internal server error). Therefore this attribute will not be explored further.

Bytes: The number of bytes transferred from the web server to the user's computer. This will not be used in the AUBA tool.

Server name: The name of the server that the user has accessed. Since the log files only record access to the a500 server this attribute has the value "a500.aub.auc.dk" in all log lines. Therefore the value of this attribute is not of value to the project and will not be used.

Referrer: The URL of the site that the client reports having been referred from. This is likely to be a page that links to or includes the requested object. The attribute can be used to find out how a user found Auboline. Since the referrer attribute contains the value of the URL that the user visited before the one in the current log line, it can also be used to piece together the pages of a session.

Browser: This is the identifying information that the browser reports about itself. Sometimes the text in this field indicates that it is not a regular user but a search bot or a web crawler, such as Googlebot. These log lines should be ignored as they are not an indication of user behavior on the web site and will only interfere with the results of the web log analysis. The most common browser is Internet Explorer, which is used in 91 percent of all log lines. There are 2295 different values in the browser field.

The general pattern of the log lines is as follows. This is the pattern that the post-processor, which is described in Chapter 6, recognizes.

```
<ip_address> <ident> <authuser> [<date>:<time> <time_zone>]
"<method> <request_url> <protocol>" <status> <bytes>
<server_name> "<referrer>" "<browser>"
```

Example:

```
193.162.54.8 - - [25/Apr/2003:07:51:03 +0200] "GET /F/VA7GU
1PDMRLYLX5B3E9AVMUD1G9H37TR66Q3451CM2MYIKT1ST-00673
?func=file&file_name=find-b&local_base=AUBOLINE HTTP/1.0"
200 12681 a500.aub.auc.dk "http://a500.aub.auc.dk/F/VA7GU1P
DMRLYLX5B3E9AVMUD1G9H37TR66Q3451CM2MYIKT1ST-00644?func=file
&file_name=base-list" "Mozilla/4.0 (compatible; MSIE 5.01;
Windows NT 5.0; YComp 5.0.0.0; DKDD)"
```

The attributes that will be used in the AUBA tool are IP address, date, time of day, method, request URL, referrer and browser. The information in the rest of the attributes is not relevant to the implementation of AUBA tool.

3.3.1 Understanding Requests

Unfortunately, the AUB staff do not have any documentation on Auboline and do not have access to the source code of the system, so they have not been able to contribute information about the meaning of the contents of the request URLs. Therefore the semantics of the contents of the request URLs used in the implementation of the AUBA tool has been accomplished by experimenting with the web site, and it has not been verified by AUB. Not all of the information in the request URLs is used in the process of examining which pages are requested by the user, since some of the information appears to be specifically for the underlying implementation and not the appearance of the web site. Like most web sites, Auboline is not implemented with web usage mining in mind. Therefore it is not possible to get all the information that could be useful to have in the data webhouse from the request URLs in the log files. For instance, when a user reads a book description it is not possible to see which book she is showing interest in in the request URL.

The request URL is the relative URL requested by the user. The server name is implicit in a relative URL so it only contains a path and maybe a filename or a query. In the log lines from Auboline only image and style-sheet request URLs have filenames. Since these requests are ignored in the analysis the focus will be on the path and the query. The requests to the Auboline system can take the following forms.

/F/
/F/<session tag>/
/F/<session tag>-<serial>
/F/<session tag>-<serial>?<query>
/F/<session tag>?<query>
/F/-/?<query>

The “/F/” part is present every request made to Auboline. Requests to other systems on the same server look differently from this. For instance, there are a few requests to an administration system in the log files. The URLs of these requests start with “/S/” instead of “/F/”. The session tag is a fifty character long random combination of letters and numbers that the Auboline system uses to distinguish sessions from each other. The serial is a number that is unique for each session. The query part of a request URL in the Auboline system is usually very interesting, since it describes a lot of what the user is doing on the web site. For instance, it describes which function he or she has just used and which search terms he or she entered in a search. The log files do not always contain all the information that is needed, but sometimes the missing information can be constructed by putting the information together with other information gathered in the same session.

As described above, the log lines that are used for the analysis use either the GET or the POST method. When the *GET* method is used, the interesting information is to be found in the query part of the request URL. This query contains a number of variables and values that describe the page that is requested by the user. If the *POST* method is used, the request URL has no query part [Srivastava et al., 2000], so the interesting information is to be derived from the referrer part of the log line. Fortunately there are no pages in Auboline with more than one link or button to a POST method, so if a POST method request is executed from the page that a user comes to before the actual reservation, the POST method request can only be a reservation. The POST method is used in Auboline for log-in, reservation and saving a book in the basket. Below is a description of the different variables in the request URLs that are used to establish the page functions that the users access in Auboline.

func: This variable is present in all URL queries in Auboline. It describes the general function used. The func variable is found to have 52 different values in Auboline. Examples of values are “find-b” and “bor-loan” which indicate the page functions “basic search” and “loans list”, respectively.

file_name: The file_name variable is present in the request URL if the func variable has the value “file”. In this case the value of the func variable is not enough to find the page function. The page function is usually some kind of form that the user needs to fill out. For instance, if the file_name variable has the value “find-b” the page is a form where the user can enter data for basic searching. When the user submits the information entered in the basic search form, the func variable has the value “find-b” on the next page, which shows the results of the search.

find_code: The find_code variable is used when the func variable indicates some kind of search such as a basic search. It indicates what field the search is made on. For instance, if a user makes a basic search on title, the find_code variable has the value “WTI”.

scan_code: The scan_code variable is present in the URL query when a user makes an index scan. It is used in the same way as the find_code variable.

action: The URL query sometimes contains different variables that all indicate some kind of action in Auboline. For instance, if the func variable has the value “history-action” the query will also have variables to indicate which history action is executed. The presence of a action_delete.x variable indicates that the action is “delete” which means that the function performed on that page is that a search result has been deleted from the history list. These variables come in pairs in the queries. If there is an action_delete.x variable, there is always an action_delete.y variable as well. These variables have integer values, but the meaning of the values are not known by AUB and it has not been possible to establish the semantics of the values.

3.3.2 Use of Log File Information

It is not legal for the library to keep information about which user is logged on to their web site at a certain time [Datatilsynet, 2005]. Therefore the user information is not kept in the log files but in a separate file that is emptied frequently. Since the library uses neither session cookies nor permanent cookies, it is not possible to establish the identity of a user or a computer that has visited the web sites [Kimball and Merz, 2000].

The only information in the log file that could be used to identify the computer from which the request came from is the IP address. But as explained, an IP address cannot be trusted to always belong to the same computer. A way to get around this problem could be to combine the IP address with the browser attribute in the log file. This would decrease the uncertainty compared to just using the IP address, but still it is not precise. When such a big part of the library borrowers are from the university there are a lot of proxies. There are also many people who use the same browser, since so many people use Internet Explorer, which is included as a part of Windows.

When users cannot be recognized from session to session it is not possible to establish the success of a loan, since there is no information about which users borrowed which books, etc. However, it is still possible to establish the success rates of the different kinds of searches and the recommendation service without being able to recognize users.

The purpose of the AUBA tool is to increase the loan numbers, so the task is to find out what kind of analysis of the web log files could potentially lead to increased book loan numbers. Auboline is the only system on AUB's web site that is directly linked to book loans. The users can not borrow books on the web site, but they can reserve them and it is a reasonable assumption that most reservations lead to loans and that an increase in reservations will lead to an increase in loans. By using the log files to analyze the users' interaction with Auboline, it is possible to find out what makes some users reserve books, while other users leave the web site without having reached their goal. The patterns in the interaction, such as which kinds of searches are used most and which are most efficient, can be found, and AUB can use this knowledge to improve their web services to help users find the books that they are looking for.

3.4 Wishes and Ideas

One of the main wishes of AUB is to improve the services that they offer their borrowers. The book recommendation service is meant to be part of this service improvement, since its objective is to help borrowers find relevant books. To find out if the implementation of the recommendation service is a success, AUB would like a tool to analyze the usage of the recommendation service by the web site visitors. At the same time they would like to be able to analyze the use of the Auboline system and compare the uses of the two systems. The staff at AUB are hoping that they can use the analysis results to further improve their services to the borrowers and thereby increase the loan numbers.

AUB would like the AUBA Tool to be developed on the same development platform and database management system as was used in the implementation of the book recommendation service. The group of students that developed the recommendation service chose to use Java Servlets as the platform and PostgreSQL as the DBMS [Ly et al., 2004]. These choices were made mainly because the tools are free. The same platforms have been used for the development of the AUBA Tool. The Java Servlets version used for the AUBA tool is J2EE version 1.3 [J2EE 1.3, 2004] and PostgreSQL is version 7.4 [PostgreSQL 7.4, 2004]. The book recommendation service was developed on a Windows platform using the Cygwin framework as a bridge to PostgreSQL which does not have a free version for Windows. A free PostgreSQL version for Windows being developed and will be available soon, but the first version will probably not be as stable as the one for Linux [PostgreSQL 7.5, 2004]. Cygwin slows down the transactions, and the system will not be running on a Windows machine when implemented on the AUB web server, so the AUBA tool has been implemented on a Debian Linux machine. PostgreSQL has some limitations compared to big commercial DBMS products such as Oracle. For instance, it does not have materialized views so it is necessary to do more manual programming in order to achieve the same query performance as can be achieved with a commercial DBMS.

Since this version of the AUBA tool will only be used to analyze past behavior, the data does not need to be extracted from the log files and loaded into the data webhouse in real time while the users are active. Instead this will be done nightly when the load on the web server is low. Each night the server, that Auboline is running on, makes a switch to write the access logs to a new file. This process can be combined with the process of extracting the logs from the log file used during the previous 24 hours, transforming the data and loading it into the data webhouse.

3.5 Project Goals

The long term project goals are to make a tool that enables analysis of the Auboline user behavior, including the use of the recommendation service and comparisons between the way that users find books using the standard Auboline features and the recommendation service. Because the recommendation service has not yet been integrated with Auboline, it is not possible to include it in the analysis. Instead a tool for the current version of Auboline that can easily be extended to enable analysis of the recommendation service as well will be implemented.

It is important that the AUB staff can trust the results of the analyses that they make with the AUBA tool. Furthermore, the finished AUBA tool should be easy to use without knowledge of the underlying program and database and response times should be acceptable for interactive use. Loading the data from the log files to the database should not take more than a few minutes each night, and it should not slow down other processes on the web server.

CHAPTER 4

Data Warehousing

In this chapter the concepts and principles behind this project are discussed. The concepts include data warehousing, dimensional modeling, data webhousing and web usage mining.

4.1 Data Warehousing

Businesses often have very large databases where they collect all kinds of internal and external data, such as personnel skills or product sales. All of this data can be used for decision support if it is analyzed properly. It is important to make sure that the data is stored and accessed in a way that ensures usable and correct results from the analyses. Having better access to data enables the business to make better decisions faster. A data warehouse is a very large database where data from the operational databases is specifically structured for query and analysis performance and ease-of-use [Kimball and Ross, 2002a].

Dimensional modeling is a technique that has been used for many years and after the first edition of Ralph Kimball's book "The Data Warehouse Toolkit" [Kimball and Ross, 2002a], it has been broadly accepted as the dominant technique for data warehouse modeling. The goals of dimensional modeling are user understandability and query performance. Therefore the multidimensional data model is good when the objective of the database system is to analyze data [Pedersen and Jensen, 2001]. A dimensional model is very simple, while supporting very good performance on the types of queries most often done on data warehouses [Kimball and Ross, 2002a].

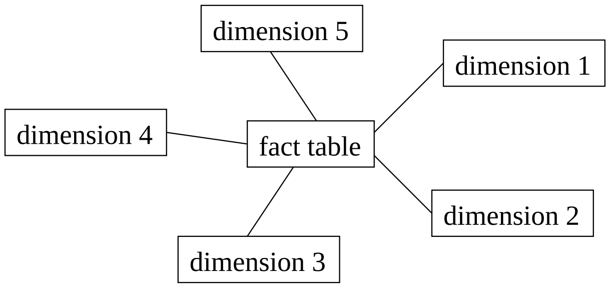


Figure 4.1: Relational representation of a star schema.

A *star schema* is the type of schema commonly used in *dimensional modeling* [Levene and Loizou, 2003]. The star schema has its name because it looks like a star with a fact table in the center and dimension tables around it as depicted in figure 4.1. The *fact table* has one record for each *fact*, such as a product sale. It contains foreign keys to dimension tables along with measures of the facts, for instance the dollar amount of a sale. A fact table generally has several thousands or millions of records and is therefore very large. Additional information, such as different types of textual descriptions of the facts, is kept in *dimension tables*. For example, the textually descriptive information about the date of the sale is kept in a date dimension table. Such a dimension table could contain a lot of information about the date such as week day, holiday and major events. It contains all usable information about the day of the sale.

Each dimension table has a surrogate primary key, a single attribute integer, that is referenced by a foreign key in the fact table. Since the fact table should not contain null references to dimensions, dimension tables can also contain records for cases where the value is not known. Dimension tables are generally very wide with many attributes but have few records compared to the fact table, so they are often much smaller than the fact table.

The *grain* of a fact table is the level of detail associated with the facts. Declaring the grain is an important step in the design of a star schema [Kimball and Ross, 2002a]. It is preferable to have a star schema with the most atomic grain possible. Because the facts cannot be subdivided any further, no information that could be useful at a later point will be lost. If there is a need for a fact table with a coarser granularity another star schema can be designed with this granularity.

Query performance can be further enhanced by use of materialized views to pre-calculate aggregates that are accessed often. This can be done by the developer when he knows what kinds of queries are often used, but it can also be done automatically by the database management system [Zaharioudakis et al., 2000]. Unfortunately, PostgreSQL does not support materialized views, so in this project materialized view have been programmed manually by using PL/pgSQL. This is explained in Chapter 9.

Entity-Relation (ER) modeling is a widely used technique in traditional database design. One of its main goals is to create a normalized database schema to avoid redundancy while still preserving all dependencies among the data. Some of the advantages of this is that updating the data in the database is much faster and the database uses less storage space. For example, a customer address only appears in one record in one table in the database with foreign keys from all other tables that are related to it. When a customer moves to another address, it is very easy to update the address information and there is no risk that the address appears differently in different places in the database.

Even for a very simple system, an ER diagram quickly becomes very complex and hard to understand, and the resulting database schema often has more tables than the number of entities and relations in the diagram. While updating a database designed with the ER model is very fast, querying the database is often very complex and time consuming. That is why ER modeling is not suited for data warehouse design. Data warehouses are very large and query performance is very important, while update performance is of lesser priority. That is why dimensional modeling is a better technique for data warehousing [Kimball, 1997]. Redundancy is accepted to improve query performance and user understandability. Since redundancy usually only appears in the dimensions which commonly take up only one to five percent of the storage space, this has an inconsiderable effect on the overall storage usage [Pedersen and Jensen, 2001]. The dimensional model is much more predictable and easy to understand than the ER model, so it fits the demands of data warehousing.

PostgreSQL is the DBMS that manages the data warehouse used by the AUBA Tool. PostgreSQL does not have a multidimensional engine included. Therefore, the AUBA Tool is a ROLAP system, which uses relational database technology for storing the data [Pedersen and Jensen, 2001].

4.2 Data Webhousing

A data webhouse [Kimball and Merz, 2000] is a special kind of data warehouse. In a data webhouse the main data source is not the operational databases like in a normal data warehouse. Instead, the primary part of the data in a data webhouse is derived from the access log files that record all access to a web server. The information that can be derived from a data webhouse relates to user behavior on the websites that the access logs relate to. To transfer the data from the access logs to the data webhouse a post-processor, that reads the log files and reformats the data to fit the data webhouse schema, has been implemented. Once the data is in the data webhouse it can be queried in the same way as a normal data warehouse. The data webhouse can be combined with data from a regular data warehouse. For instance it is possible to combine customer information with web site user information if the customers can be recognized when they visit the website.

The schemas of a data webhouse should be designed to match the focus of the web usage mining tool to be implemented. Some of the common data webhouse schemas include page event [Kimball and Merz, 2000], sequence [Demiriz, 2002] and sub-session [Andersen et al., 2000] schemas. A page event schema is a general type of schema, while the two latter types focus on the paths that the user takes through the web site. The schemas used in the data webhouse for the AUBA tool are discussed in the following chapter.

CHAPTER 5

Data Warehouse Schemas

In this chapter the design of the data warehouse will be discussed. The data warehouse has two fact tables with dimensions that have been designed to enable the different kinds of analyses described in Chapter 2. The AUBA tool helps the AUB staff analyze the fundamental user behavior in Auboline that is similar to user behavior on other web sites, but it can also assist analysis of user behavior that is specific for Auboline. Listed below are examples of questions that can be answered by use of the AUBA tool. Chapter 8 gives examples of how these questions can be answered with the graphical user interface. The rest of the current chapter will discuss the data warehouse schemas and the thoughts behind the design of the fact and dimension tables.

- Non-domain specific user behavior
  - On which pages do the users start their sessions?
  - On which pages do the users end their sessions?
  - How many pages do the user sessions consist of?
  - How does the user activity vary during the course of the week?
  - How does the user activity vary during the course of the day?
- Domain specific user behavior
  - How does the user activity vary during the course of the semester?
  - How does the frequency of book reservations vary during the course of the semester?
  - What types of page functions are used most frequently?
  - How do users find books with Auboline?
  - How many book descriptions are read after the different kinds of searches?
  - How many books are put in the basket after the different kinds of searches?
  - How many books are reserved after the different kinds of searches?
  - Which kinds of searches are most likely to lead to reservations?

As mentioned in Section 4.1, declaring the grain of the fact table is an important step of dimensional modeling. The data source of the AUBA tool is a collection of web server log files, and each of these log files consists of log lines that each describes a request to the web server. The log lines are the most atomic information since they cannot be subdivided any further. A log line can describe a request for a page, but it can also describe for instance an image or frame request. The actual number of requests that is needed for a page view is beyond the control of the user, so in the users' world a page view is the most atomic information [Srivastava et al., 2000]. The first star schema of the data webhouse for the AUBA tool is designed with a single page event as the grain. A page event is often used as the grain of a fact table in data webhouses [Kimball and Merz, 2000], because it is similar to a page view, but incorporates the knowledge that the user does not always click on a web page to view a new page. Sometimes a request is also sent to the underlying system, such as when a user makes a request to reserve a book.

The kind of star schema that has a page event fact table often includes information about the time of day, date, page, user and session in the dimension tables, so a page event schema can be used for both the non-domain specific and some of the domain specific behavior analysis listed above. The thoughts behind the design of the page event star schema are described in Section 5.1.

The domain specific task of analyzing the ways the users deal with books would be very complex using only the page event star schema. For example, it would still be very difficult to make the analysis of how the web site users most often find the books that they reserve on the web site. A search and a reservation in the page event schema are different types of pages, and a reservation can only be made after a search has been made in the same session. But since a session can have more than one search there is no direct connection to link the two page functions together to aid the analysis of which kind of search led the user to find the book that was reserved. To ease this kind of analysis an additional star schema has also been designed. This schema has a coarser granularity as there is an entry in the fact table for each search made on the web site. It is described in Section 5.2. The two star schemas have three dimensions in common, and therefore they can be depicted together as a so-called galaxy [Kimball and Ross, 2002a] as illustrated in Figure 5.1.

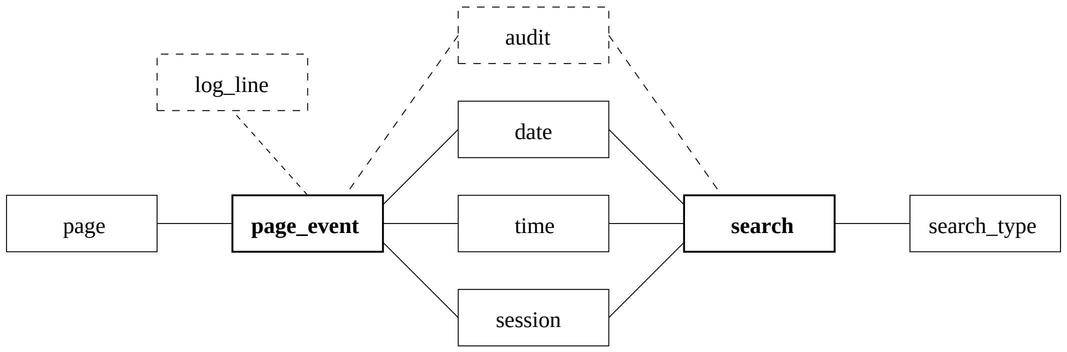


Figure 5.1: The galaxy constellation of the two star schemas.

5.1 Page Event Schema

The page event star schema has a page event fact table and four dimensions. The log line table is a lineage table, which means that it stores information that can be used to trace a particular log line back to its origin. The audit table is used for quality assurance, which is described in Chapter 7. Neither the log line table nor the audit table will be used in regular queries as dimension tables although they are associated with the fact table through a primary key / foreign key relationship just like dimensions. The other four dimensions are date, time of day, page and session.

The page event star schema is illustrated in Figure 5.2. It can be used for many types of queries, but mostly to analyze the basic behavior of the users of the web site. For example, the activity on the web site measured by week of semester can be analyzed and combined with an analysis of the use of the different types of page function to find out if the users behave in different ways at different points of the semester.

5.1.1 Fact Table

The page event fact table is a fact-less fact table [Kimball and Ross, 2002a]. The only columns of the page event table are the foreign keys to the four dimension tables and the log_line and audit tables. It has been designed without measures because the goals set in this project do not require any measures in the page event fact table. If it is later discovered that certain measures are needed to achieve new goals, they can be added to the fact table without changing the rest of the schema [Kimball and Ross, 2002a]. An example of a fact that could possibly help achieve new goals is *dwell time*. This fact could help analysis on how long time the users of the web site spend on each page. If the dwell time fact were to be added it would be necessary to use the log lines describing image requests in order to calculate when a page has finished loading and use this together with the time of the next page event to calculate the dwell time of the current page event [Kimball and Merz, 2000]. At the moment, there is no interest in how much time the users spend on each page, so dwell time is not necessary in the page event fact table.

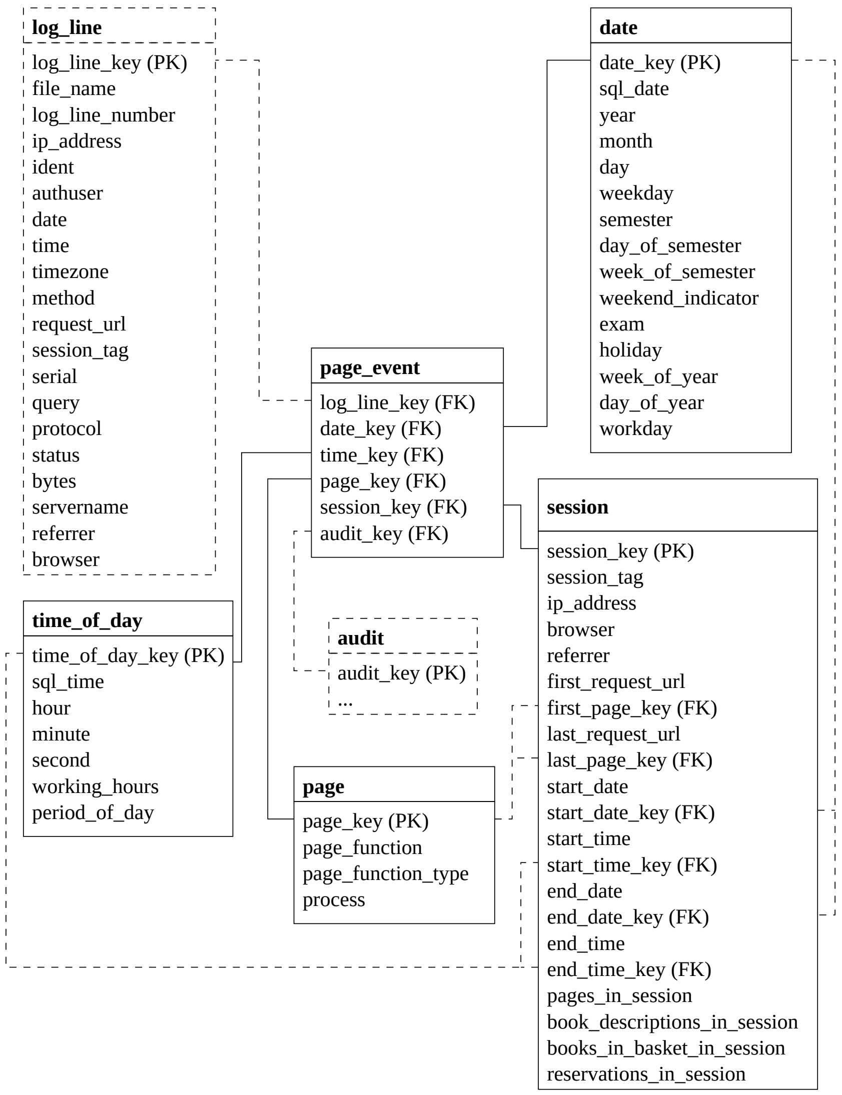


Figure 5.2: Relational representation of the page event star schema.

5.1.2 Log Line Table

If a result is reached in the analysis that seems strange, it is nice to be able to go back to the source of the information to check if it is correct. This is a big task when dealing with information gathered from thousands of log lines, but if it is from only a few log lines it would be nice to be able to see how the log lines looked originally. This is also a good thing to have during implementation of the AUBA tool where an error might occur and the log line that caused the error is needed in order to be able to find out what the cause of the problem was. Such problems often occur when users behave in ways that were not intended in the system, such as clicking the back button of the browser to go back to a session that has expired.

The log line table is a lineage table. It is used to trace information in the database back to its originating log line. It contains a row for each log line that has not been ignored by the post-processor.[1] Because each of these log lines corresponds to a page event, the page event fact table has exactly the same number of rows as the log line table. A row in the log line table contains all the information from the corresponding line in the log file, but it is easier to read since it is organized into labeled columns.

[1]This is explained in Section 6.1.

A lineage table is a good tool to have when doing outlier analysis. For instance when analyzing the number of pages in sessions it appears that a few sessions consist hundreds of pages, even though most sessions contain less than 30 pages. This information may make the analyst curious about the correctness of the analysis. By use of the log line lineage table the analyst can view the information from all the log lines that are represented in a particularly long session. The log lines may reveal that a user has spent two hours clicking through Auboline at a steady pace searching for books, reading book descriptions and making reservations. Using the lineage dimension for outlier analysis enables the user of the AUBA tool to check up on analysis results that seem strange and thereby encourages the users to trust the results produced.

The log_line table will not be used as a regular dimension table to query the data. All the information from the log lines that is relevant in the different analyses is also present in the real dimension tables. Most of the columns in the log_line table are parallel to the attributes in the log files described in Section 3.3. In addition to these columns the log_line table contains a primary key, two columns that refer to the log line in the originating log file and the three attributes session_tag, serial and query columns as described in section 3.3.

log_line_key: Surrogate primary key of the log_line table.

filename: The name of the log file that the log line originated from. The names of the log files have the format “access_log.”<date>. Each filename is unique because it contains the date from when the file was created. For instance, the log file with log lines from 25 February, 2003 has the filename “access_log.20030225”.

log_line_number: The number of the line in the log file where the log line can be found.

session_tag: A tag that the Auboline system places in the URL in order to identify sessions.

serial: A session-unique number that the Auboline system places after the session tag in the URL.

query: The part of the URL that contains variables and values that describe the specific attributes of the page.

5.1.3 Date Dimension

The date dimension has a row for each unique date in the time period that the log lines span over. The values of most of the columns can be computed when more date rows are added but some values, such as school vacation, have to be specified manually. The date dimension has the following columns.

date_key: Surrogate primary key of the date dimension. The primary key of a date dimension should always be sorted by date [Kimball and Ross, 2002a].

sql_date: The sql representation of the date. The sql_date column can be used to make queries on single dates or intervals of dates.

year: The four digit integer year of the date.

month: The integer month of the date ranging from 1 to 12.

day: The day of the month. This integer attribute can have values between 1 and 31.

week_day: The name of the day of the week. The attribute is textual and can have the values “monday”, “tuesday”, “wednesday”, “thursday”, “friday”, “saturday” and “sunday”.

semester: A domain specific attribute specifying the name of the semester that the date belongs to.

day_of_semester: A domain specific attribute specifying the number of the day as calculated from the beginning of the semester that the date belongs to. The first day of a semester is the first week day in February for the spring semesters and the first week day in September for the fall semesters. In this implementation the first day of the semester is specified manually, but it is a possibility to calculate it automatically in future versions.

week_of_semester: A domain specific attribute specifying the number of the week, that the date belongs to, as calculated from the beginning of the semester. The first week of a semester is the week which the first day of the semester is in.

weekend: This column has the value "week day" if the week day is between Monday and Friday and "weekend" if it is Saturday or Sunday.

exam: This is a domain specific attribute specifying that has the value "exam" if the date is in January or June where the regular exam periods are, "reexam" in August and "no exam" in the remaining months of the year.

public_holiday: This attribute indicates if the date is a public holiday. It has the value "holiday" if the date is a Danish public holiday and "no holiday" otherwise. The public holidays have to be specified manually when adding new date rows to the dimension.

school_vacation: A domain specific attribute that indicates if the date is in a period where there is a vacation at the university or not with the values "vacation" and "no vacation", respectively. The different institutes of the university do not always have vacation at the same time so the AUB staff have to make a judgment on which vacations they want to include in this attribute. The opening hours of the library are reduced during vacation so the staff could possibly choose to add vacations to the data warehouse in the same periods as the opening hours are reduced. In this implementation there is a vacation in all of July and August and two weeks around Christmas and New Year.

day_of_year: The number of the day as calculated from the beginning of the year that the date belongs to. The values of this attribute are between 1 and 366.

week_of_year: The number of the week, that the date belongs to, as calculated from the beginning of the year. The first week of the year is the first week with at least four days in that year. That means that if January 1st is a Monday, Tuesday, Wednesday or Thursday, the week that January 1st is in will be the first week of the year. On the other hand if January 1st is a Friday, Saturday or Sunday, the week is the last week of the preceding year. The attribute can have values between 1 and 53.

workday: This attribute has the value "workday" if the date is not a weekend, public holiday or vacation. Otherwise it has the value "no workday".

Restrictions on the date dimension can be used to group the page events by day, month, year, week of semester, week day, workday, etc. to find patterns in how much Auboline is used at different times. For instance, it would be interesting to see how much Auboline is used in the beginning of the semesters compared to the middle and the end. Since most of the borrowers are students, one could assume that the use of Auboline would reflect the pattern that many students have a period in the beginning of the semester where they gather information and therefore use Auboline to find relevant books. Toward the end of the semesters there is probably not as much traffic on AUB's web sites, since the students have all the information that they need and are busy finishing their projects. The result of this analysis is discussed in Chapter 10.

5.1.4 Time of Day Dimension

The time_of_day dimension has a row for each second in a 24 hour period. This corresponds to the accuracy of the time of day in the log files. So far it contains only the following columns, but more can be added without touching any of the other tables and without consulting the originating log files. The time_of_day dimension contains 86400 entries, which is the number of seconds in a 24 hour period.

time_of_day_key: Surrogate primary key of the time_of_day dimension.

sql_time: The sql representation of the time of day. The values of this attribute range from 00:00:00 to 23:59:59. This column can be used to sort query results by time or to query specific time of day intervals.

hour: The hour of the time of day. The values of this integer attribute range from 0 to 23.

minute: The minute of the time of day. The values of this integer attribute range from 0 to 59.

second: The second of the time of day. The values of this integer attribute range from 0 to 59.

working_hours: This attribute has the value “working hours” if the time of day is between 8 and 16 which is the most common working hours. For the rest of the day it has the value “not working hours”. Since the time_of_day dimension does not distinguish workdays from days that are not workdays, this attribute should be used in conjunction with the workday attribute of the date dimension to determine if the time of day is actually within working hours. An alternative solution would be to have two rows for each second between 8 and 16, one used on workdays and one used on days that are not workdays. That way the user of the database will not risk getting an inaccurate result if he or she is unaware that this attribute depends on the value of the workday attribute of the date dimension. Only a tenth of the page events in the data set have taken place on a day that is not a work day.

period_of_day: This attribute indicates what part of the day the time is in. It has the value “night” if the time of day is between 00:00:00 and 05:59:59, “morning” between 06:00:00 and 11:59:59, “afternoon” between 12:00:00 and 17:59:59 and “evening” between 18:00:00 and 23:59:59.

Restrictions on the time of day dimension can be used to find out if most reservations are made during working hours or after hours when the borrowers are not able to go to the library. Since the opening hours vary between the different branches of the library and days of the week there will not be a column to distinguish opening hours from when the libraries are closed. Instead the working_hours column can be used in conjunction with the work_day column of the date dimension to get an idea about whether the users reserve more books when they are at work or in school or in their spare time.

5.1.5 Page Dimension

Some of the most interesting information in a log line is found in the query part of the request URL. The information in this query can be used to find out what type of page was viewed and what the function of the page was. There are 95 entries in the page dimension, and the columns of the table are described in the following.

page_key: Surrogate primary key of the page dimension

page_function: As described in Section 3.3, the function of a page is derived from the values of the variables in the request URL. This attribute can have values such as “reserve book”, “full view of a record” and “basic search on title”. There are 95 different values of the page_function attribute.

page_function_type: The page_function_type attribute is a generalization of the page_function attribute. The page functions can be grouped into different types. For instance, page functions like “basic search on title”, “multi-field search” and “browse author index” are all different types of searches, so their page function types are “search”. Many of the page functions are alike. The page function type can have one of 32 possible values.

process: A process that the page is a part of. There are 10 different processes in Auboline. Examples are “reservation”, “search” and “login”.

Restrictions on the page dimension can be used to distinguish between which kinds of page functions are used most frequently and it can be combined with the time or date dimension to find out if the use of the page functions varies according to for instance period of day or week day.

The attributes of the page dimension form a hierarchy that enables users of the database to drill up and down [Kimball and Ross, 2002a] to get a more general or more detailed analysis. The hierarchy of the page dimension is depicted in Figure 5.3. In the figure, the T represents the whole dimension. The right part of the figure shows examples of values for the different columns.

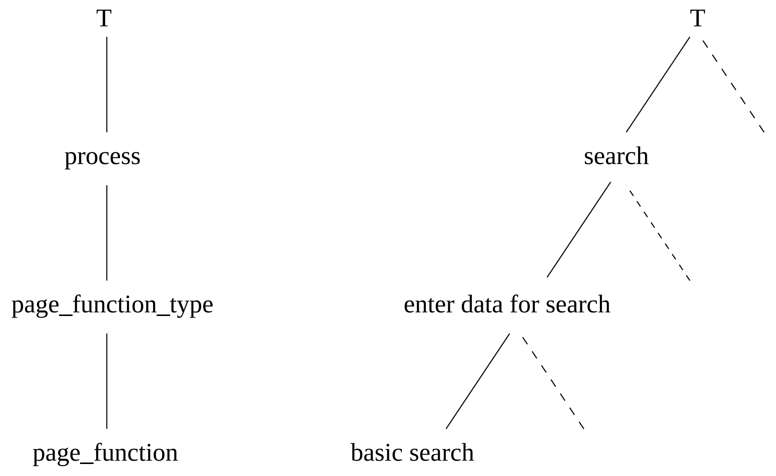


Figure 5.3: Page hierarchy.

5.1.6 Session Dimension

The session dimension contains information about the users and about when and where the users start and end their sessions. The attributes of the session dimension are described below.

session_key: Surrogate primary key of the session dimension table.

session_tag: The session tag created by Auboline for the session, as described in Section 3.3.1. The session tag is a fifty character long string of letters and digits.

ip_address: The ip address of the host accessing Auboline in this session.

browser: The browser of the host accessing Auboline in this session. Since different Internet browsers behave in different ways when reading the same web site, this attribute can be used to find out which browsers Auboline should be aimed at. 96.7 percent of the browsers that have used Auboline during the time period that the source data spans over are different versions of Microsoft Internet Explorer so this is the primary browser that AUB should adjust Auboline to.

referrer: The referrer is the referrer field of the log line of the first page event in the session. It shows where the user came from before entering Auboline. The referrer field is empty if a user is redirected from AUB's web site, uses a browser bookmark or favorite or if the URL is entered in the address field of the browser, so it is only visible when a user has clicked on a link on another web site that does not link to http://a500.aub.auc.dk which is the URL of Auboline that redirects to the Auboline start page. Only ten percent of all sessions in the data set have a value in the referrer field.

first_request_url: The first URL requested in this session. This attribute shows that 49 percent of all sessions start on the page that a visitor is redirected to when clicking on Auboline on the general AUB web site or typing http://a500.aub.auc.dk directly in the address field of the browser.

first_page_key: An outrigger to the primary key of the page dimension that allows combining session information with information about the first page in the session without joining both tables with the big page_event fact table. An outrigger is a foreign key from a dimension table to the primary key of another dimension table. By having outriggers between dimensions, it is possible to join the dimensions without involving the large fact table and thereby it is possible to improve performance on certain queries [Kimball and Ross, 2002b].

last_request_url: The last URL requested in this session. This attribute is not as comparable as the first request URL. Whenever a session consists of more than one page event, the last request

URL will have a session tag. Therefore the value of this field is unique for every session with more than one click.

last_page_key: An outrigger to the primary key of the page dimension that allows combining session information with information about the last page in the session without joining both tables with the big page_event fact table.

start_date: The SQL date of the first request in this session. This attribute has the same value as the sql_date attribute of the date dimension for the first page event in the session.

start_date_key: An outrigger to the primary key of the date dimension that allows combining session information with information about the start date of the session without joining both tables with the big page_event fact table.

start_time: The SQL time of day of the first request in this session. This attribute has the same value as the sql_time attribute of the time_of_day dimension for the first page event in the session.

start_time_key: An outrigger to the primary key of the time dimension that allows combining session information with information about the start time of the session without joining both tables with the big page_event fact table.

end_date: The SQL date of the last request in this session. This attribute has the same value as the sql_date attribute of the date dimension for the last page event in the session. In most cases this will be the same as start_date, but if the session starts before midnight and ends after, the end_date can also be the date after the start_date.

end_date_key: An outrigger to the primary key of the date dimension that allows combining session information with information about the end date of the session without joining both tables with the big page_event fact table.

end_time: The SQL time of day of the last request in this session. This attribute has the same value as the sql_time attribute of the time_of_day dimension for the last page event in the session.

end_time_key: An outrigger to the primary key of the time_of_day dimension that allows combining session information with information about the end time of the session without joining both tables with the big page_event fact table.

pages_in_session: The number of page requests in this session. Most sessions (75 percent) have less than ten clicks but some sessions are very long and contain up to 222 pages. The average number of pages in a session is 9.7.

book_descriptions_in_session: The number of book descriptions in this session. The average number of book descriptions in a session is 1.4.

books_in_basket_in_session: The number of books that are put in the basket in this session. The average number of books in the basket in a session is 0.05.

reservations_in_session: The number of reservations in this session. The average number of reservations in a session is 0.3.

5.2 Search Schema

The main purpose of the AUBA Tool is to assist the AUB staff in finding success rates of the different ways that users can find books with Auboline. At the moment the only way to find a book with Auboline is to make a search. Therefore a schema with a search fact table can make this kind of analysis very easy. When the book recommendation service is integrated with Auboline it will be simple to add a new “recommendation” search type to the search type dimension and use it to calculate the success rate of the recommendation service. The search star schema is depicted in Figure 5.4.

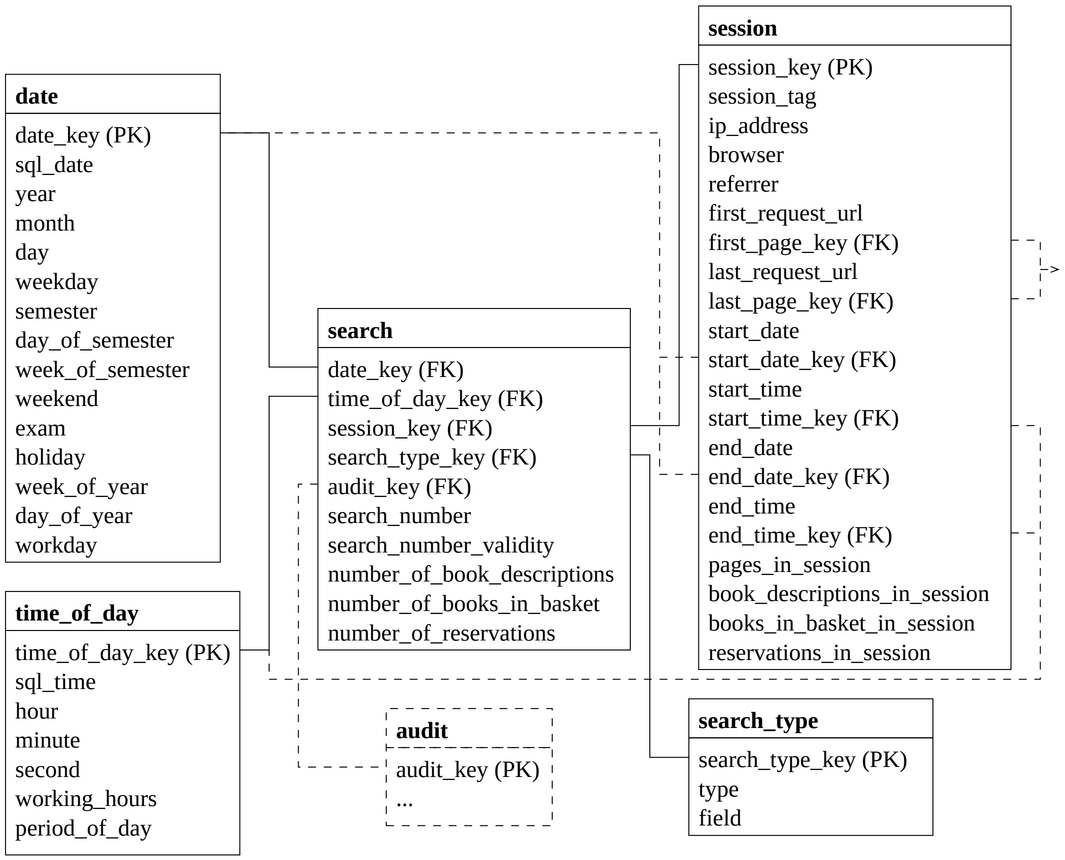


Figure 5.4: Relational representation of the search star schema.

5.2.1 *Fact Table*

The search star schema has the date, time of day and session dimensions in common with the page event star schema. Therefore it is possible to use these dimensions to slice and dice the data that deals with searches [Pedersen and Jensen, 2001]. For instance it is possible to find out how the number of reservations is distributed in the course of a semester, or which types of searches are most common in the beginning versus the end of a semester. It would be interesting to see if there are more searches on key words in the beginning of the semesters and then more precise searches, for instance on ISBN number or title toward the ends of the semesters.

date_key: Foreign key to the date dimension describing the date of the search. This can be used to join the search fact table with the date dimension to see if the most common types of searches varies according to the different attributes of the date dimension.

time_key: Foreign key to the time dimension describing the time of the search. This can be used to see for instance how the number of reservations vary during the course of a day.

session_key: Foreign key to the session dimension describing the session that the search was made in. This can be used to see for instance if the same types of searches are commonly used in the same sessions or the users try different kinds of searches to find the books that they are looking for.

search_type_key: Foreign key to the search_type dimension describing the type of search made. This can be used to make an analysis of what types of searches the users use and which book attribute fields they most often search in.

search_number: The number of the search (degenerate dimension) either given by Auboline or the post-processor. A degenerate dimension is neither a fact nor a foreign key to a dimension table. It is a column that can be used to group measures. The search number can be used to group together searches that originate from the same search. This way if a user chooses a search from the search history, the facts can be added with the facts from when the search was first made. The problem is that this can only be done if the valid search number, from when the search was first made, is present. The search number is not unique so it has to be combined with the session key to be sure that the searches with the same search number refer to the same search.

search_number_validity: Describes whether the search number is valid (given by Auboline) or temporary (given by the post-processor because the valid search number has not appeared in the request URL). The search number only appears in the request URL after the user has clicked on one of the search results, so the search number is not always known and therefore it is not always possible to recognize a search that is chosen from the search history.

number_of_book_descriptions: The number of books found by way of this search where the user has read the book description. This can be used to see how many book descriptions the users read after the different kinds of searches. If a user reads a book description it indicates that the book looked relevant to the user.

number_of_books_in_basket: The number of books found by way of this search that the user has put in the basket. If a user puts a book in his or her basket it is an indication that the user is considering reserving the book later on in the session.

number_of_reservations: The number of books found by way of this search that the user reserved. This is very central in measuring the success criteria of the different searches, since a reservation indicates that a user has found a book that he or she wants to borrow. Unfortunately it is not possible to see if a book that has been reserved is the same book as one that was previously put in the basket.

5.2.2 Search Type Dimension

The search_type dimension contains 80 entries. The search type dimension is a very important dimension in the data webhouse because it describes the kinds of searches that are made in Auboline. Auboline has different types of searches and with some of the search types the user can choose one or more fields to make the search on.

search_type_key: Surrogate primary key of the search type dimension table.

type: The type of search performed. The different types of searches in Auboline are “basic search”, “multi-base search”, “multi-field search”, “browse index” and “CCL search”. CCL is short for “Common Command Language”. With this function it is possible to use command language to make a search on one or more fields at a time. An index is an alphabetical list of authors, titles, UDK classification numbers or subjects. If a user makes a search in the author index on the name “Andersen” she gets a list of ten author names, namely the name “Andersen” and the next nine names in the alphabetical author index. From this page the user can go to the previous ten or the next ten authors until he or she finds the right one. When the user uses the search history function in Auboline it is not always possible to know which of the previous searches in the session is used.[2] Therefore the search type attribute will have the value “history” if the user clicks on a search from the history function. Likewise the value in the type column will be “basket” if the user clicks on a book in the basket, because the request URL contains no indication of which of the previously saved books is clicked on.

field: The field that was used in the search. When a borrower wants to make a search, she can choose which field, for instance “title” or “subject”, to search in, from a drop down box. The

[2] When a user clicks on a search in the search history, the search number appears in the request URL, but this number can not always be recognized from a previous search because the search number does not appear in the request URL when the search is made but only later, if the search results are used, i.e. a book description is read.

field column can also have the value “no value” if the user made a search without specifying a particular field or “all fields” if the user made a multi-field search the user has chosen to search in all fields.

type_with_field: This attribute contains a combination of the two attributes above. For instance if the type is “multi-base search”, and the field is “author”, then this attribute has the value “multi-base search on author”.

The attributes of the search type dimension form a hierarchy as illustrated in Figure 5.5. The type and field attributes can be used separately or combined.

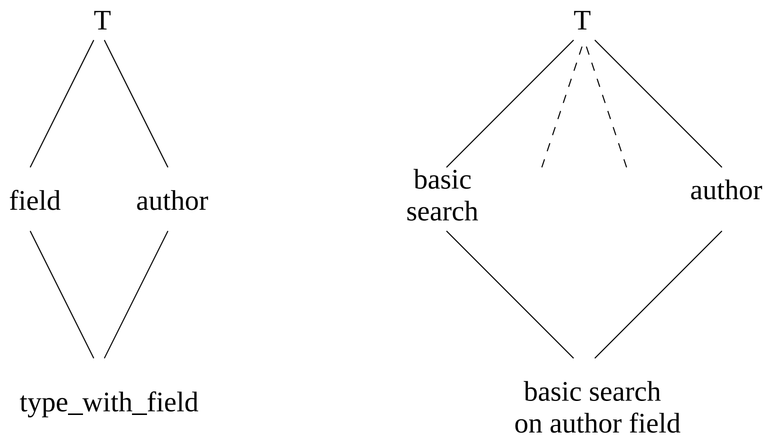


Figure 5.5: Search type hierarchy. To the left is an example of possible values of the attributes.

CHAPTER 6

Post-Processor Implementation

The post-processor is the main part of the implementation of the AUBA tool. It handles the extract-transformation-load (ETL) processes that are necessary to transfer the information from the log files to the database. This chapter describes how the post-processor is implemented and how it handles the ETL processes, especially the transformation part where page functions, sessions and searches are recognized. At the end of the chapter the performance of the post-processor is discussed.

The post-processor processes one log line at a time. The following is a summary of the tasks of the post-processor. The tasks will be explained in the next sections.

- Split each log line up in the attributes described in Section 3.3 and convert them to the appropriate formats as described in Chapter 5
- Ignore image and style-sheet requests
- Ignore requests from search bots and web crawlers
- Ignore requests to administrative systems
- Transfer the information from all recognized log lines to a log line table with appropriate columns in the database
- Find the keys of the appropriate rows in the dimension tables to describe the page event of the particular log line
- Find the keys of the appropriate rows in the dimension tables to describe the search if the request in the log line is a search
- Insert a new entry in the page_event table, and in the search table if the log line describes a search, with the keys found
- Update the information in the record in the session table that the log line belongs to or insert a new session record if necessary
- Update the measures of the search fact table if needed

All the processes and the database tables that are involved in the transformation of information from the source data to the final dimensional database are also referred to as the data staging area in data warehousing. The data presentation area contains the final tables of the dimensional database. The post-processor has two main functions as described above. The first one, which takes care of log line recognition and transformation to the log line table in the database, is described in the following section. The process of transforming the data from the log lines to the tables of the dimensional database is described in Section 6.2. The two processes are combined in the implementation, because it is faster to finish processing of each log line while it is in main memory instead of saving all log lines in the log line table and then starting over by scanning this table to find dimension keys, etc.

6.1 Log File Recognition and Transformation

When the post-processor is started, it makes a chronological scan through all the access log files in the directory, where these files are stored. The post-processor reads each of these files one line at a time. For each line it calls the constructor of the LogLine java class (Section E.6) to construct a LogLine object with the information in the log line string. If the constructor recognizes the string as a valid log line, it returns a LogLine object. This LogLine object is then used to insert a row in the log_line table in the database using the insertLogLine method of the *Database* class in Section E.2. The log lines are not inserted in the actual database table right away, because this would cause too many writes to the database, which should be avoided for the sake of the performance of the log line recognition process. Instead the log lines are written to a file, and the PostgreSQL copy command is used to move the data from the file to the log line table in the database. This is done each time the end of a log file is reached before the post-processor continues with the next file.

Some of the information in the log lines is adjusted before it is saved in the database. For instance, the date is converted to an SQL date which makes it easier to sort by the column in the database. The data definition of the log_line table can be found in Section A.2.

6.2 Transformation to Dimensional Schema

The second function of the post-processor is to transform each log line so it can be loaded into the dimensional database. This process will be referred to as "dimensionalizing the log lines" throughout the rest of the report. Each log line corresponds to a page event. Therefore the process of transferring the information from each log line to the tables of the page_event star schema basically means creating a new entry in the page_event fact table with the information from the log line. In order to do this, however, it is necessary to find out which records of the dimension tables the foreign keys in the fact table should refer to. This is explained in the following sections.

6.2.1 Time and Date

The time and date dimension tables are preloaded with all the information that is needed. There is only 24 hours in a day, so the time dimension will not have more rows added later. But when the AUBA tool is to be used for new data with dates that are later than the preloaded dates, new rows have to be added to the date dimension table before the new data is loaded into the database. This has to be done manually, because the system is unable to predict all information needed to add more dates. For instance it cannot predict all the school vacations.

The primary key of the time dimension is calculated to limit input/output between the post-processor and the database and increase performance. The key of the date dimension could also be calculated, but instead the post-processor checks if the date is the same as the date of the preceding log line, and it only fetches the date key from the database if the dates are not the same. Since the log lines are processed in chronological order, the date will be the same as the previous except for at most two times in each log file where the date has changed between the two log lines.

6.2.2 Page

The page dimension has also been preloaded into the database with all the possible page functions, page function types and processes. For each page event, the values of these variables are decided in the *setVariables()* method of the *Page* class (Section E.7). It uses the variables in the query part of the request URL as described in Section 3.3.1. After finding the values of the three columns of the page dimension, the post-processor attempts to find the page key for the row of the page dimension that has the same values for the three variables. Again the input/output to and from the database has been limited. The contents of the page dimension is loaded into a vector at the beginning of the dimensionalize process, and this vector is scanned to find the appropriate page key for each page event.

6.2.3 Session

The session dimension table is the only dimension that is not preloaded into the database. The reason for this is that its content is not as predictable as the other dimension tables. The content of the session table is as dynamic as the content of the page event table since each new page event either changes a session or creates a new session. The data is loaded into the session table in parallel with the loading of the log line and page event tables to avoid processing each log line more than once.

The sessions that are active at a point in time in the post-processing process are the sessions where the end time is less than twenty minutes before the time of the log line that is being processed at that moment. All active sessions are stored in a vector during the post-processing. For each log line the post-processor needs to check the active sessions vector to find out if the new page event belongs to a session that is already in progress. If a matching session is found, the session is updated with the information from the log line and the key of this session is returned. If no matching session is found, the post-processor must create a new session. The active sessions are stored in a vector to avoid multiple access to the database, and whenever an inactive session is found in the vector, it is written to a file. The file containing inactive sessions is copied to the database using the PostgreSQL copy function at the end of the ETL process. The process is illustrated in the following algorithm. The checkIfLogLineMatchesSession() method invoked in this algorithm can be found in Algorithm 2

Algorithm 1 Scanning the active sessions vector for matching session

```
ADDPAGEEVENTTOACTIVESESSIONSVECTOR()
  sessionKey ← -1
  for all active sessions in vector until match is found do
    if session is inactive then
      write session to file
      remove session from vector
    else
      if checkIfLogLineMatchesSession(currentSession, logLine) then
        update session with log line information
        sessionKey ← currentSessionKey
      end if
    end if
  end for
  if sessionKey = -1 then
    create new session
    sessionKey ← new session key
  end if
  return sessionKey
```

The advantages of using a vector to store active sessions instead of making an insert or update to the database for every log line are clear. But other data structures, such as a hash table could also have been used to store active sessions in main memory. A vector is simple to use while still preserving the iteration of the elements. It is very useful to keep the sessions organized by when they were first added to the vector. When a new page event is going to be added to a session, the activeSessions vector is scanned to find a matching session. Because the sessions are organized by start time, the sessions that started before the session matching the new page event, will be passed in the process of locating the matching session. If one or more of the passed sessions ended more than 20 minutes before the time of the new page event, it is not necessary to find out if the page event matches these sessions, because they are no longer active. Therefore these inactive sessions are written to the copy file and deleted from the activeSessions vector. The sessions that are inactive have a high probability of having an earlier start time than the session matching the new page event, and therefore it is not necessary to search beyond the session that matches the new page event. The goal of this process is to delete expired sessions from the vector as early as possible to make the vector as small as possible without making extra scans through the vector. A disadvantage of using a vector for storing active sessions is that it is slower to use compared to for instance a hash table, because it is necessary to scan the vector from the beginning for every session that needs to be found. A hash table enables faster access to the individual elements if there is a key to search for. When looking for a session it is sometimes the session tag, sometimes the IP address and browser information and sometimes the last request URL field that needs to be compared so a hash table is not a suitable solution for storing the active sessions.

At the end of the dimensionalization process there are most likely still sessions that are active. These should not be discarded, but they should not be added to the dimensional database either, because they are unfinished and could be changed at a later time when more page events are added. Therefore they are written to a copy file and copied to an active session table in the database at the end of the dimensionalize process. When a new dimensionalize process starts, the active sessions from this table are loaded into the activeSessions vector again so new page events can be added to these sessions if they match.

The process of finding out if a page event matches an active session or a new session should be created from the page event is aided by the fact that Auboline is session-based. Auboline places a session tag in the request URL to be able to differentiate between different users that are using the system at the same time. When a user enters Auboline from the general AUB web site, he or she does not have a session tag. Only a third of all sessions have a session tag on the first page. This could, for instance, be because the user has previously saved a link to an Auboline page in the browser favorites. At the first click in Auboline, the user gets a session tag which she keeps for the rest of the session. If there is more than 20 minutes between two clicks, the system automatically logs out and the user gets a new session tag if he/she starts to use the system again. This means that the Auboline sessions conforms to our perception of a session as an uninterrupted series of clicks by the same user, where an interruption is a pause of more than 20 minutes between two clicks.

As mentioned earlier, the post-processor attempts to find a matching session for each page event. The source code of this process can be seen in the *getSessionKey()* method of the *Database* class in Section E.2 along with the *matches()* method of the *Session* class in Section E.11. The pseudo code can be found in Algorithm 2.

Algorithm 2 Finding a matching session for a new log line

```
CHECKIFLOGLINEMATCHESSESSION(CURRENTSESSION, LOGLINE)
  if currentSession has session tag then
    if currentSessionTag = logLineSessionTag then
      return true
    else
      return false
    end if
  else if currentSessionIP = logLineIP and currentSessionBrowser = logLineBrowser and currentSessionLastRequestURL = logLineReferrer then
    return true
  else
    return false
  end if
```

The session tag that Auboline has placed in the request URL after the first click can be used to find the session that a page event belongs to. The first two pages in a session, however, need to be handled differently. If the request URL of a log line does not have a session tag, it is the first page in a session. Therefore a new session should be created. So far the new session will only contain this page. If the post-processor encounters a log line with a session tag in the request URL but no session tag in the referrer field, the page is the second page of a session. To find the session that this page belongs to, it is necessary to search the active sessions vector to find a session with the same IP address and browser as the new page and where the request_url column matches the referrer of the new page. If a matching session is not found in this way, this is an indication of a new session that has a session tag on the first page (saved by the user in favorites of the browser). In this case the post-processor creates a new session with this page as the only one so far. This new session will have the same session tag as a previous session, but has a unique session key.

When looking for a matching session for a page with a session tag in the request URL but no session tag in the referrer, there will be a certain insecurity about the result. The session tag is the only certain method of matching a page event with a session, because more than one user can have the same IP address and the same browser at the same time. If there are two concurrent sessions with the same IP address and browser there is a risk of swapping the first pages of these two sessions if they start on the same page. The chance of two sessions starting on the same page is great, because the basic search page, which is the page that a user comes to first if he/she enters Auboline from the general AUB web site, is the most common start page. If the first pages of two sessions are swapped this will not affect the result of the analyses very much. The IP address, browser and request URL of the two pages that are swapped are the same, so the only things that are different are the time and the referrer. The time will be at most 20 minutes wrong since the post-processor only finds sessions that ended within 20 minutes before the time of the new page event, and it will not affect the average length of sessions since the sum of the length of the two affected sessions will stay the same. The last information that can be affected if the start pages of two sessions are swapped, is the referrer field. Since the two swapped pages are first in the sessions, the referrer field only has a value in 9 percent of all cases, so this will not affect any analyses much either.

6.2.4 Search Type

The data of the search type dimension table is preloaded into the database and when dimensionalization starts, it is loaded into a vector to avoid unnecessary communication with the database. A *SearchType* object is created by the constructor of the *SearchType* class (Section E.10) if a log line describes a search. The search type key is collected from the vector as described for the page key to improve performance.

6.2.5 Search

In order to populate the search fact table, the post-processor needs to add a new entry whenever it encounters a page event that is a search. Furthermore, the facts of the appropriate row in the search fact table should be increased when a user views a book description, puts a book in the basket or reserves a book. This process is illustrated in Algorithm 3.

Algorithm 3 Populating the search fact table and incrementing the facts

```
for all page events do
    if function type = search then
        find search type key
        add search to search fact table with search type key and keys from page_event fact
    else if function type = book description then
        increment number_of_book_descriptions measure of current search by 1
    else if function type = book in basket then
        increment number_of_books_in_basket measure of current search by the number of books
        chosen to put in basket
    else if function type = reservation then
        increment number_of_reservations measure of current search by 1
    end if
end for
```

Active searches are kept in a vector in the same way as active sessions, but they do not necessarily become inactive after twenty minutes. A search becomes inactive when the session that it belongs to becomes inactive or when a new search is added to the session. Therefore an active search is written to the search copy file and removed from the active search vector when one of these events occur.

Each search in Auboline gets a search number, but the search number is not always present in the request URL. Therefore a temporary search number sometimes needs to be added to a search in order to recognize it from the session that it belongs to during post-processing. The search number usually shows in the request URL when the user clicks on a search result, so in this case the search number of the search and the session needs to be updated to the valid search number. The search number degenerate dimension of the search fact table can be used to group together searches with the same valid search number so results of searches from history can be combined with results of the original search.

6.3 Data Staging Area

The tables that are used to keep active sessions and searches in the data staging area of the AUBA system are like the tables in the data presentation area, but some of them have extra columns to help the post-processing of the data from the log lines continue. The session table in the data staging area has two additional columns. These are used to manage the searches as the log lines are processed by the post-processor. Since the request URL of the log line does not always reveal the search number that the actions of the user are related to, it is vital to keep track of the last search of each session during the post-processing in order to know which search to add reservations, etc., to. The value of this last search number attribute changes during the post-processing of a session whenever a new log line indicates that a new search has been made. The search_number_validity column has the value "valid" if the search number in the last_search_number column is a true search number created by Auboline and the value "temporary" if the search number is created by the post-processor.

6.4 ETL Performance

The performance of the ETL process has been measured on a 850 MHz AMD Duron with 384 Mb RAM running Debian Linux. On this machine, the ETL process takes three and a half hours to read through the 391 log files with a total of 17.7 million log lines and load approximately 3,9 million valid log lines into the log line table of the database along with 3,9 million page events, 400,000 sessions and 826,000 searches. This is equivalent to 325 log line each second or an average of 32 seconds per log file. The part of the ETL processes that takes most of the time is the actual processing of the log lines where the line is recognized and the dimension keys are found. After this, the process that copies the data to the database takes the longest time. The actual execution time of the ETL processes that would be executed each night to transform the data from the last 24 hours would probably take around the same amount of time depending on the available resources on the machine. Half a minute each night is very acceptable, especially because the load on the machine is very low at the time that the processes will be running.

The following chapter discusses how audit control and quality assurance has been integrated in the post-processor. The quality assurance parts of the ETL process are not included in the performance estimations discussed in this section.

CHAPTER 7

Data Quality Assurance

In defining the ETL processes it is important to make sure that the data is extracted, transformed and loaded correctly. There are various methods of data quality assurance that help ensure this. These will be discussed in this chapter.

It is important to make sure that every row in the log file is treated correctly and that the database always contains correct information. The data quality assurance checks can be divided into three main categories [Kimball and Ross, 2002b].

- Audit control: the database must contain as many records as are expected based on the input
- Correctness of data: the ETL process should make sure that the data in the database reflects the true behavior of the users in Auboline
- Error handling: errors in the source code of the AUBA tool and outside errors such as system crashes should be caught and handled to avoid flawed data in the database.

In the following sections these three categories will be discussed along with what has been done to assure data quality in the AUBA tool.

7.1 Audit Control

The purpose of audit control is to make sure that the data flows through the different stages of the ETL process as expected. For instance, if the program is given a log file with 10,000 log lines, the program is expected to process 10,000 log lines. If 8,000 of the processed log lines are invalid these should be discarded and 2,000 new records should appear in the database. If these numbers do not match, something in the ETL process is not as expected. In this case error handling should make sure that the database is modified back to the correct state it was in before the ETL process started. Error handling may also restart the process. This is elaborated in Section 7.3.

Audit control runs in parallel with the ETL processes in the post-processor. This way, mismatches between the audit numbers are caught as early as possible and any updates to the database can be rolled back. The ETL process is stopped if an audit number mismatch occurs, because it will not be possible to end up in a correct database state if some of the audit numbers mismatch in the process. Tables 7.1 through 7.4 list the different audit numbers that are collected and matched through the ETL process. The audit number column contains a description of each audit number, while the check column describes the check that is made when the particular audit number becomes available. The numbers in the check column refer to the leftmost numbers of the table rows. These numbers are continuous among the tables because the audit numbers are sometimes matched with numbers from other tables.

	Audit number	**Check**
1	Lines in the log file	
2	Total log lines processed	2 = 1
3	Valid log lines processed	
4	Invalid log lines processed	3 + 4 = 2
5	Lines in log line copy file	5 = 3
6	New log line records in database	6 = 5

Table 7.1: Log line audit numbers and checks

The collection of log lines can be followed through the ETL process to make sure that everything adds up the way it should at each point in the process. The numbers that can be compared during the process are listed in table 7.1. The number of log lines processed (2) should be the same at the number of lines in the log file (1). This number can be divided into valid (3) and invalid (4) log lines. The valid log lines is the interesting of these numbers. It should match the number of lines in the log line copy file (5), which should then match the number of new log line records in the database (6). The number of new log line records in the database is found by counting the number of records in the log line table that refer to the same filename as the current audit record. This can take minutes when the log line table is big, so it can be omitted if performance is more important. All updates to the database, except for the audit table, are collected in a single transaction so if something goes wrong, all the changes to the database will be rolled back.

	Audit number	**Check**
7	Active page events before ETL	
8	New page events processed	8 = 3
9	Active page events processed	
10	Lines in page event copy file	7 + 8 = 9 + 10
11	Lines in active page event copy file	9 = 11
12	New page event records	10 = 12
13	New active page event records	11 = 13

Table 7.2: Page event audit numbers and checks

In the same way as for the log lines, it is possible to check if the number of page events adds up through the ETL process. The checks, listed in Table 7.2, are a bit more complex than for the log lines because the set of page events that are handled in the ETL process is split up into active and inactive page events. Active page events are the ones that belong to an active session and therefore cannot be inserted into the page event table before the session has been completed and inserted into the session table.

Because each log line corresponds to a page event, the number of new page events processed should be equal to the number of valid log lines processed. This number is not the same as the number of lines in the copy file as it was for the log lines, because existing active page events and the new page events processed are combined to the new set of page events that is again divided into active page events and finished page events. Therefore the sum of the number of active page events before (7) and the new page events processed (8) should equal the sum of the number of active page events processed (9) and the number of lines in the page event copy file (10). The number of lines in the active page event copy file (11) should be equal to the number of active page events processed (9). As for the log lines, the number of new page event records (12) and new active page event records (13) should be equal to the number of lines in the page event copy file (10) and the active page event copy file (11), respectively. The number of new page event records can be found by counting the number of page event records that refer to the current audit key.

	Audit number	**Check**
14	Active searches before ETL	
15	New searches processed	
16	Active searches processed	
17	Lines in search copy file	14 + 15 = 16 + 17
18	Lines in active search copy file	16 = 18
19	New search records	17 = 19
20	New active search records	18 = 20

Table 7.3: Search audit numbers and checks

The audit checks for the search fact table are listed in Table 7.3. Since the search table is a fact table as well as the page event table, and they have similar characteristics, the audit checks for the two dimensions are also the same. The only difference is that the number of new searches processed can not be matched with the number of valid log lines processed, as was possible for the number of new page events processed.

	Audit number	**Check**
21	Active sessions before ETL	
22	New sessions processed	
23	Active sessions processed	
24	Lines in session copy file	21 + 22 = 23 + 24
25	Lines in active session copy file	23 = 25
26	New session records	24 = 26
27	New active session records	25 = 27

Table 7.4: Session audit numbers and checks

The audit checks for the session table listed in Table 7.4 are the same as the ones for the search table. But since the session table does not reference the audit table it is not possible to count the number of new sessions in the same way as the numbers of new page events and searches are counted. Instead, the counting is done by comparing the number of records in the session table before and after the load. The number of records before the load can be found by looking at the number of records after the previous load in the audit table. The number of records after the load is found by counting the records in the session dimension.

7.2 Correctness of Data

The process described above ensures that the right amount of data is transfered to the database in the ETL process. This is a good way to find errors that occur during this process. But even if all the numbers match, the audit control cannot guarantee that the new content of the database is correct. This is very important as well so the results of the analyses made with the program can be trusted. One way to test if the database contains correct data after it has been processed by the post-processor is to check if the output is as expected when the post-processor is given a small familiar data set. In addition to this method, [Kimball and Ross, 2002b] lists five different ways of assuring data quality. One of these is audit control, which has already been discussed, and the remaining four are listed below with the manual data processing method just mentioned.

- Manual data processing
- Referential integrity
- Cross-footing
- Manual examination
- Process validation

These five methods of assuring data quality are discussed in the following sections.

7.2.1 Manual Data Processing

It is not possible to go through all the log files manually and make sure that the content of the database is exactly as expected. However, if a part of the log files is checked up against the data in the database, there is a good chance of finding possible errors in the program. If there is an error in the program, chances are that this error will lead to several inconsistencies between the log files and the content of the database. Therefore there is a good chance of finding such an error by selecting random log file passages and testing whether the information in these passages are represented in the database exactly as expected.

A possible test is to examine a small file and decide what the output of the post-processor should be, if it were given this file. After the expected output has been defined, the post-processor should be run on the file and the output should be like expected.

Another way to test if the content of the database reflects the reality that it represents is to use Auboline to make a session where all clicks are known. The progress of the session and the content of the pages in it should be recorded. When the log file for the particular day is available it can be processed by the post-processor and then it is possible to examine if the database contains a session that reflects the reality of the test session.

These kinds of tests should primarily be performed before the AUBA tool is integrated at AUB, but if changes are made to the AUBA tool or to Auboline it would be a good idea to perform these kinds of tests again to make sure that the changes do not cause the data in the database to be incorrect.

7.2.2 Referential Integrity

As mentioned in Section 7.1, audit control assures that the database contains the right number of rows in the different tables. When this is assured, it is also important to make sure that everything matches up in the database. Referential integrity can be ensured by placing foreign key constraints between the fact tables and the dimensions in the data warehouse. This means, for instance, that it is not possible to insert a record in the page_event fact table that refers to a log line that does not exist in the log_line table or a session that does not exist in the session dimension. Placing foreign key constraints on the tables slows down insertion into the referring tables and deletion from or updates to the referred tables, so it is recommended to drop the constraints before insertion of large amounts of data and place them on the tables again after the insertion. One recommendation is that foreign key constraints should be dropped if the insert affects more than 25 % of the table [Corey et al., 2001]. Therefore, the foreign keys are dropped for the initial load for the AUBA tool where approximately one year of log file data is loaded into the database, but they will be kept for incremental loads because they only affect a very small part of the data in the database.

7.2.3 Cross-Footing

With cross-footing, the data warehouse is queried to check if the data has certain important characteristics. For instance, the pages_in_session attribute of the session dimension must match the number of records in the page_event fact table that have the same session key as the session. This cannot be checked with a table constraint on the session dimension, because a session is always inserted into the database before the page events that belong to the session, and therefore there are no page_event records with the session key in question when the session is inserted. The following view and query selects the session key, number of pages in session and the number of page events belonging to the particular session for all such combinations where the two numbers are not equal.

```
CREATE VIEW page_count AS
      SELECT session_key, COUNT(*) AS page_count
      FROM page_event
      GROUP BY session_key;

SELECT session.session_key, page_count.page_count, session.pages_in_session
FROM page_count, session
WHERE session.session_key = page_count.session_key AND
      session.pages_in_session <> page_count.page_count
```

The result set of the query contains all sessions where the number of pages in the session does not match the number of page events that reference the session key. Therefore, the result set should be empty if everything is in order. When running the query on the data from the initial load in the data warehouse, the result set is empty, as it should be, and the test is a success. It takes six minutes to execute the query on the data in the initial load so it should not be executed after each load. Rather, it could be part of a test set that is executed at a weekly or monthly basis to check if everything still matches up. Other similar tests could also be executed within this test set.

7.2.4 Manual Examination

If unexpected numbers occur in the database it could be a sign of an error in the AUBA tool or in Auboline. Therefore it is a good idea to examine the database for such numbers. For instance if there is a session with 2000 clicks or if every search made on a particular day are ’basic searches’. When such an out of range number is discovered, the person responsible for the AUBA tool should investigate the cause to find out if an error has caused it or if there is another explanation. For instance, an extremely long session can be caused by a web crawler that is unknown to the AUBA tool. If such a web crawler is discovered, all log lines created by this web crawler should be discarded from the database, and the web crawler should be added to the list of known web crawlers so it will not affect the analysis results in the future.

This kind of test should be performed at a regular basis. It is important that the person performing the tests are familiar with what is normal and what is abnormal data in the database. This kind of test has been used during implementation and has resulted in for instance discovery of new search bots that were not recognized by the post-processor.

7.2.5 Process Validation

Another of the data quality assurance methods that [Kimball and Ross, 2002b] recommends is process validation. It would be a good idea to sit down with somebody who has great knowledge about Auboline and its underlying implementation. Preferably a process validation team would include someone who has extensive knowledge about the values and semantics of the variables in the URLs and someone who knows how Auboline is used at AUB. Unfortunately, it has not been possible to get information about the underlying implementation of Auboline and the variables in the URLs, and the underlying system is not developed by AUB, it has not been attempted to contact the people who have implemented it. If it had been possible to put together a process validation team, it would have been used at different stages of the implementation process. First, before anything was implemented to translate the URLs into the information that is needed for the data warehouse, then at different milestones in the implementation process, and finally to validate the ETL processes in the testing phase.

7.3 Error handling

When an error has occurred it usually means that the content of the database is not as it should be. Such a state could be caused by an error in the AUBA tool, a power outage during the ETL process, or other problems. These kinds of problems should be taken care of by the error handling part of the AUBA tool. Part of the responsibility of error handling is to make sure that the database is returned to a state where the content is valid. This can be done by rolling back the changes made to the database during the failed ETL process. Errors should be handled automatically whenever possible so human intervention is rarely necessary. For instance, in the event of a power outage the ETL process that was running can be rolled back and started over by the AUBA Tool.

The Audit dimension is a very useful tool for error handling, where information about what happened during the previous ETL processes can be found. There are two possible scenarios that can cause error and invoke the error handling process. Either the ETL process has been interrupted by a mismatch between some of the audit numbers or there is another reason, such as a power outage, that has caused the ETL process to stop before it has completed. The handling of these two types of error is described in the following.

7.3.1 Process Interrupted by Audit Number Mismatch

Audit number mismatches can be caused by either errors in the source code of the AUBA tool or by system errors, such as when the disk is full. When such an error occurs, any changes made to the database, except for the audit table, are rolled back. As described in Section 7.1, audit number mismatches can occur throughout the entire ETL process. Depending on when the error has occurred, it may be beneficial to attempt to recover the process at a point when the audit numbers matched, or maybe the process should rather be started over from scratch. Since the first parts of processing of a log file are very fast, there will not be much gain in attempting to recover the process at an early point. Quality assurance is one of the things that takes a lot of the loading time - especially the counting of the number of new lines in the database tables takes much time. Since this is the final audit check and the transaction where the changes to the database are made has to be rolled back, if there is an error in this check, the new records have to be recounted at recovery. Therefore the process is restarted from scratch each time there is an audit number mismatch. If an audit number mismatch occurs in the restarted process as well, the process should not be restarted again and again. After two attempts, the abandoned and the AUBA tool administrator should be notified of the mismatch, so the problem can be handled manually. For instance, if the audit mismatch is caused by a lack of storage space, the administrator will have to make more storage space available, before the ETL process should be restarted.

If the second attempt at processing the same log file gives exactly the same audit number mismatch, the problem could be caused by an error in the source code of the AUBA tool, that should be located and corrected, before the process is restarted. The information about which audit numbers do not match tells the administrator how far the process was when the error occurred, and thus can help the administrator locate the error in the source code. New log files should not be loaded into the database until the error has been corrected, but the AUBA tool can still be used for analysis of the data loaded before the error, because the database will look as it did before the failed load. When the error has been located, it is necessary to consider if it only has an effect on the log file in question of if it is necessary to reload all previous data after the error has been corrected, because the error has caused undetected errors in the data that are too important to the analyses to be ignored. In most cases this would not be necessary, since the error has not been caught when processing the previous data.

7.3.2 Process Not Completed

In the event of a power outage or the like, the ETL process will be disrupted without having the opportunity of reporting the problem first. Therefore, it is important to have a process outside the AUBA tool that can check up on the AUBA tool. This could be combined with a cron job that checks for new log files and starts the incremental loads. If it notices that there is an audit record that has not been completed, but no Java process is running, it may be because there has been a power outage or because the program has ended unexpectedly. In this situation, error handling should restart the process that was interrupted. Another good idea is to have a cron job that checks if a java process is stalled. In the event of a stalled process, it should be killed and restarted if the changes to the database have not been commited.

CHAPTER 8

User Interface

The AUBA tool must be easy to use. The AUB staff should not have to take a database systems course to get results with the AUBA tool. A graphical user interface has been implemented, so the users will not have to know the underlying structure of the database or anything else about databases for that matter. In the graphical user interface the users can choose which information they need from a list of options, and this information will appear on the screen as a chart. The graphical user interface has been implemented in J2EE and JSP and will be installed on one of the servers at AUB. This way the users can access the tool from their own machines using a web browser instead of having to install a program on their machines. The Servlet runs on the server, and the user interacts with it using HTTP requests and responses, and the HTML generated by the Servlet is displayed in the user's browser.

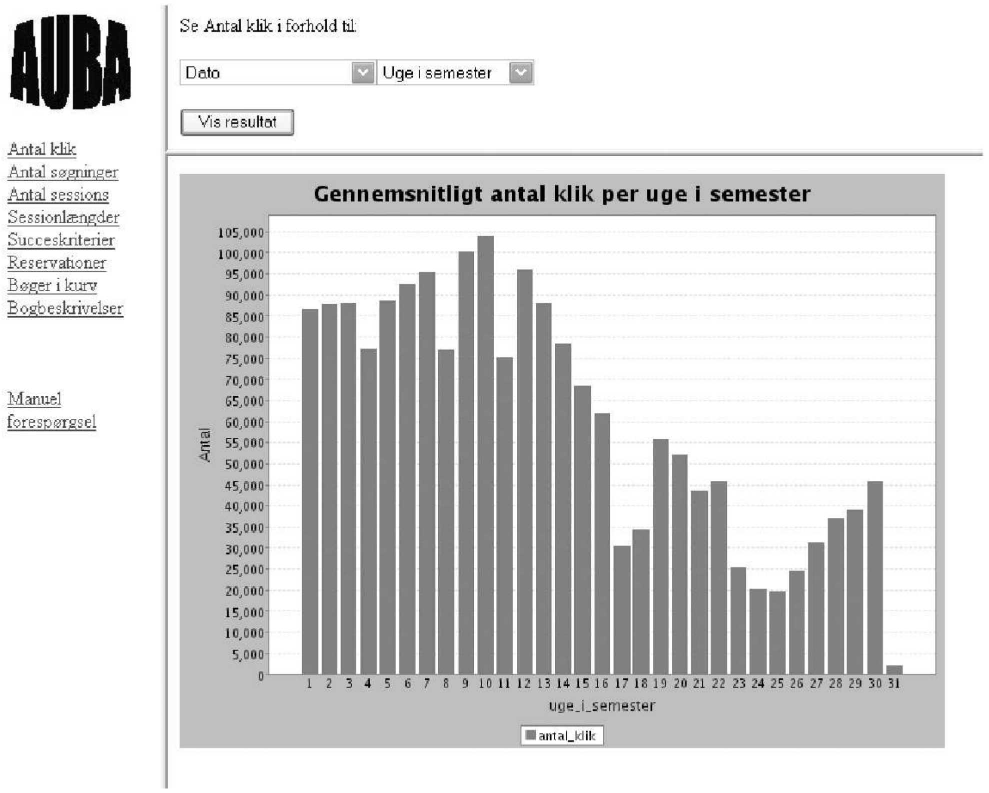


Figure 8.1: Screen shot of AUBA graphical user interface

There are three simple steps to go through to get an analysis result. First, the user selects the measure that he wants information about, for instance number of clicks. Second, he needs to choose how the information about the measure should be grouped, for instance according to date. The third step is to select the specific attribute to use to group the measure by. This could be week of semester if the user wants to see the distribution of clicks during the course of the semesters.

Figure 8.1 shows a screen shot of the simple user interface for the AUBA tool. In the first step, the user chooses a measure from one of the seven possibilities in the left frame. The two last steps take place in the top right frame. The options in the second step depend on the measure that the user chooses in step one. All measures can be analyzed by date and time of day. "Clicks" and "sessions" can further be analyzed by the different page attributes. Searches and success criteria, including book descriptions, basket saves and reservations separately, can also be analyzed by search type and session.

When the analysis parameters have been chosen, a query is generated based on these. This query is sent to the database, and the chart showing the results is generated automatically from the result set using Cewolf [Cewolf, 2004], which is an open source extension of JFreeChart [JFreeChart, 2004].

The following concrete example shows how a user would typically use the graphical interface to get information with the AUBA tool. The prototype has a menu at the left where the user can choose which measure to find information about. If the user chooses for instance “Antal klik”, he can get information about the number of clicks in Auboline. When the user has chosen a measure in the menu, the second choice appears in the top right frame as can be seen in figure 8.2.

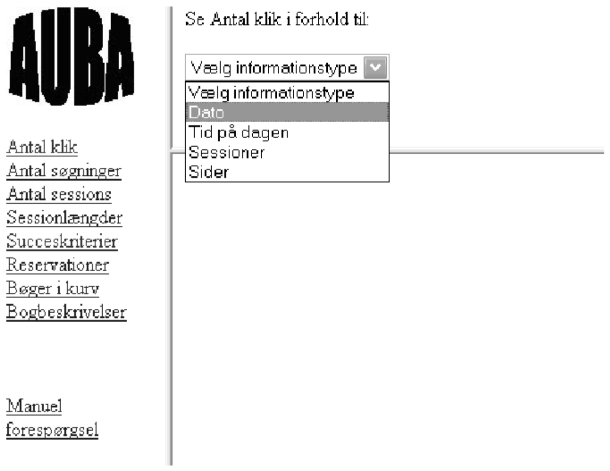


Figure 8.2: Selection of dimension.

Now the user has to choose if he wants to analyze the number of clicks with respect to date, time of day, page or session characteristics. Choosing “Dato” in the first drop down box makes a second drop down box appear. If the user wants to see the distribution of clicks per week of semester, he can choose “Uge i semester” in the second drop down box as illustrated in Figure 8.3. Clicking “Vis resultat” after these choices will send a query to the database and return a chart of the result in the bottom right frame (see figure 8.1).

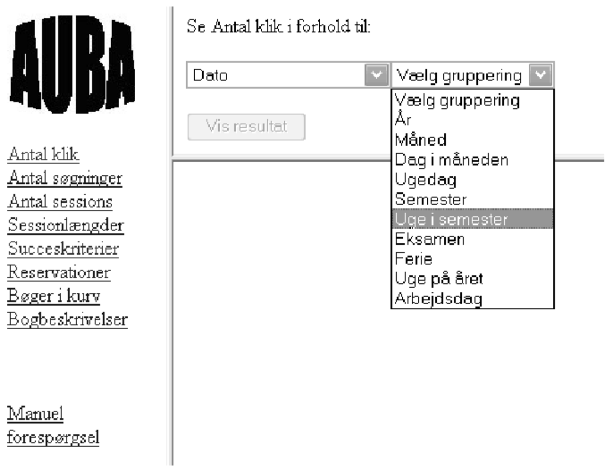
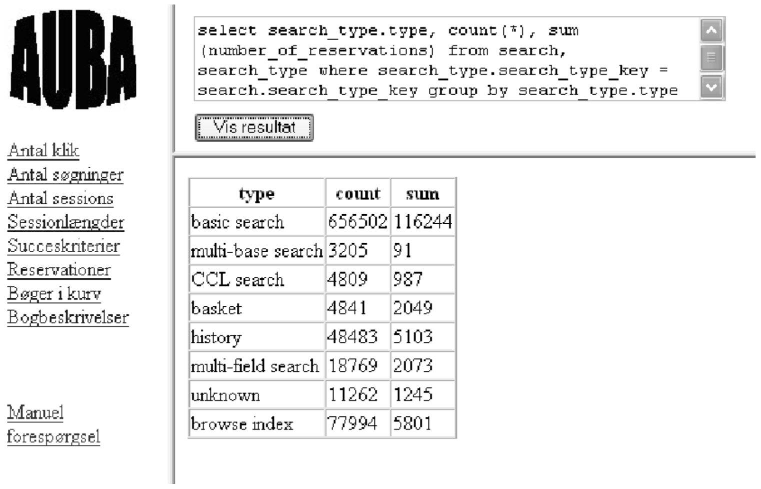


Figure 8.3: Selection of attribute.

The graphical user interface is made as simple as possible to make it easy to understand and use. The predefined queries are all of the same type because this limits the number of options that the user has and thereby simplifies the process of getting to a result. With knowledge of the tables and materialized views in the data warehouse, the advanced users can enter any query if he chooses the manual query option at the bottom of the left frame. The result of a user defined query is not guaranteed to have the format that is required to show a chart of the result, so instead the result is shown in a table as illustrated in Figure 8.4. The aesthetics of the graphical user interface has not been a priority. In a possible commercial version of the AUBA tool this would have to be improved.



```
select search_type.type, count(*), sum
(number_of_reservations) from search,
search_type where search_type.search_type_key =
search.search_type_key group by search_type.type
```

type	count	sum
basic search	656502	116244
multi-base search	3205	91
CCL search	4809	987
basket	4841	2049
history	48483	5103
multi-field search	18769	2073
unknown	11262	1245
browse index	77994	5801

Figure 8.4: Screen shot of the manual query part of the AUBA graphical user interface

CHAPTER 9

Query Performance

If the queries generated through the graphical interface access the base tables directly, the user will have to wait several minutes in many situations before the result appears on the screen. This is due to the amount of data in the database. If the users of the AUBA tool have to wait too long for the results they will loose interest in using the tool. Therefore it is necessary to have interactive response times for the queries that are generated by the graphical user interface. The two methods that are used to improve query performance in the AUBA tool are summary tables and indexing. These methods are discussed in the following.

9.1 Summary Tables

The results that are most often requested through the AUBA tool are either counts, sums or averages. These aggregate queries can be sped up considerably by using summary tables. Summary tables are materialized views where counts or sums are pre-calculated and therefore faster to access [Harinarayan et al., 1996]. For instance, if a user requests the average number of clicks in Auboline per week day, it is useful to have a summary table that contains the number of clicks per date. This summary table can be used to calculate the average number of clicks per week day. For the data set of the initial load this kind of query takes approximately one minute to join 3.9 million lines in the page_event table with 393 lines in the date table and group the result by the week_day column. It is necessary to access 86,000 disk pages to get the result. On the other hand, the summary table contains only 393 lines. Each of these lines contain the total number of clicks for a particular date, and grouping the summary table by the week_day column only takes 30 milliseconds, because it is only necessary to access 32 disk pages.

The down-side of using summary tables is that they slow down the ETL process, because they need to be updated when new information is loaded into the base tables. Having a lot of summary tables also take up a lot of storage space. Therefore, all possible summary tables should not be created to make sure that the queries run smoothly. It is a good idea to create a summary table when it represents data that is queried often and when it takes up less storage space than 25 % of the base table data that it represents or an existing materialized view that can be used to find the same result [Corey et al., 2001].

9.1.1 Basis for Selection of Summary Tables

In order to choose which summary tables to include in the data warehouse, it is important to look at the types of queries that will be used to access the data. The queries generated by the graphical user interface all deal with the measurements listed below.

- Number of clicks
- Number of searches
- Number of sessions
- Length of sessions
- Number of book descriptions read
- Number of books saved in basket
- Number of reservations

Because these measures are central in the graphical user interface, they are also central in the choice of summary tables. But there are many possible summary tables that represent these measures in different ways, so it is necessary to find out which kinds of queries will be used in order to decide which summary tables to include in the data warehouse. As mentioned, a summary table is a materialized view of an aggregate query. Aggregate queries can be grouped by one or more of the dimension attributes of the data warehouse. So the possible summary tables for the two star schemas in the data warehouse are the aggregate queries grouped by all possible combinations of dimension attributes for each star schema. Since there are 44 different attributes to group by in each star schema, there are $1,76 * 10^{13}$ candidate summary tables for each star schema if all combinations of attributes are taken into account (1 grouped by all 44 attributes, 44 grouped by 43 attributes, 946 grouped by 42 attributes, etc.). This is a total of $3,52 * 10^{13}$ possible summary tables for the whole data warehouse. Of course, all of these candidate summary tables should not be materialized. It is necessary to find a method of selecting which aggregate views to materialize in order to achieve acceptable query performance while still keeping storage space and time to update the summary tables after new loads at an acceptable level. Two questions need to be answered in order to make a good choice on which summary tables to use.

1. Which queries must have good performance?
2. Which summary tables are dependent on each other, i.e. which summary tables are not as important to use, because the same queries can be answered in acceptable time using other summary tables?

To answer the first question, it is necessary to take a look at which queries can be executed through the graphical user interface. There are two types of queries. The first type is the predefined queries that are executed when a user picks measure, dimension and attribute and clicks to get the result in a chart as described in Chapter 8. Since the users can only choose a single of a limited number of attributes to group the result by, this type of query is an aggregate query grouped on a single attribute. The second type of query can be any query. Since the users can enter the query text, there is no pattern that can be predicted, but it is reasonable to assume that each unique user defined query will not be used as often as each predefined query. Therefore the predefined queries should be prioritized when deciding which views to materialize.

The second question can be answered by looking at the result sets of the different queries and finding out which result sets are subsets of other result sets. This can help to limit the number of summary tables to use. For instance, if there is a summary table that contains the total number of reservations per search type per date, this summary table can also be used to find the total number of reservations per search type. This is because the result set of the latter query can be derived from that of the former. Similarly, the total number of clicks during working hours can be derived from the total number of clicks per hour, since it is simply the sum of the number of clicks where the hour is from 8 through 15. The knowledge of which candidate summary tables are dependent on each other is utilized in the Greedy algorithm [Harinarayan et al., 1996]. This algorithm can be used to decide which views to materialize as discussed in the following section.

9.1.2 The Greedy Algorithm

As explained, summary tables are very efficient for improving query performance. If every possible summary table is materialized, query performance would be very good because every query would just be a lookup in one of the summary tables. But the downside to having materialized summary tables is that they take up storage space and it takes time to update them when the base tables are updated. Therefore it is not a good idea to materialize all possible summary tables. The best compromise is to materialize enough summary tables to improve query performance to an acceptable level and keeping storage space and ETL performance at an acceptable level as well.

The Greedy algorithm in Algorithm 4 can be used to decide which of all possible views are most beneficial to materialize. The algorithm calculates the benefit of materializing each view. The benefit of materializing a view is the difference between the number of records that are necessary to access before and after materialization of the view. In this calculation the dependencies between views are taken into account by adding the benefit of views that can be computed from other views. When all benefits have been calculated, the view with the highest benefit is chosen for materialization. After this, a new round is started where all benefits are recalculated. This is done because some of the benefits may have changed because the views have received benefit from the prior choice for materialization. The view with the highest benefit is again chosen for materialization. This is continued until every view that has a benefit higher than zero has been chosen for materialization. The result is a list of views in the order in which they should be materialized. This list can be used to materialize views in the specified order until an acceptable balance between query performance and ETL performance has been reached. Materializing more views improves query performance but slows down ETL, so the choice of how many views to materialize will have to be a compromise.

Algorithm 4 The Greedy Algorithm [Harinarayan et al., 1996]

```
GREEDY()
  S ← {top view}
  for all i = 1 to k do
    select that view v not in S such that B(v,S) is maximized
    S ← S union {v}
  end for
  resulting S is the greedy selection
```

It is necessary to calculate the number of records in the result set of each candidate summary table in order to be able to calculate the benefits. As mentioned, the number of possible summary tables is huge, so it is not possible to calculate the number of records in each possible summary table. A possible solution to the problem is to estimate the number of records in the candidate summary tables. This way the long time it takes to query the database to get the number of records in the results of the candidate summary tables can be avoided. A method of estimating storage space of different subsets of the set of possible summary tables is suggested in [Shukla et al., 1996]. The paper also suggests limiting the set of candidate summary tables to have a maximum of one attribute from each hierarchy in each summery table. This would decrease the maximum number of attributes in a candidate summary table to 21, but there are still 44 different attributes, so the set of candidate summary tables is still very large. This is because a big part of the date and session dimension attributes cannot be organized in hierarchies.

Another method of solving the problem of limiting the time used to estimate the sizes of candidate summary tables is to limit the number of candidate summary tables substantially by utilizing knowledge about the data and how it will be used. It is known that most of the dimension tables are very small. The search type and page dimensions each have less than 100 records and the date dimension has less than 500 records. The dimension tables that are not as small have the potential of being smaller because it is not likely that all of their attributes will be used as group by attributes. The time of day dimension has 86400 records, but it is not necessary to have summary tables where the measures are grouped by second. Grouping by hour is sufficient because the time of day is the interesting part - not the exact minute or second. Grouping the time dimension by hour still preserves information about period of day and working hours. This leaves only 24 aggregated records in the time of day dimension. The session dimension is the largest dimension table. It has approximately four hundred thousand records. In many ways, the session dimension is similar to a fact table. The attributes pages_in_session, book_descriptions_in_session, books_in_basket_in_session and reservations_in_session are additive and are therefore like measures and the outriggers to the page, date and time of day dimensions are like foreign keys from a fact table to dimension tables. But the session dimension also has a lot of dimension-like attributes such as browser and referrer. However, the outriggers and the measure-like attribute are the ones from the session dimension that are of interest in the GUI, so the session attributes will not be grouped with the attributes from the page event and search fact tables. Therefore the simplest solution is to make three schemas when deciding which combination of attributes to group by in the summary tables. The first schema is the page event star schema, with the exception that the session table is removed and the group by attributes are the keys from the page and date dimensions and the hour attribute from the time of day dimension. The search schema looks like the page event schema, except that there is no page key, but a search type key in stead. Since the session information has been removed from the two star schemas, there is a third schema that contains all the session information that is of interest. It has four group by attributes - start_date_key, hour from start_time, first_page_key and last_page_key.

In the following sections, these three parts of the data warehouse will be analyzed and the Greedy algorithm will be used to decide which summary tables to include in the data warehouse.

Page Event Summary Tables

Each possible summary table is a combination of attributes from the dimensions of the page event star schema. The session dimension is not considered at this point because it will be looked at separately. To limit the enormous amount of combinations of attributes, only the most granular attribute that will be used from each of the remaining three dimensions is considered. The most granular attributes of both the page and the date dimension (page function and day, respectively) are of interest, so these dimensions can be grouped by their keys. But the time of day dimension will be grouped by the hour attribute, because minute and second information is not of interest. Figure 9.1 shows the views that will be considered to materialize. The name of each view is an abbreviation of its group by attributes. Thus, dpt is a view that is grouped by date_key, page_key and time_of_day.hour. The numbers in parentheses below each view name is the number of records in the view. These numbers are used by the Greedy algorithm to establish the benefit of materializing the different views.

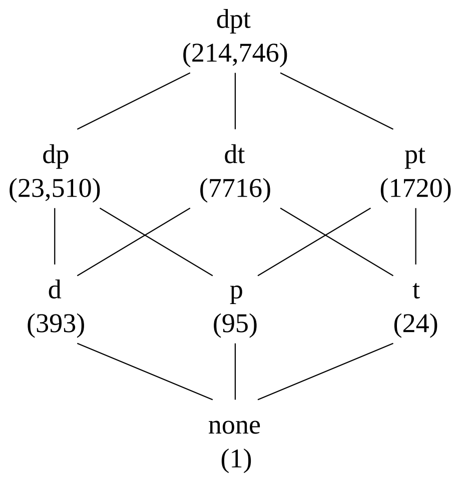


Figure 9.1: The combinations of page event attributes and the sizes of each of the aggregated views.

Algorithm 4 is the Greedy algorithm. In each round, it calculates the benefit of materializing each view as the difference between the number of records that were necessary to fetch to calculate the view and its dependent views before and after materialization of the view. The result is a prioritized list of all the views that would be beneficial to materialize. The root view of Figure 9.1 should be materialized because all the views are dependent on it, and after that a number of views from the prioritized list should be materialized. The Greedy algorithm has been implemented to find out which views are most beneficial to materialize for the AUBA tool, and the source code can be found in Appendix D. The Greedy algorithm does not consider frequencies of use of the different views, so in its original form, it assumes that each view is used an equal number of times. Since it is predicted that most queries will be single-attribute queries, this has been taken into consideration in the implemented Greedy algorithm. The implemented Greedy algorithm assumes that each single-attribute view is used ten times more often than the multi-attribute views. This is done by multiplying the sub-benefit that relates to single-attribute views for each view by ten. For page event summary tables, this customization of the Greedy algorithm changes the benefits but the order in which the views should be materialized is the same.

A method of choosing the right number of views to materialize is to materialize one view at a time until the desired balance between query performance and storage space has been reached.

Choice	Group by attributes	Benefit
1	page_key, time_of_day.hour	4,473,546
2	date_key, time_of_day.hour	2,277,330
3	date_key, page_key	191,236
4	date_key	73,230
5	time_of_day.hour	16,960
6	page_key	16,250

Table 9.1: Greedy result for the page event schema without the session dimension

The order in which the page event views should be materialized and the benefit of each materialization is listed in Table 9.1. The execution time has been tested with one single attribute query for each attribute. When testing the execution time of different queries it is learned that single attribute group by queries take up to five minutes to execute. This cost will be even worse when more data is added to the database, so it is reasonable to use summary tables. Because the dimensions are small, it is possible to achieve good response time even though only a few of the views are materialized. The response time should be better than needed so it will still be good if there is much load on the machine and it will not take up too much processor power from other processes on the machine. Therefore the acceptable response time for each of the test queries is set to one second. The summary tables are also fairly small due to the small dimensions and therefore query execution time is improved greatly already at materializing of the top level view (dpt). The single value group by test queries now take up to five seconds to execute. This is a great improvement, but it can be even better. When materializing the two first priorities on the list (pt and dt), query execution time drops to below 0.2 seconds for each query in the test set. This is very satisfying, so no more views have to be materialized.

Search Summary Tables

The dependencies of the search attributes and the sizes of the views are shown in Figure 9.2. When the customized Greedy algorithm is used on these values, the resulting materialization order is as shown in Table 9.2. This order is not exactly the same as it would be if the Greedy algorithm had not been customized. Before customization, the Greedy recommendation was to materialize all the two-attribute views before the single-attribute views, but because the single-attribute views are used more often, the benefit of materializing the view that is only grouped by date_key is now higher than the ones where date_key is one of two group by attributes.

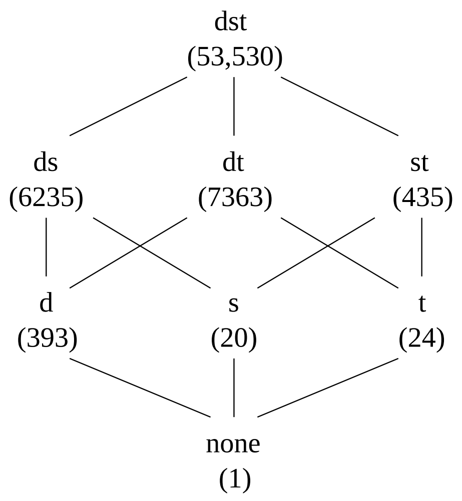


Figure 9.2: The combinations of search attributes and the sizes of each of these aggregated views.

Choice	Group by attributes	Benefit
1	search_type_key, time_of_day.hour	1,114,995
4	date_key	531,370
2	date_key, search_type_key	47,295
3	date_key, time_of_day_key	46,167
5	search_type_key	4150
6	time_of_day.hour	4110

Table 9.2: Greedy result for the search schema without the session dimension

The query performance for the search schema has been tested in the same way as the page event schema. The search schema is very small so even when querying the base tables directly, the longest execution time is only 32 seconds. In this case it is enough to materialize the top view, which brings query execution time down below 0.6 seconds for each query in the test set.

Session Summary Tables

The session dimension has four attributes that have been selected as interesting as group by attributes for the summary tables. These are date_key, time_of_day.hour, first_page_key and last_page_key. The lattice for the session part illustrated in Figure 9.3 is bigger than the two previous lattices, because there is an extra attribute that gives more combinations. When inputting these views and sizes in the Greedy algorithm it gives the result shown in Table 9.3. Again, the customization of the Greedy algorithm has caused some changes in the order of the views on the list. The smaller views have been moved up on the list compared to where they were prior to customization.

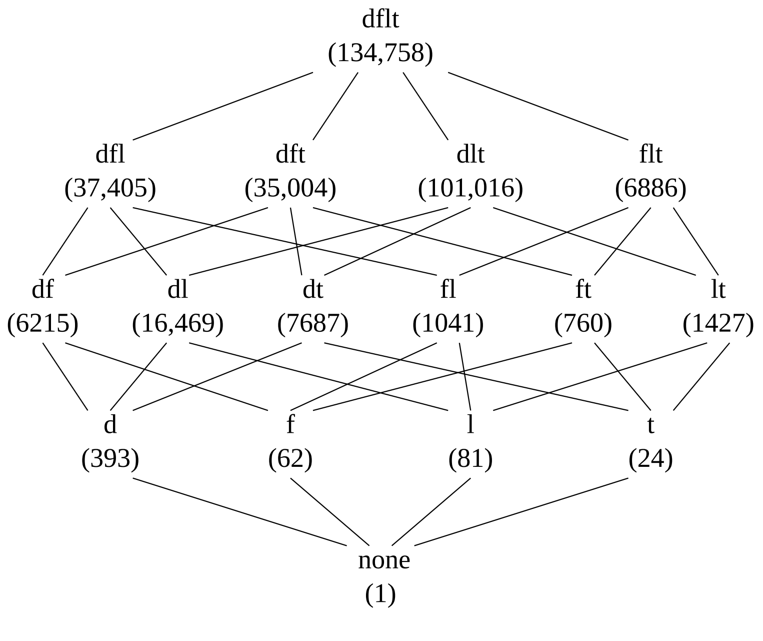


Figure 9.3: The combinations of session attributes and the sizes of each of these aggregated views.

The test set queries take up to two minutes to execute when querying the base tables directly. Materializing the top view (dflt) brings query execution down to a maximum of 13 seconds. After materializing the two first priorities on the list (flt and df) as well, query execution time drops to below 0.2 seconds for each query in the test set.

Choice	Group by attributes	Benefit
1	first_page_key, last_page_key, time_of_day.hour	4,347,648
2	date_key, first_page_key	1,420,683
3	date_key, first_page_key, time_of_day.hour	199,508
4	date_key, first_page_key, last_page_key	194,706
5	first_page_key, time_of_day.hour	121,936
6	last_page_key	68,050
7	date_key	58,220
8	date_key, last_page_key, time_of_day.hour	33,742
9	date_key, time_of_day.hour	27,317
10	date_key, last_page_key	20,936
11	time_of_day.hour	7,360
12	first_page_key	6,980
13	first_page_key, last_page_key	5,845
14	last_page_key, time_of_day.hour	5,459

Table 9.3: Greedy result for the session part of the schemas

Materializing summary tables for the AUBA tool has improved query execution time a great deal. Only seven views have been chosen for materialization and storage space is not a problem, because the summary tables are fairly small. The largest summary table is the dpt table for the page event schema, which has 214,746 rows. The seven summary tables have a total of 425,571 rows which is not much more than a tenth of the rows in the page event fact table. The only problem is the update time of the summary tables. At the moment they are recalculated from scratch each time, and this takes several minutes per view. The PL/SQL functions that handle this can be seen in Appendix C. Optimally the materialized views should be updated incrementally. This is important in to be able to update the materialized views regularly.

9.2 Indexing

Another way to improve query performance is to index the tables heavily. Indexing can help access the tables faster, so to improve performance many of the columns that are constrained upon in queries should be indexed. Indexes are good to use when only a small part of the records of a table is to be accessed but might slow down query time if they are used to access a big part of a table. For instance, if there is an index on the year column of the date dimension table. If the table is queried to find records from 2003 it would not be an advantage to use the index. This is because there are only two different years in the data set. When the index is used to find all dates in 2003, the table will be accessed randomly, and because more than half of the dates in the tables are from 2003, it would be quicker to scan through the whole table. Therefore indexes should be chosen with care. Indexes can also be used on more than one column at a time. For instance, it is possible to have an index on the day and month columns together. Such an index can be used to access the day column or the day and month columns together but does not improve performance on accessing the month column alone because the index is sorted by day. Unfortunately, indexing slows down the loading of data into the database, but this is acceptable compared to the advantages when querying the tables.

CHAPTER 10

User Behavior Survey

To find out if the result of the analyses made with the AUBA tool match how the users see their own behavior in Auboline, a user survey has been carried out. A questionnaire has been constructed (see Appendix F). 32 questionnaires have been answered by borrowers at the main department of AUB. In this chapter, the answers given by the borrowers will be compared to the results of analysis made with the AUBA tool.

It is expected that the same general patterns found with the AUBA tool will also be found in the user analysis, but it cannot be expected to get exactly the same results. There are many reasons for this [Kvale, 1994, Alvesson, 2003]. First, the 32 borrowers that answered the questionnaires represent a very small part of the borrowers whose behavior is reflected in the AUBA tool analysis. Second, many of the answers come from borrowers that were studying physically at AUB on the day of the survey and therefore may represent a more uniform use of Auboline than if all borrowers had been asked. Third, there will always be a difference between how people behave and how they describe their own behavior. Finally, some of the answers may be wrong because borrowers misinterpret the questions or are not very familiar with the jargon used. Suspected misinterpretations of the questions are discussed along with the answers in the following sections.

10.1 Frequency of Use and Entrance

In question 1 the borrowers were asked to estimate the frequency of their use. This question was asked to get a picture of how often the borrowers use Auboline. The answers for this question are depicted in the chart in Figure 10.1.

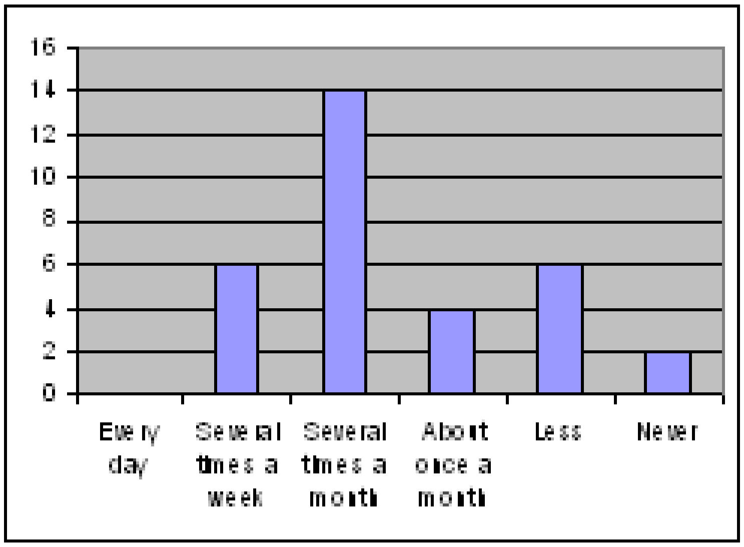


Figure 10.1: Answers for question 1: How often do you use Auboline?

The chart shows that none of the borrowers use Auboline every day. Most of the borrowers use Auboline more than once a month. This result cannot be compared to the AUBA results, since it is not possible to recognize individual users and thereby decide how often they use Auboline.

In question 2 the borrowers were asked how they most often enter Auboline. This is another thing that the AUBA tool cannot give an exact answer for because of the automatic redirect from the Auboline main page to the basic search form. Because of this redirect, it is only possible to establish the origin of sessions that do not start on the start page of Auboline. The answers to question 2 in Figure 10.2 show that almost all users access Auboline through the AUB website. There are also a few users that have Auboline in the favorites of their browser or use a search engine to find Auboline.

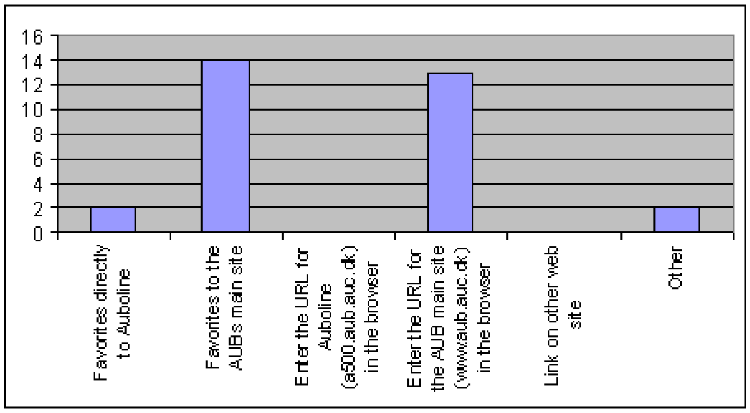


Figure 10.2: Answers for question 2: How do you enter Auboline?

10.2 Distribution of Use

In question 3 the borrowers were asked if they use Auboline mostly during working hours or in their spare time. As the chart in Figure 10.3 illustrates, more than 80 percent of the users use Auboline mainly during working hours and 20 percent use it more in their spare time. This is very close to the average number of clicks in an hour outside working hours, which is 17 percent of the average number of clicks during working hours. Actually, according to the AUBA results, 78.4 percent of all clicks in Auboline occur during working hours.

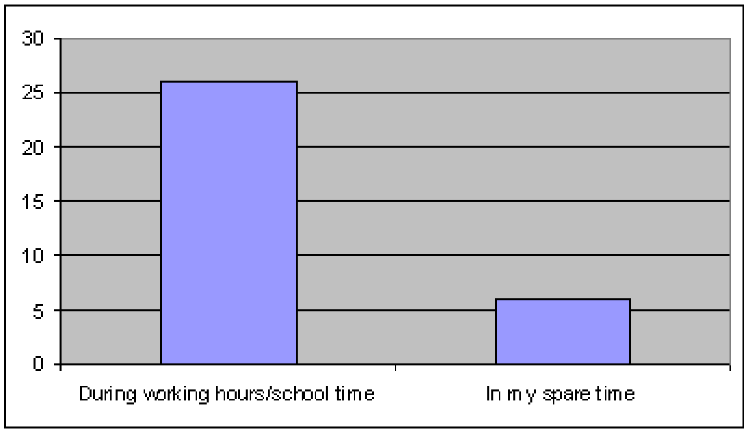


Figure 10.3: Answers for question 3: What time of day do you use Auboline most?

Question 4 is about the distribution of use of Auboline during the week. This has been analyzed with the AUBA tool as well. As illustrated in Figure 10.4, the analysis showed that Auboline is used most in the beginning and middle of the week. On Thursdays and Fridays the use starts to decrease and it is very low in the weekends.

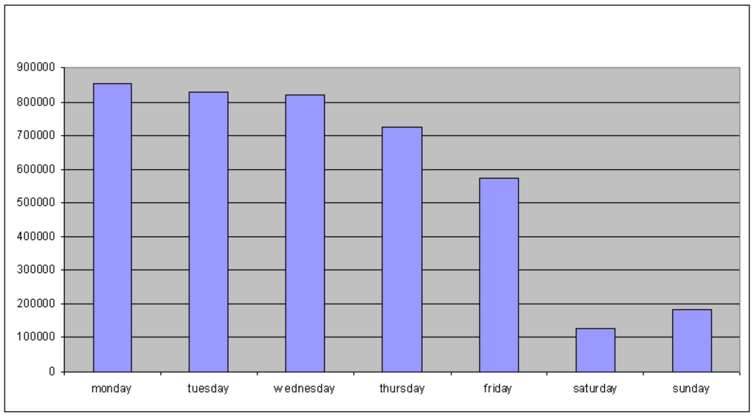


Figure 10.4: Total weekly distribution of activity measured in page events per week day.

When asking the users, the same general pattern formed. As illustrated in Figure 10.5 the users say that they use Auboline most in the beginning of the weeks. But the chart of the user behavior shows that their use drops steeply in the middle and end of the week as well as the weekend. The chart is therefore different from that in Figure 10.4. There are two possible explanations for this, besides the uncertainty that the borrowers do not remember exactly which days they use Auboline most. One explanation is that the survey was conducted on a Tuesday and that the borrowers that are at the library on a Tuesday may often use the library and Auboline in that part of the week, so if the survey had been conducted on a Friday, the pattern might have been that the borrowers used Auboline most toward the end of the weeks. A second explanation for the differences between the two results is the way they have been measured. The AUBA result shows the number of clicks on the different week days and is therefore very precise. If it had been possible to ask the borrowers to assign percentage of use to each day of the week, the result would have been more comparable with the result from the AUBA tool. Instead the question was simplified by not having as many options to assign values to and by asking the borrowers to prioritize the choices instead of estimating the percentage of their use on each week day. Therefore they were asked to rank the four possibilities “beginning of week”, “middle of week”, “late in the week” and “weekend” according to which time of the week they used Auboline most. There may be different interpretations of these values, such that some borrowers interpret “beginning of week” as only Monday, while others may interpret it as Monday and Tuesday.

To be able to put the result of this question in a bar chart, the answers were given points, so if a borrower put a 1 for “weekend”, four points were given to the “weekend” answer. The second ranked answer got three points, the third got two and the fourth got one. This way of counting points for the four possible answers has many uncertainties, because it assumes that the use of Auboline drops with the same amount between the answers with the different ranks. That is, if a user uses Auboline 80 percent in the middle of the week, 10 percent in the beginning, 7 percent at the end of the week and 3 percent in weekends, the points will only show which time Auboline is used most. The points are therefore only precise if the use is distributed with 40, 30, 20 and 10 percent. The result might have been closer to the AUBA result if the users were asked to give a percentage distribution of their use, but this would also have been difficult to estimate. Another reason that the result shows so much action in the beginning of the week is that some users have not ranked the options but has checked only one of the possibilities. For such an answer the checked possibility is given four points and the other zero. “Beginning of week” is the answer that has most often been checked, and therefore it gets a high score.

In Question 5 the borrowers were asked to estimate how much they use Auboline in the beginning, middle and end of the semesters, respectively. The result is illustrated in Figure 10.6. This result is very much like the result from the AUBA tool when the distribution of activity is measured by week of semester. This is illustrated in Figure 10.7, and the similarity of the two results is even more obvious when the activity in Auboline is grouped in three groups of ten weeks each for the beginning,

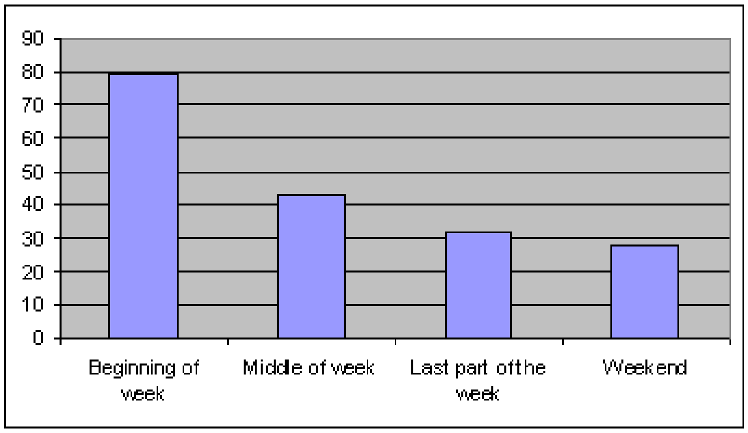


Figure 10.5: Answers for question 4: What time of week do you use Auboline most?

middle and end of the semesters as illustrated in Figure 10.8.

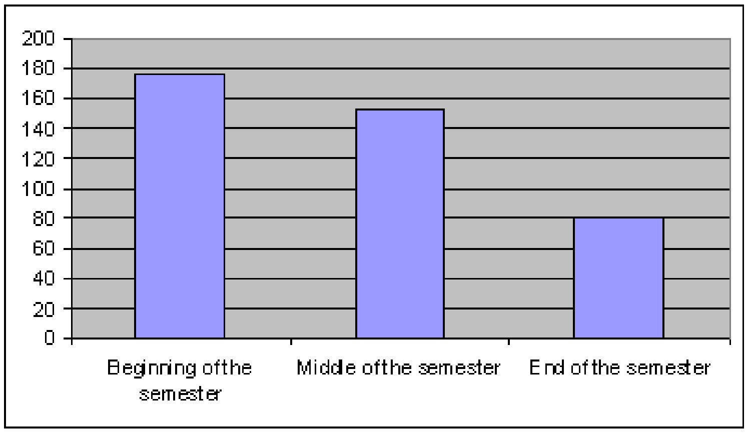


Figure 10.6: Answers for question 5: How does your use of Auboline vary during the semesters?

10.3 Purpose of Use

In Question 6 the borrowers were asked to check which of the central functions they use in Auboline. The answers for this question are collected in Figure 10.9. The result is difficult to compare with results from the AUBA tool because the users were not asked how often they use the different functions. And since it is not possible to recognize borrowers in Auboline there is no method of finding out how many users use each function on a regular basis. Instead it is possible to use the AUBA tool to find out how often each page function is used. The result of this analysis can be seen in Figure 10.10.

10.4 Types of Searches

The borrowers were asked to check the search types that they use in Question 7. As Figure 10.11 illustrates, basic search is the search type that is used most. This is also true according to the AUBA tool analysis, as illustrated in Figure 10.12, but in the AUBA result the difference between the fre-

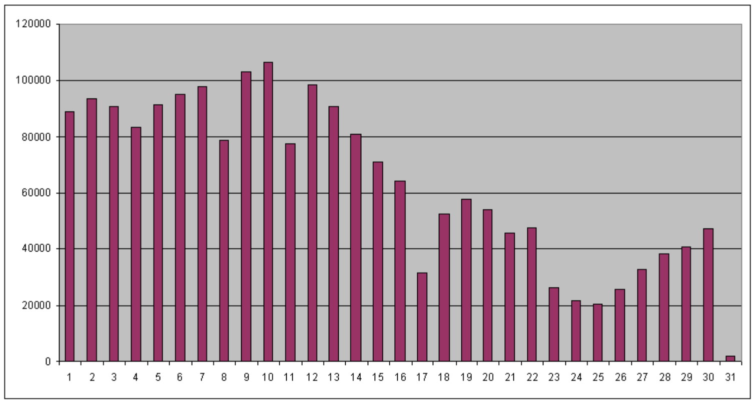


Figure 10.7: Average distribution of activity measured in page events per week of semester.

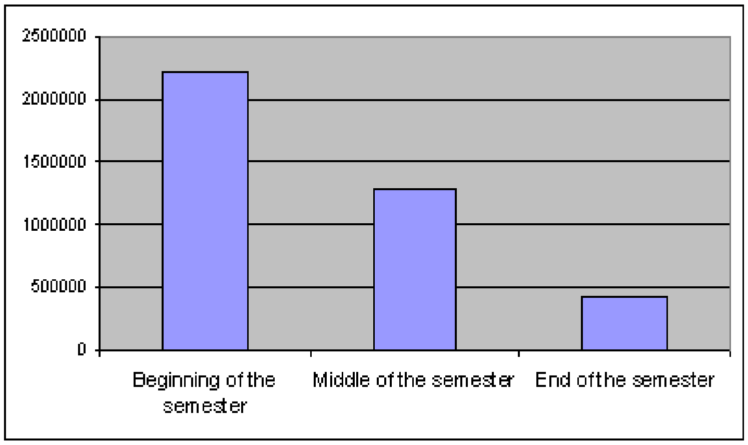


Figure 10.8: Distribution of total activity in Auboline where the semester is split into three periods

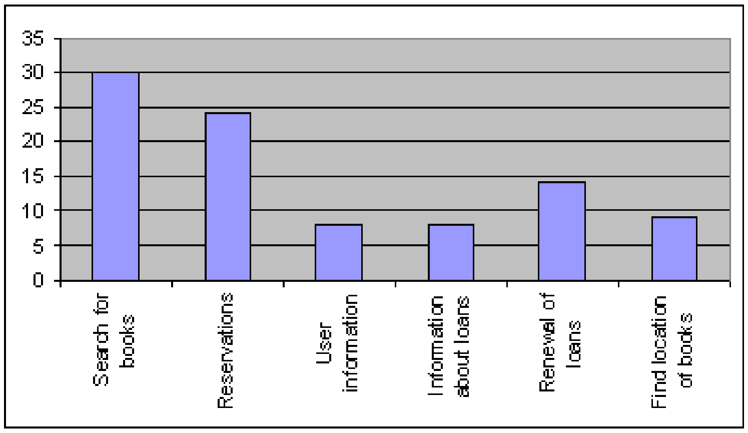


Figure 10.9: Answers for question 6: Which functions do you use most in Auboline?

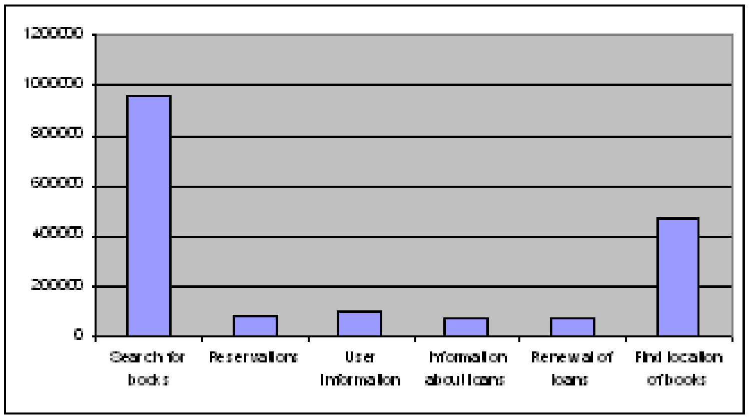


Figure 10.10: The frequency of use of different central functions in Auboline

quency of use of the basic search and the other search types is greater than it is in the survey result. Again, this could be because many borrowers use basic search in most cases, but occasionally use other search types, and therefore checked the other search types in the questionnaire as well. It is also very reasonable to assume that some borrowers do not notice what the search types are called when they use Auboline and therefore check the possibilities that seem familiar in the questionnaire. For instance, five of the borrowers say that they use multi-field search. In the Danish version of Auboline the term that is used for this search type and was also used in the questionnaire can be translated to “search on several fields”. If the borrowers do not notice that the search that they use is called “basic search” and they use it to make searches on different fields, they may think that they often use multi-field search. Index search was checked by more than a third of the borrowers. This could also be because they are not all aware of what index search means, but recognize the fields that are listed in the parentheses (title, author, subject, etc.) because they often make searches on these fields.

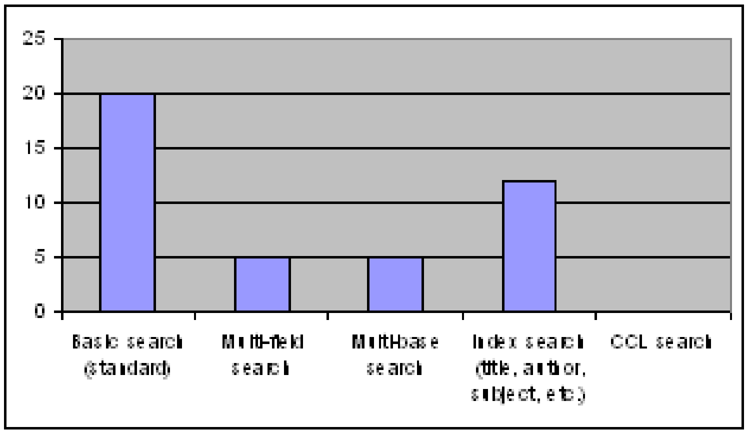


Figure 10.11: Answers for question 7: Which search types do you use?

In Question 8 the borrowers were asled about which search fields they use. Figure 10.13 shows that most borrowers make searches on all fields but that the title and author fields are also very popular. As can be seen in the AUBA result for the same question in Figure 10.14, these three are also the most popular according to the AUBA analysis. The main difference between the two results is that the AUBA results shows a much more frequent use of “all fields” than the other search fields.

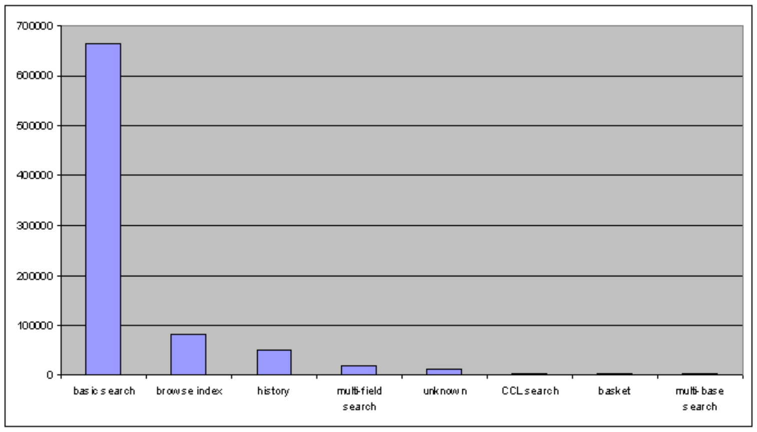


Figure 10.12: The use of the different search types in Auboline measured by the number of times they are each used in the data set.

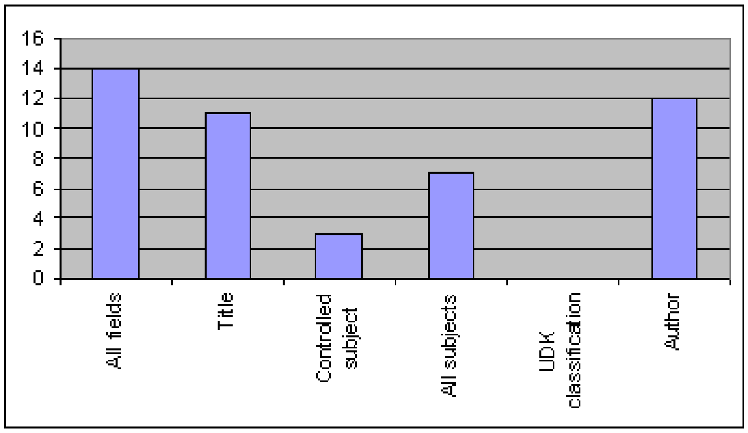


Figure 10.13: Answers for question 8: Which fields do you most often search?

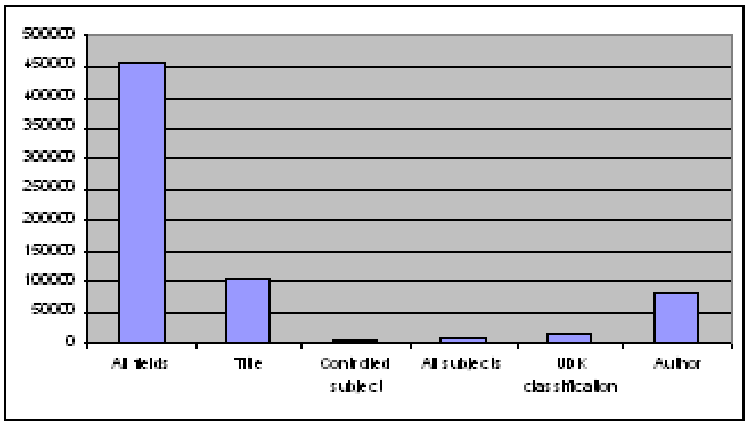


Figure 10.14: Frequency of use of the different search fields

10.5 Use of Special Features

It is known from the AUBA analysis that the special features of Auboline are not used as frequently as the other functions, but it is not possible to use the AUBA tool to find out why they are not used. In Question 9 and 10, the borrowers were asked about this.

Figure 10.15 shows the borrowers' answers for Question 9 about if, why and how they use the history function in Auboline. It is clear that most of the borrowers do not even know that the function exists and that most of the ones that know about it do not use it. According to the AUBA results, the history function is only used in 7 percent of all sessions.

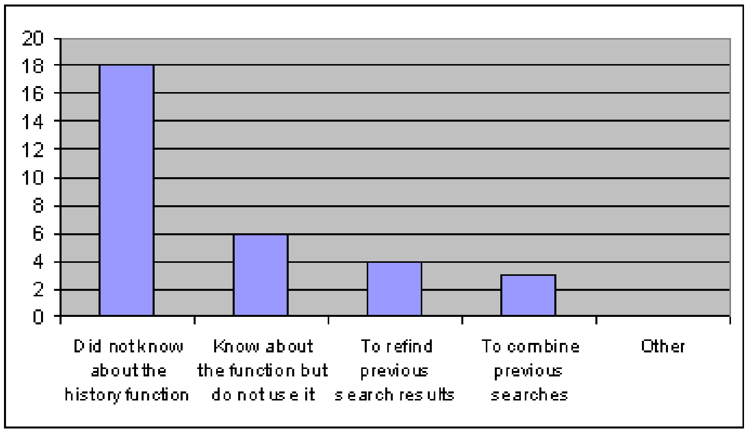


Figure 10.15: Answers for question 9: In which way do you use the history function?

In Question 10, the borrowers were asked about their use of the basket function. According to the AUBA results, this function is only used in 0.8 percent of all sessions. Figure 10.16 shows that five of the borrowers who answered the questionnaires say that they use the basket function. This difference might be because the users only use the basket function in a small part of their sessions.

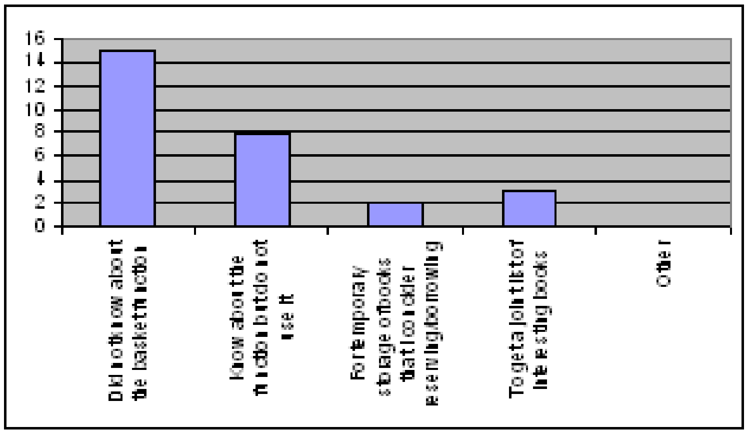


Figure 10.16: Answers for question 10: In which way do you use the basket function?

10.6 Different Types of User Behavior Analysis

As the results from the AUBA analysis and the survey illustrate, there are differences between the results that can be found with different types of methods of analyzing user behavior [Kvale, 1994].

These differences are discussed in the following.

10.6.1 The AUBA Tool

The AUBA tool deals with facts. It uses the actual clicks in Auboline, so it is very precise and can process a large amount of data in a short period of time. The drawback to this method is that the reasons for the results are not possible to find in the bare facts. For instance, the AUBA tool can be used to find out that the basket function is used very rarely, but the reason for this is not available. If the AUBA tool is the only method used for analyzing user behavior in Auboline, it is not possible to know if the users do not know about the basket function or if they do not use it because they have no interest in using it. The AUBA tool is very good for analyzing facts and correlations among facts. It is very good for asking about what happens in Auboline but needs to be combined with other methods there is a need for knowing why.

10.6.2 Interviews

Using interviews to analyze user behavior in Auboline is very different from using the AUBA tool. An interview takes a long time to analyze, but there is a possibility of both asking why and receiving much more information than requested. For instance, it is possible to discuss the basket function with a borrower and find out why she does not use the function and maybe she even has some suggestions about what could be done to make the basket function more attractive to use. Misinterpretations and misunderstandings are not very common in interviews because there is a chance of asking about what is meant and asking extra questions if the answers are not fulfilling. Many of the facts that are easy to find with the AUBA tool can not be found with interviews because it is not possible to ask all borrowers about their behavior in Auboline and they will not be able to tell exactly how their behavior is. But interviews would be a good method to combine with the AUBA analysis to find possible answers to some of the many questions that arise from the AUBA results.

10.6.3 Questionnaire

A questionnaire survey has a combination of some of the characteristics of the AUBA analysis and interviews. The questionnaires are faster to analyze than interviews but not as fast as the AUBA tool. Because the users are asked directly, it is possible to ask them about their own opinions of their behavior and why they behave as they do. There is, however, a risk of misinterpretations because the borrowers answer the questionnaires alone and there is no dialog where misunderstandings can be caught. A questionnaire is good to use in conjunction with the AUBA tool for asking simple questions that are aimed at understanding the reasoning behind the results of the AUBA analysis.

CHAPTER 11

Book Information Extension

There are many possible extensions to the AUBA tool. In its current state the data warehouse only contains data collected from the web log files. To extend the AUBA tool it would be interesting to combine the data from the web log files with data from the different AUB databases, for instance borrowers and books. Possible extensions are discussed in this chapter, but they have not been implemented, so they are not part of the current version of the AUBA tool.

11.1 Possibilities of Borrower Information

Unfortunately it is not possible to recognize borrowers in the log files. If it were possible this could lead to much interesting information, because it would be possible to recognize borrowers from session to session and maybe even include information about what they study and how far along they are in their education. It would be very interesting to see if borrowers that have similar characteristics also behave similarly in Auboline. It will not be possible to include borrower information in the data warehouse in the future, because it is illegal for AUB to save information about how the borrowers use their web sites [Datatilsynet, 2005].

11.2 Possibilities of Book Information

Book information, however, is possible to include in the data warehouse and combine with the data from the log files. In many cases it is possible to identify the books that the borrowers deal with in Auboline, but it is not always possible. For instance, when a borrower clicks on a book in a search result to see the book description, the request URL contains a number that refers to the search (the set number) and a number that refers to the ranking of the book in the search result (the set entry), for instance 5 for the fifth book on the search result list. These numbers are generated by the underlying program and refer to a temporary set of search results that can only be accessed during the duration of the session, so they cannot be used to identify a book. However, if the user clicks to see if the book is available, the request URL contains a so-called “doc_number”. This number can be used to identify the book that the borrower clicked on. For some reason, each book has two different doc numbers. One is used in some situations, such as viewing holdings[1] for a book or putting it in the basket, and the other is used on other pages, such as request for reservation or view reservations. AUB has a database table that can be used to translate between the two different doc numbers to find out exactly which books are dealt with in Auboline.

Adding book information to the data warehouse allows the users to combine the information already available with the AUBA tool with information about book titles, authors, UDK classification numbers and publication year. Below is a few examples of questions that can be answered with the extended AUBA tool when book information is combined with the web log information in the data warehouse.

- Which ten authors have written most of the books that are reserved in the start of the semesters?
- Which UDK numbers do books typically have, if they are found with a search on UDK number?
- Which search types lead to most reservations of books that are published in 2000 or later?

[1]The book holdings page in Auboline gives information about the quantity, stock and placement of the book

11.3 How to Include Book Information

To include book information in the data warehouse, two questions are necessary to answer:

- Which page events deal with books?
- Which books do these page events deal with?

The first question can be answered by looking at the page functions of the page events. There are nine different page functions where the users deal with specific books. These page functions will be referred to as book actions. They are listed below.

- Request reservation
- Reservation
- View reservation
- Delete reservation
- Book description
- Book holdings
- Put book in basket
- View loan status
- Prolong loan

As mentioned, it is not always possible to recognize which books the users deal with in Auboline from the request URL. As an example, a reservation is made with a POST method request, and therefore no information except for the session tag appears in the request URL. But since a reservation can only occur after a request for a reservation, which contains all information that is needed, the doc number for the reservation can be collected from the referrer field of the log line instead. Similarly, the book a user is interested in when reading a book description cannot be recognized from the request URL because the doc number is not present. Therefore the book will always be unknown for book descriptions when the particular log line is processed. But if the user puts the book in the basket afterwards or views book holdings, the doc number appears in this request URL and it is possible to go back and change the information about the book description when the doc number is known. In 66 % of all cases, the book description page function is followed by a page function that enables recognition of the book. As can be seen in Table 11.1, the book can be recognized from the request URL for all other page functions.

Page function	**Book recognized**	**How recognized**
Request reservation	Always	Request URL
Reservation	Always	Referrer
View reservation	Always	Request URL
Delete reservation	Always	Request URL
Book description	66 %	Later in session
Book holdings	Always	Request URL
Put book in basket	Always	Request URL
View loan status	Always	Request URL
Prolong loan	Always	Request URL

Table 11.1: Availability of book information for book functions that deal with books

As mentioned above, the specific books that are dealt with in these book actions can be found using the set numbers and doc numbers in the request URLs. The AUB book database should be converted into a book dimension table that contains the attributes book_key, isbn, title, author, udk and year. It is necessary to have a conversion table to translate each doc number into the book key that references that specific book in the book dimension table.

The book dimension cannot simply be added to the page event star schema to include book action information in the data warehouse. It is only a small part of the page events that are book actions (nine out of 74 page functions), and some page events can actually represent more than one book action. This is because it is possible to put more than one book in the basket or prolong all of a user's current loans with just one click. Therefore it is necessary to model the book actions with a new star schema. In this schema the book actions will be represented in the fact table, which refers to the book dimension table along with the date, time of day, session, search type and page dimension tables. The page dimension table is used for this star schema even though only nine of the records will be referred. This is done to make it clear that even though there are not as many possible values of pages that are book actions, it is still the same type of information that is wanted as for the page event fact schema. The book action fact table will also refer to the audit dimension that contains information about the load that each book action belongs to. The book action star schema is depicted in Figure 11.1.

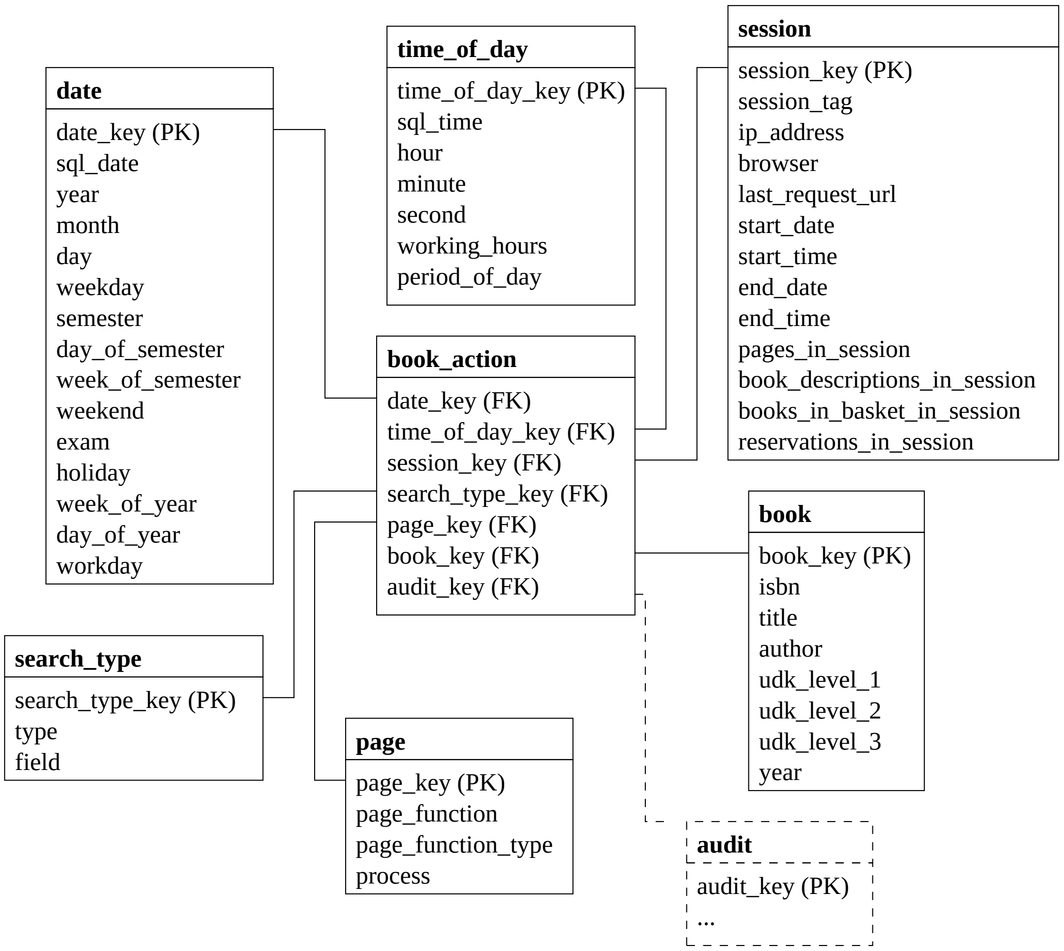


Figure 11.1: Relational representation of the book action star schema

11.4 Updating of New Books

If the book dimension table is created from the AUB book database before the AUBA tool is taken into use, it will be static and will not automatically be updated when new books are added to the AUB database. This is a problem because all books, including new books that are added to the AUB collection later, can be found with Auboline. When the AUBA tool has been in use for a while, a book action may appear that refers to a book that is not in the conversion table and therefore not in the book dimension, either. In this case it is necessary to get information about the new book from the AUB database. A possible way to get this information is to use a web service that has been implemented at AUB. This web service uses the SOAP protocol to allow access to the information in the databases from outside users without direct access to the databases. With this web service it is possible to query the AUB book database to find all information about the book that has the new doc number found in the log file. This information should be added to the conversion table and the book dimension table, so it is available the next time the book shows up in a log file.

11.5 Future Work

When book information is integrated with the AUBA tool as described above, it is also necessary to extend other parts of the AUBA tool. For instance, it is necessary to add information about book actions and books to the audit dimension and extend error handling to also handle this data. The other kinds of quality control described in Chapter 7 should also be extended to include book actions and books.

An additional part of the AUBA tool that should be extended after integrating book actions is the summary tables. Query performance should be measured for the book action star schema and views should be materialized if query performance does not match expectations. This should be done in the same way as for the page event and search star schemas.

CHAPTER 12

Conclusion

In this project a web usage mining tool has been designed and implemented. The result is a tool that can assist analysis of the user behavior in Auboline.

Analysis of the log files and experimentation with the different functions of Auboline has lead to knowledge about the possibilities of the analysis of the data. To store the information from the log lines in a logical structure that is understandable and easily accessible, the data in the database has been logically structured in two star schemas. The structure of the page event star schema is common in data webhouses. It enables analysis of sessions and page functions as well as variations in activity according to different time and date parameters. The search star schema is structured specifically for Auboline, because searches are the central part of the system. With the search star schema it is possible to make analyses of the use of different types of searches and relate the number of book descriptions, basket saves and reservations to the specific searches that lead to the activity. It is possible to extend the data webhouse schema whenever there is a need for new types of analysis or new information becomes available.

As the main part of the implementation, the post-processor has been implemented to handle the ETL processes. It transforms the pure text input files to fit the format of the data webhouse. The performance of the post-processor is satisfactory due to the limited amount of input/output activity between the java program and the database. Extensive work has been done to assure that the data flows through the ETL processes as expected and that the database will not contain flawed data. Each incremental load is done in a single transaction to assure that the data is either fully loaded if there are no problems or not loaded at all if there is an error, such as a power outage, so the process can be restarted from scratch when the problem has been fixed.

Part of the motivation behind the implementation of the AUBA tool was to enable the AUB staff to analyze the success of the book recommendation system to be integrated with Auboline. Therefore one of the goals of the project has been to prepare the AUBA tool for analysis of the use of book recommendations. The dimensional database schema has been designed in a way that the recommendation service usage data will easily fit into. Unfortunately it has not been possible to analyze the use of the recommendation service since it has not yet been integrated with Auboline.

A simple graphical user interface has been implemented to make sure that the AUB staff can find results quickly without knowing the underlying structure of the database. Point-and-click can be used to find the results of predefined queries while advanced users can query the database manually for more complicated requests. The AUB staff can use the results that can be obtained with the graphical user interface to gain knowledge about the behavior of the users of Auboline.

The outriggers from the session dimension to the date, time of day and page dimensions have improved query performance, but the most substantial query performance improvement has been achieved by using summary tables in the AUBA tool. This is a very important improvement because the users of the AUBA tool will most likely loose interest if they have to wait several minutes for each requested result. The materialized views have been hand coded because PostgreSQL dost not include the option of materializing views. Updating the materialized views takes a long time because they are recalculated from scratch. This is a temporary solution that is important to improve. It will be much faster to update the summary tables incrementally instead of recalculating them.

The user survey was conducted to illustrate the difference in the kind of information that can be gained with different methods and to compare the borrowers' perception of their behavior in Auboline with the results from the AUBA tool. Each method has its own advantages and drawbacks, and therefore it is a good idea to use a combination of several methods of gathering information to get a more clear picture of what is going on. For instance, it is a good idea to make a thorough analysis of the user behavior in Auboline with the AUBA tool and then interview the borrowers about why they behave the way they do.

Because the AUBA tool has been developed specifically for AUB it can give the staff analysis results that are targeted directly on the structure and content of their system. The AUB staff that have been involved in the project have shown a lot of interest in the AUBA tool, because it gives them information that they have not previously been able to get. Furthermore, they find the future possibilities of the tool very exciting.

There are many possibilities of improving and expanding the AUBA tool. One of the very interesting possibilities of expanding the AUBA tool is to make a part of the system analyze user behavior in real-time to categorize the users of Auboline while they are still on the web site. This can be used to make an adaptive environment in Auboline where the users are aided in their search for books. For instance, an adaptive search tip feature could be implemented.

The possibilities of using the AUBA tool to find interesting information about the use of Auboline will be further increased when book information is included in the implementation. There are many possibilities of expanding the AUBA tool to get even more information for different analyses. The technical staff at AUB are interested in developing the AUBA tool further after it has been taken into use and usage tests have given them an idea of which extra features will be interesting to explore.

Bibliography

[Aleph, 2004] Aleph (Current as of 30 June 2004). Aleph web site. Internet, `http://www.aleph.co.il`.

[Alvesson, 2003] Alvesson, M. (2003). Beyond Neo-Positivists, Romantics and Localists - A Reflexive Approach to Interviews in Organization Research. *Academy of Management Review*, 28(1).

[Andersen et al., 2000] Andersen, J., Giversen, A., Jensen, A. H., Larsen, R. S., Pedersen, T. B., and Skyt, J. (2000). Analyzing clickstreams using subsessions. In *Proceedings of the 3rd ACM international workshop on Data warehousing and OLAP*, pages 25–32. ACM Press.

[Apache, 2004a] Apache (Current as of 30 July 2004a). Apache http server: Log files. Internet, `http://httpd.apache.org/docs/logs.html`.

[Apache, 2004b] Apache (Current as of 30 July 2004b). Apache website. Internet, `http://httpd.apache.org/`.

[AUB, 2004] AUB (Current as of 30 July 2004). Aub website. Internet, `http://www.aub.auc.dk`.

[Auboline, 2004] Auboline (Current as of 30 July 2004). Auboline. Internet, `http://a500.aub.auc.dk`.

[Cewolf, 2004] Cewolf (Current as of 7 December 2004). Cewolf - chart enabling web object framework. Internet, `http://cewolf.sourceforge.net`.

[Corey et al., 2001] Corey, M. J., Abbey, M., Taub, B., and Abramson, I. (2001). *Oracle8i Data Warehousing*. McGraw-Hill Companies, first edition.

[Datatilsynet, 2005] Datatilsynet (Current as of 12 January 2005). Lov om behandling af personoplysninger. Internet, `http://www.datatilsynet.dk/lovgivning/personoplysninger/lovtekst.txt`.

[Demiriz, 2002] Demiriz, A. (2002). webSPADE: A Parallel Sequence Mining Algorithm to Analyze Web Log Data. *2002 IEEE International Conference on Data Mining (ICDM'02)*, pages 755–758.

[Due, 2004] Due, L. (2004). Analyzing User Behavior in Auboline with Web Usage Mining. Premaster's thesis (INF7), Department of Computer Science, Aalborg University.

[Harinarayan et al., 1996] Harinarayan, V., Rajaraman, A., and Ullman, J. D. (1996). Implementing Data Cubes Efficiently. *In Proceedings of ACM SIGMOD '96*, pages 205–216.

[IETF, 2004] IETF (Current as of 30 July 2004). RFC1413: IDENT. Internet, `http://www.ietf.org/rfc/rfc1413.txt?number=1413`.

[J2EE 1.3, 2004] J2EE 1.3 (Current as of 30 July 2004). Java 2 Platform, Enterprice Edition (J2EE) 1.3. Internet, `http://java.sun.com/j2ee/1.3/`.

[JFreeChart, 2004] JFreeChart (Current as of 7 December 2004). Jfreechart web site. Internet, `http://www.object-refinery.com/jfreechart/`.

[Kimball, 1997] Kimball, R. (1997). A Dimensional Modeling Manifesto: Drawing the Line Between Dimensional Modeling and ER Modeling Techniques. *DBMS Online*, 10(9).

[Kimball and Merz, 2000] Kimball, R. and Merz, R. (2000). *The Data Webhouse Toolkit: Building the Web-Enabled Data Warehouse*. Wiley Computer Publishing, first edition.

[Kimball and Ross, 2002a] Kimball, R. and Ross, M. (2002a). *The Data Warehouse Toolkit: The Complete Guide to Dimensional Modeling*. Wiley Computer Publishing, second edition.

[Kimball and Ross, 2002b] Kimball, R. and Ross, M. (2002b). *The Data Warehouse Toolkit: The Complete Guide to Dimensional Modeling*. Wiley Computer Publishing, second edition.

[Kosala and Blockeel, 2000] Kosala, R. and Blockeel, H. (2000). Web Mining Research: A Survey. *ACM SIGKDD Explorations Newsletter*, 2(1):1–15.

[Kvale, 1994] Kvale, S. (1994). *Interview - En introduktion til det kvalitative forskningsinterview*. Hans Reitzels Forlag.

[Levene and Loizou, 2003] Levene, M. and Loizou, G. (2003). Why is the Snowflake Schema a Good Data Warehouse Design? *Information Systems*, 28(3):225–240.

[Ly et al., 2003] Ly, T. H., Mogensen, J., and Skouboe, K. R. (2003). Building a Business Intelligence System for AUB. Pre-master's thesis (DAT5), Department of Computer Science, Aalborg University.

[Ly et al., 2004] Ly, T. H., Mogensen, J., and Skouboe, K. R. (2004). Building a Business Intelligence System for AUB (second edition). Master's thesis, Department of Computer Science, Aalborg University.

[Pedersen and Jensen, 2001] Pedersen, T. B. and Jensen, C. S. (2001). Multidimensional Database Technology. *IEEE Computer Magazine*, 34(12):40–46.

[PostgreSQL 7.4, 2004] PostgreSQL 7.4 (Current as of 30 July 2004). Postgresql 7.4 documentation. Internet, `http://www.postgresql.org/docs/7.4/static/index.html`.

[PostgreSQL 7.5, 2004] PostgreSQL 7.5 (Current as of 30 July 2004). Postgresql on windows. Internet, `http://techdocs.postgresql.org/guides/Windows`.

[Shukla et al., 1996] Shukla, A., Deshpande, P., Naughton, J. F., and Ramasamy, K. (1996). Storage estimation for multidimensional aggregates in the presence of hierarchies. In *The VLDB Journal*, pages 522–531.

[Srivastava et al., 2000] Srivastava, J., Cooley, R., Deshpande, M., and Tan, P.-N. (2000). Web Usage Mining: Discovery and Applications of Usage Patterns from Web Data. *SIGKDD Explorations*, 1(2).

[W3C, 2004] W3C (Current as of 30 July 2004). World wide web consortium - protocol. Internet, `http://www.w3.org/Protocols/rfc2616/rfc2616-sec5.html#sec5.1.1`.

[Zaharioudakis et al., 2000] Zaharioudakis, M., Cochrane, R., Lapis, G., Pirahesh, H., and Urata, M. (2000). Answering Complex SQL Queries Using Automatic Summary Tables. *Proceedings of the 2000 ACM SIGMOD international conference on Management of data*, pages 105–116.

APPENDIX A

PostgreSQL data definition

A.1 page_event fact table

```
CREATE TABLE page_event (
        log_line_key      INTEGER REFERENCES log_line,
        date_key          INTEGER REFERENCES date,
        time_of_day_key   INTEGER REFERENCES time_of_day,
        page_key          INTEGER REFERENCES page,
        session_key       INTEGER REFERENCES session,
        audit_key         INTEGER REFERENCES audit
);
```

A.2 log_line table

```
CREATE TABLE log_line (
        log_line_key     INTEGER       PRIMARY KEY,
        filename         VARCHAR(19)   NOT NULL,
        log_line_number  INTEGER       NOT NULL,
        ip_address       VARCHAR(15)   NOT NULL,
        ident            VARCHAR(1)    NOT NULL,
        authuser         VARCHAR(1)    NOT NULL,
        date             DATE          NOT NULL,
        time             TIME          NOT NULL,
        timezone         VARCHAR(5)    NOT NULL,
        method           VARCHAR(4)    NOT NULL,
        request_url      TEXT          NOT NULL,
        session_tag      VARCHAR(100) NOT NULL,
        serial           VARCHAR(5)    NOT NULL,
        query            TEXT          NOT NULL,
        protocol         VARCHAR(8)    NOT NULL,
        status           INTEGER       NOT NULL,
        bytes            INTEGER       NOT NULL,
        servername       VARCHAR(15)   NOT NULL,
        referrer         TEXT          NOT NULL,
        browser          TEXT          NOT NULL,
        CONSTRAINT unique_line UNIQUE (filename, log_line_number)
);
```

A.3 date and time dimension tables

```
CREATE TABLE date (
        date_key           INTEGER     PRIMARY KEY,
        sql_date           DATE        UNIQUE NOT NULL,
        year               INTEGER     ,
        month              INTEGER     ,
        day                INTEGER     ,
        week_day           VARCHAR(10) ,
        semester           VARCHAR(11) ,
        day_of_semester    INTEGER     ,
        week_of_semester   INTEGER     ,
        weekend            VARCHAR(10) ,
        exam               VARCHAR(10) ,
        public_holiday     VARCHAR(15) ,
        school_vacation    VARCHAR(20) ,
        day_of_year        INTEGER     ,
        week_of_year       INTEGER     ,
        workday            VARCHAR(15)
);
```

```
CREATE TABLE time_of_day (
        time_of_day_key   SERIAL          PRIMARY KEY,
        sql_time          TIME            UNIQUE NOT NULL,
        hour              INTEGER         ,
        minute            INTEGER         ,
        second            INTEGER         ,
        working_hours     VARCHAR(20) ,
        period_of_day     VARCHAR(20)
);
```

A.4 page dimension table

```
CREATE TABLE page (
        page_key            INTEGER     PRIMARY KEY,
        page_function       VARCHAR(100) NOT NULL DEFAULT 'unknown',
        page_function_type  VARCHAR(30) NOT NULL DEFAULT 'unknown',
        process             VARCHAR(30) NOT NULL DEFAULT 'unknown',
        CONSTRAINT unique_page UNIQUE (page_function, page_function_type, process)
);
```

A.5 session dimension table

```
CREATE TABLE session (
        session_key                   INTEGER        PRIMARY KEY,
        session_tag                   VARCHAR(100) NOT NULL DEFAULT 'unknown',
        ip_address                    VARCHAR(15)  NOT NULL DEFAULT 'unknown',
        browser                       TEXT           NOT NULL DEFAULT 'unknown',
        referrer                      TEXT           NOT NULL DEFAULT 'unknown',
        first_request_url             TEXT           NOT NULL DEFAULT 'unknown',
        first_page_key                INTEGER        REFERENCES page,
        last_request_url              TEXT           NOT NULL DEFAULT 'unknown',
        last_page_key                 INTEGER        REFERENCES page,
        start_date                    DATE           NOT NULL,
        start_date_key                INTEGER        REFERENCES date,
        start_time                    TIME           NOT NULL,
        start_time_key                INTEGER        REFERENCES time_of_day,
        end_date                      DATE           NOT NULL,
        end_date_key                  INTEGER        REFERENCES date,
        end_time                      TIME           NOT NULL,
        end_time_key                  INTEGER        REFERENCES time_of_day,
        pages_in_session              INTEGER        NOT NULL DEFAULT 0,
        book_descriptions_in_session  INTEGER        NOT NULL DEFAULT 0,
        books_in_basket_in_session    INTEGER        NOT NULL DEFAULT 0,
        reservations_in_session       INTEGER        NOT NULL DEFAULT 0
);
```

A.6 search fact table

```
CREATE TABLE search (
        date_key                      INTEGER   REFERENCES date,
        time_of_day_key               INTEGER   REFERENCES time_of_day,
        session_key                   INTEGER   REFERENCES session,
        search_type_key               INTEGER   REFERENCES search_type,
        search_number                 INTEGER   NOT NULL DEFAULT 0,
        search_number_validity        VARCHAR(9) NOT NULL DEFAULT 'temporary',
        number_of_book_descriptions   INTEGER   NOT NULL DEFAULT 0,
        number_of_books_in_basket     INTEGER   NOT NULL DEFAULT 0,
        number_of_reservations        INTEGER   NOT NULL DEFAULT 0
);
```

A.7 search_type dimension table

```
CREATE TABLE search_type (
        search_type_key      INTEGER        PRIMARY KEY,
        type                 VARCHAR(18)  NOT NULL DEFAULT 'unknown',
        field                VARCHAR(18)  NOT NULL DEFAULT 'unknown',
        type_with_field      VARCHAR(50)  NOT NULL DEFAULT 'unknown',
        CONSTRAINT unique_type UNIQUE (type, field)
);
```

A.8 audit table

```
CREATE TABLE audit (
        audit_key                    INTEGER       PRIMARY KEY,
        filename                     VARCHAR(19),
        etl_start_time               TIMESTAMP,
        etl_end_time                 TIMESTAMP,

        log_file_line_count          INTEGER,
        total_lines_processed        INTEGER,
        valid_lines_processed        INTEGER,
        invalid_lines_processed      INTEGER,
        copy_log_line_file_count     INTEGER,
        new_log_line_records         INTEGER,

        active_sessions_before       INTEGER,
        session_records_before       INTEGER,
        new_sessions_processed       INTEGER,
        active_sessions_processed    INTEGER,
        copy_session_file_count      INTEGER,
        session_records_after        INTEGER,
        new_session_records          INTEGER,

        active_searches_before       INTEGER,
        search_records_before        INTEGER,
        new_searches_processed       INTEGER,
        active_searches_processed    INTEGER,
        copy_search_file_count       INTEGER,
        search_records_after         INTEGER,
        new_search_records           INTEGER,

        active_page_events_before    INTEGER,
        page_event_records_before    INTEGER,
        new_page_events_processed    INTEGER,
        active_page_events_processed INTEGER,
        copy_page_event_file_count   INTEGER,
        page_event_records_after     INTEGER,
        new_page_event_records       INTEGER,

        success_status               VARCHAR(500),
        proceed                      VARCHAR(500)
);
```

A.9 active_page_event table

```
CREATE TABLE active_page_event (
        log_line_key       INTEGER REFERENCES log_line,
        date_key           INTEGER REFERENCES date,
        time_of_day_key    INTEGER REFERENCES time_of_day,
        page_key           INTEGER REFERENCES page,
        session_key        INTEGER REFERENCES active_session,
        audit_key          INTEGER REFERENCES audit
);
```

A.10 active_session dimension table

```
CREATE TABLE active_session (
        session_key          INTEGER PRIMARY KEY,
        session_tag          VARCHAR(100) NOT NULL DEFAULT 'unknown',
        ip_address           VARCHAR(15) NOT NULL DEFAULT 'unknown',
        browser              TEXT         NOT NULL DEFAULT 'unknown',
        first_request_url    TEXT         NOT NULL DEFAULT 'unknown',
        first_page_key       INTEGER      NOT NULL DEFAULT 1,
        last_request_url     TEXT         NOT NULL DEFAULT 'unknown',
        last_page_key        INTEGER      NOT NULL DEFAULT 1,
        referrer                      TEXT      NOT NULL DEFAULT 'unknown',
        start_date                    DATE      NOT NULL,
        start_date_key                INTEGER   NOT NULL,
        start_time                    TIME      NOT NULL,
        start_time_key                INTEGER   NOT NULL,
        end_date                      DATE      NOT NULL,
        end_date_key                  INTEGER   NOT NULL,
        end_time                      TIME      NOT NULL,
        end_time_key                  INTEGER   NOT NULL,
        pages_in_session              INTEGER   NOT NULL DEFAULT 0,
        book_descriptions_in_session  INTEGER   NOT NULL DEFAULT 0,
        books_in_basket_in_session    INTEGER   NOT NULL DEFAULT 0,
        reservations_in_session       INTEGER   NOT NULL DEFAULT 0,
        last_search_number            INTEGER   NOT NULL DEFAULT 0,
        search_number_validity        VARCHAR(9) NOT NULL DEFAULT 'temporary'
);
```

A.11 active_search fact table

```
CREATE TABLE active_search (
        date_key                      INTEGER   REFERENCES date,
        time_of_day_key               INTEGER   REFERENCES time_of_day,
        session_key                   INTEGER,
        search_type_key               INTEGER   REFERENCES search_type,
        search_number                 INTEGER   NOT NULL DEFAULT 0,
        search_number_validity        VARCHAR(9) NOT NULL DEFAULT 'temporary',
        number_of_book_descriptions   INTEGER   NOT NULL DEFAULT 0,
        number_of_books_in_basket     INTEGER   NOT NULL DEFAULT 0,
        number_of_reservations        INTEGER   NOT NULL DEFAULT 0
        --CONSTRAINT unique_search UNIQUE (session_key, search_type_key,
        --                          search_number, search_number_validity )
);
```

APPENDIX B

PostgreSQL Views

```
CREATE VIEW page_event_dpt AS
          SELECT date_key, page_key, time_of_day.hour,
                    COUNT(*) AS number_of_page_events
          FROM page_event, time_of_day
          WHERE page_event.time_of_day_key = time_of_day.time_of_day_key
          GROUP BY date_key, page_key, time_of_day.hour;

CREATE VIEW page_event_dp AS
          SELECT date_key, page_key, COUNT(*) AS number_of_page_events
          FROM page_event
          GROUP BY date_key, page_key;

CREATE VIEW page_event_dt AS
          SELECT date_key, time_of_day.hour, COUNT(*) AS number_of_page_events
          FROM page_event, time_of_day
          WHERE page_event.time_of_day_key = time_of_day.time_of_day_key
          GROUP BY date_key, time_of_day.hour;

CREATE VIEW page_event_pt AS
          SELECT page_key, time_of_day.hour, COUNT(*) AS number_of_page_events
          FROM page_event, time_of_day
          WHERE page_event.time_of_day_key = time_of_day.time_of_day_key
          GROUP BY page_key, time_of_day.hour;

CREATE VIEW page_event_d AS
          SELECT date_key, COUNT(*) AS number_of_page_events
          FROM page_event
          GROUP BY date_key;

CREATE VIEW page_event_p AS
          SELECT page_key, COUNT(*) AS number_of_page_events
          FROM page_event
          GROUP BY page_key;

CREATE VIEW page_event_t AS
          SELECT time_of_day.hour, COUNT(*) AS number_of_page_events
          FROM page_event, time_of_day
          WHERE page_event.time_of_day_key = time_of_day.time_of_day_key
          GROUP BY time_of_day.hour;

CREATE VIEW search_dst AS
          SELECT date_key, search_type_key, time_of_day.hour,
                    COUNT(*) AS number_of_searches,
                    SUM(number_of_book_descriptions) AS total_book_descriptions ,
                    SUM(number_of_books_in_basket) AS total_books_in_basket,
                    SUM(number_of_reservations) AS  total_reservations
          FROM search, time_of_day
          WHERE search.time_of_day_key = time_of_day.time_of_day_key
          GROUP BY date_key, search_type_key, time_of_day.hour;

CREATE VIEW search_ds AS
          SELECT date_key, search_type_key, COUNT(*) AS number_of_searches,
                    SUM(number_of_book_descriptions) AS total_book_descriptions ,
                    SUM(number_of_books_in_basket) AS total_books_in_basket,
                    SUM(number_of_reservations) AS  total_reservations
          FROM search
          GROUP BY date_key, search_type_key;

CREATE VIEW search_dt AS
          SELECT date_key, time_of_day.hour, COUNT(*) AS number_of_searches,
                    SUM(number_of_book_descriptions) AS total_book_descriptions ,
                SUM(number_of_books_in_basket) AS total_books_in_basket,
                SUM(number_of_reservations) AS  total_reservations
        FROM search, time_of_day
        WHERE search.time_of_day_key = time_of_day.time_of_day_key
        GROUP BY date_key, time_of_day.hour;

CREATE VIEW search_st AS
        SELECT search_type_key, time_of_day.hour, COUNT(*) AS number_of_searches,
                SUM(number_of_book_descriptions) AS total_book_descriptions ,
                SUM(number_of_books_in_basket) AS total_books_in_basket,
                SUM(number_of_reservations) AS  total_reservations
        FROM search, time_of_day
        WHERE search.time_of_day_key = time_of_day.time_of_day_key
        GROUP BY search_type_key, time_of_day.hour;

CREATE VIEW search_d AS
        SELECT date_key, COUNT(*) AS number_of_searches,
                SUM(number_of_book_descriptions) AS total_book_descriptions ,
                SUM(number_of_books_in_basket) AS total_books_in_basket,
                SUM(number_of_reservations) AS  total_reservations
        FROM search
        GROUP BY date_key;

CREATE VIEW search_s AS
        SELECT search_type_key, COUNT(*) AS number_of_searches,
                SUM(number_of_book_descriptions) AS total_book_descriptions ,
                SUM(number_of_books_in_basket) AS total_books_in_basket,
                SUM(number_of_reservations) AS  total_reservations
        FROM search
        GROUP BY search_type_key;

CREATE VIEW search_t AS
        SELECT time_of_day.hour, COUNT(*) AS number_of_searches,
                SUM(number_of_book_descriptions) AS total_book_descriptions ,
                SUM(number_of_books_in_basket) AS total_books_in_basket,
                SUM(number_of_reservations) AS  total_reservations
        FROM search, time_of_day
        WHERE search.time_of_day_key = time_of_day.time_of_day_key
        GROUP BY time_of_day.hour;

CREATE VIEW session_dflt AS
        SELECT start_date_key, first_page_key ,  last_page_key ,  time_of_day.hour,
                COUNT(*) AS number_of_sessions
        FROM session, time_of_day
        WHERE session.start_time_key = time_of_day.time_of_day_key
        GROUP BY start_date_key, first_page_key ,  last_page_key ,  time_of_day.hour;

CREATE VIEW session_dfl AS
        SELECT start_date_key, first_page_key ,  last_page_key ,
                COUNT(*) AS number_of_sessions
        FROM session
        GROUP BY start_date_key, first_page_key ,  last_page_key ,  pages_in_session ;

CREATE VIEW session_dft AS
        SELECT start_date_key, first_page_key , time_of_day.hour,
                COUNT(*) AS number_of_sessions
        FROM session, time_of_day
        WHERE session.start_time_key = time_of_day.time_of_day_key
        GROUP BY start_date_key, first_page_key ,  time_of_day.hour;

CREATE VIEW session_dlt AS
        SELECT start_date_key, last_page_key ,  time_of_day.hour,
                COUNT(*) AS number_of_sessions
        FROM session, time_of_day
        WHERE session.start_time_key = time_of_day.time_of_day_key
        GROUP BY start_date_key, last_page_key , time_of_day.hour;

CREATE VIEW session_flt AS
        SELECT first_page_key, last_page_key ,  time_of_day.hour,
                COUNT(*) AS number_of_sessions
        FROM session, time_of_day
        WHERE session.start_time_key = time_of_day.time_of_day_key
        GROUP BY first_page_key, last_page_key , time_of_day.hour;

CREATE VIEW session_df AS
        SELECT start_date_key, first_page_key , COUNT(*) AS number_of_sessions
        FROM session
        GROUP BY start_date_key, first_page_key ;

CREATE VIEW session_dl AS
        SELECT start_date_key, last_page_key , COUNT(*) AS number_of_sessions
        FROM session
        GROUP BY start_date_key, last_page_key;

CREATE VIEW session_dt AS
        SELECT start_date_key, time_of_day.hour, COUNT(*) AS number_of_sessions
        FROM session, time_of_day
        WHERE session.start_time_key = time_of_day.time_of_day_key
        GROUP BY start_date_key, time_of_day.hour;

CREATE VIEW session_fl AS
        SELECT first_page_key, last_page_key , COUNT(*) AS number_of_sessions
        FROM session
        GROUP BY first_page_key, last_page_key;

CREATE VIEW session_ft AS
        SELECT first_page_key, time_of_day.hour, COUNT(*) AS number_of_sessions
        FROM session, time_of_day
        WHERE session.start_time_key = time_of_day.time_of_day_key
        GROUP BY first_page_key, time_of_day.hour;

CREATE VIEW session_lt AS
        SELECT last_page_key, time_of_day.hour, COUNT(*) AS number_of_sessions
        FROM session, time_of_day
        WHERE session.start_time_key = time_of_day.time_of_day_key
        GROUP BY last_page_key, time_of_day.hour;

CREATE VIEW session_d AS
        SELECT start_date_key, COUNT(*) AS number_of_sessions
        FROM session
        GROUP BY start_date_key;

CREATE VIEW session_f AS
        SELECT first_page_key, COUNT(*) AS number_of_sessions
        FROM session
        GROUP BY first_page_key;

CREATE VIEW session_l AS
        SELECT last_page_key, COUNT(*) AS number_of_sessions
        FROM session
        GROUP BY last_page_key;

CREATE VIEW session_t AS
        SELECT time_of_day.hour, COUNT(*) AS number_of_sessions
        FROM session, time_of_day
        WHERE session.start_time_key = time_of_day.time_of_day_key
        GROUP BY time_of_day.hour;
```

APPENDIX C

PostgreSQL View Functions

C.1 Create Materialized View

```
CREATE OR REPLACE FUNCTION create_matview(NAME, NAME)
RETURNS VOID
SECURITY DEFINER
LANGUAGE plpgsql AS '
DECLARE
    matview ALIAS FOR $1;
    view_name ALIAS FOR $2;
    entry  matviews%ROWTYPE;
BEGIN
    SELECT * INTO entry FROM matviews WHERE mv_name = matview;

    IF  FOUND THEN
        RAISE EXCEPTION ''Materialized view ''''%'''' already  exists . '' ,
           matview;
    END IF;

    EXECUTE ''REVOKE ALL ON '' || view_name || '' FROM PUBLIC'';

    EXECUTE ''GRANT SELECT ON '' || view_name || '' TO PUBLIC'';

    EXECUTE ''CREATE TABLE '' || matview || '' AS SELECT * FROM '' || view_name;

    EXECUTE ''REVOKE ALL ON '' || matview || '' FROM PUBLIC'';

    EXECUTE ''GRANT SELECT ON '' || matview || '' TO PUBLIC'';

    INSERT INTO matviews (mv_name, v_name, last_refresh)
      VALUES (matview, view_name, CURRENT_TIMESTAMP);

    RETURN;
END
' ;
```

C.2 Refresh Materialized View

```
CREATE OR REPLACE FUNCTION refresh_matview(name) RETURNS VOID
SECURITY DEFINER
LANGUAGE plpgsql AS '
DECLARE
    matview ALIAS FOR $1;
    entry  matviews%ROWTYPE;
BEGIN

    SELECT * INTO entry FROM matviews WHERE mv_name = matview;

    IF  NOT FOUND THEN
        RAISE EXCEPTION ''Materialized view % does not exist.'' , matview;
    END IF;

    EXECUTE ''DELETE FROM '' || matview;
    EXECUTE ''INSERT INTO '' || matview
         ||  ''  SELECT * FROM '' || entry.mv_view;

    UPDATE matviews
        SET  last_refresh =CURRENT_TIMESTAMP
        WHERE mv_name=matview;
    RETURN;
END
';
```

C.3 Drop Materialized View

```
CREATE OR REPLACE FUNCTION drop_matview(NAME) RETURNS VOID
SECURITY DEFINER
LANGUAGE plpgsql AS '
DECLARE
    matview ALIAS FOR $1;
    entry  matviews%ROWTYPE;
BEGIN

    SELECT * INTO entry FROM matviews WHERE mv_name = matview;

    IF  NOT FOUND THEN
        RAISE EXCEPTION ''Materialized view % does not exist.'', matview;
    END IF;

    EXECUTE ''DROP TABLE '' || matview;
    DELETE FROM matviews WHERE mv_name=matview;

    RETURN;
END
';
```

APPENDIX D

Source Code for Greedy

D.1 Greedy Class

```
public class Greedy {

    static View[] views = null;
    static boolean done = false;

    public static void calculateBenefit (View candidateView) {
        int benefit = 0;
        for (int v = 0; v < views.length; v++) {
            boolean dependency = true;
            for (int d = 0; d < views[v].getViewName().length(); d++) {
                if (candidateView.getViewName().indexOf(views[v].getViewName()
                                                        .charAt(d)) == -1) {
                    dependency = false;
                }
            }
            if (dependency) {
                int costBefore = views[v].getCost();
                int costAfter = candidateView.getSize();
                int gain = costBefore - costAfter;
                if (gain > 0) {
                    if (views[v].getViewName().length() == 1) {
                        gain *= 10;
                    }
                    benefit += gain;
                }
            }
        }
        candidateView.setBenefit(benefit);
    }

    public static void materializeBestView() {
        View bestView = views[0];
        for (int v = 1; v < views.length; v++) {
            if (views[v].getBenefit() > bestView.getBenefit()) {
                bestView = views[v];
            }
        }
        if (bestView.getBenefit() == 0) {
            done = true;
        }
        else {
            bestView.materialize();
            System.out.println("Materializing␣" + bestView.getViewName() +
                               "␣(benefit:␣" + bestView.getBenefit() + ")");
            for (int v = 0; v < views.length; v++) {
                boolean dependency = true;
                for (int d = 0; d < views[v].getViewName().length(); d++) {
                    if (bestView.getViewName().indexOf(views[v].getViewName()
                                                       .charAt(d)) == -1) {
                        dependency = false;
                    }
                }
                if (dependency) {
                    if (views[v].getCost() > bestView.getSize()) {
                        views[v].setCost(bestView.getSize());
                    }
                }
            }
        }
    }

    public static void main (String [] args) {
        String schema = args [0];

        if (schema == null || schema.equals("")) {
            System.out. println ("Error:␣schema␣name␣must␣be␣input");
        }
        else if (schema.equals("pageevent")) {
            // page event without session with time grouped by hour
            views = new View[] {new View("dpt", 214746, 214746),
                                new View("dp",   23510, 214746),
                                new View("dt",    7716, 214746),
                                new View("pt",    1720, 214746),
                                new View("d",      393, 214746),
                                new View("p",       95, 214746),
                                new View("t",       24, 214746)};
            views [0]. materialize ();
        }
        else if (schema.equals("search")) {
            // search without session with time grouped by hour
            views = new View[] {new View("dst", 53530, 53530),
                                new View("ds",   6235, 53530),
                                new View("dt",   7363, 53530),
                                new View("st",    435, 53530),
                                new View("d",     393, 53530),
                                new View("s",      20, 53530),
                                new View("t",      24, 53530)};
            views [0]. materialize ();
        }
        else if (schema.equals("session")) {
            // session with time grouped by hour
            views = new View[] {new View("dflt", 134758, 134758),
                                new View("dfl",   37405, 134758),
                                new View("dft",   35004, 134758),
                                new View("dlt",  101016, 134758),
                                new View("flt",    6886, 134758),
                                new View("df",     6215, 134758),
                                new View("dl",    16469, 134758),
                                new View("dt",     7687, 134758),
                                new View("fl",     1041, 134758),
                                new View("ft",      760, 134758),
                                new View("lt",     1427, 134758),
                                new View("d",       393, 134758),
                                new View("f",        62, 134758),
                                new View("l",        81, 134758),
                                new View("t",        24, 134758)};
            views [0]. materialize ();
        }
        else {
            System.out. println ("Schema␣name␣not␣recognized.␣Should␣be␣one␣of:");
            System.out. println ("basicpageevent");
            System.out. println ("pageeventwithsession");
            System.out. println ("pageevent");
            System.out. println ("search");
            System.out. println ("session");
            return;
        }

        int choice = 0;

        while (! done) {
            boolean allViewsMaterialized = true;
            for (int v = 0; v < views.length; v++) {
                if (! views[v]. isMaterialized ()) {
                    allViewsMaterialized = false;
                }
            }

            if (! allViewsMaterialized ) {
                choice++;
                System.out.print("CHOICE " + choice + ": ");

                for (int v = 0; v < views.length; v++) {
                    if (!views[v].isMaterialized()) {
                        calculateBenefit(views[v]);
                    }
                    else {
                        views[v].setBenefit(0);
                    }
                }
                materializeBestView();
            }
            else {
                done = true;
            }
        }
        System.out.println("\n\n");
    }
}
```

D.2 View Class

```
public class View {

    private String _viewName, _parentViewName, _dimensions;
    private int _size, _cost, _benefit;
    private boolean _materialized;

    public View(String viewName, int size, int cost) {
        _viewName = viewName;
        _size = size;
        _cost = cost;
        _benefit = size;
        _materialized = false;
    }

    public String getViewName() {
        return _viewName;
    }

    public int getSize() {
        return _size;
    }

    public int getCost() {
        return _cost;
    }

    public int getBenefit() {
        return _benefit;
    }

    public boolean isMaterialized() {
        return _materialized;
    }

    public void materialize() {
        _materialized = true;
    }

    public void setBenefit(int newBenefit) {
        _benefit = newBenefit;
    }

    public void setCost(int newCost) {
        _cost = newCost;
    }
}
```

APPENDIX E

Source Code for Java Classes

E.1 PostProcessor Servlet Class

```java
package aub;

import java.io.IOException;
import java.io.PrintWriter;
import java.io.BufferedReader;
import java.io.FileOutputStream;
import java.io.FileReader;
import java.io.FileNotFoundException;
import java.net.MalformedURLException;
import java.sql.Time;
import java.sql.ResultSet;
import java.sql.ResultSetMetaData;
import java.util.GregorianCalendar;
import java.util.Vector;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class PostProcessor extends HttpServlet {

    PrintWriter _out;
    int _numberOfLogLinesTogether = 100;
    String _runningOn = "baerbar";
    // String _runningOn = " stationaer ";
    String _filePath = "";
    String _pathToCopyFiles = "";
    Database _database;
    int _tempSearchNumber = 0;
    int _logLineKey = 0;
    int _lastEmptiedLogLineKey = 0;
    int _auditKey = 0;
    int _sessionRecordsBefore = 0;
    int _searchRecordsBefore = 0;
    int _pageEventRecordsBefore = 0;
    int _activeSessionsBefore = 0;
    int _activeSearchesBefore = 0;
    int _activePageEventsBefore = 0;
    String _successStatus = "";
    java.util.Date _etlStartDate = null;
    String _logFileName = "";
    int _fileLineCount = 0;
    int _totalLinesProcessed = 0;
    int _validLinesProcessed = 0;
    int _invalidLinesProcessed = 0;
    int _sessionRecordsAfter = 0;
    int _searchRecordsAfter = 0;
    int _pageEventRecordsAfter = 0;
    int _newSessionsProcessed, _newSearchesProcessed, _newPageEventsProcessed;
    int _activeSessionsProcessed, _activeSearchesProcessed, _activePageEventsProcessed;
    int _newSessionRecords, _newSearchRecords, _newPageEventRecords;
    int _linesInCopyFile = 0;
    int _copySessionFileCount = 0;
    int _copySearchFileCount = 0;
    int _copyPageEventFileCount = 0;
    int _newLogLineRecords = 0;
    int _numberOfLogLines = 0;
    java.util.Date _etlEndDate = null;
    boolean _etlSuccess = true;
    private String[] _ignoreList = {".jpg", ".gif", ".ico", ".css"};
    private String[] _searchBotList = {"Ant␣Movie␣Catalog␣using␣Indy␣Library",
                                       "Gigabot", "Girafabot", "Googlebot",
                                       "Microsoft␣Data␣Access␣Internet␣Publishing",
                                       "Microsoft␣URL␣Control",
                                       "MicrosoftPrototypeCrawler",
                                       "MS␣Search␣4.0␣Robot", "MSIECrawler", "MSNBOT",
                                       "NaverRobot", "NPBot", "NutchOrg", "Openbot",
                                       "psbot", "RPT-HTTPClient/0.3-3", "TurnitinBot",
                                       "Tutorial␣Crawler", "VoilaBot", "Web␣Crawler",
                                       "WebCrawler", "www.troutfarmer.dk",
                                       "Xenu␣Link␣Sleuth", "Xenu's␣Link␣Sleuth",
                                       "ZyBorg"};
    private String[] _filenames = null;
    private String[] _columnNames = null;
    private String[] _columnValues = null;

    public PostProcessor(Database database, int auditKey, int logLineKey,
                         String filePath, String pathToCopyFiles) {
        _database = database;
        _auditKey = auditKey;
        _logLineKey = logLineKey;
        _filePath = filePath;
        _pathToCopyFiles = pathToCopyFiles;
    }

    void countLinesInLogFile(String filePath, String filename) {
        try {
            _fileLineCount = 0;
            BufferedReader logLineReader = new BufferedReader(new FileReader
                (filePath +
                 filename));
            while (logLineReader.readLine() != null) {
                _fileLineCount++;
            }

            logLineReader.close();
            _columnNames = new String[] {" log_file_line_count "};
            _columnValues = new String [] { Integer . toString ( _fileLineCount )};
            _database .updateAuditDimension(_auditKey, _columnNames,
                                            _columnValues,
                                            "Lines␣in␣log␣file␣counted");
        }
        catch ( Exception e ) {
            System.out. println (e );
            System.out. println (e .getMessage ());
            System.out. println ("Error␣counting␣lines␣in␣log␣file ." );
        }
    }


    void getActiveInfo () {
        _activePageEventsBefore = _database . getActivePageEvents ();
        _activeSearchesBefore = _database . getActiveSearches ();
        _activeSessionsBefore = _database . getActiveSessions ();

        _columnNames = new String[] {" active_sessions_before ",
                                     " active_searches_before ",
                                     " active_page_events_before " };
        _columnValues = new String [] { Integer . toString ( _activeSessionsBefore ),
                                        Integer . toString ( _activeSearchesBefore ),
                                        Integer . toString ( _activePageEventsBefore )};
        _database .updateAuditDimension(_auditKey, _columnNames, _columnValues,
                                        " active ␣info␣ collected " );

        _database . getPages ();
        _database . getSearchTypes ();
    }

    private BufferedReader [] getBufferedLogFileReaders ( String filePath ) {
        // boolean stillMoreFiles = true ;
        Vector fileReaders = new Vector ();
        Vector filenameVector = new Vector ();
        String filename ;
        BufferedReader reader = null ;
        try {
            for ( GregorianCalendar date = new GregorianCalendar (2003, 1, 25);
                  true ; date .add(date . DAY_OF_MONTH, 1)) {
                                                // filename = " accesstest .20030225";

                filename = "access_log ." + date . get ( date . YEAR);
                if ( date . get ( date . MONTH) < 9) {
                    filename += "0";
                }
                filename += ( date . get ( date . MONTH) + 1);
                if ( date . get ( date . DAY_OF_MONTH) < 10) {
                    filename += "0";
                }
                filename += date . get ( date . DAY_OF_MONTH);
                reader = new BufferedReader(new FileReader( filePath +
                                                            filename ));
                filenameVector .add( filename );
                fileReaders .add( reader );
                if ( date . get ( date . DAY_OF_MONTH) == 26) {
                    throw new FileNotFoundException("");
                }
            }
        }
        catch ( FileNotFoundException fnfe ) {
            // System.out. println (" File not found : " + filename );
            // stillMoreFiles = false ;
        }
        catch ( Exception e ) {
            System.out. println ("Exception␣in␣PostProcessor . getFilenames ()" );
            System.out. println (e );
        }
        BufferedReader [] readerArray = new BufferedReader[ fileReaders . size ()];
        for ( int r = 0; r < readerArray . length ; r++) {
            readerArray [ r ] = ( BufferedReader ) fileReaders . get ( r );
        }
        _filenames = new String[ filenameVector . size ()];
        for ( int f = 0; f < _filenames . length ; f++) {
            _filenames [ f ] = ( String ) filenameVector . get ( f );
        }
        return readerArray ;
    }

    int getCopyPageEventFileCount() {
        return _copyPageEventFileCount;
    }

    int getCopySessionFileCount () {
        return _copySessionFileCount;
    }

    int getCopySearchFileCount () {
        return _copySearchFileCount;
    }

    int [] getRecordCountsAfterLoad() {
        _pageEventRecordsAfter = _database . getNumberOfPageEventsAfter(_auditKey);
        _searchRecordsAfter = _database . getNumberOfSearchesAfter(_auditKey);
        _sessionRecordsAfter = _database . getNumberOfSessionsAfter(_auditKey);
        _columnNames = new String[] {" session_records_after ",
                                     " search_records_after ",
                                     " page_event_records_after "};
        _columnValues = new String [] { Integer . toString (_sessionRecordsAfter ),
                                        Integer . toString (_searchRecordsAfter ),
                                        Integer . toString (_pageEventRecordsAfter)};
        _database . updateAuditDimension(_auditKey, _columnNames, _columnValues,
                                         " records␣after␣counted");
        return new int [] { _pageEventRecordsAfter,
                            _searchRecordsAfter ,
                            _sessionRecordsAfter };
    }

    int [] getRecordCountsBeforeLoad() {
        _pageEventRecordsBefore = _database . getNumberOfPageEventsBefore();
        _searchRecordsBefore = _database . getNumberOfSearchesBefore();
        _sessionRecordsBefore = _database . getNumberOfSessionsBefore();
        System.out . println ("Records␣before:␣page␣event:␣" + _pageEventRecordsBefore +
                              ".␣search:␣" + _searchRecordsBefore + ".␣session:␣"
                              + _sessionRecordsBefore );

        _columnNames = new String[] {" session_records_before ",
                                     " search_records_before ",
                                     " page_event_records_before "};
        _columnValues = new String [] { Integer . toString (_sessionRecordsBefore ),
                                        Integer . toString (_searchRecordsBefore ),
                                        Integer . toString (_pageEventRecordsBefore)};
        _database . updateAuditDimension(_auditKey, _columnNames, _columnValues,
                                         " records␣before␣load␣counted");
        return new int [] { _pageEventRecordsBefore,
                            _searchRecordsBefore,
                            _sessionRecordsBefore };
    }

    private boolean isOnIgnoreList ( String requestUrl ) {
        for ( int i = 0; i < _ignoreList . length ; i++)
            if ( requestUrl . indexOf( _ignoreList [ i ]) != −1) {
                return true;
            }
        return false ;
    }

    private boolean isSearchBot ( String browser) {
        for ( int b = 0; b < _searchBotList . length ; b++)
            if ( browser . indexOf(_searchBotList [b ]) != −1) {
                return true;
            }
        return false ;
    }
```

```
    private void loadAndDimensionalize(String filePath ) {
        BufferedReader [] fileReaders = getBufferedLogFileReaders( filePath );
        _database . prepareStatements ();

        long _numberOfLogLines = 0;

        for ( int f = 0; f < fileReaders . length ; f++) {
            _auditKey = _database . newAuditRecord(_filenames[f ],
                                                   "New␣audit␣record␣created",
                                                   "Recover");

            System.out . println ("Filename:␣" + _filenames [ f ]);
            getActiveInfo ();
            getRecordCountsBeforeLoad();

            countLinesInLogFile( filePath , _filenames [ f ]);

            _database . resetFiles ();
            processLogFile ( fileReaders [ f ], _filenames [ f ]);
            // getRecordCountsAfterLoad();

            _newPageEventRecords = _pageEventRecordsAfter − _pageEventRecordsBefore;
            String pageEventSuccess = "";
            if ( _newPageEventRecords != _copyPageEventFileCount) {
                pageEventSuccess =
                    "␣Number␣of␣new␣page␣event␣records␣does␣not␣"+
                    "match␣number␣of␣lines␣in␣copy␣file.";
                _etlSuccess = false ;
            }
            _newSessionRecords = _sessionRecordsAfter − _sessionRecordsBefore;
            String sessionSuccess = "";
            if ( _newSessionRecords != _copySessionFileCount ) {
                sessionSuccess =
                    "␣Number␣of␣new␣session␣records␣does␣not␣"+
                    "match␣number␣of␣lines␣in␣copy␣file.";
                _etlSuccess = false ;
            }
            _newSearchRecords = _searchRecordsAfter − _searchRecordsBefore;
            String searchSuccess = "";
            if ( _newSearchRecords != _copySearchFileCount) {
                searchSuccess =
                    "␣Number␣of␣new␣search␣records␣does␣not␣"+
                    "match␣number␣of␣lines␣in␣copy␣file.";
                _etlSuccess = false ;
            }

            _columnNames = new String[] {"new_session_records",
                                         "new_search_records",
                                           "new_page_event_records"};
        _columnValues = new String [] { Integer . toString (_newSessionRecords),
                                        Integer . toString (_newSearchRecords),
                                        Integer . toString (_newPageEventRecords)};

        _database . updateAuditDimension(_auditKey, _columnNames, _columnValues,
                                        "number␣of␣new␣session,␣search␣and␣" +
                                        "page␣event␣records␣counted." +
                                        pageEventSuccess + sessionSuccess +
                                        searchSuccess );

        if ( _etlSuccess ) {
            _successStatus = "ETL␣process␣completed␣succesfully";
        }
        else {
            _successStatus = " Failure ";

            System.out. println ("Success␣status :␣" + _successStatus );
            System.out. println ();
        }
    }
}

/**
 * Preloads all dates between fromDate and toDate to the date dimension table .
 * Only dates that are defined as vacations or public holidays in the Date class
 * will be recognized as vacations or public holidays .
 */
private void loadDates( java . sql . Date fromDate, java . sql . Date toDate ) {
    try {
        GregorianCalendar currentDate = new GregorianCalendar ();
        GregorianCalendar endDate = new GregorianCalendar ();
        endDate.setTime(toDate );
        int dateKey = 0;

        for ( currentDate .setTime(fromDate ); ! currentDate . after (endDate);
              currentDate .add( currentDate .DAY_OF_YEAR, 1)) {
            dateKey++;
            Date date = new Date(dateKey, currentDate );
            _database. insertDate ( date );
        }
        _database . copyToDatabase("date" );
    }
    catch ( Exception e ) {
        System.out. println ( e );
    }
}

/**
 * Preloads a row into the time of day dimension for each second in a 24 hour day.
 */
private void loadTimes () throws java . io . IOException {
    long startLoadingTime = new java. util . Date (). getTime ();
    GregorianCalendar currentTime = new GregorianCalendar ();
    GregorianCalendar endTime = new GregorianCalendar();
    endTime. setTimeInMillis (Time.valueOf("23:59:59" ). getTime ());

    for ( currentTime . setTimeInMillis (Time.valueOf("00:00:00" ). getTime ());
          !currentTime . after (endTime); currentTime . add(currentTime . SECOND, 1)) {
        Time sqlTime = new Time(currentTime. getTimeInMillis ());
        int hour = currentTime . get (currentTime . HOUR_OF_DAY);
        int minute = currentTime . get (currentTime . MINUTE);
        int second = currentTime . get (currentTime . SECOND);

        String workingHours = "not␣working␣hours";
        if (8 <= hour && hour < 16) {
            workingHours = "working␣hours";
        }

        String periodOfDay;
        if (0 <= hour && hour < 6) {
            periodOfDay = " night ";
        }
        else if (6 <= hour && hour < 12) {
            periodOfDay = "morning";
        }
        else if (12 <= hour && hour < 18) {
            periodOfDay = " afternoon ";
        }
        else {
            periodOfDay = "evening" ;
        }
        _database . insertTime (sqlTime , hour , minute , second,
                                workingHours, periodOfDay);
    }
    _database . copyToDatabase("time_of_day");
    long endLoadingTime = new java. util . Date (). getTime ();
    long loadingTimeInMillis = endLoadingTime − startLoadingTime;
    String loadingTime = millisecondsToTime( loadingTimeInMillis );
    // System.out. println ("Time to load time_of_day : " + loadingTimeInMillis +
    //             " milliseconds (" + loadingTime + ").");
}

/**
 * Takes a long millisecond value as input and returns a string in hh:mm:ss.mmm
 * format representing a more human readable value of the time represented by
 * the millisecond value .
 */
```

```
private String millisecondsToTime(long milliseconds) {
    long millisecond = milliseconds % 1000;
    long timeInSeconds = milliseconds / 1000;
    long second = timeInSeconds % 60;
    long timeInMinutes = timeInSeconds / 60;
    long minute = timeInMinutes % 60;
    long hour = timeInMinutes / 60;
    return "" + hour + ":" + minute + ":" + second + "." + millisecond;
}

boolean processLogFile(BufferedReader logFile, String filename) {
    try {
        int currentLineNumber = 0;
        _totalLinesProcessed = 0;
        _validLinesProcessed = 0;
        _invalidLinesProcessed = 0;
        _database.resetNewInfo();

        // BufferedReader reader = new BufferedReader
        //( new FileReader( _filePath + _filenames[f]));
        LogLine logLine = null;
        String line = null;

        while ((line = logFile.readLine()) != null) {
            currentLineNumber++;
            _totalLinesProcessed++;

            try {
                if (! isOnIgnoreList(line) && !isSearchBot(line) &&
                    line.indexOf("/F/") != −1) {
                    logLine = new LogLine(_logLineKey, filename,
                                          currentLineNumber, line);
                    _logLineKey++;
                    _validLinesProcessed++;
                    if (! _database.insertLogLine(logLine)) {
                        _logLineKey−−;
                        _validLinesProcessed−−;
                        _invalidLinesProcessed++;
                        System.out.println("Error␣inserting␣log␣line␣" +
                                           currentLineNumber);
                    }

                    int dateKey = 0;
                    int timeKey, pageKey, sessionKey;
                    java.sql.Date date = null;

                    if (logLine != null) {
                        _numberOfLogLines++;
                        UrlQuery urlQuery = new UrlQuery(logLine.getQuery());

                        // page_event schema
                        int logLineKey = logLine.getLogLineKey();

                        if (! logLine.getDate().equals(date)) {
                            date = logLine.getDate();
                            dateKey = _database.getDateKey(date);
                            if (dateKey == −1) {
                                System.out.println("No␣new␣log␣files␣" +
                                                   "can␣be␣added␣until␣" +
                                                   "new␣dates␣have␣been␣" +
                                                   " specified ");
                                throw new Exception("Add␣new␣dates␣to␣" +
                                                    "system␣before␣new␣" +
                                                    "log␣lines␣can␣be␣" +
                                                    " input ");
                            }
                        }

                        timeKey = _database.getTimeKey(logLine.getTime());

                        Page page = new Page(logLine);
                        pageKey = _database.getPageKey(page);

                        sessionKey = _database.getSessionKey(logLine, pageKey,
                                                             dateKey, timeKey);

                        PageEvent pageEvent = new PageEvent(logLineKey,
                                                            dateKey,
                                                            timeKey,
                                                            pageKey,
                                                            sessionKey,
                                                            _auditKey);
                        _database.insertPageEvent(pageEvent, logLine);

                        // search schema
                        String searchNumberValidity = "temporary";
                        int searchNumber = urlQuery.getIntValue
                            ("set_number");
                        if (searchNumber != −1) {
                            searchNumberValidity = " valid ";
                        }

                        try {
                            SearchType searchType = new SearchType
                                (new UrlQuery(logLine.getQuery()));
                            int searchTypeKey = _database
                                .getSearchTypeKey(searchType);
            int numberOfBookDescriptions = 0;
            int numberOfBooksInBasket = 0;
            int numberOfReservations = 0;

            if (searchNumber == −1) {
                _tempSearchNumber++;
                searchNumber = _tempSearchNumber;
                searchNumberValidity = "temporary";
            }

            Search search = new Search(dateKey,
                                       timeKey, sessionKey,
                                       searchTypeKey,
                                       searchNumber,
                                       searchNumberValidity,
                                       numberOfBookDescriptions,
                                       numberOfBooksInBasket,
                                       numberOfReservations);
            _database. insertSearch (search);
        }
        catch (NotSearchTypeException e) {
            if (searchNumberValidity. equals(" valid ")) {
                _database.setLastSearchNumber(sessionKey,
                                              searchNumber,
                                              searchNumberValidity);
            }
        }
        catch (Exception e) {
            System.out. println ("Exception␣in␣search␣" +
                                 " part␣of␣dimensionalize." );
            System.out. println (e);
            System.out. println (e.getMessage());
        }

        String pageFunctionType = page
            .getPageFunctionType();
        if (pageFunctionType.equals("book␣description")) {
            _database.incrementBookDescriptions(sessionKey);
        }
        else if (pageFunctionType.equals("book␣in␣basket")) {
            _database.incrementBooksInBasket
                (sessionKey, page.getNumberOfCheckedBoxes());
        }
        else if (pageFunctionType.equals(" reservation ")) {
            _database. incrementReservations (sessionKey);
        }
    }
}
else {
            _invalidLinesProcessed ++;
        }
    } catch (InvalidLogLineException ille) {
        // fejl i oprettelse af logline objekt
        _invalidLinesProcessed ++;
    } catch (MalformedURLException mue) {
        // fejl i oprettelse af logline objekt
        _invalidLinesProcessed ++;
        System.out. println ("Malformed␣URL␣in␣" + filename +
                             ",␣line␣" + currentLineNumber);
    }
}

logFile . close ();

_columnNames = new String[] {" total_lines_processed ",
                             " valid_lines_processed ",
                             " invalid_lines_processed "};
_columnValues = new String [] { Integer . toString ( _totalLinesProcessed ),
                               Integer . toString ( _validLinesProcessed ),
                               Integer . toString ( _invalidLinesProcessed )};
boolean match = false ;

if ( _fileLineCount == _totalLinesProcessed ) {
    if ( _totalLinesProcessed == _validLinesProcessed +
        _invalidLinesProcessed ) {
        _successStatus = "Processing␣of␣log␣lines␣complete";
        match = true;
    }
    else {
        _successStatus = "Mismatch␣between␣total␣number␣of␣lines␣" +
            "processed␣and␣sum␣of␣valid␣and␣invalid␣lines␣processed";
        _database.completeAuditRecord(_auditKey, _successStatus , "Proceed");
    }
}
else {
    if ( _totalLinesProcessed == _validLinesProcessed +
        _invalidLinesProcessed ) {
        _successStatus = "Sum␣of␣valid␣and␣invalid␣lines␣processed␣" +
            " is␣equal␣to␣total␣lines␣processed,␣but␣this␣is␣not␣" +
            "equal␣to␣number␣of␣lines␣in␣log␣file";
        _database.completeAuditRecord(_auditKey, _successStatus , "Proceed");
    }
    else {
        _successStatus = "The␣total␣lines␣processed␣is␣neither␣" +
            "equal␣to␣the␣number␣of␣lines␣in␣the␣log␣file␣or␣the␣" +
            "sum␣of␣valid␣and␣invalid␣log␣lines␣processed";
        _database.completeAuditRecord(_auditKey, _successStatus , "Proceed");
    }
```

```
}

_database.updateAuditDimension(_auditKey, _columnNames, _columnValues,
                                _successStatus );
/*
  System.out.println ("Total number of log lines processed : " +
   _totalLinesProcessed );
  System.out.println ("Number of valid log lines processed : " +
   _validLinesProcessed );
  System.out.println ("Number of invalid log lines processed : " +
   _invalidLinesProcessed );*/

if (! match) {
    throw new AuditMismatchException(_successStatus);
}

_newSessionsProcessed = _database.getNumberOfNewSessions();
_newSearchesProcessed = _database.getNumberOfNewSearches();
_newPageEventsProcessed = _database.getNumberOfNewPageEvents();

_columnNames = new String[] {"new_sessions_processed",
                             "new_searches_processed",
                             "new_page_events_processed"};
_columnValues = new String [] { Integer . toString (_newSessionsProcessed),
                                Integer . toString (_newSearchesProcessed),
                                Integer . toString (_newPageEventsProcessed)};

_database.updateAuditDimension(_auditKey, _columnNames, _columnValues,
                               "number␣of␣new␣sessions,␣searches␣and␣" +
                               "page␣events␣processed␣counted");


int _activeSessionsProcessed = _database.saveActiveSessions ();
int _activeSearchesProcessed = _database.saveActiveSearches ();
int _activePageEventsProcessed = _database.saveActivePageEvents ();

_columnNames = new String[] {" active_sessions_processed ",
                             " active_searches_processed ",
                             " active_page_events_processed "};
_columnValues = new String [] { Integer . toString ( _activeSessionsProcessed ),
                                Integer . toString ( _activeSearchesProcessed ),
                                Integer .
                                toString ( _activePageEventsProcessed )};
_successStatus = "Active␣sessions,␣searches␣and␣page␣events␣saved";

_database.updateAuditDimension(_auditKey, _columnNames,
                               _columnValues, _successStatus );

try {
```

```
// count number of lines in log line copy file
BufferedReader copyFileReader = new BufferedReader
     (new FileReader(_pathToCopyFiles + " log_line . data" ));
_linesInCopyFile = 0;
_database. flushFile (" log_line . data" );
while ( copyFileReader.readLine () != null ) {
     _linesInCopyFile ++;
}

_columnNames = new String[] {" copy_log_line_file_count "};
_columnValues = new String [] { Integer . toString ( _linesInCopyFile )};

if ( _linesInCopyFile == _validLinesProcessed ) {
     _successStatus = "Lines␣in␣log␣line␣copy␣file␣counted";
}
else {
     _successStatus = "Number␣of␣lines␣in␣log␣line␣copy␣file␣does␣" +
          "not␣match␣number␣of␣valid␣lines␣processed";
     _etlSuccess = false ;
     match = false ;
}

_database.updateAuditDimension(_auditKey, _columnNames,
                               _columnValues, _successStatus );
if (! match) {
     throw new AuditMismatchException (_successStatus);
}

// count number of lines in session copy file
copyFileReader. close ();
copyFileReader = new BufferedReader
     (new FileReader(_pathToCopyFiles + " session . data" ));
_copySessionFileCount = 0;
_database. flushFile (" session . data" );
while ( copyFileReader.readLine () != null ) {
     _copySessionFileCount++;
}

_columnNames = new String[] {" copy_session_file_count "};
_columnValues = new String [] { Integer . toString (_copySessionFileCount )};

_database.setCopySessionFileCount(_copySessionFileCount );

if ( _activeSessionsBefore + _newSessionsProcessed ==
     _activeSessionsProcessed + _copySessionFileCount ) {
     _successStatus = "Lines␣in␣session␣copy␣file␣counted";
}
else {
     _successStatus = "Number␣of␣lines␣in␣session␣copy␣file␣does␣" +
```

                "not␣match␣number␣of␣sessions␣processed";
            _etlSuccess = **false**;
            match = **false**;
        }

        _database.updateAuditDimension(_auditKey, _columnNames,
                                       _columnValues, _successStatus);
        if (! match) {
            _etlSuccess = **false**;
            throw new AuditMismatchException(_successStatus);
        }

        // count number of lines in search copy file
        copyFileReader.close();
        copyFileReader = **new** BufferedReader
            (**new** FileReader(_pathToCopyFiles + "search.data"));
        _copySearchFileCount = 0;
        _database.flushFile("search.data");
        while (copyFileReader.readLine() != **null**) {
            _copySearchFileCount++;
        }

        _columnNames = **new** String[] {"copy_search_file_count"};
        _columnValues = **new** String[] { Integer.toString(_copySearchFileCount)};

        _database.setCopySearchFileCount(_copySearchFileCount);

        if (_activeSearchesBefore + _newSearchesProcessed ==
            _activeSearchesProcessed + _copySearchFileCount) {
            _successStatus = "Lines␣in␣search␣copy␣file␣counted";
        }
        else {
            _successStatus = "Number␣of␣lines␣in␣search␣copy␣file␣does␣" +
                "not␣match␣number␣of␣valid␣searches␣processed";
            _etlSuccess = **false**;
            match = **false**;
        }

        _database.updateAuditDimension(_auditKey, _columnNames,
                                       _columnValues, _successStatus);
        if (! match) {
            _etlSuccess = **false**;
            throw new AuditMismatchException(_successStatus);
        }

        // count number of lines in page event copy file
        copyFileReader.close();
        copyFileReader = **new** BufferedReader
            (**new** FileReader(_pathToCopyFiles + "page_event.data"));
        _copyPageEventFileCount = 0;
        _database.flushFile("page_event.data");
        while (copyFileReader.readLine() != **null**) {
            _copyPageEventFileCount++;
        }

        copyFileReader.close();
        _database.setCopyPageEventFileCount(_copyPageEventFileCount);

        _columnNames = **new** String[] {"copy_page_event_file_count"};
        _columnValues = **new** String[] { Integer.toString(_copyPageEventFileCount)};

        _database.setCopyPageEventFileCount(_copyPageEventFileCount);

        if (_activePageEventsBefore + _newPageEventsProcessed ==
            _activePageEventsProcessed + _copyPageEventFileCount) {
            _successStatus = "Lines␣in␣page_event␣copy␣file␣counted";
        }
        else {
            _successStatus = "Number␣of␣lines␣in␣page_event␣copy␣file␣" +
                "does␣not␣match␣number␣of␣valid␣page_events␣processed";
            _etlSuccess = **false**;
            match = **false**;
        }

        _database.updateAuditDimension(_auditKey, _columnNames,
                                       _columnValues, _successStatus);
        if (! match) {
            _etlSuccess = **false**;
            throw new AuditMismatchException(_successStatus);
        }
    }
    catch (Exception e) {
        System.out.println(e);
    }

    // return currentLineNumber;
    // Skriv fÃ¸rst antallet af gyldige og ugyldige processerede log
    // linier i stabil hukommelse, eller send med.

    System.out.println("Start␣copying␣to␣database");

    String [] tablesToCopy = **new** String[] { "log_line", "session",
                                             "page_event", "search",
                                             "active_session", "active_search",
                                             "active_page_event"};
    _etlSuccess = _database.copyToDatabase(tablesToCopy, _auditKey);

    System.out.println("End␣copying␣to␣database");

```
 // count new records in log_line table
_newLogLineRecords = _database.getNumberOfLogLinesFromFile(filename);

_columnNames = new String[] {"new_log_line_records" };
_columnValues = new String [] { Integer . toString (_newLogLineRecords)};

 if ( _validLinesProcessed == _newLogLineRecords) {
      _successStatus = "New␣records␣in␣log_line␣table␣counted";
 }
 else {
      _successStatus = "Number␣of␣new␣records␣in␣log_line␣table␣does␣" +
            "not␣match␣number␣of␣valid␣log␣lines␣processed";
      _etlSuccess = false ;
      match = false ;
 }

_database .updateAuditDimension(_auditKey, _columnNames, _columnValues,
                                        _successStatus );
 if (! match) {
      _etlSuccess = false ;
      throw new AuditMismatchException(_successStatus);
}

 // count new records in session table
/∗
   _newSessionRecords = _database . getNumberOfSessionsFromRun(auditKey);

   _columnNames = new String[] {"new_log_line_records "};
   _columnValues = new String [] { Integer . toString (_newLogLineRecords)};

    if ( _validLinesProcessed == _newLogLineRecords) {
    _successStatus = " New records in log_line table counted";
    }
    else {
    _successStatus = " Number of new records in log_line table does " +
   "not match number of valid log lines processed ";
   match = false ;
    }

   _database . updateAuditDimension(_auditKey, _columnNames, _columnValues,
    _successStatus );
    if (! match) {
   throw new AuditMismatchException(_successStatus);
    }

    // count new records in search table
   _newLogLineRecords = _database.getNumberOfLogLinesFromFile(filename);
   _columnNames = new String[] {"new_log_line_records "};
   _columnValues = new String [] { Integer . toString (_newLogLineRecords)};

    if ( _validLinesProcessed == _newLogLineRecords) {
    _successStatus = " New records in log_line table counted";
    }
    else {
    _successStatus = " Number of new records in log_line table does " +
   "not match number of valid log lines processed ";
   match = false ;
    }

   _database . updateAuditDimension(_auditKey, _columnNames, _columnValues,
    _successStatus );
    if (! match) {
   throw new AuditMismatchException(_successStatus);
    }

    // count new records in page_event table
   _newLogLineRecords = _database.getNumberOfLogLinesFromFile(filename);

   _columnNames = new String[] {"new_log_line_records "};
   _columnValues = new String [] { Integer . toString (_newLogLineRecords)};

    if ( _validLinesProcessed == _newLogLineRecords) {
    _successStatus = " New records in log_line table counted";
    }
    else {
    _successStatus = " Number of new records in log_line table does " +
   "not match number of valid log lines processed ";
   match = false ;
    }

   _database . updateAuditDimension(_auditKey, _columnNames, _columnValues,
    _successStatus );
    if (! match) {
   throw new AuditMismatchException(_successStatus);
    }
∗/

// System.out. println ("Number of new lines in the log_line table : " +
//                _newLogLineRecords);

/∗ boolean noProblems = _fileLineCount == _totalLinesProcessed &&
      _totalLinesProcessed == ( _validLinesProcessed +
      _invalidLinesProcessed ) &&
      _validLinesProcessed == _linesInCopyFile &&
      _linesInCopyFile ==_newLogLineRecords;
```

```
                if (noProblems) {
                _successStatus = "ETL process succesfully finished .";
                }
                else {
                _successStatus = "Problem during ETL process. Data not correct .";
                }*/
        }
        catch (AuditMismatchException ame) {
            System.out.println ("Audit␣mismatch");
            errorHandling ("audit␣mismatch");
        }
        catch (Exception e) { // hvis der sker en fejl i læsningen af en fil
            System.out.println ("PostProcessor.insertLogFile()␣ failed ..." );
            System.out.println (e);
            for (int s = 0; s < e.getStackTrace().length; s++)
                System.out.println (e.getStackTrace()[s]);
        }
        return _etlSuccess;
    }

    void setAuditKey(int newAuditKey) {
        _auditKey = newAuditKey;
    }

    /**
     * Presents the query result as a table in html.
     */
    private void viewQueryResult(String query) {
        try {
            ResultSet resultSet = _database.query(query);

            if (resultSet != null) {
                ResultSetMetaData metaData = resultSet.getMetaData();

                _out.println ("<table␣border=\"1\"␣cellspacing=\"0\">");
                _out.println ("<tr>");
                for (int c = 1; c <= metaData.getColumnCount(); c++) {
                    _out.println ("<th>" + metaData.getColumnName(c) + "</th>");
                }
                _out.println ("</tr>");
                while (resultSet.next()) {
                    _out.println ("<tr>");
                    for (int c = 1; c <= metaData.getColumnCount(); c++) {
                        _out.println ("<td>" + resultSet.getString(c) + "</td>");
                    }
                    _out.println ("</tr>");
                }
                _out.println ("</table>");
            }
        } catch (Exception e) {
            _out.println ("PostProcessor.viewQueryResult␣failed ..." );
            _out.println (e);
        }
    }

    public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {

        long startTime = System.currentTimeMillis();
        System.out.println ("\n\n\nNEW␣RUN␣AT␣" + (new java.util.Date(startTime)));
        System.out.println ();

        response.setContentType("text/html");
        _out = response.getWriter();

        if (_runningOn.equals("baerbar")) {
            System.out.println ("Running␣on␣baerbar");
            _pathToCopyFiles = "/var/lib/pgsql/";
            // _database = new Database("test", "louise", _pathToCopyFiles);
            _database = new Database("aub", "aub", _pathToCopyFiles);
            _filePath = "/home/louise/projekt/logfiles2/";
        }
        else if (_runningOn.equals("stationaer")) {
            System.out.println ("Running␣on␣stationaer");
            _pathToCopyFiles = "/pack/postgres/";
            _database = new Database("aub", "louise", _pathToCopyFiles);
            // _database = new Database("aubtest", "louise", _pathToCopyFiles);
            // _database = new Database("test", "louise", _pathToCopyFiles);
            _filePath = "/pack/louise/projekt/logfiles/";
        }
        else {
            System.out.println ("ERROR!␣WHERE␣AM␣I␣RUNNING???");
        }

        String task = (String) request.getParameter("task");
        String query = (String) request.getParameter("query");
        if (query == null) {
            query = "SELECT␣*␣FROM␣log_line";
        }
        String filename = (String) request.getParameter("filename");
        if (filename == null)
            filename = "access_log.20030225";

        int logLineFromLine = 0;
        int logLineToLine = 1000;
        String logLineFromLineString = request.getParameter("loglinesfromrow");
        if (logLineFromLineString != null)
            logLineFromLine = Integer.parseInt(logLineFromLineString);
```

```
String logLineToLineString = request.getParameter("loglinestorow");
if (logLineToLineString != null)
    logLineToLine = Integer.parseInt(logLineToLineString);

String fromDateString = request.getParameter("fromdate");
if (fromDateString == null)
    fromDateString = "2003−02−25";
java.sql.Date fromDate = java.sql.Date.valueOf(fromDateString);
String toDateString = request.getParameter("todate");
if (toDateString == null)
    toDateString = "2004−07−01";
java.sql.Date toDate = java.sql.Date.valueOf(toDateString);

_out.println("<html>");
_out.println("<head><title>PostProcessor␣on␣" + _runningOn + "</title></head>");
_out.println("<body>");
_out.println("<table␣cellpadding=\"10\"␣border=\"1\">");
_out.println("<tr>");
_out.println("<td>");
_out.println("<form␣action=\"PostProcessor\"␣method=\"get\">");
_out.println("<input␣type=\"hidden\"␣name=\"task\"␣value=\"insertloglines\">");
_out.println("Insert␣rows␣from␣<input␣type=\"text\"␣" +
             "name=\"loglinesfromrow\"␣value=\"" +
             logLineFromLine + "\"␣size=\"10\">" +
             "␣to␣<input␣type=\"text\"␣name=\"loglinestorow\"␣value=\"" +
             logLineToLine + "\"␣size=\"10\">");
_out.println("␣from␣file␣<input␣type=\"text\"␣name=\"filename\"␣value=\"" +
             filename + "\">");
_out.println("<input␣type=\"submit\"␣value=\"Insert\">");
_out.println("</form>");
_out.println("</td>");
_out.println("<td␣align=\"center\">");
_out.println("<form␣action=\"PostProcessor\"␣method=\"get\">");
_out.println("<input␣type=\"hidden\"␣name=\"task\"␣value=\"emptyloglines\">");
_out.println("<input␣type=\"submit\"␣value=\"Empty␣log␣lines\">");
_out.println("</form>");
_out.println("</td>");
_out.println("</tr>");
_out.println("<tr>");
_out.println("<td␣colspan=\"2\">");
_out.println("<form␣action=\"PostProcessor\"␣method=\"get\">");
_out.println("<input␣type=\"hidden\"␣name=\"task\"␣" +
             "value=\"insertallloglines\">");
_out.println("<input␣type=\"submit\"␣value=\"Insert␣All␣Log␣Files\">");
_out.println("</form>");
_out.println("</td>");
_out.println("</tr>");
_out.println("<tr>");
_out.println("<td␣colspan=\"2\">");
_out.println("<form␣action=\"PostProcessor\"␣method=\"get\">");
_out.println("<input␣type=\"hidden\"␣name=\"task\"␣value=\"query\">");
_out.println("<textarea␣name=\"query\"␣cols=\"50\"␣rows=\"5\">" +
             query + "</textarea>");
_out.println("<input␣type=\"submit\"␣value=\"Query\">");
_out.println("</form>");
_out.println("</td>");
_out.println("</tr>");
_out.println("<tr>");
_out.println("<td>");
_out.println("<form␣action=\"PostProcessor\"␣method=\"get\">");
_out.println("<input␣type=\"hidden\"␣name=\"task\"␣value=\"loaddates\">");
_out.println("<input␣type=\"text\"␣name=\"fromdate\"␣value=\"2003−02−25\">");
_out.println("<input␣type=\"text\"␣name=\"todate\"␣value=\"2004−07−01\">");
_out.println("<input␣type=\"submit\"␣value=\"Load␣dates\">");
_out.println("</form>");
_out.println("</td>");
_out.println("<td>");
_out.println("<form␣action=\"PostProcessor\"␣method=\"get\">");
_out.println("<input␣type=\"hidden\"␣name=\"task\"␣value=\"loadtimes\">");
_out.println("<input␣type=\"submit\"␣value=\"Load␣times\">");
_out.println("</form>");
_out.println("</td>");
_out.println("</tr>");
_out.println("<tr>");
_out.println("<td>");
_out.println("<form␣action=\"PostProcessor\"␣method=\"get\">");
_out.println("<input␣type=\"hidden\"␣name=\"task\"␣value=\"dimensionalize\">");
_out.println("<input␣type=\"submit\"␣value=\"Dimensionalize\">");
_out.println("</form>");
_out.println("</td>");
_out.println("<td>");
_out.println("<form␣action=\"PostProcessor\"␣method=\"get\">");
_out.println("<input␣type=\"hidden\"␣name=\"task\"␣value=\"reconnect\">");
_out.println("<input␣type=\"submit\"␣value=\"Reconnect␣to␣database\">");
_out.println("</form>");
_out.println("</td>");
_out.println("</tr>");
_out.println("<tr>");
_out.println("<td␣colspan=\"2\">");
_out.println("<form␣action=\"PostProcessor\"␣method=\"get\">");
_out.println("<input␣type=\"hidden\"␣name=\"task\"␣" +
             "value=\"insertanddimensionalize\">");
_out.println("<input␣type=\"submit\"␣value=\"Insert␣and␣Dimensionalize\">");
_out.println("</form>");
_out.println("</td>");
_out.println("</tr>");
_out.println("</table>");
_out.flush();
```

```
        if ( _database.openConnection()) {
            if ( task != null ) {
                if ( task.equals(" insertloglines ")) {
                    System.out.println ("task␣ insertloglines ␣not␣available");
                    /*
                      _logLineKey = _database.getMaxLogLineKey() + 1;
                      // _lastEmptiedLogLineKey = 0;
                      try {
                      BufferedReader reader = new BufferedReader(new FileReader
                      ( _filePath +
                      filename ));
                      int linesInserted = insertLogFile ( reader , filename ,
                      logLineFromLine,
                      logLineToLine);
                      _out.println ("Read to line " + linesInserted +
                      " in the log file .");
                      }
                      catch ( FileNotFoundException fnfe ) {
                      _out.println (" File " + filename + " not found in " +
                      _filePath );
                      }
                    */
                }
                else if ( task.equals(" insertallloglines ")) {
                    System.out.println ("task␣ insertallloglines ␣not␣available");
                    /*
                      _logLineKey = _database.getMaxLogLineKey() + 1;
                      // _lastEmptiedLogLineKey = 0;
                      BufferedReader[] fileReaders = getBufferedLogFileReaders
                      ( _filePath );
                      for ( int r = 0; r < fileReaders.length ; r++) {
                      // BufferedReader reader = new BufferedReader(new FileReader
                      //                                  ( _filePath +
                      //                                    _filenames[s ]));
                       insertLogFile ( fileReaders [r ], _filenames[r ], 0, 10000000);
                      }
                    */
                }
                else if ( task.equals(" insertanddimensionalize ")) {
                    _logLineKey = _database.getMaxLogLineKey() + 1;
                    _database.setSessionKey ();
                    loadAndDimensionalize( _filePath );
                    // BufferedReader[] fileReaders =
                    // getBufferedLogFileReaders( _filePath );
                    // for ( int r = 0; r < fileReaders.length ; r++) {
                    // BufferedReader reader = new BufferedReader(new FileReader
                    //                              ( _filePath +
                    //                                _filenames[s ]));
                    // insertLogFile ( fileReaders [r ], _filenames[r ], 0, 10000000);
                    //}
                }
                else if ( task.equals("query")) {
                    viewQueryResult(query);
                }
                else if ( task.equals(" loaddates ")) {
                    loadDates(fromDate, toDate );
                }
                else if ( task.equals(" loadtimes ")) {
                    loadTimes();
                }
                else if ( task.equals(" dimensionalize ")) {
                    System.out.println ("task␣dimensionalize␣not␣available");
                    /*
                      _out.flush ();
                      dimensionalizeLogLines();
                    */
                }
                else if ( task.equals(" reconnect ")) {
                    _database.closeConnection ();
                    _database.openConnection();
                }
                else
                    _out.println ("Error!␣Task␣does␣not␣exist.<br>");
            }
            _database.closeConnection ();
        }
        else {
            _out.println ("Connection␣to␣database␣failed<br>");
        }
        _out.println ("</body></html>");

        _out.close ();
        long endTime = System.currentTimeMillis ();
        System.out.println ("\n\nFINISHED␣AT␣" + (new java.util.Date(endTime)));
        long totalTime = endTime − startTime;
        System.out.println ("Total␣time:␣" + totalTime + "␣milliseconds␣(" +
                        millisecondsToTime( totalTime ) + ")." );
    }
}
```

E.2 Database Class

```
package aub;

import java.io.*;
import java.lang.*;
import java.net.*;
import java.sql.*;
import java.util.*;

public class Database {
    String _databaseName, _username;
    Connection _connection;
    Statement _statement;
    ResultSet _resultSet;
    int _numberOfOvermathedLogLines;
    int _sessionKey, _pageKey;
    Vector _activePageEvents, _activeSessions, _activeSearches, _pages, _searchTypes;
    FileOutputStream _fileOutputStream;
    OutputStreamWriter _outputStreamWriter;
    PrintWriter _logLineFile, _sessionFile, _pageEventFile, _searchFile,
        _searchTypeFile, _dateFile, _timeOfDayFile, _activePageEventFile,
        _activeSessionFile, _activeSearchFile; //, _pageFile;
    String _pathToCopyFiles;
    int _copyPageEventFileCount, _copySessionFileCount, _copySearchFileCount;
    int _newSessions, _newSearches, _newPageEvents;
    int _newSessionRecords, _newSearchRecords, _newPageEventRecords;
    int _sessionRecordsBefore, _searchRecordsBefore, _pageEventRecordsBefore;
    int _sessionRecordsAfter, _searchRecordsAfter, _pageEventRecordsAfter;
    String [] _columnNames, _columnValues;

    public Database (String databaseName, String username, String pathToCopyFiles) {
        _databaseName = databaseName;
        _username = username;
        _sessionKey = 0;
        _pageKey = 0;
        _numberOfOvermathedLogLines = 0;
        _activePageEvents = new Vector();
        _activeSessions = new Vector();
        _activeSearches = new Vector();
        _pages = new Vector();
        _searchTypes = new Vector();
        _pathToCopyFiles = pathToCopyFiles;

        try {
            resetFiles ();

            // _fileOutputStream = new FileOutputStream(_pathToCopyFiles +
            //                                      "page.data");
            // _outputStreamWriter = new OutputStreamWriter(_fileOutputStream, "UTF-8");
            // _pageFile = new PrintWriter(new BufferedWriter(_outputStreamWriter));

            _fileOutputStream = new FileOutputStream(_pathToCopyFiles + "date.data");
            _outputStreamWriter = new OutputStreamWriter(_fileOutputStream, "UTF-8");
            _dateFile = new PrintWriter(new BufferedWriter(_outputStreamWriter));
            _fileOutputStream = new FileOutputStream(_pathToCopyFiles +
                                                 "time_of_day.data");
            _outputStreamWriter = new OutputStreamWriter(_fileOutputStream, "UTF-8");
            _timeOfDayFile = new PrintWriter(new BufferedWriter(_outputStreamWriter));
        }
        catch (Exception e) {
            System.out.println ("Database␣constructor␣failed ...");
            System.out.println ("Exception:␣" + e);
            System.out.println (e.getMessage());
        }
    }

    public boolean openConnection() {
        boolean success = false;
        try {
            Class.forName("org.postgresql.Driver");
            _connection = DriverManager.getConnection("jdbc:postgresql:" +
                                                      _databaseName,
                                                      _username, "");
            if (_connection != null) {
                System.out.println ("Connection␣open");
                System.out.println ("dababaseName:␣" + _databaseName +
                                    ",␣username:␣" + _username);
                success = true;
            }
            _statement = _connection.createStatement ();
        } catch (SQLException se) {
            System.out.println ("Database.openConnection()␣failed ...");
            System.out.println ("SQLException:␣" + se.getMessage());
            System.out.println ("SQLState:␣␣␣␣␣" + se.getSQLState());
            System.out.println ("VendorError:␣␣" + se.getErrorCode());
        } catch (Exception e) {
            System.out.println ("Database.openConnection()␣failed ...");
            System.out.println (e);
        }
        return success;
    }

    public void closeConnection () {
        try {
            _connection.close ();
        } catch (SQLException se) {
            System.out.println ("Database.closeConnection()␣failed ...");
            System.out.println ("SQLException:␣" + se.getMessage());
            System.out.println ("SQLState:␣␣␣␣␣" + se.getSQLState());
            System.out.println ("VendorError:␣␣" + se.getErrorCode());
        } catch (Exception e) {
            System.out.println ("Database.closeConnection()␣failed ...");
            System.out.println (e);
            System.out. println (e.getMessage());
        }
    }

    public void copyToDatabase(String table ) {
        try {
            PrintWriter file = null;
            if ( table . equals (" log_line " )) {
                file = _logLineFile ;
            }
            else if ( table . equals (" session " )) {
                file = _sessionFile ;
                System.out. println ("SESSION␣FILE");
            }
            else if ( table . equals ("page_event" )) {
                file = _pageEventFile;
            }
            else if ( table . equals (" search" )) {
                file = _searchFile ;
            }
            else if ( table . equals (" search_type" )) {
                file = _searchTypeFile ;
            }
            else if ( table . equals ("time_of_day" )) {
                file = _timeOfDayFile;
            }
            else if ( table . equals (" date" )) {
                file = _dateFile ;
            }
            else if ( table . equals (" active_page_event " )) {
                file = _activePageEventFile ;
            }
            else if ( table . equals (" active_session " )) {
                file = _activeSessionFile ;
            }
            else if ( table . equals (" active_search " )) {
                file = _activeSearchFile ;
            }
            // else if ( table . equals (" page ")) {
            //     file = _pageFile;
            //}
            file . write (" \\.\ n");
            file . flush ();
            file . close ();
            update("COPY␣" + table + "␣FROM␣'"
                    + _pathToCopyFiles + table + ". data '" );
        }
        catch (Exception e ) {
            System.out. println ("Database.copyToDatabase␣failed ..." );
            System.out. println (e );
            System.out. println (e.getMessage());
        }
    }

    public boolean copyToDatabase(String [] tables , int auditKey ) {
        boolean success = true;
        try {
            _connection.setAutoCommit(false);
            deleteActiveInfo ();
            for ( int t = 0; t < tables . length ; t++) {
                PrintWriter file = null;
                if ( tables [ t ]. equals (" log_line " )) {
                    file = _logLineFile ;
                }
                else if ( tables [ t ]. equals (" session " )) {
                    file = _sessionFile ;
                }
                else if ( tables [ t ]. equals ("page_event" )) {
                    file = _pageEventFile;
                }
                else if ( tables [ t ]. equals (" search" )) {
                    file = _searchFile ;
                }
                else if ( tables [ t ]. equals (" search_type" )) {
                    file = _searchTypeFile ;
                }
                else if ( tables [ t ]. equals ("time_of_day" )) {
                    file = _timeOfDayFile;
                }
                else if ( tables [ t ]. equals (" date" )) {
                    file = _dateFile ;
                }
                else if ( tables [ t ]. equals (" active_page_event " )) {
                    file = _activePageEventFile ;
                    // throw new SQLException("Interrupted by user ");
                }
                else if ( tables [ t ]. equals (" active_session " )) {
                    file = _activeSessionFile ;
                }
                else if ( tables [ t ]. equals (" active_search " )) {
                    file = _activeSearchFile ;
                }
                                                                    // else if ( tables [ t ]. equals (" page ")) {
                                                                    //      file = _pageFile;
                                                                    //}
                file . write (" \\.\ n");
                file . flush ();
                file . close ();
                _statement.addBatch("COPY␣" + tables[t] + "␣FROM␣'" +
                                         _pathToCopyFiles + tables[t] + ".data'");
            }
            _statement.executeBatch();

            if (success) {
                completeAuditRecord(auditKey, "ETL␣process␣" +
                                         "completed␣succesfully", "Proceed");
                _connection.commit();
            }
            else {
                _connection.rollback();
            }
        }
        catch (SQLException sqle) {
            try {
                success = false;
                System.out.println("Rollback");
                _connection.rollback();
            }
            catch (Exception e) {
                System.out.println(e);
            }
            success = false;
            System.out.println("Database.copyToDatabase␣failed ...");
            System.out.println(sqle);
            System.out.println(sqle.getMessage());
            System.out.println(sqle.getNextException());
        }
        catch (Exception e) {
            success = false;
            System.out.println("Database.copyToDatabase␣failed ...");
            System.out.println(e);
            System.out.println(e.getMessage());
        }
        try {
            _connection.setAutoCommit(true);
        }
        catch (Exception e) {
            System.out.println(e);
        }
        return success;
    }

    private void deleteActiveInfo() {
        update("DELETE␣FROM␣active_page_event");
        update("DELETE␣FROM␣active_search");
        update("DELETE␣FROM␣active_session");
    }

    /*
     public void emptyCopyFiles() {
     try {
     copyToDatabase("page_event");
     // _fileOutputStream = new FileOutputStream(_pathToCopyFiles +
     //                                          "page_event.data");
     // _outputStreamWriter = new OutputStreamWriter(_fileOutputStream, "UTF-8");
     // _pageEventFile = new PrintWriter(new BufferedWriter(_outputStreamWriter));

     copyToDatabase("search");
     // _fileOutputStream = new FileOutputStream(_pathToCopyFiles + "search.data");
     // _outputStreamWriter = new OutputStreamWriter(_fileOutputStream, "UTF-8");
     // _searchFile = new PrintWriter(new BufferedWriter(_outputStreamWriter));

     copyToDatabase("session");
     // _fileOutputStream = new FileOutputStream(_pathToCopyFiles +
     //                                          "session.data");
     // _outputStreamWriter = new OutputStreamWriter(_fileOutputStream, "UTF-8");
     // _sessionFile = new PrintWriter(new BufferedWriter(_outputStreamWriter));
     }
     catch (Exception e) {
     System.out.println("Database.emptyCopyFiles() failed ...");
     System.out.println(e);
     System.out.println(e.getMessage());
     }
     }
    */

    public void emptyLogLineFile() {
        try {
            copyToDatabase("log_line");
            _fileOutputStream = new FileOutputStream(_pathToCopyFiles +
                                                     "log_line.data");
            _outputStreamWriter = new OutputStreamWriter(_fileOutputStream, "UTF-8");
            _logLineFile = new PrintWriter(new BufferedWriter(_outputStreamWriter));
        }
        catch (Exception e) {
            System.out.println("Database.emptyLogLineFile()␣failed ...");
            System.out.println(e);
            System.out.println(e.getMessage());
        }
    }

    public void execute(String executeString) {
        try {
            _statement = _connection.createStatement();
            _statement.execute(executeString);
        } catch (SQLException se) {
            System.out. println ("Database.execute()␣ failed  ..."  );
            System.out. println ("ExecuteString :␣" +  executeString );
            System.out. println ("SQLException:␣" + se.getMessage());
            System.out. println ("SQLState:␣␣␣␣␣" + se.getSQLState());
            System.out. println ("VendorError:␣␣" + se . getErrorCode ());
        } catch ( Exception  e ) {
            System.out. println ("Database.execute()␣ failed  ..."  );
            System.out. println ("ExecuteString :␣" +  executeString );
            System.out. println ("Statement:␣" +  _statement );
            System.out. println (e );
            System.out. println (e . getMessage ());
        }
    }

    public  void   flushFile ( String   filename ) {
        try {
            if  ( filename . equals (" log_line . data" )) {
                _logLineFile . flush ();
            }
            else   if   ( filename . equals (" session . data" )) {
                _sessionFile . flush ();
            }
            else   if   ( filename . equals ("page_event. data" )) {
                _pageEventFile. flush ();
            }
            else   if   ( filename . equals (" search . data" )) {
                _searchFile . flush ();
            }
            else   if   ( filename . equals (" search_type . data" )) {
                _searchTypeFile . flush ();
            }
            else   if   ( filename . equals ("time_of_day. data" )) {
                _timeOfDayFile. flush ();
            }
            else   if   ( filename . equals (" date . data" )) {
                _dateFile . flush ();
            }
            else   if   ( filename . equals (" active_page_event . data" )) {
                _activePageEventFile . flush ();
            }
            else   if   ( filename . equals (" active_session  . data" )) {
                _activeSessionFile  . flush ();
            }
            else   if   ( filename . equals (" active_search  . data" )) {
                _activeSearchFile  . flush ();
            }
            // else   if   ( filename . equals ("page. data ")) {
            //     _pageFile . flush ();
            //}
        }
        catch  ( Exception  e ) {
            System.out. println ("Database.  flushFile ␣ failed  ..."  );
            System.out. println (e );
            System.out. println (e . getMessage ());
        }
    }

    public  boolean completeAuditRecord(int  auditKey ,  String   successStatus ,
                                        String  proceed ) {
        update ("UPDATE␣audit␣SET␣etl_end_time␣=␣CURRENT_TIMESTAMP,␣" +
                " success_status ␣=␣'" +  successStatus  +  "',␣proceed␣=␣'" +
                proceed  +  "'␣WHERE␣audit_key␣=␣" + auditKey);
        System.out. println ("Success␣status :␣" +  successStatus );
        return  true ;
    }

    public  int   getActivePageEvents () {
        try {
            query("EXECUTE␣select_active_page_events");

            while  (  _resultSet  . next ()) {
                _activePageEvents . add(getPageEventFromResultSet ());
            }
            // update (" DELETE FROM active_page_event");
            _resultSet  . close ();
        }
        catch  ( Exception  e ) {
            System.out. println ("Database. getActivePageEvents()␣ failed  ..."  );
            System.out. println (e );
            System.out. println (e . getMessage ());
        }
        return   _activePageEvents . size ();
    }

    public  int   getActiveSearches () {
        try {
            query("EXECUTE␣select_active_searches");

            while  (  _resultSet  . next ()) {
                _activeSearches . add(getSearchFromResultSet ());
            }
            // update (" DELETE FROM active_search");
            _resultSet  . close ();
        }
        catch  ( Exception  e ) {
            System.out. println ("Database. getActiveSearches ()␣ failed  ..."  );
            System.out. println (e );
            System.out. println (e . getMessage ());
        }
        return _activeSearches . size ();
    }

    public int getActiveSessions () {
        try {
            query("EXECUTE␣select_active_sessions");

            while ( _resultSet . next ()) {
                _activeSessions .add(getSessionFromResultSet ());
            }
            _resultSet . close ();
        }
        catch (Exception e) {
            System.out. println ("Database. getActiveSessions ()␣ failed ..." );
            System.out. println (e);
            System.out. println (e.getMessage());
        }
        return _activeSessions . size ();
    }

    public Audit getAudit(int auditKey) {
        Audit audit = null;
        try {
            query("EXECUTE␣select_audit");

            if ( _resultSet . next ()) {
                audit = new Audit(auditKey,
                                  _resultSet . getString ("filename"),
                                  _resultSet . getString (" etl_start_time "),
                                  _resultSet . getString ("etl_end_time"),
                                  _resultSet . getInt (" log_file_line_count "),
                                  _resultSet . getInt (" total_lines_processed "),
                                  _resultSet . getInt (" valid_lines_processed "),
                                  _resultSet . getInt (" invalid_lines_processed "),
                                  _resultSet . getInt (" copy_log_line_file_count "),
                                  _resultSet . getInt ("new_log_line_records"),
                                  _resultSet . getString (" success_status "),
                                  _resultSet . getString ("proceed"),
                                  _resultSet . getInt ("min_log_line_key"),
                                  _resultSet . getInt ("max_log_line_key"));
            }
            _resultSet . close ();
        }
        catch (Exception e) {
            System.out. println ("Database.getAudit ..." );
            System.out. println (e);
            System.out. println (e.getMessage());
        }
        return audit;
    }

    public Audit [] getAuditForUnfinishedLoads () {
        Audit [] audits = null;
        try {
            query("SELECT␣*␣FROM␣audit␣WHERE␣proceed␣=␣'fix␣this␣before␣proceeding'");

            _resultSet . last ();
            int numberOfRows = _resultSet.getRow();
            audits = new Audit[numberOfRows];
            _resultSet . beforeFirst ();

            int row = 0;
            while ( _resultSet . next ()) {
                audits [row] = new Audit( _resultSet . getInt ("audit_key"),
                                          _resultSet . getString ("filename"),
                                          _resultSet . getString (" etl_start_time "),
                                          _resultSet . getString ("etl_end_time"),
                                          _resultSet . getInt (" log_file_line_count "),
                                          _resultSet . getInt (" total_lines_processed "),
                                          _resultSet . getInt (" valid_lines_processed "),
                                          _resultSet . getInt (" invalid_lines_processed "),
                                          _resultSet . getInt (" copy_log_line_file_count "),
                                          _resultSet . getInt ("new_log_line_records"),
                                          _resultSet . getString (" success_status "),
                                          _resultSet . getString ("proceed"),
                                          _resultSet . getInt ("min_log_line_key"),
                                          _resultSet . getInt ("max_log_line_key"));
                row++;
            }
            _resultSet . close ();
        }
        catch (Exception e) {
            System.out. println ("Database.getAuditForUnfinishedLoads ()..." );
            System.out. println (e);
            System.out. println (e.getMessage());
        }
        return audits;
    }

    public int getDateKey(java. sql .Date date) {
        int dateKey = −1;

        try {
            query("EXECUTE␣select_date_key␣('" + date + "')");

            if ( _resultSet . next ()) {
                dateKey = _resultSet . getInt ("date_key");
            }
            _resultSet . close ();
        } catch (Exception e) {
            System.out. println ("Database.getDateKey()␣failed ..." );
            System.out. println (e);
            System.out. println (e.getMessage());
        }
        return dateKey;
    }

    public String getLastSuccessfulLoadFilename () {
        String filename = null;
        try {
            query("SELECT␣filename␣FROM␣audit␣WHERE␣proceed␣=␣'Proceed'␣"+
                    "ORDER␣BY␣audit_key␣DESC␣LIMIT␣1");

            if ( _resultSet .next ()) {
                filename = _resultSet . getString (1);
            }
        }
        catch (Exception e) {
            System.out. println ("Database.getLastSuccessfulLoadFilename()␣failed ..." );
            System.out. println (e);
            System.out. println (e.getMessage());
        }
        return filename;
    }

    public String [] getLoadedLogFileNames() {
        String [] filenames = null;
        try {
            query("SELECT␣filename␣FROM␣audit␣WHERE␣proceed␣=␣'Proceed'␣"+
                    "ORDER␣BY␣audit_key");

            _resultSet . last ();
            int numberOfRows = _resultSet.getRow();
            filenames = new String[numberOfRows];
            _resultSet . beforeFirst ();

            for (int f = 0; f < numberOfRows && _resultSet.next(); f++) {
                filenames[f] = _resultSet . getString (1);
            }
        }
        catch (Exception e) {
            System.out. println ("Exception␣in␣Database.getLoadedLogFileNames...");
            System.out. println (e);
            System.out. println (e.getMessage());
        }
        return filenames;
    }

    public LogLine getLogLine(int logLineKey) {
        LogLine logLine = null;
        try {
            query("EXECUTE␣select_log_line␣(" + logLineKey + ")");

            if ( _resultSet .next ()) {
                logLine = getLogLineFromResultSet();
            }
            _resultSet . close ();
        } catch (Exception e) {
            System.out. println ("Database.getLogLine()␣failed ..." );
            System.out. println (e);
            System.out. println (e.getMessage());
        }
        return logLine;
    }

    public LogLine[] getLogLines(int numberOfLogLines, int firstLogLineKey ) {
        LogLine[] logLineArray = new LogLine[numberOfLogLines];
        try {
            int lastLogLineKey = (( firstLogLineKey / numberOfLogLines) *
                                      numberOfLogLines) + numberOfLogLines − 1;
            query("EXECUTE␣select_log_lines␣(" + firstLogLineKey + ",␣" +
                    lastLogLineKey + ")" );
            for (int l = firstLogLineKey; l <= lastLogLineKey; l++) {
                if ( _resultSet .next ()) {
                    logLineArray[l%numberOfLogLines] = getLogLineFromResultSet();
                }
            }
            _resultSet . close ();
        } catch (Exception e) {
            System.out. println ("Database.getLogLine()␣failed ..." );
            System.out. println (e);
            System.out. println (e.getMessage());
        }
        return logLineArray;
    }

    private LogLine getLogLineFromResultSet() {
        LogLine logLine = null;
        try {
            int logLineKey = _resultSet . getInt ("log_line_key");
            String filename = _resultSet . getString ("filename");
            int logLineNumber = _resultSet . getInt ("log_line_number");
            String ipAddress = _resultSet . getString ("ip_address");
            String ident = _resultSet . getString ("ident");
            String authuser = _resultSet . getString ("authuser");
            java . sql . Date  date  =  _resultSet  . getDate( " date " );
            Time time  =  _resultSet  . getTime("time" );
            String  timezone =  _resultSet  . getString ( "timezone" );
            String  method =  _resultSet  . getString ( "method");
            String  requestUrl  =  _resultSet  . getString ( " request_url " );
            String  sessionTag  =  _resultSet  . getString ( " session_tag " );
            String  serial  =  _resultSet  . getString ( " serial " );
            String  query =  _resultSet  . getString ( "query" );
            String  protocol  =  _resultSet  . getString ( " protocol " );
            int  status  =  _resultSet  . getInt ( " status " );
            int  bytes  =  _resultSet  . getInt ( " bytes " );
            String  servername =  _resultSet  . getString ( "servername" );
            String  referrer  =  _resultSet  . getString ( " referrer " );
            String  browser =  _resultSet  . getString ( "browser" );

            logLine  =  new LogLine(logLineKey, filename ,  logLineNumber, ipAddress,
                                   ident ,  authuser ,  date ,  time ,  timezone ,  method,
                                   requestUrl ,  sessionTag ,  serial  ,  query ,  protocol ,
                                   status  ,  bytes ,  servername ,  referrer  ,  browser );
        }
        catch  ( Exception  e ) {
            System.out . println ( "Database.getLogLineFromResultSet()␣ failed  ..."  );
            System.out . println ( e );
            System.out . println ( e . getMessage ());
        }
        return  logLine ;
    }

    public  int  getMaxAuditKey() {
        int  maxAuditKey = 0;
        try  {
            query( "SELECT␣max(audit_key)␣FROM␣audit");
            if  (  _resultSet  . next ()) {
                maxAuditKey = _resultSet . getInt  (1);
            }
            _resultSet  . close ();
        }  catch  ( Exception  e ) {
            System.out . println ( "Database.getMaxAuditKey␣failed ..." );
            System.out . println ( e );
            System.out . println ( e . getMessage ());
        }
        return  maxAuditKey;
    }

    public  int  getMaxLogLineKey() {
        int  maxLogLineKey = −1;
        try  {
            query( "SELECT␣max(log_line_key)␣FROM␣log_line");
            if  (  _resultSet  . next ()) {
                maxLogLineKey = _resultSet. getInt  (1);
            }
            _resultSet  . close ();
        }  catch  ( Exception  e ) {
            System.out . println ( "Database.getMaxLogLineKey␣failed..." );
            System.out . println ( e );
            System.out . println ( e . getMessage ());
        }
        return  maxLogLineKey;
    }

    public  int  getMaxLogLineNumber() {
        int  maxLogLineNumber = −1;
        try  {
            query( "SELECT␣max(log_line_number)␣FROM␣log_line");
            if  (  _resultSet  . next ()) {
                maxLogLineNumber = _resultSet.getInt (1);
            }
            _resultSet  . close ();
        }  catch  ( Exception  e ) {
            System.out . println ( "Database.getMaxLogLineNumber␣failed...");
            System.out . println ( e );
            System.out . println ( e . getMessage ());
        }
        return  maxLogLineNumber;
    }

    public  void  setSessionKey () {
        int  maxSessionKey = 0;
        int  maxActiveSessionKey = 0;
        try  {
            query( "SELECT␣max(session_key)␣FROM␣session");
            if  (  _resultSet  . next ()) {
                maxSessionKey =  _resultSet . getInt  (1);
            }
            query( "SELECT␣max(session_key)␣FROM␣active_session");
            if  (  _resultSet  . next ()) {
                maxActiveSessionKey =  _resultSet . getInt  (1);
            }
        }
        catch  ( Exception  e ) {
            System.out . println ( "Database.getMaxSessionKey␣failed ..." );
            System.out . println ( e );
            System.out . println ( e . getMessage ());
        }
        if  ( maxSessionKey > maxActiveSessionKey) { //brug max−funktion
            _sessionKey  =  maxSessionKey;
        }
        else  {
            _sessionKey = maxActiveSessionKey;
        }
    }

    public int [] getMinAndMaxLogLineKey() {
        int [] minAndMax = {0, 0};
        try {
            query("SELECT␣min(log_line_key),␣max(log_line_key)␣FROM␣log_line");
            if ( _resultSet .next ()) {
                minAndMax[0] = _resultSet. getInt (1);
                minAndMax[1] = _resultSet. getInt (2);
            }
            _resultSet . close ();
        } catch ( Exception e ) {
            System.out. println ("Database.getMinLogLineKey␣failed..." );
            System.out. println (e );
            System.out. println (e .getMessage ());
        }
        return minAndMax;
    }

    public int getNumberOfLogLinesFromFile(String filename) {
        int numberOfLogLines = 0;
        try {
            query("SELECT␣count(∗)␣FROM␣log_line␣WHERE␣filename␣=␣'" +
                    filename + "'" );
            if ( _resultSet .next ()) {
                numberOfLogLines = _resultSet . getInt (1);
            }
        }
        catch ( Exception e ) {
            System.out. println ("Database.getNumberOfLogLinesFromFile␣failed..." );
            System.out. println (e );
            System.out. println (e .getMessage ());
        }
        return numberOfLogLines;
    }

    public int getNumberOfNewPageEvents() {
        return _newPageEvents;
    }

    public int getNumberOfNewSearches() {
        return _newSearches;
    }

    public int getNumberOfNewSessions() {
        return _newSessions;
    }

    public int getNumberOfPageEventsAfter(int auditKey ) {
        _pageEventRecordsAfter = 0;
        try {
            query("SELECT␣count(∗)␣FROM␣page_event");
            if ( _resultSet . next ()) {
                _pageEventRecordsAfter = _resultSet . getInt (1);
            }
        }
        catch ( Exception e ) {
            System.out. println ("Database.getNumberOfPageEventsAfter␣failed ..." );
            System.out. println (e );
            System.out. println (e .getMessage ());
        }
        return _pageEventRecordsAfter;
    }

    public int getNumberOfSearchesAfter(int auditKey ) {
        _searchRecordsAfter = 0;
        try {
            query("SELECT␣count(∗)␣FROM␣search");
            if ( _resultSet . next ()) {
                _searchRecordsAfter = _resultSet . getInt (1);
            }
        }
        catch ( Exception e ) {
            System.out. println ("Database.getNumberOfSearchesAfter␣failed ..." );
            System.out. println (e );
            System.out. println (e .getMessage ());
        }
        return _searchRecordsAfter ;

    }

    public int getNumberOfSessionsAfter(int auditKey ) {
        int _sessionRecordsAfter = 0;
        try {
            query("SELECT␣count(∗)␣FROM␣session");
            if ( _resultSet . next ()) {
                _sessionRecordsAfter = _resultSet . getInt (1);
            }
        }
        catch ( Exception e ) {
            System.out. println ("Database.getNumberOfSessionsAfter␣failed ..." );
            System.out. println (e );
            System.out. println (e .getMessage ());
        }
        return _sessionRecordsAfter ;
    }

    public int getNumberOfPageEventsBefore() {
        _pageEventRecordsBefore = 0;
        try {
            query("SELECT␣page_event_records_after␣FROM␣audit␣WHERE␣proceed␣=␣" +
                  "'Proceed'␣ORDER␣BY␣audit_key␣DESC␣LIMIT␣1");
            if ( _resultSet .next ()) {
                _pageEventRecordsBefore = _resultSet . getInt (" page_event_records_after " );
            }
        }
        catch (Exception e) {
            System.out. println ("Database.getNumberOfPageEventsBefore␣failed ..." );
            System.out. println (e );
            System.out. println (e.getMessage ());
        }
        return _pageEventRecordsBefore;
    }

    public int getNumberOfSearchesBefore() {
        _searchRecordsBefore = 0;
        try {
            query("SELECT␣search_records_after␣FROM␣audit␣WHERE␣proceed␣=␣" +
                  "'Proceed'␣ORDER␣BY␣audit_key␣DESC␣LIMIT␣1");
            if ( _resultSet .next ()) {
                _searchRecordsBefore = _resultSet . getInt (" search_records_after " );
            }
        }
        catch (Exception e) {
            System.out. println ("Database.getNumberOfSearchesBefore␣failed ..." );
            System.out. println (e );
            System.out. println (e.getMessage ());
        }
        return _searchRecordsBefore;

    }

    public int getNumberOfSessionsBefore() {
        _sessionRecordsBefore = 0;
        try {
            query("SELECT␣session_records_after␣FROM␣audit␣WHERE␣proceed␣=␣" +
                  "'proceed'␣ORDER␣BY␣audit_key␣DESC␣LIMIT␣1");
            if ( _resultSet .next ()) {
                _sessionRecordsBefore = _resultSet . getInt (" session_records_after " );
            }
        }
        catch (Exception e) {
            System.out. println ("Database.getNumberOfSessionsBefore␣failed ..." );
            System.out. println (e );
            System.out. println (e.getMessage ());
        }
        return _sessionRecordsBefore;

    }

    public PageEvent getPageEventFromResultSet() {
        PageEvent pageEvent = null;
        try {
            int logLineKey = _resultSet . getInt ("log_line_key" );
            int dateKey = _resultSet . getInt ("date_key" );
            int timeOfDayKey = _resultSet. getInt ("time_of_day_key");
            int pageKey = _resultSet . getInt ("page_key");
            int sessionKey = _resultSet . getInt ("session_key" );
            int auditKey = _resultSet . getInt ("audit_key" );

            pageEvent = new PageEvent(logLineKey, dateKey, timeOfDayKey,
                                      pageKey, sessionKey, auditKey);
        }
        catch (Exception e) {
            System.out. println ("Database.getPageEventFromResultSet()␣failed ..." );
            System.out. println (e );
            System.out. println (e.getMessage ());
        }
        return pageEvent;
    }



    public int getPageKey(Page page) {
        try {
            for ( int p = 0; p < _pages. size (); p++) {
                Page pageInVector = ( Page) _pages. get(p);
                if ( page.getPageFunction (). equals(pageInVector.getPageFunction()) &&
                    page.getPageFunctionType (). equals (pageInVector
                                                        .getPageFunctionType()) &&
                    page. getProcess (). equals(pageInVector. getProcess ())) {
                    return pageInVector.getPageKey();
                }
            }
            _pageKey++;
            page.setPageKey(_pageKey);
            _pages.add(page);
            System.out. println ("No␣entry␣for␣page␣\"" + page + "\"␣in␣database." );
            // _pageFile . write (page. toString () + "\ n");
            return 1; // _pageKey;
        }
        catch (Exception e) {
            System.out. println ("Database.getPageKey(Page)␣failed ..." );
            System.out.println(e);
            System.out.println(e.getMessage());
        }
        return -1;
    }

    public void getPages() {
        try {
            query("EXECUTE␣select_pages");
            while (_resultSet.next()) {
                Page newPage = new Page(_resultSet.getInt("page_key"),
                                        _resultSet.getString("page_function"),
                                        _resultSet.getString("page_function_type"),
                                        _resultSet.getString("process"));
                _pages.add(newPage);
                if (newPage.getPageKey() > _pageKey) {
                    _pageKey = newPage.getPageKey();
                }
            }
        }
        catch (Exception e) {
            System.out.println("Database.getPages()␣ failed ...");
            System.out.println(e);
            System.out.println(e.getMessage());
        }
    }

    public QueryQuestion[] getQueryQuestions() {
        QueryQuestion[] queryQuestions = new QueryQuestion[0];
        try {
            query("select_query_questions␣AS␣SELECT␣*␣FROM␣" +
                  "query_questions␣ORDER␣BY␣id");

            _resultSet.last();
            int numberOfRows = _resultSet.getRow();
            queryQuestions = new QueryQuestion[numberOfRows];
            _resultSet.beforeFirst();

            while (_resultSet.next()) {
                QueryQuestion newQueryQuestion =
                    new QueryQuestion(_resultSet.getInt("id"),
                                      _resultSet.getString("question"),
                                      _resultSet.getString("query"));
                queryQuestions[_resultSet.getRow() - 1] = newQueryQuestion;
            }
        }
        catch (Exception e) {
            System.out.println("Database.getQueryQuestions()␣ failed ...");
            System.out.println(e);
            System.out.println(e.getMessage());
        }
        return queryQuestions;
    }

    public Search getSearch(int sessionKey) {
        Search search = null;
        try {
            for (int s = 0; s < _activeSearches.size(); s++) {
                search = (Search) _activeSearches.get(s);

                // Tjek evt. at den ikke er null
                if (search.getSessionKey() == sessionKey) {
                    return search;
                }
            }
        } catch (Exception e) {
            System.out.println("Database.getSearch(sessionKey)␣ failed ...");
            System.out.println(e);
            System.out.println(e.getMessage());
        }
        return null;
    }

    public Search getSearch(int sessionKey, int searchNumber,
                            String searchNumberValidity) {
        Search search = null;
        try {
            for (int s = 0; s < _activeSearches.size(); s++) {
                search = (Search) _activeSearches.get(s);
                if (search.getSessionKey() == sessionKey &&
                    search.getSearchNumber() == searchNumber &&
                    search.getSearchNumberValidity().equals(searchNumberValidity)) {
                    return search;
                }
            }
        } catch (Exception e) {
            System.out.println("Database.getSearch(sessionKey,␣searchNumber,␣" +
                               "searchNumberValidity)␣ failed ...");
            System.out.println(e);
            System.out.println(e.getMessage());
        }
        return null;
    }

    private Search getSearchFromResultSet() {
        Search search = null;
        try {
            int dateKey = _resultSet.getInt("date_key");
            int timeOfDayKey = _resultSet. getInt ("time_of_day_key");
            int sessionKey = _resultSet . getInt ("session_key" );
            int searchTypeKey = _resultSet . getInt ("search_type_key" );
            int searchNumber = _resultSet . getInt ("search_number");
            String searchNumberValidity = _resultSet . getString
                ("search_number_validity" );
            int numberOfBookDescriptions = _resultSet . getInt
                ("number_of_book_descriptions" );
            int numberOfBooksInBasket = _resultSet. getInt ("number_of_books_in_basket");
            int numberOfReservations = _resultSet . getInt ("number_of_reservations" );

            search = new Search(dateKey, timeOfDayKey, sessionKey, searchTypeKey,
                                 searchNumber, searchNumberValidity,
                                 numberOfBookDescriptions, numberOfBooksInBasket,
                                 numberOfReservations);
        }
        catch (Exception e) {
            System.out. println ("Database.getSearchFromResultSet()␣failed ..." );
            System.out. println (e );
            System.out. println (e.getMessage ());
        }
        return search;
    }

    public int getSearchTypeKey(SearchType searchType) {
        try {
            for ( int s = 0; s < _searchTypes. size (); s++) {
                SearchType searchTypeInVector = ( SearchType) _searchTypes. get(s );
                if ( searchTypeInVector .getTypeWithField ()
                    . equals (searchType. getTypeWithField ())) {
                    return searchTypeInVector .getSearchTypeKey();
                }
            }
        } catch (Exception e) {
            System.out. println ("Database.getSearchTypeKey()␣failed ..." );
            System.out. println (e );
            System.out. println (e.getMessage ());
        }
        return 1;
    }

    public void getSearchTypes () {
        try {
            query("EXECUTE␣select_search_types");
            while ( _resultSet .next ()) {
                SearchType searchType =
                    new SearchType(_resultSet
                                  . getInt ("search_type_key" ),
                                  _resultSet . getString ("type" ),
                                  _resultSet . getString (" field " ),
                                  _resultSet
                                  . getString (" type_with_field " ));
                _searchTypes.add(searchType );
            }
        }
        catch (Exception e) {
            System.out. println ("Database.getSearchTypes()␣failed ..." );
            System.out. println (e );
            System.out. println (e.getMessage ());
        }
    }

    public Session getSession (int sessionKey) {
        Session session = null;

        try {
            for ( int s = 0; s < _activeSessions . size () && session == null ; s++) {
                if ((( Session ) _activeSessions . get (s )). getSessionKey () == sessionKey) {
                    session = ( Session ) _activeSessions . get (s );
                }
            }
        } catch (Exception e) {
            System.out. println ("Database. getSession ()␣failed ..." );
            System.out. println (e );
            System.out. println (e.getMessage ());
        }
        return session ;
    }

    private Session getSessionFromResultSet () {
        Session session = null;
        try {
            int sessionKey = _resultSet . getInt ("session_key" );
            String sessionTag = _resultSet . getString (" session_tag " );
            String ipAddress = _resultSet . getString (" ip_address" );
            String browser = _resultSet . getString ("browser" );
            String firstRequestUrl = _resultSet . getString (" first_request_url " );
            int firstPageKey = _resultSet . getInt (" first_page_key " );
            String lastRequestUrl = _resultSet . getString (" last_request_url " );
            int lastPageKey = _resultSet . getInt ("last_page_key" );
            String referrer = _resultSet . getString (" referrer " );
            java . sql .Date startDate = _resultSet .getDate(" start_date " );
            int startDateKey = _resultSet . getInt (" start_date_key " );
            Time startTime = _resultSet .getTime(" start_time " );
            int startTimeKey = _resultSet . getInt (" start_time_key " );
            java . sql .Date endDate = _resultSet .getDate("end_date" );
            int endDateKey = _resultSet . getInt ("end_date_key");
            Time endTime = _resultSet .getTime("end_time");
```

```
                int endTimeKey = _resultSet.getInt("end_time_key");
                int pagesInSession = _resultSet.getInt("pages_in_session");
                int bookDescriptionsInSession =
                    _resultSet.getInt(" book_descriptions_in_session ");
                int booksInBasketInSession =
                    _resultSet.getInt(" books_in_basket_in_session ");
                int reservationsInSession = _resultSet.getInt(" reservations_in_session ");
                int lastSearchNumber = _resultSet.getInt("last_search_number");
                String searchNumberValid = _resultSet.getString(" search_number_validity ");

                session = new Session(sessionKey, sessionTag, ipAddress, browser,
                                      firstRequestUrl, firstPageKey, lastRequestUrl,
                                      lastPageKey, referrer, startDate, startDateKey,
                                      startTime, startTimeKey, endDate, endDateKey,
                                      endTime, endTimeKey, pagesInSession,
                                      bookDescriptionsInSession,
                                      booksInBasketInSession, reservationsInSession,
                                      lastSearchNumber, searchNumberValid);
            }
            catch (Exception e) {
                System.out.println("Database.getSessionFromResultSet()␣ failed ...");
                System.out.println(e);
                System.out.println(e.getMessage());
            }
            return session;
        }

        public int getSessionKey(LogLine logLine, int pageKey, int dateKey, int timeKey) {
            int logLineSessionKey = −1;
            Session currentSession;
            if (logLine == null) {
                return −1;
            }
            try {
                if (! logLine.hasSessionTag()) {
                    _sessionKey++;
                    _activeSessions.add(new Session(_sessionKey, logLine, pageKey,
                                                    dateKey, timeKey));
                    _newSessions++;
                    logLineSessionKey = _sessionKey;
                }
                else {
                    long longLogLineTime = logLine.getDate().getTime() +
                        logLine.getTime().getTime();
                    for (int v = 0; v < _activeSessions.size() &&
                            logLineSessionKey == −1; v++) {
                        currentSession = (Session) _activeSessions.get(v);
                        if (currentSession.getEndLong() < longLogLineTime − 1200000) {
                            // Terminated session found
                            _sessionFile.write(currentSession.toString() + "\n");
                            removeInactivePageEvents(currentSession.getSessionKey());
                            removeInactiveSearches(currentSession.getSessionKey());
                            _activeSessions.remove(v);
                            v−−;
                            Search inactiveSearch = getSearch
                                (currentSession.getSessionKey(),
                                 currentSession.getLastSearchNumber(),
                                 currentSession.getSearchNumberValidity());
                            if (inactiveSearch != null) {
                                _searchFile.write(inactiveSearch.toString() + "\n");
                                _activeSearches.remove(inactiveSearch);
                            }
                        }
                        else if (currentSession.matches(logLine)) {
                            currentSession.addPageEvent(logLine, pageKey,
                                                        dateKey, timeKey);
                            logLineSessionKey = currentSession.getSessionKey();
                        }
                    }
                    if (logLineSessionKey == −1) {
                        _sessionKey++;
                        _activeSessions.add(new Session(_sessionKey, logLine, pageKey,
                                                        dateKey, timeKey));
                        _newSessions++;
                        logLineSessionKey = _sessionKey;
                    }
                }
            } catch (Exception e) {
                System.out.println("Database.getSessionKey()␣ failed ...");
                System.out.println(e);
                System.out.println(e.getMessage());
            }
            return logLineSessionKey;
        }

        public int getTimeKey(Time sqlTime) {
            try {
                String timeString = sqlTime.toString();
                int hour = Integer.parseInt(timeString.substring(0,2));
                int minute = Integer.parseInt(timeString.substring(3,5));
                int second = Integer.parseInt(timeString.substring(6));

                int timeKey = 1 + second + (60 ∗ minute) + (60 ∗ 60 ∗ hour);
                return timeKey;
            }
            catch (Exception e) {
                System.out.println("Database.getTimeKey()␣failed ...");
                System.out.println(e);
            System.out. println (e.getMessage ());
        }
        return −1;
    }

    public boolean incrementBookDescriptions( int sessionKey ) {
        boolean success = false ;
        try {
            Session session = getSession (sessionKey );
            if ( session == null ) {
                System.out. println ("Session␣is␣null␣in␣" +
                                    "incrementNumberOfBookDescriptions(" +
                                    sessionKey + ")" );
            }
            else {
                int searchNumber = session .getLastSearchNumber();
                String searchNumberValidity = session .getSearchNumberValidity ();
                Search search = getSearch (sessionKey , searchNumber,
                                           searchNumberValidity );
                if ( search != null ) {
                    search .incrementNumberOfBookDescriptions(1);
                    session . incrementBookDescriptionsInSession (1);
                    return true;
                }
            }
        }
        catch ( Exception e ) {
            System.out. println ("Database.incrementNumberOfBookDescriptions()␣" +
                                 " failed ..." );
            System.out. println (e );
            System.out. println (e .getMessage ());
        }
        return false ;
    }

    public boolean incrementBooksInBasket(int sessionKey , int amount) {
        boolean success = false ;
        try {
            Session session = getSession (sessionKey );
            int searchNumber = session .getLastSearchNumber();
            String searchNumberValidity = session .getSearchNumberValidity ();

            Search search = getSearch (sessionKey , searchNumber, searchNumberValidity );
            if ( search != null ) {
                search .incrementNumberOfBooksInBasket(amount);
                session .incrementBooksInBasketInSession(amount);
                return true;
            }
        }
        catch ( Exception e ) {
            System.out. println ("Database.incrementNumberOfBooksInBasket()␣" +
                                 " failed ..." );
            System.out. println (e );
            System.out. println (e .getMessage ());
        }
        return false ;
    }

    public boolean incrementReservations ( int sessionKey ) {
        boolean success = false ;
        try {
            Session session = getSession (sessionKey );
            int searchNumber = session .getLastSearchNumber();
            String searchNumberValidity = session .getSearchNumberValidity ();

            Search search = getSearch (sessionKey , searchNumber, searchNumberValidity );
            if ( search != null ) {
                search .incrementNumberOfReservations(1);
                session . incrementReservationsInSession (1);
                return true;
            }
        }
        catch ( Exception e ) {
            System.out. println ("Database.incrementNumberOfReservations" +
                                 " failed ..." );
            System.out. println (e );
            System.out. println (e .getMessage ());
        }
        return success;
    }

    public boolean insertDate (Date date ) {
        boolean success = false ;
        try {
            _dateFile . write (date . toString () + "\n");
        } catch ( Exception e ) {
            System.out. println ("Database. insertDate ()␣ failed ..." );
            System.out. println (e );
            System.out. println (e .getMessage ());
        }
        return success;
    }

    public boolean insertLogLine (LogLine logLine ) {
        boolean success = false ;
        try {
            if ( logLine .getLogLineKey() % 1000 == 0) {
                System.out. println ("Log␣line␣key:␣" + logLine .getLogLineKey());
            }
            _logLineFile.write(logLine.toString() + "\n");
            success = true;
        } catch (Exception e) {
            System.out.println("Database.insertLogLine()␣failed ...");
            System.out.println(e);
            System.out.println(e.getMessage());
        }
        return success;
    }

    public boolean insertPageEvent(PageEvent pageEvent, LogLine logLine) {
        boolean success = false;
        try {
            _activePageEvents.add(pageEvent);
            _newPageEvents++;
            success = true;
        } catch (Exception e) {
            System.out.println("Database.insertPageEvent()␣failed ...");
            System.out.println(e);
            System.out.println(e.getMessage());
        }
        return success;
    }

    public boolean insertSearch(Search search) {
        try {
            Session activeSession = getSession(search.getSessionKey());
            if (activeSession != null && activeSession.getLastSearchNumber() != -1) {
                Search activeSearch = getSearch(search.getSessionKey());
                if (activeSearch != null) {
                    activeSearch.deactivate();
                }
            }
            _activeSearches.add(search);
            _newSearches++;
            activeSession.setLastSearchNumber(search.getSearchNumber());
            activeSession.setSearchNumberValidity
                (search.getSearchNumberValidity());
            activeSession.incrementSearchesInSession(1);
            return true;
        } catch (Exception e) {
            System.out.println("Database.insertSearch()␣failed ...");
            System.out.println(e);
            System.out.println(e.getMessage());
        }
        return false;
    }

    public boolean insertTime(Time sqlTime, int hour, int minute, int second,
                              String workingHours, String periodOfDay) {
        boolean success = false;
        try {
            int timeKey = getTimeKey(sqlTime);
            _timeOfDayFile.write(timeKey + "\t" +
                                 sqlTime + "\t" +
                                 hour + "\t" +
                                 minute +"\t" +
                                 second +"\t" +
                                 workingHours +"\t" +
                                 periodOfDay + "\n");
            return true;
        }
        catch(Exception e) {
            System.out.println("database.insertTime()␣failed ...");
            System.out.println(e);
            System.out.println(e.getMessage());
        }
        return false;
    }

    public int newAuditRecord(String filename, String successStatus,
                              String proceed) {
        int maxAuditKey = getMaxAuditKey();
        update("INSERT␣INTO␣audit␣(audit_key,␣filename,␣etl_start_time,␣" +
                " success_status,␣proceed)␣VALUES␣(" + (maxAuditKey + 1) + ",␣'" +
                filename + "',␣CURRENT_TIMESTAMP,␣'" + successStatus + "',␣'"
                + proceed + "')");
        return (maxAuditKey + 1);
    }

    public void prepareStatements() {
        try {
            execute("PREPARE␣select_date_key␣(DATE)␣AS␣SELECT␣date_key␣FROM␣date␣" +
                    "WHERE␣sql_date␣=␣$1");
            execute("PREPARE␣select_time_of_day_key␣(TIME)␣AS␣" +
                    "SELECT␣time_of_day_key␣FROM␣time_of_day␣WHERE␣sql_time␣=␣$1");
            execute("PREPARE␣select_log_line␣(INTEGER)␣AS␣SELECT␣*␣FROM␣log_line␣" +
                    "WHERE␣log_line_key␣=␣$1");
            execute("PREPARE␣select_log_lines␣(INTEGER,␣INTEGER)␣AS␣SELECT␣*␣FROM␣" +
                    " log_line␣WHERE␣log_line_key␣BETWEEN␣$1␣AND␣$2");
            execute("PREPARE␣select_pages␣AS␣SELECT␣*␣FROM␣page");
            execute("PREPARE␣select_search_types␣AS␣SELECT␣*␣FROM␣search_type");
            execute("PREPARE␣select_page_key␣(VARCHAR(60),␣VARCHAR(30),␣VARCHAR(30))" +
                    "␣AS␣SELECT␣page_key␣FROM␣page␣WHERE␣page_function␣=␣$1␣AND␣" +
                    "page_function_type␣=␣$2␣AND␣process␣=␣$3");
            execute("PREPARE␣select_search_type_key␣(VARCHAR(18),␣VARCHAR(18))␣AS␣" +
                    "SELECT␣search_type_key␣FROM␣search_type␣" +
                    "WHERE␣type␣=␣$1␣AND␣field␣=␣$2");
            execute ("PREPARE␣select_audit␣(INTEGER)␣AS␣SELECT␣filename,␣" +
                    " etl_start_time ,␣etl_end_time ,␣ log_file_line_count ,␣" +
                    " total_lines_processed ,␣ valid_lines_processed ,␣" +
                    " invalid_lines_processed ,␣ copy_log_line_file_count ,␣" +
                    "new_log_line_records,␣ success_status ,␣proceed␣" +
                    "FROM␣audit␣" +
                    "WHERE␣audit_key␣=␣$1");
            execute ("PREPARE␣select_active_searches␣AS␣SELECT␣*␣FROM␣active_search␣" +
                    "ORDER␣BY␣date_key,␣time_of_day_key");
            execute ("PREPARE␣select_active_sessions␣AS␣SELECT␣*␣FROM␣active_session␣" +
                    "ORDER␣BY␣end_date,␣end_time");
            execute ("PREPARE␣select_active_page_events␣AS␣SELECT␣*␣FROM␣" +
                    " active_page_event␣ORDER␣BY␣date_key,␣time_of_day_key");
            // execute ("PREPARE select_query_questions AS SELECT * FROM " +
            //    "query_questions  ORDER BY id");
        }
        catch (Exception e) {
            System.out. println ("database . prepareStatements␣failed  ..." );
            System.out. println (e);
            System.out. println (e.getMessage ());
        }
    }

    public ResultSet query( String queryString ) {
        try {
            _statement = _connection. createStatement ();
            _resultSet = _statement .executeQuery( queryString );
        }
        catch (SQLException se) {
            System.out. println ("Database.query()␣ failed  ..." );
            System.out. println ("SQLException:␣" + se.getMessage ());
            System.out. println ("SQLState:␣␣␣␣␣" + se.getSQLState());
            System.out. println ("VendorError:␣␣" + se . getErrorCode ());
            System.out. println ("query:␣" + queryString );
        }
        catch (Exception e) {
            System.out. println ("Database.query()␣ failed  ..." );
            System.out. println (e);
            System.out. println (e. getMessage ());
            System.out. println ("Query:␣" + queryString );
        }
        return _resultSet ;
    }

    public void removeInactivePageEvents( int inactiveSessionKey ) {
         // System.out. println ("RemoveInactivePageEvents(" + inactiveSessionKey  + ")");
        PageEvent pageEvent = null;
        try {
            for ( int p = 0; p < _activePageEvents . size (); p++) {
                pageEvent = ( PageEvent) _activePageEvents . get (p);
                if ( pageEvent.getSessionKey () == inactiveSessionKey ) {
                    _pageEventFile . write (pageEvent. toString () + "\n");
                    _activePageEvents .remove(p);
                    p−−;
                }
            }
        }
        catch (Exception e) {
            System.out. println ("Database.removeInactivePageEvents()␣ failed  ..." );
            System.out. println (e);
            System.out. println (e.getMessage ());
        }
    }

    public void removeInactiveSearches ( int inactiveSessionKey ) {
         // System.out. println ("RemoveInactiveSearches(" + inactiveSessionKey  + ")");
        Search search = null;
        try {
            for ( int s = 0; s < _activeSearches . size (); s++) {
                search = ( Search ) _activeSearches . get (s );

                // Tjek evt . at den ikke er null
                if ( search .getSessionKey () == inactiveSessionKey ) {
                    _searchFile . write (search . toString () + "\n");
                    _activeSearches .remove(s);
                    s−−;
                }
            }
        }
        catch (Exception e) {
            System.out. println ("Database. removeInactiveSearches ()␣ failed  ..." );
            System.out. println (e);
            System.out. println (e.getMessage ());
        }
    }

    public void resetFiles () {
        try {
            if ( _logLineFile != null ) {
                _logLineFile . close ();
            }
            _fileOutputStream = new FileOutputStream(_pathToCopyFiles +
                                                      " log_line . data" );
            _outputStreamWriter = new OutputStreamWriter(_fileOutputStream , "UTF−8");
            _logLineFile = new PrintWriter (new BufferedWriter(_outputStreamWriter ));

            if ( _sessionFile != null ) {
```

```
            _sessionFile . close ();
        }
        _fileOutputStream = new FileOutputStream(_pathToCopyFiles +
                                                  " session . data" );
        _outputStreamWriter = new OutputStreamWriter(_fileOutputStream , "UTF−8");
        _sessionFile = new PrintWriter (new BufferedWriter(_outputStreamWriter ));

        if ( _pageEventFile != null ) {
            _pageEventFile. close ();
        }
        _fileOutputStream = new FileOutputStream(_pathToCopyFiles +
                                                  "page_event. data" );
        _outputStreamWriter = new OutputStreamWriter(_fileOutputStream , "UTF−8");
        _pageEventFile = new PrintWriter (new BufferedWriter(_outputStreamWriter ));

        if ( _searchFile != null ) {
            _searchFile . close ();
        }
        _fileOutputStream = new FileOutputStream(_pathToCopyFiles + "search . data" );
        _outputStreamWriter = new OutputStreamWriter(_fileOutputStream , "UTF−8");
        _searchFile = new PrintWriter (new BufferedWriter(_outputStreamWriter ));

        if ( _activePageEventFile != null ) {
            _activePageEventFile . close ();
        }
        _fileOutputStream = new FileOutputStream(_pathToCopyFiles +
                                                  " active_page_event . data" );
        _outputStreamWriter = new OutputStreamWriter(_fileOutputStream , "UTF−8");
        _activePageEventFile = new PrintWriter (new BufferedWriter
            (_outputStreamWriter ));

        if ( _activeSessionFile != null ) {
            _activeSessionFile . close ();
        }
        _fileOutputStream = new FileOutputStream(_pathToCopyFiles +
                                                  " active_session . data" );
        _outputStreamWriter = new OutputStreamWriter(_fileOutputStream , "UTF−8");
        _activeSessionFile = new PrintWriter (new BufferedWriter
            (_outputStreamWriter ));

        if ( _activeSearchFile != null ) {
            _activeSearchFile . close ();
        }
        _fileOutputStream = new FileOutputStream(_pathToCopyFiles +
                                                  " active_search . data" );
        _outputStreamWriter = new OutputStreamWriter(_fileOutputStream , "UTF−8");
        _activeSearchFile = new PrintWriter (new BufferedWriter
            (_outputStreamWriter ));
    }
        catch ( Exception e ) {
            System.out. println (" database . resetFiles ()␣ failed ..." );
            System.out. println ("Exception:␣" + e );
            System.out. println (e . getMessage ());
        }
    }

    public void resetNewInfo () {
        _newSessions = 0;
        _newSearches = 0;
        _newPageEvents = 0;
    }

    public int saveActivePageEvents () {
        // System.out. println ("Number of active page events :" + _activePageEvents . size ());
        int activePageEvents = 0;
        try {
            while ( _activePageEvents . size () > 0) {
                _activePageEventFile . write ((( PageEvent) _activePageEvents . get (0)). toString () +
                                             "\n" );
                _activePageEvents . remove(0);
                activePageEvents++;
            }
        }
        catch ( Exception e ) {
            System.out. println ("Database.saveActivePageEvents()␣ failed ..." );
            System.out. println (e );
            System.out. println (e . getMessage ());
        }
        return activePageEvents ;
    }

    public int saveActiveSearches () {
        int activeSearches = 0;
        try {
            while ( _activeSearches . size () > 0) {
                _activeSearchFile . write ((( Search) _activeSearches . get (0)). toString () +
                                          "\n" );
                _activeSearches .remove(0);
                activeSearches ++;
            }
        }
        catch ( Exception e ) {
            System.out. println ("Database. saveActiveSearches ()␣ failed ..." );
            System.out. println (e );
            System.out. println (e . getMessage ());
        }
        return activeSearches ;
    }
```

```
public int saveActiveSessions () {
    int activeSessions = 0;
    try {
        while ( _activeSessions . size () > 0) {
            _activeSessionFile . write ((( Session ) _activeSessions . get (0))
                                     . activeSessionToString () + "\n");
            _activeSessions .remove(0);
            activeSessions ++;
        }
    }
    catch ( Exception e ) {
        System.out. println ("Database. saveActiveSessions ()␣ failed ..." );
        System.out. println (e );
        System.out. println (e .getMessage ());
    }
    return activeSessions ;
}

public void setCopyPageEventFileCount(int newFileCount) {
    _copyPageEventFileCount = newFileCount;
}

public void setCopySearchFileCount( int newFileCount) {
    _copySearchFileCount = newFileCount;
}

public void setCopySessionFileCount( int newFileCount) {
    _copySessionFileCount = newFileCount;
}

public boolean setLastSearchNumber(int sessionKey , int searchNumber,
                                   String searchNumberValidity ) {
    boolean success = false ;
    try {
        Session session = getSession (sessionKey );
        int sessionSearchNumber = −1;
        String sessionSearchNumberValidity = "";
        if ( session == null ) {
            System.out. println ("Error␣in␣setLastSearchNumber:␣Can␣not␣find␣" +
                                 " session␣with␣key␣" + sessionKey);
            return false ;
        }
        else {
            sessionSearchNumber = session .getLastSearchNumber();
            sessionSearchNumberValidity = session .getSearchNumberValidity ();
            if (( searchNumberValidity . equals (" valid ") &&
                  sessionSearchNumberValidity . equals ("temporary" )) ||
                 (searchNumberValidity . equals (" valid ") &&
                  sessionSearchNumberValidity . equals (" valid ") &&
                  searchNumber != sessionSearchNumber)) {
                session .setLastSearchNumber(searchNumber);
                session . setSearchNumberValidity (searchNumberValidity );
            }
            else {
                return false ;
            }
        }
        Search search = getSearch (sessionKey , sessionSearchNumber,
                                   sessionSearchNumberValidity );
        if ( search == null ) {
            if ( sessionSearchNumber != −1) {
                System.out. println ("Error␣in␣setLastSearchNumber:␣Can␣not␣find␣" +
                                     "search␣with␣sessionKey␣" + sessionKey +
                                     ",␣search␣number␣" + sessionSearchNumber +
                                     "␣and␣searchNumberValidity␣" +
                                     sessionSearchNumberValidity );
            }
            return false ;
        }
        else {
            search .setSearchNumber(searchNumber);
            search . setSearchNumberValidity (searchNumberValidity );
        }
        return true;
    } catch ( Exception e ) {
        System.out. println ("Database. getSearch (setLastSearchNumber()␣ failed ..." );
        System.out. println (e );
        System.out. println (e .getMessage ());
    }
    return false ;
}

public int update( String updateString ) {
    int rowsUpdated = 0;
    try {
                // System.out. println ( updateString );
        rowsUpdated = _statement .executeUpdate( updateString );
    } catch ( SQLException se) {
        System.out. println ("Database.update()␣ failed ..." );
        System.out. println ("UpdateString:␣" + updateString );
        System.out. println ("SQLException:␣" + se.getMessage ());
        System.out. println ("SQLState:␣␣␣␣␣" + se.getSQLState());
        System.out. println ("VendorError:␣␣" + se . getErrorCode ());
    } catch ( Exception e ) {
        System.out. println ("Database.update()␣ failed ..." );
        System.out. println ("UpdateString:␣" + updateString );
        System.out. println ("Statement:␣" + _statement );
```

```
            System.out. println (e );
            System.out. println (e.getMessage());
        }
        return rowsUpdated;
    }

    public int updateAuditDimension(int auditKey , String [] columnNames,
                                    String [] newValues, String newSuccessStatus) {
        /∗System.out. println ("updateAuditDimension:");
          for ( int c = 0; c < columnNames.length; c++) {
          System.out. println (columnNames[c ] + ": " + newValues[c ]);
          }∗/
        System.out. println ("Success␣status:␣" + newSuccessStatus);

        String updateString = "UPDATE␣audit␣SET";
        // try {
        for ( int c = 0; c < columnNames.length; c++) {
            updateString += "␣" + columnNames[c] + "␣=␣'" + newValues[c] + " ',";
        }
        updateString += "␣success_status␣=␣'" + newSuccessStatus +
            "'␣WHERE␣audit_key␣=␣" + auditKey;
        // System.out. println ( updateString );
        return update( updateString );
        /∗}
          catch ( SQLException se) {
          System.out. println ("Database.update () failed ...");
          System.out. println ("UpdateString : " + updateString );
          System.out. println ("SQLException: " + se .getMessage());
          System.out. println ("SQLState:        " + se .getSQLState ());
          System.out. println ("VendorError:    " + se .getErrorCode ());
          } catch ( Exception e ) {
          System.out. println ("Database.update () failed ...");
          System.out. println ("UpdateString : " + updateString );
          System.out. println ("Statement : " + _statement );
          System.out. println (e );
          System.out. println (e.getMessage());
          }∗/
    }
}
```

E.3 DataExtractor Class

```
package aub;

import java. text .SimpleDateFormat;
import java. util .Date;
import java. util .Map;
import java.io. Serializable ;
import org. jfree . data . CategoryDataset;
import org. jfree . data . DefaultCategoryDataset ;
import de. laures .cewolf. DatasetProduceException;
import de. laures .cewolf. DatasetProducer;
import de. laures .cewolf.CategoryItemLinkGenerator;

import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;
import org. jfree . chart . entity . CategoryItemEntity ;
import org. jfree . chart . tooltips .CategoryToolTipGenerator;

import aub.∗;
import java. sql .∗;
import java. io .∗;

public class DataExtractor implements DatasetProducer, Serializable {

    // These values would normally not be hard coded but produced by
    // some kind of data source like a database or a file
    private String [] categories ;
    private String [] seriesNames;
    private String xAxisLabel = "X␣axis";
    private ResultSet resultSet ;
    private String _resultDescription = "no␣description";
    private String _query = " default␣query";
    private Database _database = null;


    private Database getDatabase () {
        if ( _database == null ) {
            // _database = new Database("aub ", " aub ", "/ var/ lib /pgsql /");// baerbar
            _database = new Database("aub", " louise " , "/pack/ postgres /" ); // stationaer
            _database . openConnection();
            System.out. println ( _database );
        }

        return _database;

    }

    private void closeDatabase () {
        if ( _database != null ) {
            System.out. println ("Closing␣database␣connection");
            _database . closeConnection ();
            _database = null;
        }
    }

    public void setGraphParameters () {
        try {
            Database database = getDatabase();

            System.out.println(_query);
            resultSet = database.query(_query);
            System.out.println( resultSet );

            resultSet.last();
            int numberOfRows = resultSet.getRow();
            categories = new String[numberOfRows];
            System.out.println(numberOfRows);
            resultSet.beforeFirst();

            ResultSetMetaData metaData = resultSet.getMetaData();
            int numberOfColumns = metaData.getColumnCount();
            if (numberOfColumns < 2) {
                throw (new Exception("Error!␣The␣resultset␣must␣have␣at␣least␣two␣columns"));
            }
            xAxisLabel = metaData.getColumnName(1);
            seriesNames = new String[numberOfColumns − 1];
            for (int s = 1; s < numberOfColumns; s++) {
                seriesNames[s − 1] = metaData.getColumnName(s + 1);
            }
        }
        catch (SQLException sqle) {
            System.out.println("SQLException␣caught:␣" + sqle.getMessage());
        }
        catch (NullPointerException npe) {
            System.out.println("NullPointerException␣caught:␣" + npe.getMessage());
        }
        catch (Exception e) {
            System.out.println("Exception␣caught:␣" + e.getMessage());
        }

    }

    public Object produceDataset(Map params) throws DatasetProduceException {
        DefaultCategoryDataset dataset = new DefaultCategoryDataset();
        try {
            int c = 0;
            while (resultSet.next()) {
                // for (int c = 0; c < columnNames.length; c++) {
                System.out.println("Value:␣" + resultSet.getFloat(2));
                categories[c] = resultSet.getString(1);
                dataset.addValue((double) resultSet.getFloat(2), seriesNames[0], categories[c]);
                c++;
            }
        }
        catch (SQLException sqle) {
            System.out.println("SQLException␣caught:␣" + sqle.getMessage());
            sqle.printStackTrace();
        }
        catch (NullPointerException npe) {
            System.out.println("NullPointerException␣caught:␣" + npe.getMessage());
        }
        catch (Exception e) {
            System.out.println("Exception␣caught:␣" + e.getMessage());
        }

        closeDatabase();
        return dataset;
    }

    public boolean hasExpired(Map params, Date since) {
        // log.debug(getClass().getName() + "hasExpired()");
        return (System.currentTimeMillis() − since.getTime()) > 5000;
    }

    public String getProducerId() {
        return "PageViewCountData␣DatasetProducer";
    }


    public String getType() {
        return "verticalbar";
    }

    public String getXAxis() {
        return xAxisLabel;
    }

    public String getYAxis() {
        return "Antal";
    }


    public void setResultDescription(String r) {
        _resultDescription = r;
    }

    public String getResultDescription() {
        return _resultDescription;
    }

    public void setQuery(String q) {
        _query = q;
        System.out.println("query␣has␣been␣set");
    }
    public void setQueryKeys(String measure, String dimension, String attribute ) {
        Database database = getDatabase ();  // The database is closed by produceDataset

        String query =
            " select ␣ result ,query␣from␣gui_queries␣where␣measure='"+measure+"'"+
            "␣and␣dimension='"+dimension+"'"+
            "␣and␣ attribute ='"+ attribute +"';" ;

        System.out. println (query );

        try {

            ResultSet result = database .query(query );
            System.out. println ( result );

            if ( result . first ()) {
                String resultDescription = result . getString (" result " );
                setResultDescription ( resultDescription );

                String graphQuery = result . getString ("query" );
                setQuery(graphQuery);

            } else {
                System.out. println ("No␣query␣has␣been␣set");
            }

        } catch ( SQLException sqle) {
            System.out. println ("SQLException␣caught:␣" + sqle.getMessage ());
            sqle . printStackTrace ();
        }
        catch ( NullPointerException npe) {
            System.out. println (" NullPointerException ␣caught:␣" + npe.getMessage ());
            npe. printStackTrace ();
        }
        catch ( Exception e ) {
            System.out. println ("Exception␣caught:␣" + e.getMessage ());
            e. printStackTrace ();
        }


    }


}
```

E.4 Date Class

```
package aub;

import java. util .*;

public class Date {

    private GregorianCalendar _calendar , _firstDayOfSemester;
    private String _weekDay, _semester, _weekend, _exam, _publicHoliday,
        _schoolVacation , _workday;
    private int _dateKey, _year , _month, _day,_dayOfSemester, _weekOfSemester,
        _dayOfYear, _weekOfYear;
    long _millisIntoSemester ;
    private java . sql . Date _sqlDate;

    public Date ( int dateKey, GregorianCalendar calendar ) {
        _dateKey = dateKey;
        _calendar = calendar ;
    }

    public java . sql . Date getSqlDate () {
        _sqlDate = new java. sql . Date(_calendar . getTimeInMillis ());
        return _sqlDate;
    }

    public int getYear () {
        _year = _calendar . get ( _calendar . YEAR);
        return _year;
    }

    public int getMonth() {
        _month = _calendar . get ( _calendar . MONTH) + 1;
        return _month;
    }

    public int getDay () {
        _day = _calendar . get ( _calendar . DAY_OF_MONTH);
        return _day;
    }

    public String getWeekDay() {
        if ( _calendar . get ( _calendar . DAY_OF_WEEK) == _calendar.MONDAY) {
            _weekDay = "monday";
        }
        else if ( _calendar . get ( _calendar . DAY_OF_WEEK) == _calendar.TUESDAY) {
            _weekDay = "tuesday";
        }
        else if ( _calendar . get ( _calendar . DAY_OF_WEEK) == _calendar.WEDNESDAY) {
            _weekDay = "wednesday";
        }
        else if ( _calendar . get ( _calendar . DAY_OF_WEEK) == _calendar.THURSDAY) {
            _weekDay = "thursday";
        }
        else if ( _calendar . get ( _calendar . DAY_OF_WEEK) == _calendar.FRIDAY) {
            _weekDay = "friday";
        }
        else if ( _calendar . get ( _calendar . DAY_OF_WEEK) == _calendar.SATURDAY) {
            _weekDay = "saturday";
        }
        else if ( _calendar . get ( _calendar . DAY_OF_WEEK) == _calendar.SUNDAY) {
            _weekDay = "sunday";
        }
        else {
            _weekDay = "unknown";
        }
        return _weekDay;
    }

    public String getSemester () {
        if ( _calendar . after (new GregorianCalendar(2003, 0, 31)) &&
             _calendar . before (new GregorianCalendar (2003, 8, 1))) {
            _semester = "spring␣2003";
        }
        else if ( _calendar . after (new GregorianCalendar(2003, 7, 31)) &&
                  _calendar . before (new GregorianCalendar (2004, 1, 1))) {
            _semester = " fall ␣2003";
        }
        else if ( _calendar . after (new GregorianCalendar(2004, 0, 31)) &&
                  _calendar . before (new GregorianCalendar (2004, 8, 1))) {
            _semester = "spring␣2004";
        }
        else {
            _semester = "unknown";
        }
        return _semester;
    }

    public GregorianCalendar getFirstDayOfSemester () {
        if ( _calendar . after (new GregorianCalendar(2003, 0, 31)) &&
             _calendar . before (new GregorianCalendar (2003, 8, 1))) {
            _firstDayOfSemester = new GregorianCalendar (2003, 1, 1);
        }
        else if ( _calendar . after (new GregorianCalendar(2003, 7, 31)) &&
                  _calendar . before (new GregorianCalendar (2004, 1, 1))) {
            _firstDayOfSemester = new GregorianCalendar (2003, 8, 1);
        }
        else if ( _calendar . after (new GregorianCalendar(2004, 0, 31)) &&
                  _calendar . before (new GregorianCalendar (2004, 8, 1))) {
            _firstDayOfSemester = new GregorianCalendar (2004, 1, 1);
        }
        else {
            _firstDayOfSemester = null;
        }
        return _firstDayOfSemester;
    }

    public int getDayOfSemester() {
         _millisIntoSemester = _calendar . getTime (). getTime() −
             getFirstDayOfSemester (). getTime (). getTime ();
        _dayOfSemester = (int ) ( _millisIntoSemester / (24∗60∗60∗1000)) + 1;
        return _dayOfSemester;
    }

    public int getWeekOfSemester() {
        _weekOfSemester = ((getDayOfSemester () − 1) / 7) + 1;
        return _weekOfSemester;
    }

    public String getWeekend() {
        getWeekDay();
        if ( _weekDay.equals("saturday" ) || _weekDay.equals("sunday")) {
            _weekend = "weekend";
        }
        else {
            _weekend = "week␣day";
        }
        return _weekend;
    }

    public String getExam() {
        getMonth();
        if ( _month == 1 || _month == 6) {
            _exam = "exam";
        }
        else if ( _month == 8) {
            _exam = "reexam";
        }
        else {
            _exam = "no␣exam";
        }
        return _exam;
    }

    public String getPublicHoliday () {
        GregorianCalendar [] publicHolidays = { new GregorianCalendar (2003, 0, 1),
                                                new GregorianCalendar (2003, 3, 13),
                                        new GregorianCalendar (2003, 3, 17),
                                        new GregorianCalendar (2003, 3, 18),
                                        new GregorianCalendar (2003, 3, 20),
                                        new GregorianCalendar (2003, 3, 21),
                                        new GregorianCalendar (2003, 4, 16),
                                        new GregorianCalendar (2003, 4, 29),
                                        new GregorianCalendar (2003, 5, 5),
                                        new GregorianCalendar (2003, 5, 8),
                                        new GregorianCalendar (2003, 5, 9),
                                        new GregorianCalendar (2003, 11, 25),
                                        new GregorianCalendar (2003, 11, 26),
                                        new GregorianCalendar (2004, 0, 1),
                                        new GregorianCalendar (2004, 3, 4),
                                        new GregorianCalendar (2004, 3, 8),
                                        new GregorianCalendar (2004, 3, 9),
                                        new GregorianCalendar (2004, 3, 11),
                                        new GregorianCalendar (2004, 3, 12),
                                        new GregorianCalendar (2004, 4, 7),
                                        new GregorianCalendar (2004, 4, 20),
                                        new GregorianCalendar (2004, 4, 30),
                                        new GregorianCalendar (2004, 4, 31),
                                        new GregorianCalendar (2004, 5, 5)};
        for ( int g = 0; g < publicHolidays.length && _publicHoliday == null ; g++) {
            if ( _calendar.equals( publicHolidays[g ])) {
                _publicHoliday = "holiday";
            }
        }
        if ( _publicHoliday == null ) {
            _publicHoliday = "no␣holiday";
        }
        return _publicHoliday;
    }

    public String getSchoolVacation () {
        if (( _calendar. after (new GregorianCalendar(2003, 5, 30)) &&
              _calendar. before (new GregorianCalendar  (2003, 7, 1))) ||
            ( _calendar. after (new GregorianCalendar(2004, 5, 30)) &&
              _calendar. before (new GregorianCalendar (2004, 7, 1)))){
            _schoolVacation = "vacation";
        }
        else {
            _schoolVacation = "no␣vacation";
        }
        return _schoolVacation;
    }

    public int getDayOfYear() {
        _dayOfYear = _calendar.get( _calendar.DAY_OF_YEAR);
        return _dayOfYear;
    }

    public int getWeekOfYear() {
        _calendar.setMinimalDaysInFirstWeek(4);
        _weekOfYear = _calendar.get( _calendar.WEEK_OF_YEAR);
        return _weekOfYear;
    }

    public String getWorkday() {
        _workday = "workday";
        if ( getWeekend().equals("weekend" ) ||
             getPublicHoliday (). equals("holiday" ) ||
             getSchoolVacation (). equals("vacation" )) {
            _workday = "no␣workday";
        }
        return _workday;
    }

    public String toString () {
        StringBuffer buffer = new StringBuffer ();
        buffer = buffer.append(_dateKey + "\t" );
        buffer = buffer.append(getSqlDate () + "\t" );
        buffer = buffer.append(getYear () + "\t" );
        buffer = buffer.append(getMonth() + "\t" );
        buffer = buffer.append(getDay () + "\t" );
        buffer = buffer.append(getWeekDay() + "\t" );
        buffer = buffer.append(getSemester () + "\t" );
        buffer = buffer.append(getDayOfSemester() + "\t" );
        buffer = buffer.append(getWeekOfSemester() + "\t" );
        buffer = buffer.append(getWeekend() + "\t" );
        buffer = buffer.append(getExam() + "\t" );
        buffer = buffer.append(getPublicHoliday () + "\t" );
        buffer = buffer.append(getSchoolVacation () + "\t" );
        buffer = buffer.append(getDayOfYear() + "\t" );
        buffer = buffer.append(getWeekOfYear() + "\t" );
        buffer = buffer.append(getWorkday());
        return buffer.toString ();
    }
}
```

E.5 IncrementalLoad Class

```
package aub;

import java.io.BufferedReader;
import java.util.Vector;
import java.util.GregorianCalendar;
import java.io.FileReader;
import java.io.FileNotFoundException;

public class IncrementalLoad {
    static Database _database;
    static int _logLineKey;
    // static String _pathToCopyFiles = "/var/lib/pgsql/";// baerbar
    static String _pathToCopyFiles = "/pack/postgres/"; // stationaer
    // static String _filePath = "/home/louise/projekt/logfiles2/";// baerbar
    static String _filePath = "/pack/louise/projekt/logfiles/";
    static String[] _filenames;
    static PostProcessor _postProcessor;
    static int _auditKey;
    static String _successStatus;
    static String[] _columnNames;
    static String[] _columnValues;
    static boolean _etlSuccess;
    static int _newSessionRecords;
    static int _newSearchRecords;
    static int _newPageEventRecords;
    static int _pageEventRecordsBefore;
    static int _pageEventRecordsAfter;
    static int _searchRecordsBefore;
    static int _searchRecordsAfter;
    static int _sessionRecordsBefore;
    static int _sessionRecordsAfter;
    static int _copyPageEventFileCount;
    static int _copySessionFileCount;
    static int _copySearchFileCount;

   public static BufferedReader[] getUnprocessedFileReaders(String filePath) {
       System.out.println("getUnprocessedFileReaders()");
       Vector fileReaders = new Vector();
       Vector filenameVector = new Vector();
       String filename = "";
       BufferedReader reader = null;
       String[] loadedFiles = _database.getLoadedLogFileNames();
       String lastLoadedFilename = _database.getLastSuccessfulLoadFilename();
       GregorianCalendar date = new GregorianCalendar(2003, 1, 25);
       if (lastLoadedFilename != null) {
           int year = Integer.parseInt(lastLoadedFilename.substring(11, 15));
           int month = Integer.parseInt(lastLoadedFilename.substring(15, 17));
           int day = Integer.parseInt(lastLoadedFilename.substring(17));
           date = new GregorianCalendar(year, month − 1, day);
       }
       try {
           while (true) {
               date.add(date.DAY_OF_MONTH, 1);
                                                   // filename = "accesstest.20030225";

               filename = "access_log." + date.get(date.YEAR);
               if (date.get(date.MONTH) < 9) {
                   filename += "0";
               }
               filename += (date.get(date.MONTH) + 1);
               if (date.get(date.DAY_OF_MONTH) < 10) {
                   filename += "0";
               }
               filename += date.get(date.DAY_OF_MONTH);
               System.out.println(filename);

               boolean fileLoaded = false;
               for (int f = 0; f < loadedFiles.length && !fileLoaded; f++) {
                   fileLoaded = filename.equals(loadedFiles[f]);
               }
               if (!fileLoaded) {
                   System.out.println("File␣is␣not␣loaded");
                   reader = new BufferedReader(new FileReader(filePath +
                                                            filename));
                   filenameVector.add(filename);
                   fileReaders.add(reader);
               }
               else {
                   System.out.println("File␣has␣already␣been␣loaded.");
               }
           }
       }
       catch (FileNotFoundException fnfe) {
           System.out.println("File␣not␣found:␣" + filename);
           // stillMoreFiles = false;
       }
       catch (Exception e) {
           System.out.println("Exception␣in␣IncrementalLoad.getUnprocessedFileReaders()");
           System.out.println(e);
       }
       BufferedReader[] readerArray = new BufferedReader[fileReaders.size()];
       for (int r = 0; r < readerArray.length; r++) {
           readerArray[r] = (BufferedReader) fileReaders.get(r);
       }
       _filenames = new String[filenameVector.size()];
       for (int f = 0; f < _filenames.length; f++) {
           _filenames[f] = (String) filenameVector.get(f);
       }
       return readerArray;
   }

   public static boolean noOtherLoadsRunning() {
       return true;
   }
```

```

  public static void main ( String [] args ) {
    if ( noOtherLoadsRunning()) {
      _etlSuccess = true;
      // _database = new Database("aub ", " aub ", _pathToCopyFiles);
      _database = new Database("aub" , " louise " , _pathToCopyFiles);
      _database . openConnection();
      _database . setSessionKey ();
      _logLineKey = _database . getMaxLogLineKey() + 1;
      _postProcessor = new PostProcessor( _database , _auditKey , _logLineKey,
                                          _filePath , _pathToCopyFiles);
      BufferedReader [] fileReaders = getUnprocessedFileReaders( _filePath );
      if ( fileReaders . length > 0) {
        _database . prepareStatements ();
      }
      long _numberOfLogLines = 0;
      for ( int f = 0; f < fileReaders . length && _etlSuccess ; f++) {
        _auditKey = _database . newAuditRecord(_filenames[f ],
                                               "New␣audit␣record␣created",
                                               "Recover");
        _postProcessor . setAuditKey(_auditKey);
        System.out. println ("Filename:␣" + _filenames [ f ]);
        _postProcessor . getActiveInfo ();
        int [] recordsBeforeLoad = _postProcessor . getRecordCountsBeforeLoad();
        // _pageEventRecordsBefore = recordsBeforeLoad [0];
        // _searchRecordsBefore = recordsBeforeLoad [1];
        // _sessionRecordsBefore = recordsBeforeLoad [2];

        _postProcessor . countLinesInLogFile( _filePath , _filenames [ f ]);

        _database. resetFiles ();
        _etlSuccess = _postProcessor . processLogFile( fileReaders [ f ], _filenames [ f ]);
        // int [] recordsAfterLoad = _postProcessor . getRecordCountsAfterLoad();
        // _pageEventRecordsAfter = recordsAfterLoad [0];
        // _searchRecordsAfter = recordsAfterLoad [1];
        // _sessionRecordsAfter = recordsAfterLoad [2];

        /*
          _newPageEventRecords = _pageEventRecordsAfter − _pageEventRecordsBefore;
          System.out. println ("_newPageEventRecords: " + _newPageEventRecords);
          String pageEventSuccess = "";
          _copyPageEventFileCount = _postProcessor . getCopyPageEventFileCount();
          if ( _newPageEventRecords != _copyPageEventFileCount) {
          pageEventSuccess =
              " Number of new page event records does not "+
              "match number of lines in copy file . _newPageEventRecords: " +
              _newPageEventRecords;
          _etlSuccess = false ;
          }
          _newSessionRecords = _sessionRecordsAfter − _sessionRecordsBefore;
          System.out. println ("_newSessionRecords: " + _newSessionRecords);
          String sessionSuccess = "";
          _copySessionFileCount = _postProcessor . getCopySessionFileCount ();
          if ( _newSessionRecords != _copySessionFileCount ) {
          sessionSuccess =
              " Number of new session records does not "+
              "match number of lines in copy file .";
          _etlSuccess = false ;
          }
          _newSearchRecords = _searchRecordsAfter − _searchRecordsBefore;
          System.out. println ("_newSearchRecords: " + _newSearchRecords);
          String searchSuccess = "";
          _copySearchFileCount = _postProcessor . getCopySearchFileCount();
          if ( _newSearchRecords != _copySearchFileCount) {
          searchSuccess =
              " Number of new search records does not "+
              "match number of lines in copy file .";
          _etlSuccess = false ;
          }

          _columnNames = new String[] {"new_session_records ",
          "new_search_records",
          "new_page_event_records"};
          _columnValues = new String [] { Integer . toString (_newSessionRecords),
          Integer . toString (_newSearchRecords),
          Integer . toString (_newPageEventRecords)};

          _database . updateAuditDimension(_auditKey, _columnNames, _columnValues,
          "number of new session , search and " +
          "page event records counted ." +
          pageEventSuccess + sessionSuccess +
          searchSuccess );
        */

        if ( _etlSuccess ) {
          _successStatus = "ETL␣process␣completed␣succesfully";
          _database . completeAuditRecord(_auditKey, _successStatus , "Proceed" );
        }

        // System.out. println ("Success status : " + _successStatus );
        System.out. println ();
      }
    }
  }
}
```

E.6 LogLine Class

```
package aub;

import java.io.*;
import java.sql.*;
import java.text.*;
import java.net.*;

public class LogLine {

    private int _logLineKey, _logLineNumber, _status, _bytes;
    private String _filename, _ipAddress, _ident, _authuser, _timeZone, _method,
        _sessionTag, _serial, _query, _protocol, _serverName, _browser;
    private java.sql.Date _date;
    private Time _time;
    private URL _requestUrl, _referrer ;
    private boolean _logLineValid;
    // private FileWriter _irregularLines ;

    public LogLine(int logLineKey, String filename, int logLineNumber,
                   String ipAddress, String ident, String authuser,
                   java.sql.Date date, Time time, String timeZone,
                   String method, String requestUrl, String sessionTag, String serial,
                   String query, String protocol, int status, int bytes,
                   String serverName, String referrer, String browser) {
        _logLineKey = logLineKey;
        _filename = filename;
        _logLineNumber = logLineNumber;
        _ipAddress = ipAddress;
        _ident = ident;
        _authuser = authuser;
        _date = date;
        _time = time;
        _timeZone = timeZone;
        _method = method;

        try {
            _requestUrl = new URL(requestUrl);
        } catch (MalformedURLException mue) {}

        _sessionTag = sessionTag;
        _serial = serial ;
        _query = query;
        _protocol = protocol;
        _status = status ;
        _bytes = bytes;
        _serverName = serverName;
        try {
            _referrer = new URL(referrer);
        } catch (MalformedURLException mue) {}

        _browser = browser;
    }

    /**
     * Constructs a LogLine object using the filename, linenumber and log line
     * string. Throws InvalidLogLineException if the log line is not accepted
     * as valid.
     */
    public LogLine(int logLineKey, String filename, int logLineNumber,
                   String logLine) throws InvalidLogLineException,
                   MalformedURLException {
        _logLineKey = logLineKey;
        _filename = filename;
        _logLineNumber = logLineNumber;

        try {
            // _irregularLines = new FileWriter ("/home/louise/projekt/logfiles/" +
            //                                   "irregularlines.txt", true);
            //System.out.println("irregularLines created");

            // Escape backslashes and apostrophes
            logLine = logLine.replaceAll("\\\\",␣"\\\\\\\\");
            logLine = logLine.replaceAll("'", "\\\\'");

            // remove multible spaces
            while (logLine.indexOf("␣␣") != -1) {
                logLine = logLine.replaceAll("␣␣", "␣");
            }

            int firstBracket = logLine.indexOf("[");
            int secondBracket = logLine.indexOf("]");
            int requestStart = logLine.indexOf("\"", secondBracket);
            int requestEnd = logLine.indexOf("\"", requestStart + 1);
            int referrerStart = logLine.indexOf("\"", requestEnd + 1);
            int referrerEnd = logLine.indexOf("\"", referrerStart + 1);

            // split log line into six major parts
            String ipIdentAuthuser = logLine.substring(0, firstBracket - 1);
            String timestamp = logLine.substring(firstBracket + 1, secondBracket);
            String request = logLine.substring(requestStart + 1, requestEnd);
            String statusBytesServer = logLine.substring(requestEnd + 2,
                                                         referrerStart - 1);

            try {
                _referrer = new URL(logLine.substring(referrerStart + 1, referrerEnd));
        } catch (MalformedURLException mue) {
            _referrer = null;
        }

        _browser = logLine.substring(referrerEnd + 3, logLine.length() − 1);

        String[] ipIdentAuthuserParts = ipIdentAuthuser.split("\\s");
        _ipAddress = ipIdentAuthuserParts[0];
        _ident = ipIdentAuthuserParts[1];
        _authuser = ipIdentAuthuser.substring(_ipAddress.length() +
                                              _ident.length() + 2);

        String dateString = timestamp.substring(0, 11);
        _date = sqlDateFormat(dateString);
        _time = Time.valueOf(timestamp.substring(12, 20));
        _timeZone = timestamp.substring(21);

        String[] requestParts = request.split("\\s");
        if ( requestParts.length != 3) {
                                                    // _irregularLines.write(logLine + "\n");
            throw new InvalidLogLineException("request␣too␣short");
        }

        _method = requestParts[0];

        if (_method.equals("HEAD")) {
            throw new InvalidLogLineException("HEAD␣method␣not␣accepted␣as␣" +
                                              "user␣request");
        }

        _requestUrl = new URL("http://a500.aub.auc.dk" + requestParts[1]);
        if (_requestUrl.toString().endsWith("/?func=logout")) {
            throw new InvalidLogLineException("session␣time−out");
        }
        _protocol = requestParts[2];
        _sessionTag = null;
        _serial = "";

        if (_requestUrl != null && _requestUrl.getPath() != null &&
            _requestUrl.getPath().indexOf("/F/") == −1) {
            throw new InvalidLogLineException("not␣auboline␣user␣function");
        }

        int endOfSessionTag = _requestUrl.getPath().indexOf("−");
        if (endOfSessionTag == −1) {
            endOfSessionTag = _requestUrl.getPath().length();
        }
        if (_requestUrl.getPath().length() > 3) {
            _sessionTag = _requestUrl.getPath().substring(3, endOfSessionTag);
            if (_sessionTag.length() > 50) {
                System.out.println("session␣tag␣too␣long:␣" + _sessionTag);
            }
        }
        if (endOfSessionTag != _requestUrl.getPath().length()) {
            _serial = _requestUrl.getPath().substring(endOfSessionTag + 1);
            if (_serial.length() != 5) {
                _serial = "";
            }
        }
        _query = _requestUrl.getQuery();
        if (_query == null) {
            _query = "";
        }

        String[] statusBytesServerParts = statusBytesServer.split("\\s");
        if ( statusBytesServerParts.length != 3) {
                                                    // _irregularLines.write(logLine + "\n");
            throw new InvalidLogLineException("status,␣bytes␣or␣server␣missing");
        }

        _status = Integer.parseInt( statusBytesServerParts[0]);
        if ( statusBytesServerParts[1].equals("−")) {
            throw new InvalidLogLineException("the␣bytes␣attribute␣has␣no␣value");
        }
        else {
            _bytes = Integer.parseInt( statusBytesServerParts[1]);
        }
        _serverName = statusBytesServerParts[2];

    } catch (StringIndexOutOfBoundsException stringEx) {
        System.out.println(stringEx);
        System.out.println("Log␣line␣" + logLineNumber + ":␣" + logLine);
    } catch (IOException ioe) {
        System.out.println(ioe);
    }
}

public int getLogLineKey() {
    return _logLineKey;
}

public String getFilename() {
    return _filename;
}

public int getLogLineNumber() {
    return _logLineNumber;
}
```

```
    public int getStatus () {
        return _status ;
    }

    public int getBytes () {
        return _bytes;
    }

    public String getIpAddress () {
        return _ipAddress;
    }

    public String getIdent () {
        return _ident ;
    }

    public String getAuthuser () {
        return _authuser ;
    }

    public String getTimeZone() {
        return _timeZone;
    }

    public String getMethod() {
        return _method;
    }

    public URL getRequestUrl() {
        return _requestUrl ;
    }

    public String getSessionTag () {
        return _sessionTag;
    }

    public String getSerial () {
        return _serial ;
    }

    public String getQuery () {
        return _query;
    }

    public String getProtocol () {
        return _protocol ;
    }
    public String getServerName() {
        return _serverName;
    }

    public URL getReferrer () {
        return _referrer ;
    }

    public String getBrowser () {
        return _browser;
    }

    public java.sql.Date getDate () {
        return _date;
    }

    public Time getTime() {
        return _time;
    }

    public long getLongTime() {
        return _date.getTime () + _time.getTime();
    }

    public boolean referrerHasSessionTag () {
        String referrerSessionTag = null;
        if ( _referrer != null && _referrer.getPath () != null &&
            _requestUrl.getPath ().indexOf("/F/") != -1) {

            int endOfSessionTag = _referrer.getPath ().indexOf("-");
            if ( endOfSessionTag == -1) {
                endOfSessionTag = _referrer.getPath ().length ();
            }
            if ( _referrer.getPath ().length () > 3) {
                referrerSessionTag = _referrer.getPath ().substring (3, endOfSessionTag);
            }
        }
        return ( referrerSessionTag != null && referrerSessionTag.length () == 50);
    }

    public boolean hasSessionTag () {
        return _sessionTag != null && !_sessionTag.equals ("");
    }

    public boolean isValid () {
        return _logLineValid;
    }

    public void setFilename ( String newFilename) {
        _filename = newFilename;
    }

    public void setLogLineNumber(int newLogLineNumber) {
        _logLineNumber = newLogLineNumber;
    }

    public void setStatus (int newStatus) {
        _status = newStatus;
    }

    public void setBytes(int newBytes) {
        _bytes = newBytes;
    }

    public void setIpAddress(String newIpAddress) {
        _ipAddress = newIpAddress;
    }

    public void setIdent (String newIdent) {
        _ident = newIdent;
    }

    public void setAuthuser(String newAuthuser) {
        _authuser = newAuthuser;
    }

    public void setTimeZone(String newTimeZone) {
        _timeZone = newTimeZone;
    }

    public void setMethod(String newMethod) {
        _method = newMethod;
    }

    public void setRequestUrl(String newRequestUrl) {
        try {
            _requestUrl = new URL(newRequestUrl);
        } catch (Exception e) {}
    }

    public void setRequestUrl(URL newRequestUrl) {
        _requestUrl = newRequestUrl;
    }

    public void setSessionTag(String newSessionTag) {
        _sessionTag = newSessionTag;
    }

    public void setProtocol (String newProtocol) {
        _protocol = newProtocol;
    }

    public void setServerName(String newServerName) {
        _serverName = newServerName;
    }

    public void setReferrer (String newReferrer) {
        try {
            _referrer = new URL(newReferrer);
        } catch (Exception e) {}
    }

    public void setReferrer (URL newReferrer) {
        _referrer = newReferrer;
    }

    public void setBrowser(String newBrowser) {
        _browser = newBrowser;
    }

    public void setDate(java.sql.Date newDate) {
        _date = newDate;
    }

    public void setTime(Time newTime) {
        _time = newTime;
    }

    private java.sql.Date sqlDateFormat(String dateInLogFormat) {
        SimpleDateFormat dateFormat = new SimpleDateFormat("dd/MMM/yyyy");
        java.sql.Date dateInSqlFormat = null;
        try {
            dateInSqlFormat = new java.sql.Date
                (dateFormat.parse(dateInLogFormat).getTime());
        } catch (ParseException pe) {
            System.out.println(" TestServlet.sqlDateFormat()␣failed ..." );
            System.out.println(pe);
        } catch (Exception e) {
            System.out.println(" TestServlet.sqlDateFormat()␣failed ..." );
            System.out.println(e);
        }
        return dateInSqlFormat;
    }

    public String toString () {
        StringBuffer buffer = new StringBuffer ();
        buffer.append(_logLineKey + "\t");
        buffer.append(_filename + "\t");
        buffer.append(_logLineNumber + "\t");
        buffer.append(_ipAddress + "\t");
        buffer.append(_ident + "\t");
        buffer.append(_authuser + "\t");
        buffer.append(_date + "\t");
        buffer.append(_time + "\t");
        buffer.append(_timeZone + "\t");
        buffer.append(_method + "\t");
        buffer.append(_requestUrl + "\t");
        buffer.append(_sessionTag + "\t");
        buffer.append(_serial + "\t");
        buffer.append(_query + "\t");
        buffer.append(_protocol + "\t");
        buffer.append(_status + "\t");
        buffer.append(_bytes + "\t");
        buffer.append(_serverName + "\t");
        if ( _referrer == null) {
            buffer.append("no␣value\t");
        }
        else {
            buffer.append(_referrer + "\t");
        }
        buffer.append(_browser);
        return buffer.toString();
    }
}
```

E.7 Page Class

```
package aub;

import java.net.*;
import java.util.*;

public class Page {

    private int _pageKey, _numberOfCheckedBoxes;
    private String _method, _func, _fileName, _findCode, // _funcOption,
        _scanCode, _action, _pageFunction, _pageFunctionType,
        _process, _field, _referrerFunc, _referrerQuery;

    public Page (LogLine logLine) {
        UrlQuery urlQuery = new UrlQuery(logLine.getQuery());
        _pageKey = -1;
        _method = logLine.getMethod();
        _func = urlQuery.getValue("func");
        _fileName = urlQuery.getValue("file_name");
        _findCode = urlQuery.getValue("find_code");
        _scanCode = urlQuery.getValue("scan_code");
        _action = "unknown";
        if (urlQuery.getValue("action_view.x") != null) {
            _action = "view";
        }
        if (urlQuery.getValue("action_delete.x") != null) {
            _action = "delete";
        }
        if (urlQuery.getValue("action_cross.x") != null) {
            _action = "cross";
        }
        if (urlQuery.getValue("action_short_basket_store.x") != null) {
            _action = "store␣in␣basket";
        }
        URL referrer = logLine.getReferrer();
        if ( referrer != null) {
            _referrerQuery = referrer.getQuery();
        }
        if ( _referrerQuery != null) {
            _referrerFunc = (new UrlQuery(_referrerQuery).getValue("func"));
        }
        _numberOfCheckedBoxes = 0;
        String query = logLine.getQuery();
        while (query.indexOf("=on") != -1) {
            _numberOfCheckedBoxes++;
            query = query.substring(query.indexOf("=on") + 1);
        }
        setVariables();
    }

    public Page (int pageKey, String method, String func, String fileName,
                 String findCode, String scanCode, String action, String field,
                 String pageFunction, String pageFunctionType, String process) {
        _pageKey = pageKey;
        _method = method;
        _func = func;
        _fileName = fileName;
        _findCode = findCode;
        _scanCode = scanCode;
        _action = action;
        _field = field;
        _pageFunction = pageFunction;
        _pageFunctionType = pageFunctionType;
        _process = process;
    }

    public Page (int pageKey, String pageFunction, String pageFunctionType,
                 String process) {
```

```
        _pageKey = pageKey;
        _pageFunction = pageFunction;
        _pageFunctionType = pageFunctionType;
        _process = process;
    }

    public int getPageKey() {
        return _pageKey;
    }

    public String getMethod() {
        return _method;
    }

    public String getFunc () {
        if ( _func == null || _func. equals ("" )) {
            _func = "unknown";
        }
        return _func;
    }

    public String getFileName() {
        if ( _fileName == null || _fileName. equals ("" )) {
            _fileName = "unknown";
        }
        return _fileName;
    }

    public String getFindCode() {
        if ( _findCode == null || _findCode. equals ("" )) {
            _findCode = "unknown";
        }
        return _findCode;
    }

    public String getScanCode() {
        if ( _scanCode == null || _scanCode.equals("" )) {
            _scanCode = "unknown";
        }
        return _scanCode;
    }

    public String getAction () {
        return _action;
    }

    public String getField () {
        if ( _field == null ) {
            if ( "WRD".equals(_findCode)) {
                _field = " all␣fields ";
            }
            else if ( "WTI".equals(_findCode ) || "LTI". equals (_scanCode)) {
                _field = " title ";
            }
            else if ( "WFO".equals(_findCode ) || "LFO".equals(_scanCode)) {
                _field = " author ";
            }
            else if ( "WKE".equals(_findCode)) {
                _field = " controlled␣subject ";
            }
            else if ( "WEM".equals(_findCode)) {
                _field = " all␣subjects ";
            }
            else if ( "WIS".equals(_findCode )) {
                _field = " title ";
            }
            else if ( "WAN".equals(_findCode)) {
                _field = " title ";
            }
            else if ( "LEM".equals(_scanCode)) {
                _field = "keyword";
            }
            else if ( "LCL".equals(_scanCode)) {
                _field = "UDK−classification";
            }
            else if ( "WTI".equals(_scanCode)) {
                _field = "words␣in␣title ";
            }
            else {
                _field = "no␣value";
            }
        }
        return _field ;
    }

    public String getReferrerFunc () {
        if ( _method.equals("POST")) {
            return _referrerFunc ;
        }
        else {
            return ( "no␣relevance" );
        }
    }

    public int getNumberOfCheckedBoxes() {
        if ( _numberOfCheckedBoxes == 0) {
            return 1;
        }
        else {
            return _numberOfCheckedBoxes;
        }
    }

    private void setVariables () {
        _pageFunction = "unknown";
        _pageFunctionType = "unknown";
        _process = "unknown";
        if (_func == null || _func.equals("unknown") ||
            _func.equals("null") || _func.equals("")) {
            if (_method.equals("POST") && "item−hold−request".equals(_referrerFunc)) {
                _pageFunction = "reserve␣book";
                _pageFunctionType = "reservation";
                _process = "reservation";
            }
            if (_method.equals("POST") &&
                ("BOR−LOGIN".equals(_referrerFunc) || "login".equals(_referrerFunc))) {
                _pageFunction = "login";
                _pageFunctionType = "login";
                _process = "login";
            }
        }
        else if (_func.equals("basket−delete")) {
            _pageFunction = "delete␣item␣from␣basket";
            _pageFunctionType = "basket␣function";
            _process = "basket";
        }
        else if (_func.equals("basket−delete−all")) {
            _pageFunction = "delete␣all␣items␣from␣basket";
            _pageFunctionType = "basket␣function";
            _process = "basket";
        }
        else if (_func.equals("basket−full")) {
            _pageFunction = "full␣format␣information␣on␣book␣from␣basket";
            _pageFunctionType = "book␣description";
            _process = "basket";
        }
        else if (_func.equals("basket−note")) {
            _pageFunction = "entered␣note␣for␣item␣to␣save␣in␣basket";
            _pageFunctionType = "book␣in␣basket";
            _process = "basket";
        }
        else if (_func.equals("basket−note−0")) {
            _pageFunction = "enter␣note␣before␣saving␣item␣in␣basket";
            _pageFunctionType = "basket␣function";
            _process = "basket";
        }
        else if (_func.equals("basket−short")) {
            _pageFunction = "show␣books␣saved␣in␣basket";
            _pageFunctionType = "basket␣function";
            _process = "basket";
        }
        else if (_func.equals("bor−hold")) {
            _pageFunction = "reservations␣list";
            _pageFunctionType = "reservation␣function";
            _process = "reservation";
        }
        else if (_func.equals("BOR−HOLD−DELETE")) {
            _pageFunction = "delete␣reservation";
            _pageFunctionType = "reservation␣function";
            _process = "reservation";
        }
        else if (_func.equals("BOR−HOLD−EXP")) {
            _pageFunction = "espanded␣information␣on␣a␣reservation";
            _pageFunctionType = "reservation␣function";
            _process = "reservation";
        }
        else if (_func.equals("bor−info")) {
            _pageFunction = "borrower␣information";
            _pageFunctionType = "borrower␣information";
            _process = "borrower␣information";
        }
        else if (_func.equals("bor−loan")) {
            _pageFunction = "loans␣list";
            _pageFunctionType = "borrower␣information";
            _process = "borrower␣information";
        }
        else if (_func.equals("BOR−LOAN−EXP")) {
            _pageFunction = "expanded␣view␣of␣a␣loan";
            _pageFunctionType = "borrower␣information";
            _process = "borrower␣information";
        }
        else if (_func.equals("BOR−LOAN−RENEW")) {
            _pageFunction = "renew␣loan";
            _pageFunctionType = "renew␣loan";
            _process = "borrower␣information";
        }
        else if (_func.equals("BOR−LOGIN")) {
            _pageFunction = "login␣before␣borrower␣information";
            _pageFunctionType = "login";
            _process = "login";
        }
        else if (_func.equals("bor−renew−all")) {
            _pageFunction = "renew␣all␣loans";
            _pageFunctionType = "renew␣loan";
            _process = "borrower␣information";
        }
        else if (_func.equals("bor-update")) {
            _pageFunction = "enter␣information␣for␣address␣update";
            _pageFunctionType = "borrower␣information";
            _process = "borrower␣information";
        }
        else if (_func.equals("bor-update-1")) {
            _pageFunction = "address␣information␣updated";
            _pageFunctionType = "borrower␣information";
            _process = "borrower␣information";
        }
        else if (_func.equals("direct-set")) {
            _pageFunction = "full␣format␣(after␣scan)";
            _pageFunctionType = "book␣description";
            _process = "search";
        }
        else if (_func.equals("file")) {
            if ("base-list".equals(_fileName)) {
                _pageFunction = "list␣of␣databases";
                _pageFunctionType = "list␣databases";
                _process = "search";
            }
            else if ("bor-update-password".equals(_fileName)) {
                _pageFunction = "fields␣to␣update␣password␣on␣old␣library␣card";
                _pageFunctionType = "borrower␣information";
                _process = "borrower␣information";
            }
            else if ("feedback".equals(_fileName)) {
                _pageFunction = "feedback␣form";
                _pageFunctionType = "feedback";
                _process = "feedback";
            }
            else if ("find-b".equals(_fileName)) {
                _pageFunction = "basic␣search␣form";
                _pageFunctionType = "enter␣data␣for␣search";
                _process = "search";
            }
            else if ("logout".equals(_fileName)) {
                _pageFunction = "click␣go␣to␣logout";
                _pageFunctionType = "logout";
                _process = "logout";
            }
        }
        else if (_func.equals("find-a")) {
            _pageFunction = "multi-field␣search";
            _pageFunctionType = "search";
            _process = "search";
        }
        else if (_func.equals("find-acc")) {
            _pageFunction = "full␣view␣of␣record␣(maybe␣only␣from␣scan)";
            _pageFunctionType = "book␣description";
            _process = "search";
        }
        else if (_func.equals("find-b")) {
            _pageFunctionType = "search";
            _process = "search";
            if (getField().equals("no␣value")) {
                _pageFunction = "basic␣search";
            }
            else {
                _pageFunction = "basic␣search␣on␣" + _field;
            }
        }
        else if (_func.equals("find-c")) {
            _pageFunction = "Common␣Command␣Language␣search";
            _pageFunctionType = "search";
            _process = "search";
        }
        else if (_func.equals("find-m")) {
            _pageFunctionType = "search";
            _process = "search";
            if (getField().equals("no␣value")) {
                _pageFunction = "multi-base␣search";
            }
            else {
                _pageFunction = "multi-base␣search␣on␣" + _field;
            }
        }
        else if (_func.equals("find-word")) {
            _pageFunction = "search␣word␣not␣found,␣choosen␣other␣word␣from␣list";
            _pageFunctionType = "search";
            _process = "search";
        }
        else if (_func.equals("full-mail")) {
            _pageFunction = "sent␣records␣in␣mail";
            _pageFunctionType = "mail";
            _process = "mail";
        }
        else if (_func.equals("full-mail-0")) {
            _pageFunction = "enter␣text␣to␣mail␣records";
            _pageFunctionType = "enter␣data";
            _process = "mail";
        }
        else if (_func.equals("full-set")) {
            _pageFunction = "full␣view␣of␣record␣after␣reservation";
            _pageFunctionType = "book␣description";
            _process = "search";
        }
        else if (_func.equals("full-set-set")) {
            _pageFunction = "full␣view␣of␣record␣(different␣formats)";
            _pageFunctionType = "book␣description";
            _process = "search";
        }
        else if (_func.equals("history")) {
            _pageFunction = "show␣history␣options";
            _pageFunctionType = "history";
            _process = "history";
        }
        else if (_func.equals("history-action")) {
            _pageFunctionType = "history␣function";
            _process = "history";
            if ("view".equals(_action)) {
                _pageFunction = "view␣search␣result␣from␣history";
                _pageFunctionType = "search";
            }
            else if ("delete".equals(_action)) {
                _pageFunction = "delete␣search␣result␣from␣history";
            }
            else if ("cross".equals(_action)) {
                _pageFunction = "choose␣how␣to␣cross␣search␣results";
            }
            else {
                _pageFunction = "unknown␣history␣action";
            }
        }
        else if (_func.equals("history-cross")) {
            _pageFunction = "cross␣several␣search␣results␣from␣history";
            _pageFunctionType = "history␣function";
            _process = "history";
        }
        else if (_func.equals("ill-request-1")) {
            _pageFunction = "illegal␣request";
            _pageFunctionType = "illegal";
        }
        else if (_func.equals("item-global")) {
            _pageFunction = "show␣all␣copies␣of␣a␣book";
            _pageFunctionType = "show␣copies";
            _process = "search";
        }
        else if (_func.equals("item-global-exp")) {
            _pageFunction = "item␣record␣expand␣view";
            _pageFunctionType = "book␣description";
            _process = "search";
        }
        else if (_func.equals("item-hold-request")) {
            _pageFunction = "reservation␣process";
            _pageFunctionType = "before␣reservation";
            _process = "reservation";
        }
        else if (_func.equals("login")) {
            _pageFunction = "login";
            _pageFunctionType = "login";
            _process = "login";
        }
        else if (_func.equals("logout")) {
            _pageFunction = "session␣time-out␣-␣start␣over␣with␣search␣or␣login";
            _pageFunctionType = "logout";
            _process = "logout";
        }
        else if (_func.equals("option-show")) {
            _pageFunction = "show␣formats";
            _pageFunctionType = "show␣formats";
            _process = "search";
        }
        else if (_func.equals("option-update")) {
            _pageFunction = "change␣formats";
            _pageFunctionType = "change␣formats";
            _process = "search";
        }
        else if (_func.equals("option-update-lng")) {
            _pageFunction = "change␣language␣or␣base";
            _pageFunctionType = "change␣language␣or␣base";
            _process = "search";
        }
        else if (_func.equals("scan")) {
            _pageFunctionType = "search";
            _process = "search";
            if (getField().equals("no␣value")) {
                _pageFunction = "browse␣unknown␣index";
            }
            else {
                _pageFunction = "browse␣" + _field + "␣index";
            }
        }
        else if (_func.equals("scan-list")) {
            _pageFunction = "browse␣index";
            _pageFunctionType = "enter␣data␣for␣search";
            _process = "search";
        }
        else if (_func.equals("service")) {
            _pageFunction = "lookup␣in␣other␣index␣(placement,␣author,␣or␣ISBN)";
            _pageFunctionType = "search";
            _process = "search";
        }
        else if (_func.equals("short")) {
            _pageFunction = "results␣list";
            _pageFunctionType = "results␣list";
            _process = "search";
        }
        else if (_func.equals("short−action")) {
            _pageFunction = "action␣after␣search␣" +
                "(e.g.␣ filter ␣or␣mail␣search␣results)";
            _pageFunctionType = "search␣function";
            _process = "search";
            if ("store␣in␣basket".equals(_action)) {
                _pageFunction = "store␣book(s)␣in␣basket";
                _pageFunctionType = "book␣in␣basket";
                _process = "basket";
            }
        }
        else if (_func.equals("short− filter −3")) {
            _pageFunction = " filter :␣show␣only␣available␣items";
            _pageFunctionType = " filter ";
            _process = "search";
        }
        else if (_func.equals("short− filter −y")) {
            _pageFunction = " filter ␣by␣year";
            _pageFunctionType = " filter ";
            _process = "search";
        }
        else if (_func.equals("short−mail")) {
            _pageFunction = "sent␣list␣by␣mail";
            _pageFunctionType = "mail";
            _process = "mail";
        }
        else if (_func.equals("short−refine−exec")) {
            _pageFunction = "refine␣search";
            _pageFunctionType = "refine␣search";
            _process = "search";
        }
        else if (_func.equals("short−sort")) {
            _pageFunction = "change␣sort␣order";
            _pageFunctionType = "sort";
            _process = "search";
        }
        else if (_func.equals("unknown")) {
            _pageFunction = "func␣is␣unknown";
        }
        else {
            _pageFunction = "new␣page␣function:␣" + _func;
        }
    }

    public String getPageFunction () {
        if (_pageFunction == null) {
            setVariables ();
        }
        return _pageFunction;
    }

    public String getPageFunctionType () {
        if (_pageFunctionType == null) {
            setVariables ();
        }
        return _pageFunctionType;
    }

    public String getProcess () {
        if (_process == null) {
            setVariables ();
        }
        return _process;
    }

    public void setPageKey(int newPageKey) {
        _pageKey = newPageKey;
    }

    public String toString () {
        StringBuffer buffer = new StringBuffer ();
        buffer.append(_pageKey + "\t");
        buffer.append(_pageFunction + "\t");
        buffer.append(_pageFunctionType + "\t");
        buffer.append(_process);
        return buffer.toString ();
    }


}
```

E.8 PageEvent Class

```
package aub;

public class PageEvent {

    int _logLineKey, _dateKey, _timeOfDayKey, _pageKey, _sessionKey, _auditKey;

    public PageEvent (int logLineKey, int dateKey, int timeOfDayKey,
                      int pageKey, int sessionKey, int auditKey) {
        _logLineKey = logLineKey;
        _dateKey = dateKey;
        _timeOfDayKey = timeOfDayKey;
        _pageKey = pageKey;
        _sessionKey = sessionKey;
        _auditKey = auditKey;
    }

    public int getLogLineKey() {
        return _logLineKey;
    }

    public int getDateKey() {
        return _dateKey;
    }

    public int getTimeKey() {
        return _timeOfDayKey;
    }

    public int getPageKey() {
        return _pageKey;
    }

    public int getSessionKey() {
        return _sessionKey;
    }

    public int getAuditKey() {
        return _auditKey;
    }

    public String toString () {
        StringBuffer buffer = new StringBuffer ();
        buffer .append(_logLineKey + "\t");
        buffer .append(_dateKey + "\t");
        buffer .append(_timeOfDayKey + "\t");
        buffer .append(_pageKey + "\t");
        buffer .append(_sessionKey + "\t");
        buffer .append(_auditKey);
        return buffer . toString ();
    }
}
```

E.9 Search Class

```
package aub;

public class Search {

    private int _dateKey, _timeOfDayKey, _sessionKey, _searchTypeKey, _searchNumber,
        _numberOfBookDescriptions, _numberOfBooksInBasket, _numberOfReservations;
    private String _searchNumberValidity;
    private boolean _active;

    public Search ( int dateKey, int timeOfDayKey, int sessionKey, int searchTypeKey,
                    int searchNumber, String searchNumberValidity,
                    int numberOfBookDescriptions, int numberOfBooksInBasket,
                    int numberOfReservations) {
        _dateKey = dateKey;
        _timeOfDayKey = timeOfDayKey;
        _sessionKey = sessionKey;
        _searchTypeKey = searchTypeKey;
        _searchNumber = searchNumber;
        _searchNumberValidity = searchNumberValidity;
        _numberOfBookDescriptions = numberOfBookDescriptions;
        _numberOfBooksInBasket = numberOfBooksInBasket;
        _numberOfReservations = numberOfReservations;
        _active = true;
    }

    public int getDateKey() {
        return _dateKey;
    }

    public int getTimeOfDayKey() {
        return _timeOfDayKey;
    }

    public int getSessionKey() {
        return _sessionKey;
    }

    public int getSearchTypeKey() {
        return _searchTypeKey;
    }

    public int getSearchNumber() {
        return _searchNumber;
    }

    public String getSearchNumberValidity() {
        return _searchNumberValidity;
    }

    public int getNumberOfBookDescriptions() {
        return _numberOfBookDescriptions;
    }

    public int getNumberOfBooksInBasket() {
        return _numberOfBooksInBasket;
    }

    public int getNumberOfReservations() {
        return _numberOfReservations;
    }

    public boolean getActive () {
        return _active ;
    }

    public void deactivate () {
        _active = false ;
    }

    public void incrementNumberOfBookDescriptions(int amount) {
        _numberOfBookDescriptions = _numberOfBookDescriptions + amount;
    }

    public void incrementNumberOfBooksInBasket(int amount) {
        _numberOfBooksInBasket = _numberOfBooksInBasket + amount;
    }

    public void incrementNumberOfReservations(int amount) {
        _numberOfReservations = _numberOfReservations + amount;
    }

    public void setSearchNumber(int newSearchNumber) {
        _searchNumber = newSearchNumber;
    }

    public void setSearchNumberValidity( String newSearchNumberValidity) {
        _searchNumberValidity = newSearchNumberValidity;
    }

    public String toString () {
        StringBuffer buffer = new StringBuffer ();
        buffer .append(_dateKey + "\t");
        buffer .append(_timeOfDayKey + "\t");
        buffer .append(_sessionKey + "\t");
        buffer .append(_searchTypeKey + "\t");
        buffer .append(_searchNumber + "\t");
        buffer .append(_searchNumberValidity + "\t");
        buffer .append(_numberOfBookDescriptions + "\t");
        buffer .append(_numberOfBooksInBasket + "\t");
        buffer .append(_numberOfReservations);
        return buffer . toString ();
    }
}
```

E.10 SearchType Class

```
package aub;

public class SearchType {

    private int _searchTypeKey;
    private String _type , _findCode , _scanCode, _field , _typeWithField ;

    public SearchType ( int searchTypeKey, String type , String field ,
                        String typeWithField ) {
        _searchTypeKey = searchTypeKey;
        _type = type;
        _field = field ;
        _typeWithField = typeWithField;
    }

    public SearchType (UrlQuery urlQuery ) throws NotSearchTypeException {
        if (urlQuery == null ) {
            System.out. println ("urlQuery␣is␣null" );
            throw new NotSearchTypeException("urlQuery␣is␣null");
        }
        String func = urlQuery. getValue ("func" );
        // System.out. println (" func : " + func );
        if ( func == null ) {
            throw new NotSearchTypeException("func␣is␣null");
        }

        _searchTypeKey = 0;
        _type = "unknown";
        _findCode = urlQuery . getValue ("find_code" );
        _scanCode = urlQuery . getValue ("scan_code" );

        if ( func . equals (" find -a")) {
            _type = "multi-field␣search";
        }
        else if ( func . equals (" find -b")) {
            _type = "basic␣search";
        }
        else if ( func . equals (" find -c")) {
            _type = "CCL␣search";
        }
        else if ( func . equals (" find -m")) {
            _type = "multi-base␣search";
        }
        else if ( func . equals ("scan- list" ) || func . equals ("scan" )) {
            _type = "browse␣index";
        }
        else if ( func . equals (" history -action") &&
                urlQuery . getValue ("action_view . x" ) != null ) {
            _type = " history ";
        }
        else if ( func . equals ("basket−full" )) {
            _type = "basket";
        }
        else {
            throw new NotSearchTypeException("urlQuery␣does␣not␣indicate␣search");
        }

        _field = getField ();

        _typeWithField = _type + "␣on␣" + _field + "␣field ";
    }

    public int getSearchTypeKey() {
        return _searchTypeKey;
    }

    public String getType () {
        return _type;
    }

    public String getTypeWithField () {
        return _typeWithField;
    }

    public String getFindCode() {
        return _findCode;
    }

    public String getField () {
        if ("WRD".equals(_findCode ) || "multi−field␣search" . equals (_type )) {
            _field = " all ";
        }
        else if ("WTI".equals(_findCode ) || "LTI". equals (_scanCode)) {
            _field = " title ";
        }
        else if ("WFO".equals(_findCode ) || "LFO".equals(_scanCode)) {
            _field = "author";
        }
        else if ("WKE".equals(_findCode)) {
            _field = " controlled ␣subject";
        }
        else if ("WEM".equals(_findCode)) {
            _field = " all ␣subjects ";
        }
        else if ("WIS".equals(_findCode )) {
            _field = " title ";
        }
        else if ("WAN".equals(_findCode)) {
            _field = " title ";
        }
        else if ("LEM".equals(_scanCode)) {
            _field = "keyword";
        }
        else if ("LCL".equals(_scanCode)) {
            _field = "UDK−classification";
        }
        else if ("WTI".equals(_scanCode)) {
            _field = "words␣in␣ title ";
        }
        else {
            _field = "no␣value";
        }
        return _field ;
    }
}
```

E.11 Session Class

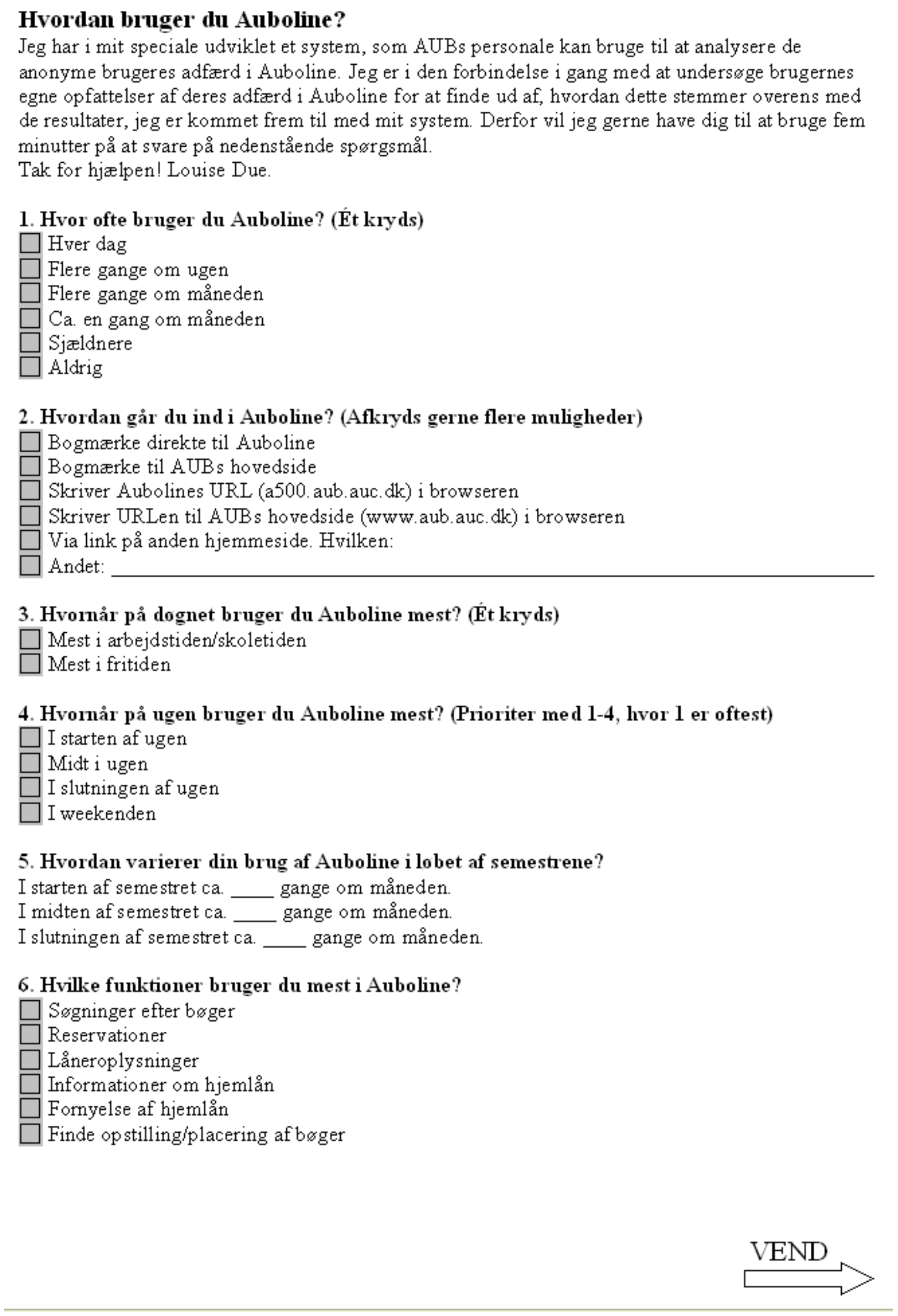
```
package aub;

import java . sql .∗;
import java . net .∗;

public class Session {

    private int _sessionKey , _pagesInSession , _lastSearchNumber, _firstPageKey ,
        _lastPageKey , _startDateKey , _startTimeKey , _endDateKey, _endTimeKey,
         _searchesInSession , _bookDescriptionsInSession , _booksInBasketInSession,
          _reservationsInSession ;
    private String _sessionTag , _ipAddress , _browser , _searchNumberValidity;
    private java . sql . Date _startDate , _endDate;
    private Time _startTime , _endTime;
    private URL _firstRequestUrl , _lastRequestUrl , _referrer ;

    public Session( int sessionKey , String sessionTag , String ipAddress , String browser,
                   String firstRequestUrl , int firstPageKey , String lastRequestUrl ,
                   int lastPageKey , String referrer , java . sql . Date startDate ,
                   int startDateKey , Time startTime , int startTimeKey,
                   java . sql . Date endDate, int endDateKey, Time endTime, int endTimeKey,
                   int pagesInSession , int _bookDescriptionsInSession ,
                   int _booksInBasketInSession , int _reservationsInSession ,
                   int lastSearchNumber, String searchNumberValidity ) {
        _sessionKey = sessionKey;
        _sessionTag = sessionTag;
        _ipAddress = ipAddress;
        _browser = browser;

        try {
            _firstRequestUrl = new URL(firstRequestUrl);
        } catch (MalformedURLException mue) {}

        _firstPageKey = firstPageKey;

        try {
            _lastRequestUrl = new URL(lastRequestUrl);
        } catch (MalformedURLException mue) {}

        _lastPageKey = lastPageKey;

        try {
            _referrer = new URL(referrer);
        } catch (MalformedURLException mue) {}

        _startDate = startDate;
        _startDateKey = startDateKey;
        _startTime = startTime;
        _startTimeKey = startTimeKey;
        _endDate = endDate;
        _endDateKey = endDateKey;
        _endTime = endTime;
        _endTimeKey = endTimeKey;
        _pagesInSession = pagesInSession;
        _searchesInSession = 0;
        _bookDescriptionsInSession = 0;
        _booksInBasketInSession = 0;
        _reservationsInSession = 0;
        _lastSearchNumber = lastSearchNumber;
        _searchNumberValidity = searchNumberValidity;
    }

    public Session(int sessionKey, String sessionTag, String ipAddress, String browser,
                   URL firstRequestUrl, int firstPageKey, URL lastRequestUrl,
                   int lastPageKey, URL referrer, java.sql.Date startDate,
                   int startDateKey, Time startTime, int startTimeKey,
                   java.sql.Date endDate, int endDateKey, Time endTime, int endTimeKey,
                   int pagesInSession, int _bookDescriptionsInSession,
                   int _booksInBasketInSession, int _reservationsInSession,
                   int lastSearchNumber, String searchNumberValidity) {
        _sessionKey = sessionKey;
        _sessionTag = sessionTag;
        _ipAddress = ipAddress;
        _browser = browser;
        _firstRequestUrl = firstRequestUrl;
        _firstPageKey = firstPageKey;
        _lastRequestUrl = lastRequestUrl;
        _lastPageKey = lastPageKey;
        _referrer = referrer;
        _startDate = startDate;
        _startDateKey = startDateKey;
        _startTime = startTime;
        _startTimeKey = startTimeKey;
        _endDate = endDate;
        _endDateKey = endDateKey;
        _endTime = endTime;
        _endTimeKey = endTimeKey;
        _pagesInSession = pagesInSession;
        _lastSearchNumber = lastSearchNumber;
        _searchNumberValidity = searchNumberValidity;
    }

    public Session(int sessionKey, LogLine logLine, int pageKey, int dateKey,
                   int timeKey) {
        _sessionKey = sessionKey;
        _sessionTag = logLine.getSessionTag();
        _ipAddress = logLine.getIpAddress();
        _browser = logLine.getBrowser();
        _firstRequestUrl = logLine.getRequestUrl();
        _firstPageKey = pageKey;
        _lastRequestUrl = _firstRequestUrl;
        _lastPageKey = _firstPageKey;
        _referrer = logLine.getReferrer();
        _startDate = logLine.getDate();
        _startDateKey = dateKey;
        _startTime = logLine.getTime();
        _startTimeKey = timeKey;
        _endDate = _startDate;
        _endDateKey = _startDateKey;
        _endTime = _startTime;
        _endTimeKey = _startTimeKey;
        _pagesInSession = 1;
        _lastSearchNumber = -1;
        _searchNumberValidity = "temporary";
    }

    public void addPageEvent(LogLine logLine, int pageKey, int dateKey, int timeKey) {
        if (!hasSessionTag()) {
            _sessionTag = logLine.getSessionTag();
        }
        _lastRequestUrl = logLine.getRequestUrl();
        _lastPageKey = pageKey;
        _endDate = logLine.getDate();
        _endDateKey = dateKey;
        _endTime = logLine.getTime();
        _endTimeKey = timeKey;
        _pagesInSession++;
    }

    public void addSearch(Search search) {
        _searchesInSession++;
        _bookDescriptionsInSession++;
        _booksInBasketInSession++;
        _reservationsInSession++;
    }

    public String getSessionTag() {
        return _sessionTag;
    }

    public boolean hasSessionTag() {
        return (getSessionTag() != null && !getSessionTag().equals("null") &&
                !getSessionTag().equals(""));
    }

    public int getSessionKey() {
        return _sessionKey;
    }

    public String getIpAddress() {
        return _ipAddress;
    }

    public String getBrowser() {
        return _browser;
    }

    public URL getFirstRequestUrl() {
        return _firstRequestUrl;
    }

    public int getFirstPageKey() {
        return _firstPageKey;
    }

    public URL getLastRequestUrl() {
        return _lastRequestUrl;
    }

    public int getLastPageKey() {
        return _lastPageKey;
    }

    public URL getReferrer() {
        return _referrer;
    }

    public java.sql.Date getStartDate() {
        return _startDate;
    }

    public int getStartDateKey() {
        return _startDateKey;
    }

    public Time getStartTime() {
        return _startTime;
    }

    public int getStartTimeKey() {
        return _startTimeKey;
    }

    public java.sql.Date getEndDate() {
        return _endDate;
    }

    public int getEndDateKey() {
        return _endDateKey;
    }

    public Time getEndTime() {
        return _endTime;
    }

    public int getEndTimeKey() {
        return _endTimeKey;
    }

    public int getPagesInSession() {
        return _pagesInSession;
    }

    public int getLastSearchNumber() {
        return _lastSearchNumber;
    }

    public boolean isLastSearchNumberKnown() {
        return (_lastSearchNumber != 0);
    }

    public String getSearchNumberValidity() {
        return _searchNumberValidity;
    }

    public void incrementSearchesInSession (int amount) {
        _searchesInSession += amount;
    }

    public void incrementBookDescriptionsInSession (int amount) {
        _bookDescriptionsInSession += amount;
    }

    public void incrementBooksInBasketInSession(int amount) {
        _booksInBasketInSession += amount;
    }

    public void incrementReservationsInSession (int amount) {
        _reservationsInSession += amount;
    }

    public boolean isLastSearchNumberValid() {
        return _searchNumberValidity.equals("valid");
    }

    public long getEndLong() {
        long endLong = _endDate.getTime() + _endTime.getTime();
        return endLong;
    }

    public long getStartLong () {
        long startLong = _startDate .getTime () + _startTime .getTime();
        return startLong;
    }

    public boolean matches(LogLine logLine) {
        boolean matchFound = false;
        try {
            if (hasSessionTag ()) {
                if (_sessionTag.equals(logLine.getSessionTag()) &&
                    getEndLong() > logLine.getLongTime() − 1200000) {
                    matchFound = true;
                }
                else if (_ipAddress.equals(logLine.getIpAddress()) &&
                        logLine. getReferrer () != null &&
                        _lastRequestUrl . toString (). equals(logLine. getReferrer ()
                                                            . toString ()) &&
                        _browser.equals(logLine.getBrowser()) &&
                        getEndLong() < logLine.getLongTime() − (20∗60∗1000)) {
                    matchFound = true;
                }
            }
            else {
                if (! logLine. referrerHasSessionTag ()) {
                    matchFound = (_ipAddress.equals(logLine.getIpAddress()) &&
                                  logLine. getReferrer () != null &&
                                  _lastRequestUrl . toString ()
                                  .equals(logLine. getReferrer (). toString ()) &&
                                  _browser.equals(logLine.getBrowser()) &&
                                  getEndLong() > logLine.getLongTime() − (20∗60∗1000));
                }
            }
        }
        catch (Exception e) {
            System.out. println ("Session.matches()␣failed ..." );
            System.out. println (e);
            System.out. println (e.getMessage());
        }
        return matchFound;
    }

    public void setLastSearchNumber(int newSearchNumber) {
        _lastSearchNumber = newSearchNumber;
    }

    public void setSearchNumberValidity(String newValidity) {
        _searchNumberValidity = newValidity;
    }

    public String toString () {
        StringBuffer buffer = new StringBuffer ();
        try {
            buffer .append(_sessionKey + "\t");
            buffer .append(_sessionTag + "\t");
            buffer .append(_ipAddress + "\t");
            buffer .append(_browser + "\t");
            if ( _referrer == null) {
                buffer .append("no␣value\t");
            }
            else {
                if ( _referrer . toString (). endsWith("\\"))␣{
                    _referrer = new URL(_referrer. toString (). replaceAll ("\\\\",␣"" ));
                }
                buffer .append( _referrer + "\t");
            }
            buffer .append( _firstRequestUrl + "\t");
            buffer .append(_firstPageKey + "\t");
            buffer .append(_lastRequestUrl + "\t");
            buffer .append(_lastPageKey + "\t");
            buffer .append( _startDate + "\t");
            buffer .append(_startDateKey + "\t");
            buffer .append(_startTime + "\t");
            buffer .append(_startTimeKey + "\t");
            buffer .append(_endDate + "\t");
            buffer .append(_endDateKey + "\t");
            buffer .append(_endTime + "\t");
            buffer .append(_endTimeKey + "\t");
            buffer .append(_pagesInSession + "\t");
            buffer .append(_bookDescriptionsInSession + "\t");
            buffer .append(_booksInBasketInSession + "\t");
            buffer .append( _reservationsInSession );
        }
        catch (Exception e) {
            System.out. println ("Session. toString ()␣ failed ..." );
            System.out. println (e);
        }
        return buffer . toString ();
    }

    public String activeSessionToString () {
        StringBuffer buffer = new StringBuffer ();
        try {
            buffer .append(_sessionKey + "\t");
            buffer .append(_sessionTag + "\t");
            buffer .append(_ipAddress + "\t");
            buffer .append(_browser + "\t");
            buffer .append( _firstRequestUrl + "\t");
            buffer .append(_firstPageKey + "\t");
            buffer .append(_lastRequestUrl + "\t");
            buffer .append(_lastPageKey + "\t");
            if ( _referrer == null) {
                buffer .append("no␣value\t");
            }
            else {
                if ( _referrer . toString (). endsWith(" \\"))␣{
                    _referrer = new URL(_referrer. toString (). replaceAll (" \\\\",␣" " ));
                }
                buffer .append( _referrer + "\t");
            }
            buffer .append(_startDate + "\t");
            buffer .append(_startDateKey + "\t");
            buffer .append(_startTime + "\t");
            buffer .append(_startTimeKey + "\t");
            buffer .append(_endDate + "\t");
            buffer .append(_endDateKey + "\t");
            buffer .append(_endTime + "\t");
            buffer .append(_endTimeKey + "\t");
            buffer .append(_pagesInSession + "\t");
            buffer .append(_bookDescriptionsInSession + "\t");
            buffer .append(_booksInBasketInSession + "\t");
            buffer .append( _reservationsInSession + "\t");
            buffer .append(_lastSearchNumber + "\t");
            buffer .append(_searchNumberValidity);
        }
        catch (Exception e) {
            System.out. println ("Session. activeSessionToString ()␣ failed ..." );
            System.out. println (e);
        }
        return buffer . toString ();
    }
}
```

E.12 UrlQuery Class

```
package aub;

public class UrlQuery {

    private String [] variables , values;

    public UrlQuery(String query) {
        String [] variablesAndValues = query. split ("&");
        variables = new String[variablesAndValues. length ];
        values = new String[variablesAndValues. length ];

        for ( int v = 0; v < variablesAndValues. length ; v++) {
            int equals = variablesAndValues[v]. indexOf("=");
            if ( equals != −1) {
                variables [v] = variablesAndValues[v]. substring (0, equals );
                values[v] = variablesAndValues[v]. substring (equals + 1);
            }
            else {
                variables [v] = "";
                values[v] = "";
            }
        }
    }

    public int getIntValue ( String variable ) {
        String stringValue = getValue( variable );
        try {
            return Integer . parseInt ( stringValue );
        } catch (Exception e) {
            return −1;
        }
    }
    public String getValue(String variable) {
        String value = null;
        for (int v = 0; v < variables.length && value == null; v++) {
            if (variables[v].equals(variable)) {
                value = values[v];
            }
        }
        return value;
    }
}
```

APPENDIX F

Questionnaire



Hvordan bruger du Auboline?
Jeg har i mit speciale udviklet et system, som AUBs personale kan bruge til at analysere de anonyme brugeres adfærd i Auboline. Jeg er i den forbindelse i gang med at undersøge brugernes egne opfattelser af deres adfærd i Auboline for at finde ud af, hvordan dette stemmer overens med de resultater, jeg er kommet frem til med mit system. Derfor vil jeg gerne have dig til at bruge fem minutter på at svare på nedenstående spørgsmål.
Tak for hjælpen! Louise Due.

1. Hvor ofte bruger du Auboline? (Ét kryds)
☐ Hver dag
☐ Flere gange om ugen
☐ Flere gange om måneden
☐ Ca. en gang om måneden
☐ Sjældnere
☐ Aldrig

2. Hvordan går du ind i Auboline? (Afkryds gerne flere muligheder)
☐ Bogmærke direkte til Auboline
☐ Bogmærke til AUBs hovedside
☐ Skriver Aubolines URL (a500.aub.auc.dk) i browseren
☐ Skriver URLen til AUBs hovedside (www.aub.auc.dk) i browseren
☐ Via link på anden hjemmeside. Hvilken:
☐ Andet: ______________________________

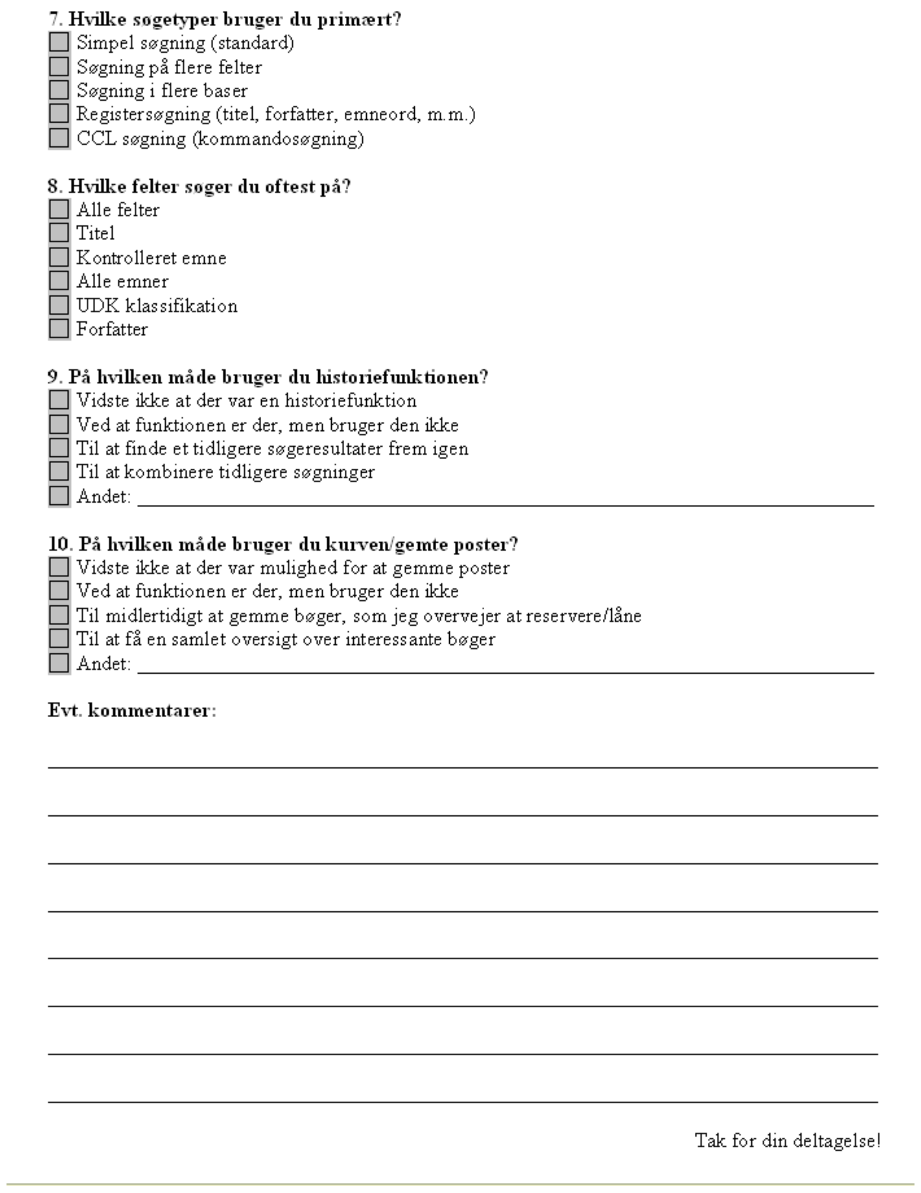
3. Hvornår på døgnet bruger du Auboline mest? (Ét kryds)
☐ Mest i arbejdstiden/skoletiden
☐ Mest i fritiden

4. Hvornår på ugen bruger du Auboline mest? (Prioriter med 1-4, hvor 1 er oftest)
☐ I starten af ugen
☐ Midt i ugen
☐ I slutningen af ugen
☐ I weekenden

5. Hvordan varierer din brug af Auboline i løbet af semestrene?
I starten af semestret ca. _____ gange om måneden.
I midten af semestret ca. _____ gange om måneden.
I slutningen af semestret ca. _____ gange om måneden.

6. Hvilke funktioner bruger du mest i Auboline?
☐ Søgninger efter bøger
☐ Reservationer
☐ Låneroplysninger
☐ Informationer om hjemlån
☐ Fornyelse af hjemlån
☐ Finde opstilling/placering af bøger

VEND ⇨



7. Hvilke sogetyper bruger du primært?
☐ Simpel søgning (standard)
☐ Søgning på flere felter
☐ Søgning i flere baser
☐ Registersøgning (titel, forfatter, emneord, m.m.)
☐ CCL søgning (kommandosøgning)

8. Hvilke felter soger du oftest på?
☐ Alle felter
☐ Titel
☐ Kontrolleret emne
☐ Alle emner
☐ UDK klassifikation
☐ Forfatter

9. På hvilken måde bruger du historiefunktionen?
☐ Vidste ikke at der var en historiefunktion
☐ Ved at funktionen er der, men bruger den ikke
☐ Til at finde et tidligere søgeresultater frem igen
☐ Til at kombinere tidligere søgninger
☐ Andet: ____________________

10. På hvilken måde bruger du kurven/gemte poster?
☐ Vidste ikke at der var mulighed for at gemme poster
☐ Ved at funktionen er der, men bruger den ikke
☐ Til midlertidigt at gemme bøger, som jeg overvejer at reservere/låne
☐ Til at få en samlet oversigt over interessante bøger
☐ Andet: ____________________

Evt. kommentarer:

Tak for din deltagelse!

APPENDIX G

Summary

In this project a web usage mining tool has been designed and implemented. The AUBA tool has been implemented to aid the staff at Aalborg University Library (Aalborg UniversitesBibliotek, AUB) in analyzing the behavior of the users of Auboline, the library's on-line system that borrowers can use to search for books. The library staff is interested in analyzing user behavior because they can use the information to improve their services to the borrowers and thereby increase the loan numbers to improve the economy of the library.

The input for the AUBA tool is a collection of web server log files. Analysis of the log files and experimentation with the different functions of Auboline has lead to knowledge about the possibilities of the analysis of the data. To store the information from the log lines in a logical structure that is understandable and easily accessible, the data in the database has been logically structured in two star schemas, page_event and search. The structure of the page event star schema is common in data webhouses. It enables analysis of sessions and page functions as well as variations in activity according to different time and date parameters. The search star schema is structured specifically for Auboline, because searches are the central part of the system. With the search star schema it is possible to make analyses of the use of different types of search types and relate the number of book descriptions, basket saves and reservations to the specific searches that lead to the activity. It is possible to extend the data webhouse schema whenever there is a need for new types of analysis or new information becomes available.

As the main part of the implementation, the post-processor has been implemented to handle the ETL processes. It transforms the pure text input files to fit the format of the data webhouse. The performance of the post-processor is satisfactory due to the limited amount of input/output activity between the java program and the database. Extensive work has been done to assure that the data flows through the ETL processes as expected and that the database will not contain flawed data. Each incremental load is done in a single transaction to assure that the data is either fully loaded if there are no problems or not loaded at all if there is an error, such as a power outage, so the process can be restarted from scratch when the problem has been fixed.

Part of the motivation behind the implementation of the AUBA tool was to enable the AUB staff to analyze the success of the book recommendation system to be integrated with Auboline. Therefore one of the goals of the project has been to prepare the AUBA tool for analysis of the use of book recommendations. The dimensional database schema has been designed in a way that the recommendation service usage data will easily fit into. Unfortunately it has not been possible to analyze the use of the recommendation service since it has not yet been integrated with Auboline.

A simple graphical user interface has been implemented to make sure that the AUB staff can find results quickly without knowing the underlying structure of the database. Point-and-click can be used to find the results of predefined queries while advanced users can query the database manually for more complicated requests. The results that can be obtained with the graphical user interface can be used by the AUB staff to gain knowledge about the behavior of the users of Auboline.

Different steps have been taken to improve performance. For the loading of the data, communication with the database has been kept at a minimum. For the query part, performance have been optimized in several ways. Outriggers have been placed between dimension tables to avoid joining with the fact table whenever possible. Indexing is also used, but the most substantial query performance improvement has been achieved by using summary tables in the AUBA tool. This is a very important improvement because the users of the AUBA tool will most likely loose interest if they have to wait 20 minutes for each requested result. The materialized views have been hand coded because PostgreSQL dost not include the option of materializing views. Updating the materialized views takes a long time because they are recalculated from scratch. This is a temporary solution that is important to improve. It will be much faster to update the summary tables incrementally instead of recalculating them.

A user survey has been conducted to illustrate the difference in the kind of information that can be gained with different methods and to compare the borrowers' perception of their behavior in Auboline with the results from the AUBA tool. Each method has its own advantages and drawbacks, and therefore it is a good idea to use a combination of several methods of gathering information to get a more clear picture of what is going on. For instance, it is a good idea to make a thorough analysis of the user behavior in Auboline with the AUBA tool and then interview the borrowers about why they behave the way they do.

Because the AUBA tool has been developed specifically for AUB it can give the staff analysis results that are targeted directly on the structure and content of their system. The AUB staff that have been involved in the project have shown a lot of interest in the AUBA tool, because it gives them information that they have not previously been able to get. Furthermore they find the future possibilities of the tool very exciting.