DAT6 project
Aalborg University
Spring 2004

Building a Business Intelligence System for

(second edition)

Group members:	Trien Huy Ly	|	jianfai@cs.auc.dk
	Jacob Mogensen	|	jacobm@cs.auc.dk
	Kristian Reesen Skouboe	|	atommax@cs.auc.dk

Faculty of Engineering and Science
Aalborg University

Department of Computer Science

TITLE:
Building a Business Intelligence System for AUB (second edition)

PROJECT PERIOD:
DAT6,

February 1st -
June 11th, 2004

PROJECT GROUP:
E3-117

GROUP MEMBERS:
Trien Huy Ly
Jacob Mogensen
Kristian Reesen Skouboe

SUPERVISOR:
Torben Bach Pedersen

NUMBER OF COPIES: 6

REPORT PAGES: 144

APPENDIX PAGES: 19

TOTAL PAGES: 163

SYNOPSIS:

This report describes the development of a business intelligence (BI) system for AUB. The foundation of the BI system is a data warehouse. It can be used for finding association rules, getting statistics, and finding book recommendations.

At first, AUB's requirements are described. Based on the requirements the data warehouse for AUB is developed.

We have implemented our own algorithm (LIQ) for finding association rules. Furthermore, we have made an hybrid out of LIQ and Apriori to improve performance by benefiting the strengths of both worlds.

In addition, we have implemented a recommendation service based on both an item-based and a content-based approach in order to improve the quality of the recommendations.

The implementations for both association rule mining and the recommendation service are described along with performance tests of the implementations.

Finally, in order to demonstrate the statistical features, association rule mining, and recommendation service, we have implemented a data access web tool for this purpose.

Authors

Trien Huy Ly

Jacob Mogensen

Kristian Reesen Skouboe

Preface

We would like to thank AUB for providing us with sample data and answering our questions during the course of building the BI system.

The source code for the BI system along with a Java API can be found at:
http://www.cs.auc.dk/~jianfai/BISystem.zip

The online data access tool developed for AUB is available at:
http://multirac.aub.auc.dk:8080/myapp/

Username: aub
Password: aub110604

Contents

1 Introduction **13**

2 Requirements **17**

2.1 Existing Services 18
2.2 Existing System Architecture 18
2.3 System Architecture Extension 19
2.4 Platform 22
2.4.1 DBMS 22
2.4.2 Web Server 24
2.4.3 Platform Experiences 25
2.5 Privacy 25
2.6 System Criteria 27

3 Data Warehouse **29**

3.1 Data Warehouse Design 30
3.1.1 The Four Steps 30
3.1.2 Data Warehouse Components 32
3.2 Implementation 42
3.2.1 Extraction 44
3.2.2 Transformation 44
3.2.3 Loading 48
3.2.4 Logging 49
3.2.5 Maintenance 49

3.3 Performance 51
3.3.1 Comparing the Implementations 52
3.3.2 Scalability 53
3.3.3 Evaluation of Execution Time 54
3.3.4 Evaluation of Memory Usage 55
3.4 Future Work 57

4 Association Rule Mining 59
4.1 Association Rules 60
4.1.1 Association Rule Example 60
4.1.2 Association Rules terms 61
4.1.3 Generalized Association Rules 62
4.1.4 Association Rules Constraints 63
4.2 Finding Large Itemsets 63
4.2.1 Apriori 64
4.2.2 LIQ 64
4.2.3 LIQ Hybrid 65
4.2.4 LIQ2 65
4.2.5 FPGrowth 66
4.3 Implementation 68
4.3.1 Large Itemset Query 70
4.3.2 LIQ2 72
4.4 Performance 74
4.4.1 Comparing Apriori, LIQ, and LIQ2 75
4.4.2 Finding the Switching Point 77
4.4.3 LIQ Hybrid 78
4.4.4 Scalability of LIQ Hybrid vs. FPGrowth 79
4.5 Extension of FPGrowth 81
4.5.1 SQL Based Frequent Pattern Mining with FPGrowth . 81
4.6 Future Work 82

4.6.1 New Hybrid between LIQ and FPGrowth 83
4.6.2 Privacy Preserving Mining of Association Rules 83
4.6.3 Sequential Patterns 85

5 Recommendation 87
5.1 Recommendation 88
5.1.1 Item-based Collaborative Filtering 88
5.1.2 Content-based Recommendation 92
5.2 Implementation 97
5.2.1 Item-based CF 98
5.2.2 Content-based Recommendation 103
5.2.3 Recommendation 106
5.3 Performance 109
5.3.1 Execution Time 109
5.3.2 Scalability 109
5.3.3 Online 111
5.4 Evaluation 115
5.4.1 Item-based CF 115
5.4.2 Content-based Recommendation 116
5.4.3 Improving the Quality 118
5.5 Related work 119
5.5.1 User-based Collaborative Filtering 119
5.5.2 Amazon's Collaborative Filtering System 119
5.6 Future Work 122

6 The Data Access Tools 125
6.1 The Librarian Service 125
6.2 Recommendation Service 131
6.3 Using the System 133
6.4 Feedback 133
6.5 Future Work 134

6.5.1 OLAP 134
6.5.2 Graphical overviews 136
6.5.3 Help and Support 136
6.5.4 Aggregate Navigator 137

7 Conclusion **139**

A Sample Data **145**

B Data Warehouse **147**
B.1 Dimensions and Fact Table Definitions 147
B.2 Table Definitions for Aggregate Tables 149

C Association Rules **153**
C.1 Loan Table 153
C.2 UDK Table 154
C.3 StatusType Table 154

D Recommendation **155**

E Bibliography **159**

1 Introduction

The library of Aalborg University (Danish: Aalborg Universitetsbibliotek, AUB) wants an extension of their current system that can recommend relevant books to the borrowers when they search for books on their web site. Furthermore, the extension should deliver additional information from AUB's current system that can assist in optimizing the support from the librarians, minimizing the expenses on irrelevant books, and targeting the purchase of popular books.

The type of system extension that AUB wants is called a *business intelligence* (BI) [29] system. BI is a broad category of applications and technologies for gathering, storing, analyzing, and providing access to data to help enterprise users make better business decisions.

AUB is a public research library for the city of Aalborg and its neighbors. Its primary task is to support research and education for Aalborg University (AAU) by having relevant information available. AAU has about 2,000 employees and about 11,000 borrowers. AUB is placed geographically together with AAU, with a small department in Esbjerg. The library has about 300,000 editions of books available for borrowing.

The BI system consists of two applications; one for the business users and another for the borrowers. The business users are in particular the librarians. The application for the librarians, the librarian service, provides so-called association rules, which is information about which books have been borrowed together with which other books. Furthermore, the librarian service provides statistical data where the librarians among other things can see statistics about loan data. For the borrowers a recommendation service has been implemented in order to recommend relevant books, when they search for books on AUB's web site. We have developed a data warehouse for AUB in order to be able to make statistical queries, association rules, and recommendations.

The recommendation service does, as the name suggests, recommend books to borrowers. Based on the similarity of books a list of similar and presumably relevant books is presented for the borrower, whenever a particular book has been selected by the borrower. For instance, we could have a borrower who has just found the book *The Hobbit*, the recommendation service will then recommend a list of books, which are similar to *The Hobbit.* This recommendation is based on the likings of the community of users which have also borrowed *The Hobbit.* A list of recommended books could for instance contain *The Lord of the Ring*, *The Silmarillion*, and *The Atlas of Middle-Earth.* We have implemented an item-based collaborative filtering algorithm from [10] that makes use of our data warehouse. Initially, it is difficult to come up with enough recommendations due to the sparse amount of loan history. In order to cope with this kind of cold-start problem, we have implemented a content-based algorithm to compensate for the initial lack of recommendations.

The librarian service for mining association rules gives the librarians a list of rules stating which books are associated with each other. An example of an association rule could be *Hound of the Baskervilles* associated with *The Dead Zone*, which means that when borrowers borrow the book *Hound of the Baskervilles*, they also tend to borrow the book *The Dead Zone.* We have implemented our own algorithm called LIQ and made a performance test of it together with the classic Apriori algorithm. From the test result an appropriate hybrid of the two algorithms was created in order to test its scalability against FP-Growth, which is known to scale well.

We have made a web site that collects statistics and association rules from the data warehouse. Therefore, the librarian service also provides statistical data by giving a level-wise presentation of the data. For instance it is possible to see the distribution of loans over years, months, weeks, and days. Furthermore, it is possible to get a list of most popular books and most active borrowers.

All the services access the data from the *data warehouse*, which is the very core of the system. Data from AUB's operational system is extracted, transformed and loaded into the data warehouse on a regular basis. The data warehouse uses PostgreSQL as the DBMS.

The structure of the report is as follows. Chapter 2 covers AUB's requirements and the choice of platform on which the system is built. Based on the requirements Chapter 3 explains the different components in the data warehouse. In Chapter 4, we get into association rule mining, where we implement our own general algorithm LIQ, Large Itemset Query, and improves on it to better the performance. Chapter 5 concerns the implementation of the recommendation system. Chapter 6 presents our data access tools. Finally, Chapter 7 concludes on the report. Furthermore, the report includes appendices about the sample data from AUB, the Data Warehouse, Association Rules, and Collaborative Filtering.

This report is a continuation of the previous one [33]. In the following the new contributions are briefly outlined. The individual chapters will further elaborate on the subjects:

- The performance of the ETL process has been significantly improved. The execution time has dropped from 4 - 5 hours to approximately 2 minutes for 10 months of loan data.
- The data warehouse design has been modified such that two mini-dimensions have emerged from the edition and borrower dimensions in order to make it easier to present the data hierarchically as described in Chapter 3, where the data warehouse design is discussed.
- Our own algorithm, LIQ, has been modified such that it is capable of finding large itemsets constrained on the status and type of the borrower. For instance, it is possible to mine association rules for borrowers from a certain field of study and semester. Furthermore, a level-wise view of association rules is also possible. In an attempt to minimize the risk for unique identification of borrowers of rarely borrowed books a minimum threshold for support has been added.
- The performance of the item-based implementations has been significantly improved. The execution time has dropped from a couple of hours based on 10 months of loan data to approximately 3 minutes. Furthermore, a content-based approach has been implemented to compensate for cold-start problems due to an initially sparse amount of loan data.

2 Requirements

AUB would like to recommend relevant books to the borrowers when they search for books on their web site. A borrower's current usage scenario is to browse onto AUB's web site, input some specific keywords for the books the borrower is searching for, let AUB's system search for books matching the keywords and then the borrower browses through the result and the borrower can then choose to reserve some of the books for borrowing.

The borrowers should be supported in the search of relevant books since studies have shown that borrowers generally do bad searches. [18] describes how 50% of all searches are done using only 2 search terms or less and 80% of all searches do not contain any operators. All of these searches results in many matching books, which the borrower then has to scan through manually in order to find the most relevant. This is not very effective and the borrower might skip highly relevant books.

Furthermore, the system should deliver additional information from AUB's current system that can assist in optimizing the support from the librarians, minimizing the expenses on irrelevant books, and targeting the purchase of popular books.

The following sections will present the current system running at AUB, what the requirements for the new system are, and which criteria for designing the system are important to AUB.

2.1 Existing Services

AUB is a research library and their web site is open to the public. The web site of AUB offers different services, among others are the below mentioned services[1].

- An online catalog called Auboline, which makes it possible to have access to books, journals, electronical journals, notes, and CDs available in the library.
- A books delivery service, which makes it possible to have books delivered at the borrowers home, if you are living outside Aalborg and Esbjerg.
- A suggest new books feature, where it is possible to suggest what new books should be bought by AUB.
- General information is available on the web site to new students.
- A newsletter service, that gives information about new services that AUB offers by sending an email to subscribers.
- A support service, where users can get support from the library via e-mail.

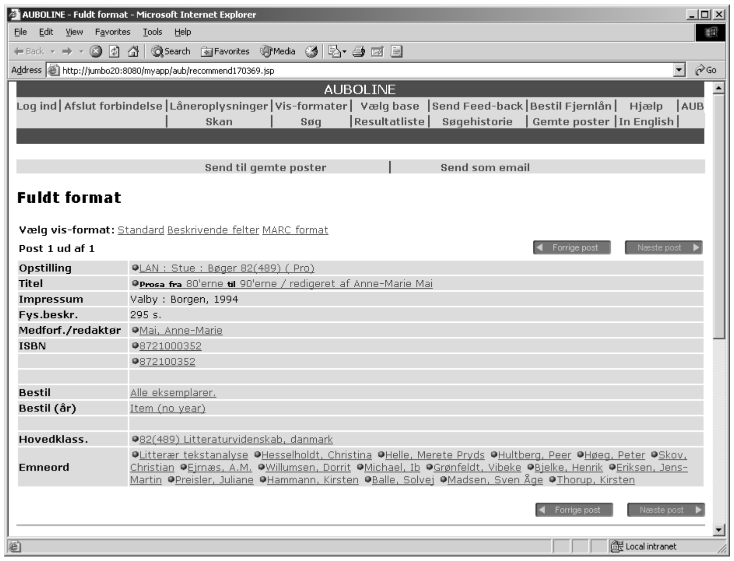
Figure 2.1 shows an example of AUB's current system which is called Auboline. It is possible to search for books given a title, author, or UDK number and then information about the books can be shown. It is when the information for a specific book is shown that it should be possible to see a list of recommendations for other books.

2.2 Existing System Architecture

AUB's web site system runs on a Tomcat server which is a web and application server. The systems runs on the following operating systems: Microsoft Windows 2000 Professional for the client part of the system and Solaris 8 or 9 running on the SPARC or x86 platforms for the server part of the system.

On the frontend, all queries are received by an Apache 1.3.28 server with a set of XML/XSLT tools, which are used in other contexts. The backend

[1]http://www.aub.auc.dk/portal/js_pane/forside/article/60


AUBOLINE - Fuldt format - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://jumbo20:8080/myapp/aub/recommend170369.jsp
Go
AUBOLINE
Log ind | Afslut forbindelse | Låneroplysninger | Vis-formater | Vælg base | Send Feed-back | Bestil Fjernlån | Hjælp | AUB
Skan | Søg | Resultatliste | Søgehistorie | Gemte poster | In English
Send til gemte poster | Send som email
Forrige post
Næste post
Local intranet

Fuldt format

Vælg vis-format: Standard Beskrivende felter MARC format

Post 1 ud af 1

Opstilling	●LAN : Stue : Bøger 82(489) (Pro)
Titel	●**Prosa fra** 80'erne **til** 90'erne / redigeret af Anne-Marie Mai
Impressum	Valby : Borgen, 1994
Fys.beskr.	295 s.
Medforf./redaktør	●Mai, Anne-Marie
ISBN	●8721000352
	●872100352
Bestil	Alle eksemplarer.
Bestil (år)	Item (no year)
Hovedklass.	●82(489) Litteraturvidenskab, danmark
Emneord	●Litterær tekstanalyse ●Hesselholdt, Christina ●Helle, Merete Pryds ●Hultberg, Peer ●Høeg, Peter ●Skov, Christian ●Ejrnæs, A.M. ●Willumsen, Dorrit ●Michael, Ib ●Grønfeldt, Vibeke ●Bjelke, Henrik ●Eriksen, Jens-Martin ●Preisler, Juliane ●Hammann, Kirsten ●Balle, Solvej ●Madsen, Sven Åge ●Thorup, Kirsten

Figure 2.1: Auboline's Web page for a Book.

is an Apache server that has a Java connector, which connects parts of the catalogs on their site with a Tomcat 4.1.18 LE server, which runs the actual portal application.

AUB's portal is run by the Apache Jakarta Jetspeed Portal software, which is based on the XML data format. XML data is being delivered using web services to an Apache Axis web service concentrator.

The whole content foundation in the portal is stored in a MySQL database. Data within it is normalized according to the XML syntax, but because MySQL is not a web service, Axis takes care of this and thereby enables MySQL to function as a web service. Using the same procedure other databases are also connected to a web service using Axis.

2.3 System Architecture Extension

The services mentioned in the beginning of the Existing Services section is common to many library web sites. AUB is interested in using their existing data to further provide new and better services for the borrowers and business users, i.e., the librarians.

AUB would like to base the new system on open source applications in order to cut costs, i.e., mainly to save license costs. Open source applications such

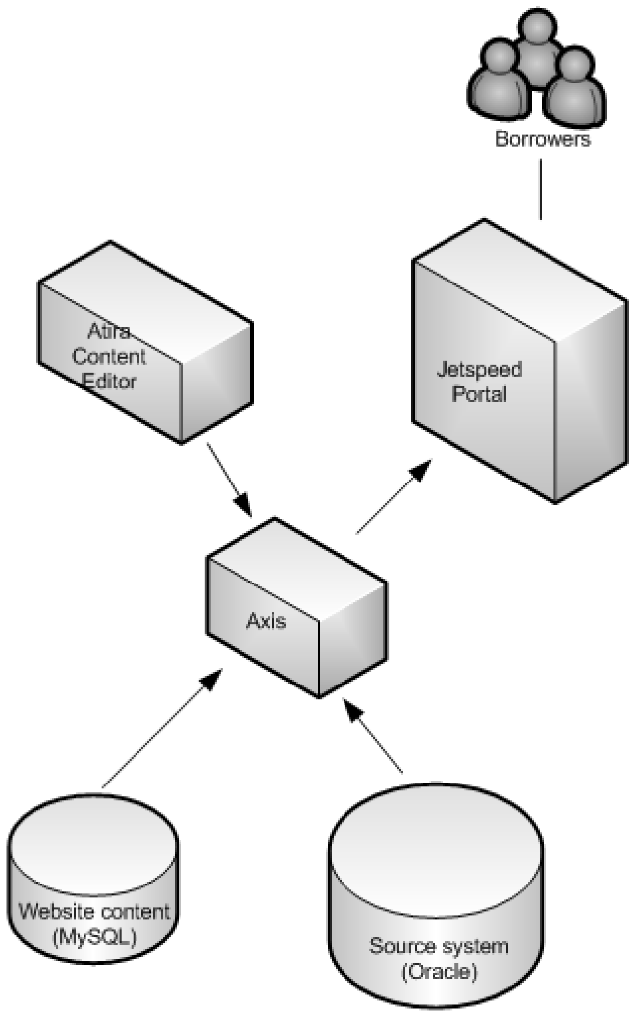


Figure 2.2: AUB's Current System Architecture.

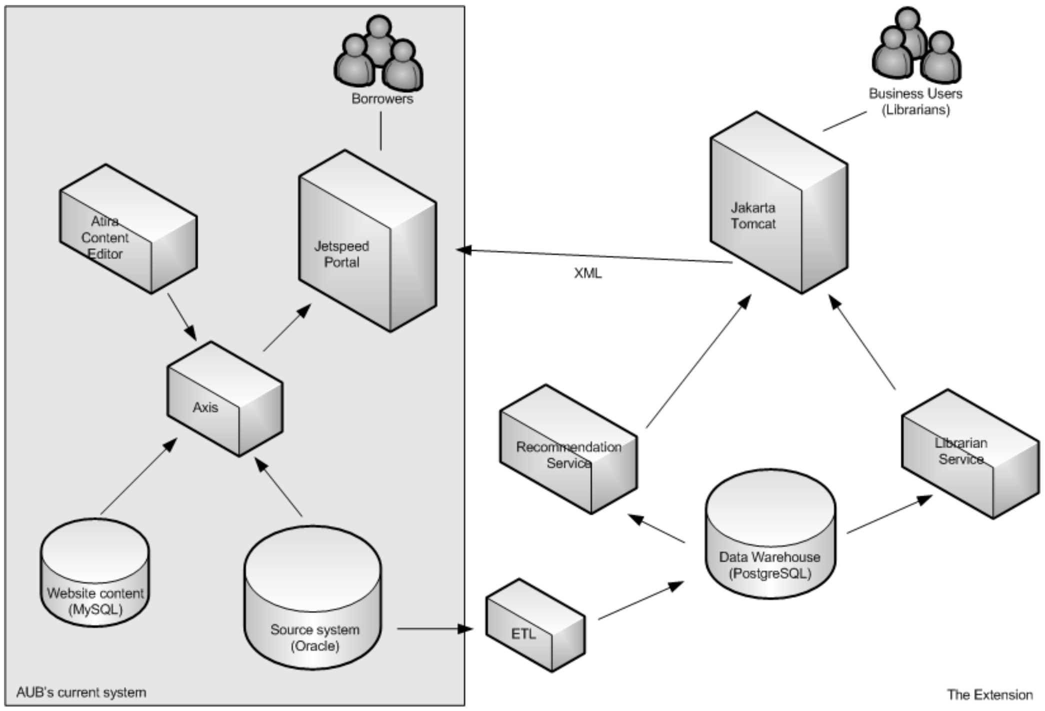


Figure 2.3: AUB's new System.

as Tomcat and PostgreSQL have also matured over the years. Today, they can be used to provide a reliable web application system thus Tomcat and PostgreSQL have been chosen to be used.

The new system extension will consist of three parts:

- A data warehouse.
- A recommendation service that can be integrated in Auboline and be available to the borrowers.
- A librarian service that provides statistics and association rules through a web site to the business users.

On Figure 2.3 the new system architecture for AUB is shown. It consists of the above three parts. The ETL extracts and transforms data from the source system that runs on an Oracle DBMS in AUB's current system. Then the ETL loads the data into the data warehouse that runs on a PostgreSQL DBMS in our system extension. The recommendation and librarian services use the data warehouse for providing data to the Jakarta Tomcat web and application server. Tomcat provides statistics and association rules to the business users. Furthermore Tomcat provides recommendations to the Jetspeed Portal using XML. The Jetspeed Portal can then provide recommendations to the borrowers using Auboline.

The users of the system are the business users and the borrowers. The system will add new services to AUB's existing system and we will now go through these services.

The new services in the system can be divided into two main parts:

1. **The Recommendation Service:**

   Recommendation of books. A recommendation of books assists the borrower in focusing on related books more conveniently. This concept of a recommendation list is inspired by Amazon.

2. **The Librarian Service:**

   Relation between co-borrowed books for the business users. The system tells which books are often borrowed together. This is done by finding association rules using an algorithm. This makes it possible for AUB to, e.g., make book collections that consist of books that are often being borrowed together.

   Statistics for the business users. It should possible to see statistics over time, e.g., number of books being borrowed per day, month, quarter, week, and year. Statistics about the categories for the books are also available.

2.4 Platform

AUB's requirements described in this chapter place some constraints on platform choices. This section will describe the chosen DBMS and web server meeting the constraints.

2.4.1 DBMS

The DBMS should adhere to the following criteria:

- Meet the SQL92 standard. Among other features it should enable the possibilities for transactions, materialized views, and referential integrities.
- Provide support for Java.
- Free to use.

PostgreSQL

PostgreSQL complies to the above mentioned criteria and has thus been chosen. PostgreSQL [26] is an enhancement of the Postgres [12] database management system, which is a next generation DBMS research prototype. PostgreSQL has the same data model as Postgres, which is relational, but it replaces the PostQuel [11] query language with an extended subset of the Structured Query Language (SQL).

PostgreSQL presently conforms to most of the Entry-level SQL92 standard, as well as many of the Intermediate- and Full-level features. Additionally, many of the features new in SQL99 are quite similar to the object-relational concepts pioneered by PostgreSQL (arrays, functions, and inheritance). It has the same features as most commercial DBMSs, like transactions, subselects, triggers, views, foreign key referential integrity, different types of indexing, row locking, and stored procedures. However materialized views are as yet not supported. A workaround is to manually apply the "CREATE TABLE AS" command to represent a materialized view. Furthermore, PostgreSQL is free and open source. It runs on most UNIX platforms, but it is portable to, e.g., Microsoft Windows by using the Cygwin application, which is the Cygnus Unix/NT porting library[2].

The speed of PostgreSQL varies according to its documentation. However, it is in some ways faster than, e.g., MySQL when having multiple users, complex queries, and a read/write query load. On the other hand, MySQL is faster than PostgreSQL when doing simple SELECT queries by few users [26]. Furthermore, PostgreSQL is considered to be very stable and well tested according to its documentation. If support is needed a large groups of developers and users can help in resolving the problems.

In general, most Unix-compatible platforms with modern libraries should be able to run PostgreSQL and it requires at least 8 MB of memory and at least 45 MB of disk space to hold the source, binaries, and user databases.

[2]http://www.cygwin.com

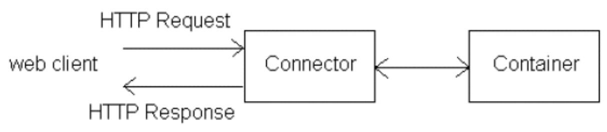


Figure 2.4: Figure showing the Catalina servlet container functionality.

2.4.2 Web Server

The web server should adhere to the following criteria:

- It should provide an easy way to deploy web applications.
- The client should be able to easily access the services via an Internet browser.
- It should be free.

Tomcat 4.1

The choice fell on Apache Tomcat 4 by Apache Software Foundation [15] as it fulfills the above mentioned criteria. The product is a standalone Web server that supports Java Servlet and JavaServer Pages technologies (JSP) [21]. Tomcat is the Jakarta Project's [16] open-source servlet container and is released under the Apache Software License [17].

Figure 2.4 illustrates how the servlet container works in Tomcat. As illustrated, a client sends a web request to the Apache Tomcat Server. Tomcat then directs the request to the appropriate servlet container. The servlet container computes result and sends it back to the client via Tomcat.

In order to develop software for Tomcat you need to install the Java programming language and use some kind of development software, e.g., Borland JBuilder Personal Edition, which is available for free. However, it is possible to develop the software just using the Java Compiler, which is available free of charge.

A user browsing a web site using Tomcat does not need any extra software except a browser in order to see the web pages.

2.4.3 Platform Experiences

When using PostgreSQL on a Windows 2000 Professional machine, you need to have Cygwin installed. This have an impact on the system and tends to slow down the DBMS, because a lot of I/O activity is used by Cygwin. However, PostgreSQL has a lot of advanced features, and is available for free. When deploying the system in a production environment, we intend to use PostgreSQL on a UNIX-based operating system, thus avoiding the overhead of having Cygwin to emulate a UNIX-like environment on Windows. It is also important that the PostgreSQL is configured for UNICODE in order to handle the international characters in the AUB data.

Regarding Tomcat it was easy to set it up on a Windows machine. However, Java Virtual Machine[25] needs to be installed since it is used by Tomcat. In order to ease the use of Tomcat you also need to install an application called Ant. Ant is also used by Tomcat in order to be able to update the JSP web pages and compile the classes more easily. Without Ant you would need to do a lot of manual work in order to get your code to run on Tomcat. Ant provide an easy way to deploy the web applications.

2.5 Privacy

When releasing data such as the information collected by the AUB business intelligence system, it is important to protect the anonymity of the individual borrowers. It is important that specific borrowers cannot be identified from the information made available from the AUB business intelligence system. It should also be impossible to link other sources of data to the information from the AUB business intelligence system and thereby indirectly identify specific borrowers. A person with access to the association rules should, e.g., not be able to link knowledge of what books a borrower has borrowed to other books borrowed by the same borrower. The privacy ensures that any persons can freely borrow books without the loan history being used for political reasons or sold for profit.

Several techniques have been developed with respect to preserving privacy. Some of the techniques consist of adding noise, swapping values, etc., while still maintaining an overall statistical property. These techniques leads to distorted data, which is not always useful when doing data mining [31]. [31] describes two techniques called suppression and generalization, which provide privacy protection while maintaining the integrity of the data. The techniques make use of the following common terms:

1. **Quasi-identifiers**
   Attributes that can be exploited for linking, i.e., used to link with attributes in external data sources for identifying borrowers.

2. **K-anonymity**
   Describes the degree of protection of data with respect to inference by linking. Using an external data source the linking could, e.g., map to three borrowers and then 3-anonymity would be upheld.

Suppression and generalization as described in [31] ensures k-anonymity for some given k value. Suppression achieves k-anonymity by removing tuples. Generalization is a technique that uses, e.g., dimensional hierarchies in order to archive k-anonymity. The values are made less informative in order to ensure k-anonymity. The best technique is to use both suppression and generalization.

A technique proposed by [7] and even further improved by [5] modifies the values of a field using:

1. **Value-Class Membership**
   Partition the values into a set of disjoint, mutually-exhaustive classes and return the class into which the true value x_i falls.

2. **Value Distortion**
   Return a value $x_i + r$ instead of x_i, where r is a random value drawn from some distribution.

3. **Amplification**
   Make it possible to guarantee limits on privacy breaches without any knowledge of the distribution of the original data.

Values-Class Membership, Value Distortion and Amplification ensures that Data Mining is possible while still preserving Privacy. The techniques alters the original data in such a way that it is not possible to accurately estimate original values.

However, by using a novel reconstruction procedure to estimate the distribution of the original values, it is possible to do Data Mining, e.g., using Decision-tree classifiers or Association Rule Mining algorithms. The classifiers' accuracy is comparable to the accuracy of classifiers built with the original data. As proven in [7] it is possible to be within 5% to 15% (absolute) of the original accuracy, even when the privacy is 100%. 100% privacy means that the true value cannot be estimated any closer than an interval of width which is the entire range for the corresponding attribute.

Though, in order to implement privacy-preserving Data Mining as proposed by [7] or [5], we will need to change, e.g., their algorithms. The reason for this is that the algorithms have been designed for Decision-Tree classification. Therefore, it would require a deeper understanding of the algorithms and we have chosen to give higher priority to other things. However, another technique proposed by [6] shows how to do privacy preserving mining of association rules.

The AUB business intelligence system must be developed with regards to quasi-identifiers and k-anonymity in order to ensure a minimum level of privacy. No techniques such as suppression and generalization have to be applied since no complete tables of data are released to the public. The data is only accessed through tools, which can ensure the privacy. For more information about what we have implemented see 4.6.2.

2.6 System Criteria

Summing up this chapter, the criteria for designing the different parts of the system are the following:

General:

- The system should be able to run on AUB's existing platforms and hardware.
- The system has to use open source applications in order to save license costs.
- The Java programming language is used.

Data Warehouse:

- The data warehouse should seamlessly synchronize with the source system. It should harvest useful information from existing data that is not already known, for instance which books are borrowed together. It should also reflect updates in already loaded data.

- The Danish law of privacy[3] has to be obeyed, e.g., the users identity have to be protected in the system in order to uphold anonymity. It is not legal to monitor a user.
- The system has to use the PostgreSQL system as a DBMS server and Tomcat as a web and application server.

Librarian Service (Business Users):

- The system should show statistics.
- The system should also show association rules.
- Help should be available to the business users.

Recommendation Service (Borrowers):

- The recommendation service should run as a recommendation service meaning that it produces XML output.
- It should be easy to integrate the recommendation service in Auboline by being able to make use of the data from the recommendation service. The Auboline system is already using different Web Services and XML.

[3]http://www.datatilsynet.dk/include/show.article.asp ?art_id=443&sub_url=/lovgivning/indhold.asp

3 Data Warehouse

AUB has a system for handling the different daily tasks at the library. Such a system is called an *operational system*. For the operational system the main priorities are processing time and availability. The queries only deal with one or a few rows at a time. The business users, i.e. librarians, use the operational system to add books, borrowers, and register loans. Thus most of the queries to the operational system are of the same type just with different values. There is, e.g., a query for registering a loan. This is a single query, which is used with different values depending on borrower, book, and time. This query is done several times each day. An operational system also has to handle a large amount of updates.

BI services do not deal with single rows. Instead the services have queries that deal with several million rows. The queries often change since they arise from the questions that the business users might ask. The business users might, e.g., be interested in loan overviews by month. The information that they get from this overview might then lead to the need of overviews by week, day, and hour. These changing queries requires a flexible system that is simple so that the business users can understand and use it.

The requirements from the BI services and the requirements listed in Chapter 2 can be met by implementing a data warehouse for AUB. The data warehouse is robust meaning that it can easily be adjusted to support new queries and it can easily be changed. The data warehouse consists of several data marts where each has associated a collection of dimensions [29]. A data mart is a simple system for decision support since it focuses on a single business process as, e.g., book loans. The simplicity is important in order to get the business users to accept and use the data warehouse.

The following sections describes how a data mart for the AUB data warehouse is designed, how it is implemented, the performance of the different implementations, and future work.

3.1 Data Warehouse Design

The design for the AUB data warehouse is described in this section following a description of the different components.

3.1.1 The Four Steps

According to Ralph Kimball et. al. [29] there are four steps for developing a data warehouse design. These four steps ensure that overall design decisions are made first before deciding on the details. The four steps are:

1. **Select the business process**
   A business process is the selected process at the business organization that is to be modeled by a data mart in the data warehouse. This can, e.g., be managing employees or book loans at AUB.

   The business process is selected by considering the business requirements from AUB and the available data in the source system at AUB. As described in Chapter 2 AUB wants to analyze the different relations regarding how the books are borrowed by using the data that is captured by the current source system.

   The business process that is to be modeled for AUB is the process of book loans by borrowers. The model will allow AUB to analyze the relations between loans, borrowers, books, time of day, and which books are borrowed together. The analysis of which books are borrowed together requires the loans to be grouped. The books that are borrowed together by the same user are grouped, just like market basket data in a data warehouse for a retail business.

   The selected business process for the data mart is very much like the traditional retail example where the business process is the point of sale, see [29] chapter 2. Here the business users want to analyze customer purchases in a store by doing market basket data analysis and looking at the relations between purchases, customers, products and time. The market basket analysis is discovering information about products bought together. The most famous result from market basket analysis was the discovery of a relation between diapers and beers. The relation showed that the customers buying diapers also had a tendency to buy beers.

2. **Declare the grain**
   Declaring the grain is selecting the level of data detail that is to be stored in the fact tables. The level of grain is set to the process of a book loan meaning that each level represents the process of a book loan with the milestones loan and return date. The fact table type used is an accumulative snapshot. It provides a clear picture of the duration between the main events, i.e, loan and returnal of a book. In a transaction fact table a process of a book loan would consist of two rows, one for loan and one for returnal, and the duration would therefore not be clear from a single row. The only difference of the two types of fact tables in the case of the AUB data warehouse is that the accumulative fact table contains extra date keys in order to represent milestones.

3. **Choose the dimensions**
   The dimensions are found by looking at the selected business process. When registering book loans the descriptive dimensions are Date, Time of day, Borrower, Status Type, Edition, UDK, and Transaction. These dimensions represent the time of day and date of the loan, borrower, edition, UDK, and the status and type of the borrower which borrowed an edition of a book. UDK is the Universal Decimal Classification (Danish: Universelle Decimal-Klassifikation, UDK) for an edition and denotes where the edition of a book is placed in the library. The Transaction and Borrower dimension are both a *degenerate dimension* (DD) [29] since the associated dimension table is empty because the information regarding the transaction or borrower is already located in the other dimensions. The Transaction dimension is only used for grouping loans into transactions.

4. **Identify the facts**
   The fact table is a factless fact table since there are no measurable facts. The fact table models a loan history, i.e., the process of a book being borrowed and returned. The fact table models two events, and events are often modeled as a factless fact table since they rarely have any obvious numeric facts associated with them.

   The rows in a factless fact table represent a process by making a relation between the foreign keys of the dimensions. The foreign keys are often a collection of different dates when the fact table is an accumulative snapshot. The relation in the fact table captures the loan history by a specific edition, borrower including date and time for each milestone.

3.1.2 Data Warehouse Components

The data warehouse consists of four components, which are developed in accordance to the design. Data is extracted from the operational system. The extracted data is transformed by a data staging. The transformed data is loaded into the data presentation area. The business users access the data in the presentation area through the data access tools. The specific components of the AUB data warehouse are described next and they are also illustrated in Figure 3.1.

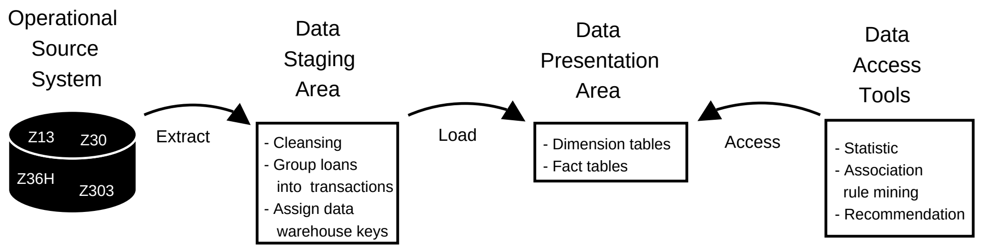


Figure 3.1: The AUB Data Warehouse Components.

Operational System

The operational system at AUB consists of an Oracle DBMS running a database containing several tables. The complete documentation containing the detailed schemas cannot be disclosed as AUB consider it as classified documentation. The tables used from the operational system is therefore only partly described in this section.

The four main tables from the operational system and the important attributes from the tables are listed in the diagram in Figure 3.2. The main tables are:

- **Z13** contains information about each book and is therefore mentioned in the following sections as the *book source* table.
- **Z30** contains information about each edition and is mentioned as the *edition source* table.

- **Z36H** contains the loan history and is mentioned as the *loan source* table. Loan history is added to this table when a book is returned.
- **Z303** contains information about each borrower and is mentioned as the *borrower source* table.

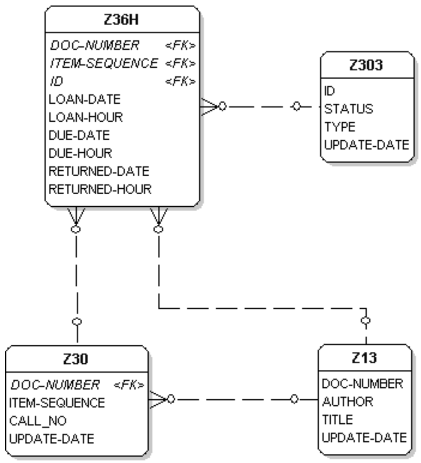


Figure 3.2: Diagram of the Main Tables from the Source System.

The primary key for the *loan source* table is the REC-KEY. The value of this attribute can be split into a DOC-NUMBER and ITEM-SEQUENCE. The DOC-NUMBER refers to a book in the *book source* table while DOC-NUMBER and ITEM-SEQUENCE are composite foreign keys referring to an edition in the *edition source* table. Since rows are only added to the *loan source* table when a book is returned, the accumulative snapshot in the AUB data warehouse does not need to be updated. Each date for every milestone is already registered when updating the data warehouse.

The date format used in the source system is YYYYMMDD, e.g., 20030125 while the time format is HHMM.

The CALL-NO attribute in the *edition source* table is the UDK for the specific edition and denotes where the edition of a book is placed in the library. An example of an UDK is 543.422.25 where 543 denotes chemistry, 422 denotes absorption analysis, and 25 denotes NMR spectroscopy.

Data Staging Area

One of the requirements from AUB was that the data warehouse should use the existing source system. These requirements lead to a data staging area

Source Attribute	**Description**
Z36H-REC-KEY	An administrative system number consisting of a DOC-NUMBER and ITEM-SEQUENCE.
Z36H-ID	A foreign key referring to borrower.
Z36H-LOAN-DATE	The date of the loan.
Z36H-LOAN-HOUR	The time of the loan.
Z36H-DUE-DATE	The date the book has to be returned.
Z36H-DUE-HOUR	The time the book has to be returned.
Z36H-RETURNED-DATE	The date the book was returned.
Z36H-RETURNED-HOUR	The time the book was returned.

Figure 3.3: Attributes from the *loan source* Table.

that has the following extract-transformation-load (ETL) processes:

1. **Extraction**

   The data is extracted from the *loan source* table and *borrower source* table. The tables and the attributes that are used for extracting data are described in Figure 3.3 and Figure 3.4. The UPDATE-DATE attribute in the *borrower source* table and the LOAN-DATE attribute in the *loan source* table are used in order to extract only the new and updated rows. The rest of the attributes including LOAN-DATE are extracted and used for loading into the presentation area.

   Data from the *edition source* table and *book source* table is not extracted directly from the source tables but instead a view is defined using the attributes from the *edition source* table and *book source*. The two tables are listed in Figure 3.5 and Figure 3.6. The view combines the *edition source* table and *book source* table in order to support the loading of data into the presentation area. This way the view is updated whenever either the edition or book data is updated.

2. **Transformation**

   The transformation cleans the data, groups the loans into transactions, assigns data warehouse keys, and handles the issue of reflecting the updates of already loaded data from the source system.

Source Attribute	**Description**
Z303-ID	The primary key.
Z303-BOR-STATUS	The status of the borrower.
Z303-BOR-TYPE	The type of the borrower.
Z303-UPDATE-DATE	The update date for the row.

Figure 3.4: Attributes from the *borrower source* Table.

Source Attribute	**Description**
Z30-DOC-NUMBER	A foreign key referring to the book source table.
Z30-ITEM-SEQUENCE	Is along with the DOC-NUMBER the primary key.
Z30-CALL-NO	The UDK for the edition.
Z30-UPDATE-DATE	The update date for the row.

Figure 3.5: Attributes from the *edition source* Table.

Source Attribute	**Description**
Z13-DOC-NUMBER	The primary key of the book source table.
Z13-AUTHOR	The author of the book.
Z13-TITLE	The title of the book.
Z13-UPDATE-DATE	The update date for the row.

Figure 3.6: Attributes from the *book source* Table.

Attribute	**Description**
Surrogate Dimension Key	The assigned key that is used in the data warehouse.
Operational Key	The operational key associated with surrogate dimension key.
Checksum	A checksum of the data from the operational system.

Figure 3.7: The Master Dimension Cross-Reference Table.

The cleansing consists of trimming the text for starting and ending blanks, spell checking, and handling special symbols. The special symbols that have to be handled are, e.g., the apostrophe in "O´ Connor" and removing escape characters. The apostrophe has to be handled in order to build SQL queries. Removing escape characters, blanks, and spell checking is necessary for not getting different author names for the same author simply because of spelling errors, blanks, or escape characters. Pattern matching is also used in order to combine author names and titles where only a few letters differ.

The transactions are defined as loans by the same borrower at the same day. So if a borrower, e.g., borrows five books on a specific day then these loans will be grouped together by having the same transaction ID. This leads to transactions that are not very fine grained but most borrowers only borrow books once per day. If the transactions, e.g., are defined to be loans within the same hour and by the same borrower then there will only be a few more transactions. It is also more interesting to have larger transactions when doing data mining such as association rule mining. With larger transactions it is easier to find patterns.

The data warehouse keys are assigned by storing operational keys along with the associated surrogate dimension key in three Master Dimension Cross-Reference Tables [29]: A table for the Borrower, Edition, and StatusType dimensions. There is also stored a checksum along with each operational key for the Edition dimension. The checksum for an edition covers author, title, and the UDK levels. The checksum is used for checking whether the data from the operational system has been updated. The design does not only rely on the update date in the *edition source* table since the update might have been of attributes of no interest for the ETL process. The schema for the Master Dimension Cross-Reference Tables are listed in Figure 3.7.

The updates of already loaded data from the *edition source* tables in the operational system are handled by doing the following procedure described in pseudo code:

```
for each extracted row from loan source
    if new operational key for edition
        assign data warehouse key
        make checksum of the attributes: Author, Title
        store operational key, data warehouse key, and checksum
        load row into presentation area using data warehouse key
    if old operational key and new checksum
            make checksum of attributes from source system
            update row in presentation area
            store new checksum for edition
    if old operational key and old checksum
        skip
```

An operational key is assigned a data warehouse key when the row containing it is extracted for the first time. It is called a new operational key in the above pseudo code. A new checksum is created when a row has been loaded and assigned an operational key but the attributes have changed. The assigned data warehouse keys and checksums are stored in the Master Dimension Cross-Reference Tables for the Edition dimension.

The update of status and type for a borrower and UDK for an edition are handled through the use of a mini dimensions [29] in the presentation area. This is done in order to ensure that the loan history for status and type of the borrowers and UDK of editions are represented correctly over time. Updating would lead to misleading history. The status and type values for the borrowers would, e.g., only be the current values. Updates are used in the case of updates of already loaded edition data since these updates often are a matter of correcting spelling errors.

3. **Load**

After extracting and transforming the data it is loaded into the presentation area. The data is stored in the dimensions and relations are created in the fact table. Rows for the Date and Time of day dimensions are not extracted from the source system but are generated in advance. This means that rows covering a day are loaded initially into the Time of day dimension and rows covering ten years are loaded into the Date dimension. The aggregate tables created for the AUB data warehouse are also updated during the load process.

Data Presentation Area

The business users can access the data stored in the presentation area. The data therefore has to be organized so that the business users can understand and use the data. The simplicity of the data model is achieved by using multidimensional modeling [29]. A multidimensional model consists of dimensions and facts. Dimensions provides context for the facts by having descriptive attributes. The facts represent the events and associated measures that are to be analyzed.

Figure 3.8 is a relational representation of the multidimensional model for the AUB data warehouse. The tables are illustrated by listing the fact and dimension tables. The schema definitions used for the PostgreSQL database are listed in Appendix B.

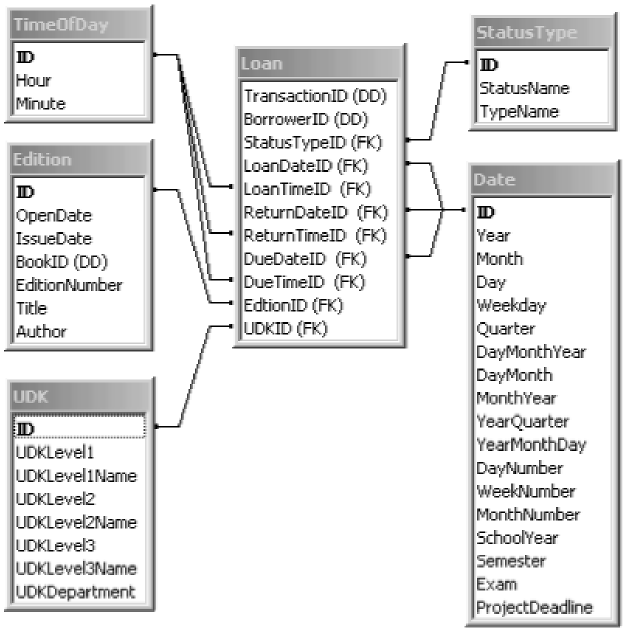


Figure 3.8: Relational Representation of the Multidimensional Schema.

Dimensions The Date and TimeOfDay dimensions are used for registering loans over time and for representing the different milestones. The StatusType, Edition, and UDK dimensions are used for registering the editions and the status and type of the borrowers. The specific loan is registered by the Loan fact table. Both StatusType and UDK dimensions [29] are mini dimensions and their purpose is described below:

The StatusType mini dimension contains every single occurrence of status and type. The type and status for a borrower is then represented by a foreign key in the Loan fact. This leads to a design that reflects the status and type for the borrowers correctly over time. If a borrower gets a new status or type then a new foreign key is used leaving the borrower's ID unchanged. This is why we have chosen not to have an explicit borrower dimension. The mini dimension is also a good entry point for doing selection on status and type.

The UDK mini dimension is used for registering the loan history for changing UDKs for editions. If the UDK of an edition is changed then new rows representing a loan of the edition in the Loan fact table just contain the new UDK foreign key. This way the loan history is not changed when an edition is assigned a new UDK. Furthermore, storing all known combinations of UDK levels provides role-up and drill-down features efficiently.

The Date, Time of day, StatusType, and Edition dimensions contain hierarchies that can be used for roll-up/drill-down operations. Drill down is, e.g., when a business user has a list of number of loans by UDK level 1 and then chooses to add more details by adding an UDK level. Roll up is when a business user has a list of number of loans by UDK level 1 and 2 and then chooses to only look at UDK level 1. This is simply moving up and down the hierarchy that is in the Edition dimension. The hierarchy in the Edition dimension is illustrated in Figure 3.11.

Hierarchies The hierarchy for the Date dimension is listed with a hierarchy schema in Figure 3.9 (a) and is illustrated by an example in Figure 3.9 (b). T represents all of the dimension and the dotted lines in the example represents the rest of the elements from the dimension that belong to the hierarchy. The hierarchy for the Date dimension consists of the levels year, quarter, month, week, and day.

The Time dimension hierarchy contains the levels hour and minute and is listed with a hierarchy schema in Figure 3.10 (a). An example of the Time dimension hierarchy is illustrated in Figure 3.10 (b).

The hierarchy in the Edition dimension consists of the three UDK levels, a book level, and an edition level. The schema for the hierarchy is illustrated in Figure 3.11 (a). An example of the hierarchy is illustrated in Figure 3.11

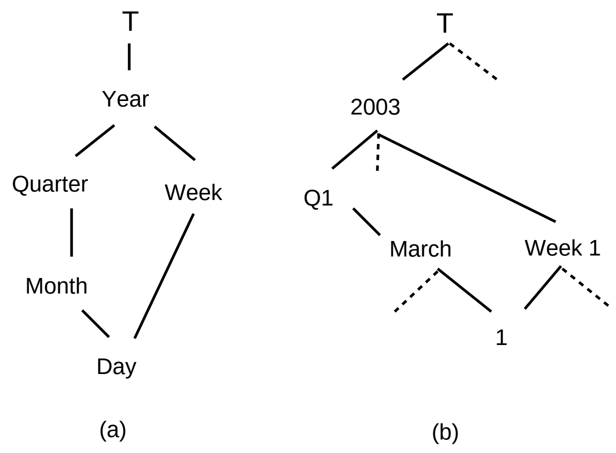


Figure 3.9: Date Hierarchy.

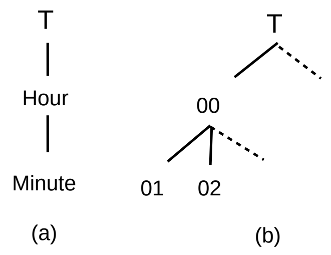


Figure 3.10: Time of Day Hierarchy.

(b). In the example the levels for UDK 543.422.25 are illustrated along with one book and edition from this specific UDK.

The StatusType hierarchy contains the levels status and type. The schema for the hierarchy is listed in Figure 3.12 (a). An example of the StatusType hierarchy is illustrated in Figure 3.12 (b). In the example of the hierarchy the types are social science and natural science and the status are PhD and Employee.

Schema Type The multidimensional model that we use is represented in the form of a star schema. A star schema is used in order to keep the design simple and to have a better performance than if a snowflake schema was used. The performance of a star schema is good because it is denormalized which leads to fewer joins when doing queries. Denormalization also leads to

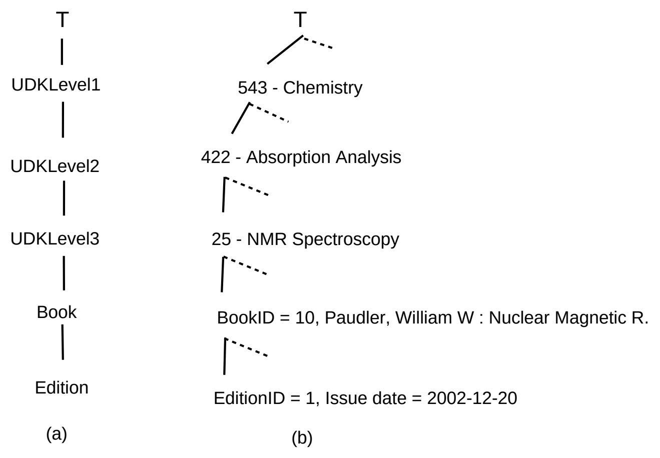

T
UDKLevel1
UDKLevel2
UDKLevel3
Book
Edition
(a)
T
543 - Chemistry
422 - Absorption Analysis
25 - NMR Spectroscopy
BookID = 10, Paudler, William W : Nuclear Magnetic R.
EditionID = 1, Issue date = 2002-12-20
(b)

Figure 3.11: Edition Hierarchy.

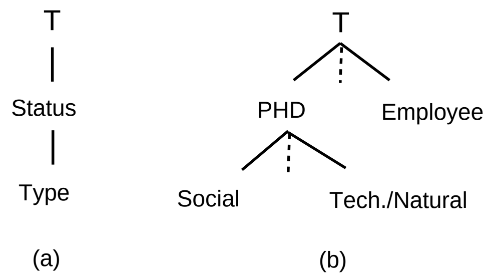


Figure 3.12: StatusType Hierarchy.

a more simple model [29].

A snowflake schema could have been chosen in order to save disk space by using normalization and thereby not having any redundancy. The snowflake schema ensures that updates only have to alter a single row while several rows have to be updated in a star schema. If, e.g., an author name is updated then only a single row is affected in a snowflake schema while the update will cover many rows if a star schema is used.

The snowflake schema is not used because it introduces more joins and it makes the design more complex. The rows in the data warehouse are very seldom updated so there is little overhead of having to update several rows instead of one row. The disk space saved by using the snowflake schema is also insignificant. This is due to the fact that disk space savings gained by using snowflake schemas are typically less than 1 percent of the total disk space needed for the overall schema, see [29] Chapter 2.

Data Access Tools

The data access tools are the front end of the data warehouse. It is through these tools that the business users access the data in the presentation area and these tools are also used for servicing the borrowers by doing book recommendation.

The data access tools can be canned, data mining, and OLAP applications. Canned applications contain prebuilt queries where the users can only set some parameters, e.g., the size of the result set and the sorting order. Data mining applications can, e.g., find patterns in the data, recommend books to borrowers, and predict future values. OLAP applications are used for viewing the data in cubes and for doing online analysis by slicing, dicing, and drilling down and up.

We have implemented a librarian service for supporting decision support for business users, i.e. librarian. The librarian service can, e.g., show an overview of loans by years, months, weeks, and days. It can also show associations between borrowed books. The implemented librarian service is described in Chapter 6 where also screen shots are listed.

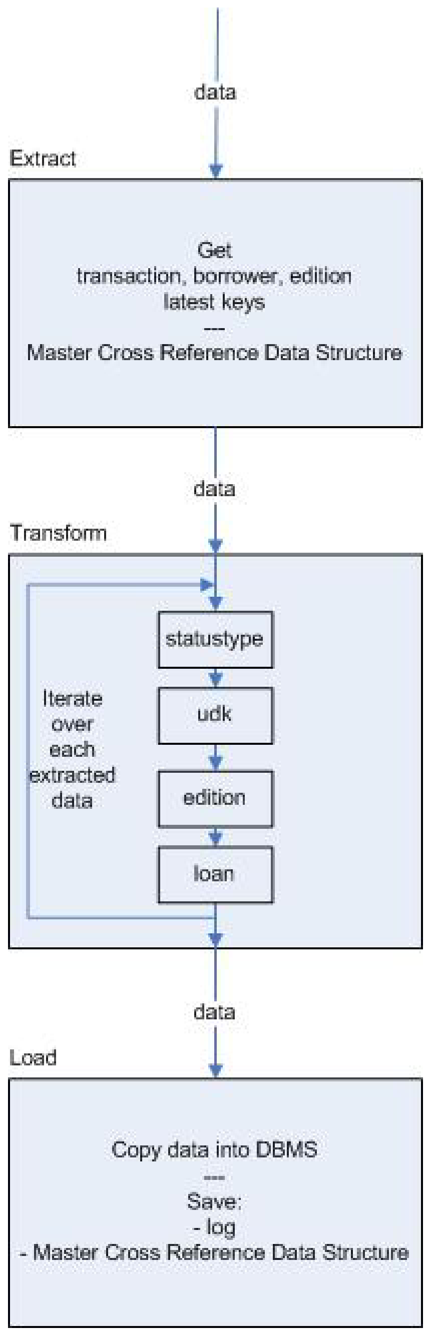
We have also implemented a recommendation service that is used for recommending books to borrowers on AUB's web site. The recommendation service simply recommends a list of similar books given a specific book.

The association rule mining and recommendation are two data mining applications and they are described in Chapter 4 and 5.

3.2 Implementation

The implementation of the ETL processes is divided into three separate responsibilities, as the acronym suggests. This is also depicted in Figure 3.13. For each extracted row the data is transformed. The transformation consists of data cleansing and data warehouse key assignment. When the transformation of all rows from the operational system has been carried out, the transformed data is loaded into the data warehouse.

We have four implementations for the ETL processes. Only the best performing implementation will be described in this section. The next Section 3.3 Performance will compare the execution times of the four implementations.


data
Extract
Get
transaction, borrower, edition
latest keys

Master Cross Reference Data Structure
data
Transform
statustype
udk
edition
loan
Iterate over each extracted data
data
Load
Copy data into DBMS

Save:
- log
- Master Cross Reference Data Structure

Figure 3.13: The ETL Processes.

3.2.1 Extraction

We have an overall method responsible for extracting the rows containing loan data from the operational system. The extraction takes outset in the timestamp of when a row was added or updated to the *loan source* table and the extracted rows are sorted by the timestamp. Only newly added loans are extracted by storing the timestamp of the last extracted row in a special table containing log information about the ETL runs. At an extraction the latest timestamp is fetched from the log table and only the newly added loans are thereby extracted. Using the timestamp in order to partition the data enables the extraction to be run at any frequency as, e.g., an hourly, daily, or weekly routine.

3.2.2 Transformation

Only TimeOfDay and Date dimensions are left untouched as they are pre-loaded during the construction of the multi-dimensional schema. We assign data warehouse keys to each unique edition and borrower by mapping from their corresponding operational key as described in Section 3.1. Furthermore, each known combination of status and type and UDK is also mapped to a data warehouse key.

Java's *HashMap* is used to provide the data structure for the mapping process. This is the implementation version of the master cross-reference table mentioned earlier in Section 3.1.2. The mapping process happens in-memory, which is more efficient than if we query the DBMS, as we avoid the overhead of invoking the query planner, which first have to parse the query in order to build an efficient execution plan. We do a look-up in the appropriate HashMap to retrieve the data warehouse key. If one of the dimension members does not exist in the appropriate HashMap a new data warehouse key is created. The mapping between the dimension member and the created data warehouse key is added to the HashMap. In order to provide an efficient way to look-up data warehouse keys in Date and TimeOfDay dimensions they are stored in separate HashMaps.

Furthermore, we delay adding dimension and fact data to the end of the transformations. All the new data for the dimension and fact tables is buffered separately, i.e., we have a buffer file for each of the dimension and fact tables.

In the following sections the transformation of editions, UDKs, borrowers, status and types, and loans are described.

Edition Transformation

Figure 3.14 depicts the process of transforming editions data. The raw edition data from the source system is initially cleansed. In order to check whether this edition already exists in the data warehouse a look-up in the edition HashMap is carried out.

If a data warehouse key is found, we check whether the changeable attributes, i.e., title and author, have been changed by comparing checksums. Each existing edition has a CRC32 [2] checksum value computed on the changeable attributes. The data warehouse key of the existing edition is then mapped to the checksum value. We, then, compute a checksum value of the newly cleansed edition and look-up the stored checksum value based on the data warehouse key. If there is a difference in the checksum value we assume that at least one of the changeable attributes has been altered and the edition must be updated accordingly. Otherwise, the newly cleansed edition data will not be further processed as no changes has been made.

On the other hand, if the look-up in the edition HashMap thus not return a data warehouse key, we have a new edition, which need to be transformed appropriately. At first we need to establish whether the edition belongs to a new or already existing book. We have a book HashMap for this task. If a data warehouse key of the book does not exist in the book HashMap a new one is created. Otherwise, the existing book data warehouse key is retrieved. In any case, the returned book data warehouse key along with the new edition data warehouse key and the transformed edition data should be added to the edition buffer.

UDK Transformation

Each edition is allocated into certain UDK levels. The original representation of UDK levels is a string containing numbers separated by a dot. The string is split around the first three dots we encounter, thus forming three UDK levels. Again, we utilize HashMap to map each UDK to different data warehouse keys. As UDK levels are represented numerically, they are not very user-friendly for ordinary users to read. After the ETL processes have finished their execution we assign names to the various UDK levels to make it understandable for ordinary users. AUB provided us with a list of UDK level names in a separate file.

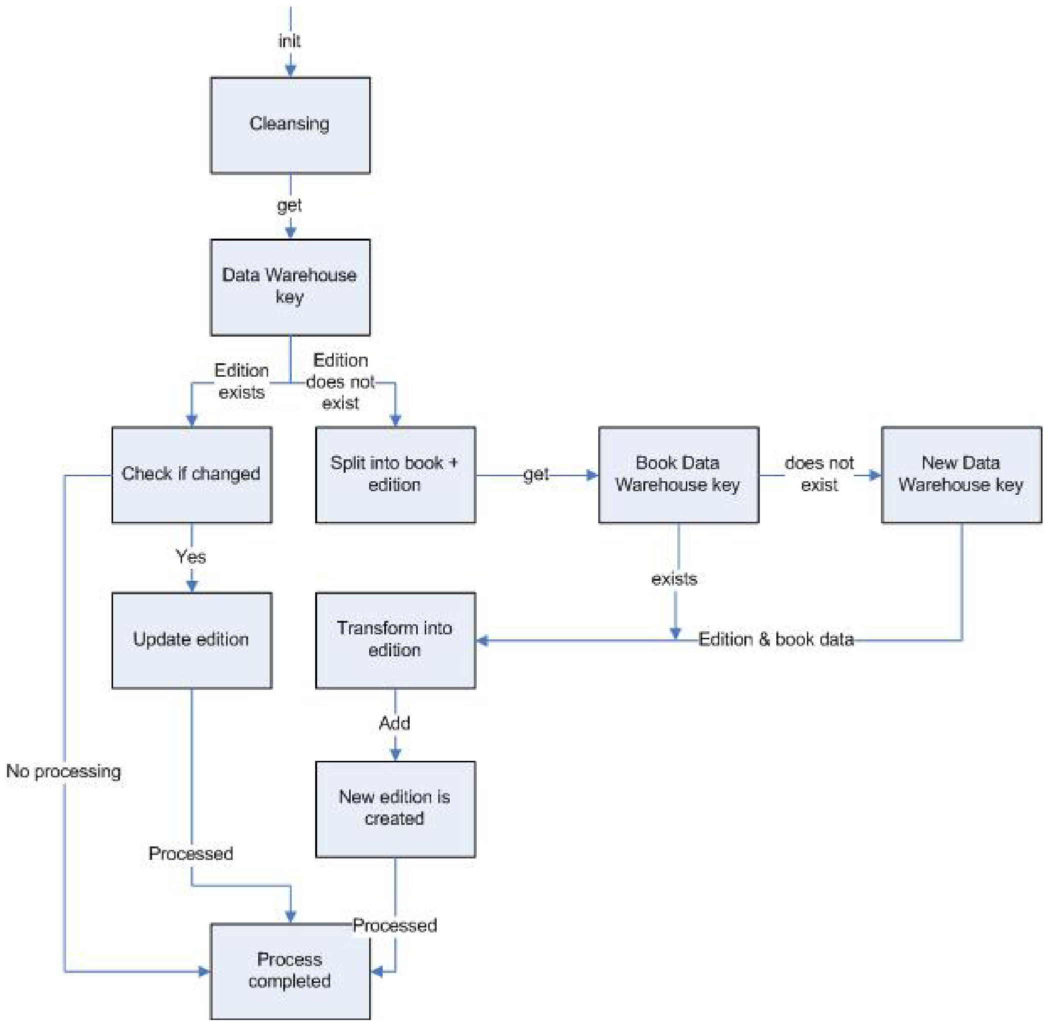


Figure 3.14: Edition Transformation Process.

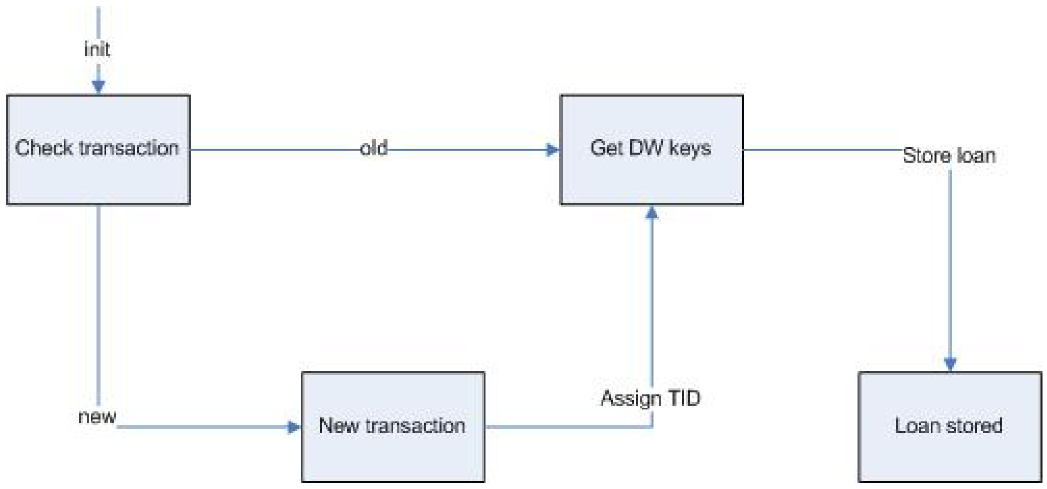


Figure 3.15: Loan Transformation Process.

Borrower and StatusType Transformation

When a borrower's data warehouse key is added to the fact table, the appropriate status and type data warehouse key is also added to the fact table. This makes it possible to track the course of change of each borrower's status and type. Furthermore, it will also make it more efficient to provide role-up and drill-down features.

Loan Transformation

Figure 3.15 depicts the process of transforming loan data. For each loan we need to check whether it belongs to an existing transaction or forms a new one. A transaction is defined as a combination of borrower data warehouse key and loan date data warehouse key. This combination is mapped to a transaction data warehouse key and the mapping is stored in a HashMap. This way we ensure that a transaction corresponds to a set of loans by the same borrower on the same day. If no transaction data warehouse key exists in the HashMap for the loan, a new transaction data warehouse key is created. Otherwise the existing transaction data warehouse key is retrieved. In any case, the returned transaction data warehouse key is added to the loan buffer along with the loan data, which consists of data warehouse keys for edition, UDK, borrower, status and type, date and time of day.

It is important to first map the data warehouse keys for edition, borrower, status and type, and UDK before adding the loan to the fact table as it depends on the mentioned data warehouse keys. This is also illustrated by the order of borrower, edition, status and type, UDK, and loan transformation in Figure 3.13.

Saving the HashMaps

When the transformation process is ended, the HashMaps for Edition, Borrower, StatusType, UDK, TimeOfDay, and Date keys are stored in separate BLOBs in the database. The HashMaps are only stored if the transformation process completes without any errors. The next invocation of the ETL processes will restore the BLOBs from the database.

The Buffered Files

During the transformation process, the data which ought to be added to the respective dimensions and fact table, is stored into files until the whole transformation process has ended. At the start of the transformation process the data files are emptied in order to ensure that the files do not contain corrupted data. Buffered files are used in order to limit the I/O. After completing the transformation process the stored data is loaded from the files into the data warehouse.

3.2.3 Loading

The data is loaded into the data warehouse using the COPY command of PostgreSQL. The COPY command copies data to and from a table in a database in PostgreSQL. When using the COPY command with a filename, it instructs the PostgreSQL backend to directly read from or write to a file. All the insertions made by the COPY command are considered as one transaction.

In order to cope with sudden errors occurring during the loading process we take an ”all or nothing” approach, i.e., either all the transformed data is properly loaded into the data warehouse or no data is loaded at all. On failure the ETL processes need to be re-started. We consider the loadings of edition, UDK, status/type, and loan data as one atomic transaction, i.e., we commit at the end of the loadings. Below is depicted the idea of ”all or nothing” in pseudocode:

```
BEGIN
    COPY new editions into data warehouse.
    COPY new status/types into data warehouse.
    COPY new UDKs into the data warehouse.
    COPY new loans into data warehouse.
    UPDATE changed editions.
COMMIT
```

The load process also updates the aggregate tables in the data warehouse by adding new rows into the aggregate tables representing the newly added data. The aggregate tables are all listed in Appendix B. They have been chosen for the AUB data warehouse in order to support the most of the queries in the data access web tool.

We have attempted to drop the indices of the dimensions and fact table prior to copying and afterwards creating the indices again. The result was a 2 seconds gain in execution time for 10 months of loan data. This indicates that we need to bulk load large amount of loan data before we can out-weight the cost in time for dropping and creating indices. As AUB plans to invoke the ETL processes frequently we cannot out-weight the cost and thus will not attempt to drop the indices before bulk loading and afterwards create them again.

3.2.4 Logging

After having executed the ETL processes, the application logs the number of borrowers, editions, UDKs, and loans inserted into the data warehouse. An example of an entry in the log is given in Figure 3.16. The execution time in milliseconds is logged as well as the number of new transactions that the loans were grouped into. Each loan contains an end time, which is a timestamp indicating when it was added to the *loan source* table. The most recent of the end times of the present ETL processes is also logged. The end time is used the next time the application is run so only the new data is extracted from the source system.

3.2.5 Maintenance

AUB can run the Java application for ETL in a daily routine by setting up a Cron job on a Unix or Linux system. Getting the Java application to run in a daily basis is done by adding it to the Cron configuration file. When the Cron job is set the Java application for ETL is run automatically after a specified periodic interval and the administrator is sent an email containing

Log	Value
Loans	64371
Borrowers	8758
Editions	43685
Transactions	29373
Execution Time (ms)	122737
End Time	200310031146566

Figure 3.16: An Example of an Entry in the ETL Log.

the output if there are any errors. For further information regarding setting up a Cron job see [13].

A Cron job can also be applied to database maintenance activity for reclaiming unused disk space. Every time an UPDATE or DELETE statement is issued the affected tuple is not removed and thus not available for re-use. To avoid infinite growth of disk space requirements it is recommendable to reclaim the unused space. This is done by using PostgreSQL's VACUUM command [4].

When loading data into our data warehouse we primarily makes use of the COPY command. In this case there would be no unused space to be released. However, we also use UPDATE statements when changes occur for an existing edition, which is already in the data warehouse. An UPDATE statement of a row does not immediately remove the old version of the row. This approach is necessary to gain the benefits of multi version concurrency control. The row must not be deleted while it is still potentially visible to other transactions. Eventually, the old row version is of no interest to any transactions and the space it occupies should be reclaimed using the VACUUM command.

PostgreSQL use statistics of table contents to choose the most appropriate query planner in order to speed up query processing. When major changes have been made to the content of the data warehouse dimensions, it is recommendable to update the statistics of table contents. For this purpose the ANALYZE command can be used [4].

An advisable approach for maintenance could be to run both VACUUM and ANALYZE when ETL processes have ended. VACUUM is a time, processor, and disk intensive activity, thus it should be run at a time of day, where the load of the server is low, typically at night. Running ANALYZE is a matter of seconds and should be run immediately after VACUUM.

3.3 Performance

In this section, we will look at the performance of four different ETL implementations. We have tried to optimize the implementation of the ETL processes in order to achieve the fastest execution time possible. Therefore, we developed different ETL implementations and tested the execution time of these.

We have applied four methods to improve the execution time of the ETL processes. The execution times of the four methods is illustrated in Figure 3.17. We have named the four methods after the technique forming the main reason for their level of execution time. The first method is the one we took outset in. Each following method contains improvements compared to their predecessors:

Index is the method which we took outset in. We have placed single column indices on the dimension and fact table attributes, which we select and constrain upon. For each loan extracted from the operational system we send several queries to the master cross-reference tables in order to check whether the borrower or the edition already exist in our data warehouse. If they exist, they are not added to our data warehouse, otherwise they are added. Finally, the loan is inserted into the loan fact table.

All in all, we send SELECT queries to test for the existence of edition, book, UDK, borrower, status, and type. Furthermore, six additional SELECT queries are issued to retrieve the data warehouse key for loan, return and due date and time, respectively. In addition, data warehouse keys for date and time of day are also queried for each loan. Each time a loan has been processed, the appropriate insertions are made to the data warehouse dimensions and fact table by doing INSERT queries. The multiple select queries result in as many parses followed by invoking the query planner to figure out an efficient way to carry out the queries. We hypothesize that this is the main reason why the execution time is higher than the three other implementations.

The number of statements in the Index implementation can be decreased by using a join in order to find the data warehouse keys. The main idea of the approach is to first load the loan data into a temporary loan fact table containing operational keys. The data warehouse keys are then found and added to the real loan fact table by doing a join between the temporary loan fact table and the master cross-reference tables. The join simply maps the operational keys to data warehouse keys and the resulting rows are then added to the loan fact table.

Copy introduces a significant improvement to the ETL processes as we reduced the number of times which our application communicated with the DBMS. Previously, each loan data extracted from the operational system resulted in one immediate INSERT query to the DBMS. Here, we defer this action by buffering the insertion statements in buffered files until the end of the ETL processes, where we do a bulk load into the DBMS using the COPY command of PostgreSQL.

PL/pgSQL function also uses the COPY command of PostgreSQL. What it does differently than the previous method is that the assignment of data warehouse key is done in the DBMS using a PL/pgSQL function. The technique is first to copy the raw data into a temporary table in the DBMS. Next, the PL/pgSQL function is executed in order to split the raw data into different temporary tables, e.g. edition and loan, assign data warehouse keys to each tuple in each table, and update the relations between the tuples. Lastly, the PL/pgSQL function appends the temporary tables to the real data warehouse tables and drops the temporary tables.

HashMap is based on the idea of significantly reducing the CPU load by the DBMS. A new data warehouse key is mapped to each new operational key of borrowers, editions, books, and status/type. The mapping data structure used is Java's HashMap. Instead of sending several queries to the DBMS for each loan, which was the case in *Index*, we "query" the relevant HashMap in memory, which significantly reduces the execution time. The new data warehouse data are still buffered to the end of the ETL processes, where the COPY command is used. Furthermore, the HashMaps are stored in the database as BLOBs until the next invocation of the ETL processes where they will be retrieved into memory.

3.3.1 Comparing the Implementations

In order to be able to make a fair comparison of using HashMaps stored in memory and using the DBMS, we have to optimize the memory usage of the DBMS [20]. The PostgreSQL DBMS has settings stored in a configuration file that specifies how much memory the DBMS is allowed to use. One

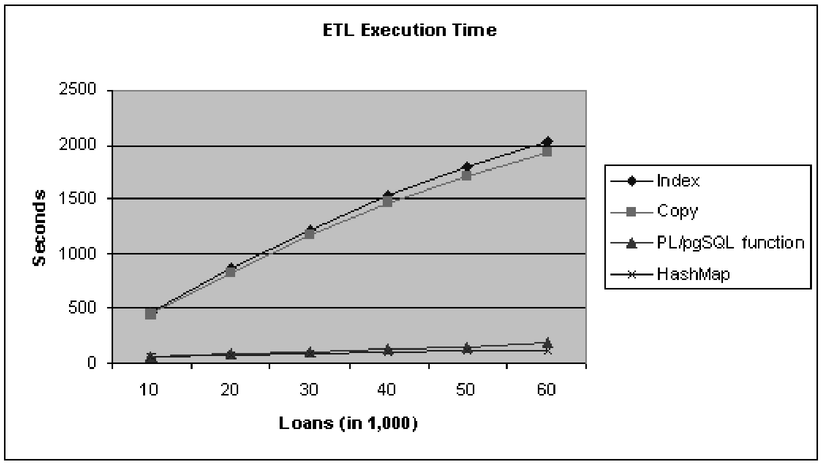


Figure 3.17: ETL Execution Time for all Four Methods.

setting is a buffer size. PostgreSQL uses a shared memory segment among its sub-threads to buffer data in memory. The PostgreSQL DBMS standard settings defaults to use about 1,000 shared buffer blocks. One buffer block is 8 kilobytes. We have therefore increased the amount of buffers to 20,000 which when monitoring the memory usage of PostgreSQL corresponds to using 300 megabytes of RAM.

The implementation of the four methods follows two fundamentally different strategies, which is expressed in Figure 3.17 where *PL/pgSQL function* and *HashMap* performs equally well and much better than *Index* and *Copy*. The difference lies in the number of times the query planner is invoked. *PL/pgSQL function* and *HashMap* reduce this to a minimum as the assignment of data warehouse keys happens either entirely in the DBMS or our Java Application. However, this is not the case for *Index* and *Copy* as they query the DBMS each time they need to assign a data warehouse key. The query planner is the bottleneck which is reflected in the difference in execution time shown in Figure 3.17.

3.3.2 Scalability

All four methods scale almost linearly. The execution time does not grow proportional to the increase of loans. Though the execution time does increase as more loans are processed. Instead they seem to perform better the more loans they have to process because much less time is spent on populating the master cross-reference tables or its equivalent, HashMaps. The heavy work is done in the initial 10,000 loans where almost each loan consists of either a new edition or borrower, which needs to be added to the HashMaps. As more and more loans are processed we encounter still fewer new editions and borrowers. This is expressed by the decrease of the slope of the graphs as we get farther to the right side of the x-axis. Figure 3.17 does not clearly express this characteristic for HashMap because it performs so much better than the three other methods and thus is pushed to the bottom of the graph. In order to show that HashMap has the same characteristics, we have depicted it in a separate graph, which can be seen in Figure 3.18.

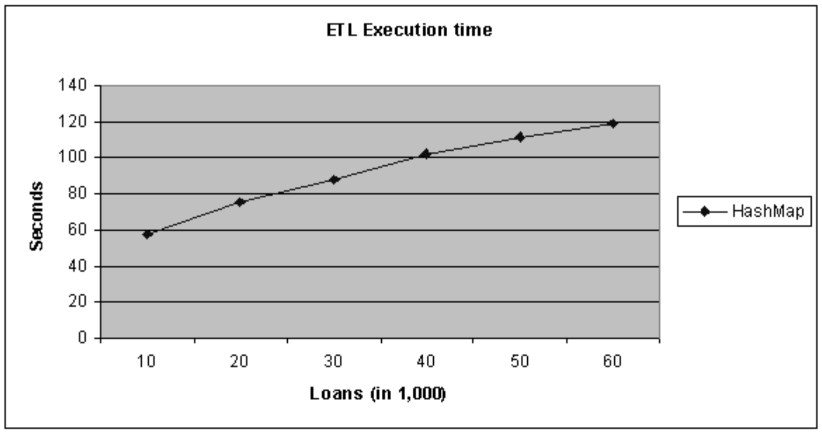


Figure 3.18: ETL Execution Time for HashMap.

3.3.3 Evaluation of Execution Time

Figure 3.19 illustrates how much faster HashMap executes compared to the three other methods with a load of 60,000 loans, which equals to approximately 10 months of loan data. Not surprisingly, HashMap is the winner. It runs the ETL processes 28 times faster than both *Copy*, and 55 times faster than *Index*. However, the PL/pgSQL function performs almost as good as HashMap. The reason why HashMap runs that much faster is because it minimizes the number of queries. we only need to communicate with the DBMS at the beginning and end of the ETL processes. The reason for this is the storing of the data warehouse keys for the edition, book, borrower, status/type, and date dimensions in BLOBs. The data warehouse keys are then stored in memory when the ETL processes are initialized. In the beginning, we need to establish a connection to the operational system in order to retrieve new loan data. Furthermore, the Blobs containing the mapping between operational and data warehouse keys need to be retrieved. Once all the processing has been carried out, we do a bulk load of the transformed data into our data warehouse, as described in the Implementation section.

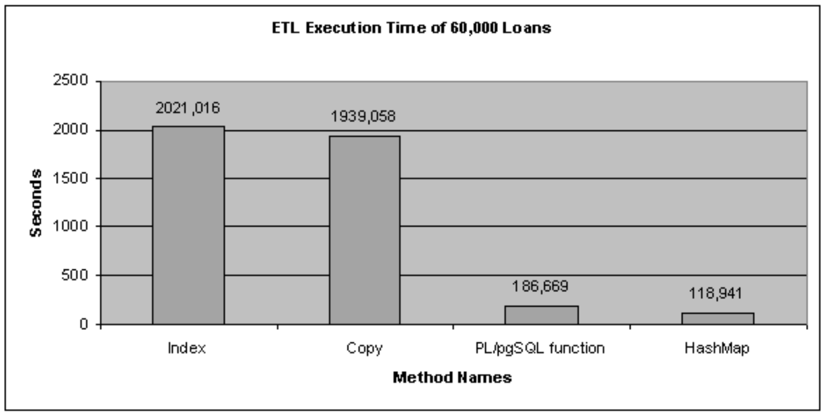


Figure 3.19: ETL Execution Time for all Four Methods with 60,000 Loans.

3.3.4 Evaluation of Memory Usage

The memory usage of the fastest ETL processes is shown in Figure 3.20. The amount of megabytes used for creating the HashMaps increases proportional with the amount of loans to be processed.

The space usage of the serialized HashMaps is shown in Figure 3.21. The amount of space used is connected with the amount of memory used when the ETL processes executes. For example if the serialized HashMaps of the keys and the data totally uses about 13 megabytes on the hard drive then the memory usage will be about 32 megabytes. The reason for the larger amount of memory used than the disk space used is the use of, e.g., keys and hash codes in Java.

The data that is going to be sent to the DBMS with a COPY command is also stored temporarily in memory in a *File Buffer*. However, this does not increase the amount of memory used by the ETL processes because of data being written to disk when the File Buffer is full.

When looking at the memory sizes of the HashMaps, we can see that the biggest HashMaps are editionCRC, editionKeys, bookKeys, and transaction-WareHouseKeys. The reason for the editionCRC being the biggest HashMap

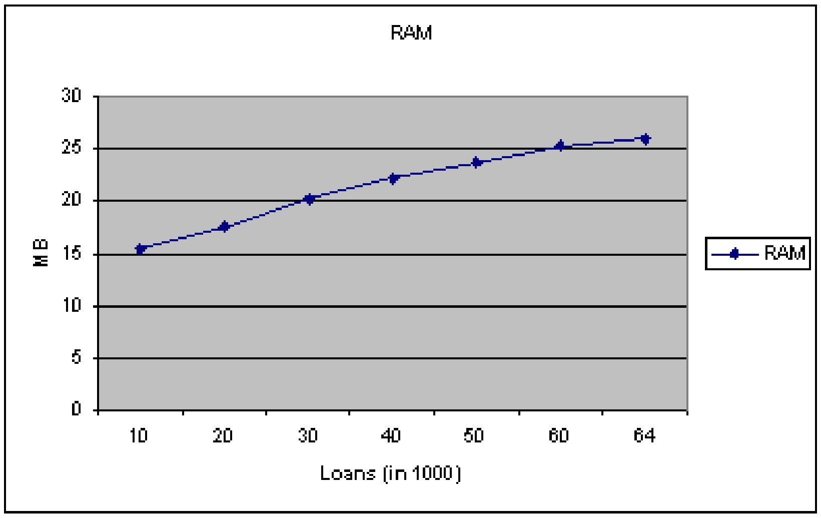


Figure 3.20: ETL Memory Usage.

is because of the stored CRC values, which is 32 bit in length for each edition (CRC32).

One can actually estimate the amount of memory used when processing a given number of loans. This can be done using linear regression. The formula calculated using linear regression for the ETL Memory Usage statistics is shown in Equation 3.1.

The Equation 3.1 takes the number of transactions to be processed (x), and returns the amount of memory in megabytes that is going to be used worst-case by the ETL-processes.

$$f(x) = 1.93 * 10^{-4} * x + 13.912 \tag{3.1}$$

The amount of memory used by the ETL processes will not be critical when processing small chunks of data. This is because the process only need to load the hash keys from the previous processed data and then only process the new chunk of data. Furthermore, the possibility for getting new data that needs to be stored in the hash maps decreases as more data is processed. This is because of, e.g., loans referring to books that are already given an ID in the corresponding HashMap.

However, processing large amounts of data, say more than 2,500,000 transactions, in one round of ETL-processes is not recommended because of high

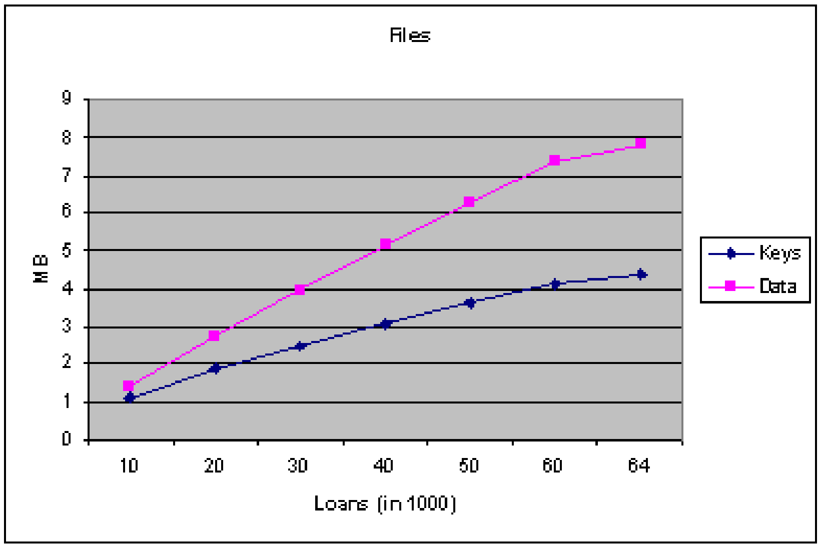


Figure 3.21: ETL File Sizes for Hashmaps.

memory usage. The amount of memory used for processing 2,500,000 transactions is about 496 megabytes using Equation 3.1. However, 2,500,000 transactions corresponds to about 25 years of transactions. Though, it is still recommended to process data in chunks in separate rounds of ETL-processes in order to reduce the memory usage. Another solution could be to use the PL/pgSQL function instead, because it relies on the DBMS to take care of the data processing.

3.4 Future Work

Not every feature listed in the design for the data staging area has been implemented due to time constraints and limited access to data from AUB. The following has not been implemented:

- Extraction of data directly from the base tables in the operational system.
- Pattern matching.
- Further optimizations of the DBMS
- PL/pgSQL function alternative implementation

The data must be extracted from the base tables in the operational system in order to make the data warehouse work properly. This part was not implemented because of only having access to a sample set of loan history. The details of the sample set can be found in Appendix A.

Pattern matching could be done by, e.g., combining author names where only a few characters differ.

Further optimizations of the DBMS can be done by tweaking the settings of the PostgreSQL configuration files. This can decrease the execution time of implementations using PL/pgSQL functions, indices and/or COPY.

An alternative implementation to the HashMap could be done by using the PL/pgSQL function. This would solve the problem with high memory usage when processing many transactions, because the PL/pgSQL function makes use of buffer and query capabilities of the DBMS.

4 Association Rule Mining

Association rule mining is a part of the data mining area and is used for finding new information from existing data, e.g., stored in a database. The information discovered by association rule mining is rules from a large collection of data. The collection of data consists of transactions and each transaction is a set of items. The rules show which set of items that coexists in the different transactions. These co-existing items are also known as *intra-transaction patterns* [30], i.e., patterns within the transactions in the database.

The association rule mining can be used with the AUB data warehouse as a foundation and thereby add further information to the AUB data warehouse. The transactions in the AUB data warehouse are collections of books borrowed together by the borrowers. The association rules mined from the AUB data warehouse then show the patterns of books, i.e, the set of books that are often borrowed together.

In the following sections association rules are introduced, different approaches for mining association rules are explained along with an implementation, and the performance of the association rule mining implementation is shown. This chapter wraps up with two sections regarding related and future work.

4.1 Association Rules

A short introduction to association rule mining is given in this section. Association rules are explained by an example, terms are introduced, and two advanced approaches to mining association rules are explained.

4.1.1 Association Rule Example

Every day a library lends several number of books and each loan is registered in the library database. It would be interesting to explore associations of books that have been borrowed together. The process of finding such sets of books is called *Association Rule Mining*. To be specific, imagine that we have a borrower who is interested in thriller and adventure literature. The borrower goes to the library and finds five books, which the borrower would like to borrow. These are:

- Lord of the Ring (*LR*)
- The Dead Zone (*DZ*)
- Harry Potter (*HP*)
- Hound of the Baskervilles (*HB*)
- The Eyes of the Dragon (*ED*)

The capitalized letters in the parentheses are abbreviations of their corresponding book titles. After the books have been registered for loan, the borrower is issued a receipt on which those five books are listed along with the time of return.

It is only interesting to know about those books which are often borrowed together, thus association rule mining takes outset from two conditions: *minimum support* and *minimum confidence*. These two conditions serves to limit the number of association rules found. A specific association rule is shown below in order to explain by example:

$$HB \Rightarrow DZ$$

The left-hand side is known as the *antecedent* and the right-hand side is the *consequent*. Minimum support states how many times a given set of books should at least be borrowed together. Minimum confidence expresses the strength of the rule and states how often the book on the right-hand side should at least be borrowed whenever the book on the left-hand side is borrowed. If the above support of the association rule is 20 and confidence is 75% it means that HB and DZ is borrowed together 20 times, and whenever HB has been borrowed, then 3 out of 4 times DZ is also borrowed together with it.

Often those association rules with the highest support and confidence are indeed very obvious, thus it is often those association rules further down the top list, which are of particular interest because they might not be that much obvious and though still have a high support and confidence. To understand why, it helps to draw a parallel to the daily shopping at the supermarket. In this domain the items in an association rule are products. For instance, milk and bread form an association rule with a very high support and confidence. This rule is as such not interesting because it is an expected fact. Those products which are not obviously correlated with each other and have high support and confidence form the interesting association rules. This is also the case with books. For instance, we presented association rules with the highest support and confidence found during a meeting with AUB. They were not surprised to see that library related books lie on top of the association rules because those books are part of a set of books which is being borrowed each semester by students from the library school. The interesting association rules came further down the top list.

4.1.2 Association Rules terms

After having introduced association rules by example we will in the following explain the terms, which are often used when talking about association rules.

Item equals to a single book.

Itemset equals to a set of books, which are borrowed together.

Candidate itemset is an itemset, where the number of occurrences of books in the database has not yet been counted, thus it is unknown whether it meets the minimum support.

Large itemset is an itemset, which meets the given minimum support.

Transaction is an event where a set of books are borrowed together on the same day by the same borrower. What constitutes a transaction is a transaction ID along with an itemset.

Association rule Given a set of *items* $I = \{I_1, I_2, ..., I_m\}$ and a database of transactions $D = \{t_1, t_2, ..., t_n\}$ where $t_i = \{I_{i1}, I_{i2}, ..., I_{ik}\}$ and $I_{ij} \in I$, an association rule is an implication of the form $X \Rightarrow Y$ where $X, Y \subset I$ are *itemsets* and $X \cap Y = \emptyset$.

Support of an item or (set of items) is the percentage of transactions in which that item (or items) occurs.

$$support(I_k) = count(I_k, D) \tag{4.1}$$

$count(I_k, D)$ is the number of transactions in which the itemset I_k occurs in the database D.

Confidence for an association rule $X \Rightarrow Y$ is the ratio of the number of transactions that contain $X \cup Y$ to the number of transactions that contain X.

$$confidence(X \Rightarrow Y) = \frac{support(X \cup Y)}{support(X)} \tag{4.2}$$

Hierarchy defines different levels for which association rule mining can be applied.

Constraints are used in order to select subsets of transactions that should be used for association rule mining.

The process of mining association rules consists of two main activities. First, large itemsets meeting the specified minimum support must be found. Second, based on these itemsets association rules meeting the specified confidence are extracted.

4.1.3 Generalized Association Rules

There are very often hierarchies over the items used for association rule mining. Hierarchies which apply to books were introduced in the Data Warehouse chapter. An example of a hierarchy is the Edition hierarchy in Figure 3.11 on page 41. In the figure the example shows that a specific edition is a book, a book belongs to a specific UDK level 3, UDK level 3 belongs to specific UDK level 2, which belongs to a specific UDK level 1.

The hierarchies over the items can also be used when mining association rules. This approach was introduced by Agrawal et. al. in [9]. Using this approach the association rule mining is not restricted to finding associations between leaf-level items in a hierarchy. The association rules can with this approach be rules covering higher levels in the hierarchy. This is valuable since rules at lower levels might not have minimum support and using the higher levels might then lead to the discovery of new interesting rules that have minimum support and thereby stronger rules.

4.1.4 Association Rules Constraints

Constraints can be used when mining for association rules. The association rules can, e.g., be mined from a subset of loans. The subset can be specified by supplying constraints such as a specific status and type of borrowers. The status and type can, e.g, be Ph.D and Social Studies. Then all the loans registered with this status and type are used for association rule mining. Other constraints can be loans from a specific UDK or from a specific date or time range.

Using constraints for association rule mining makes the association rules more focused. The librarians can, e.g., with the constraints find the loan trends of different groups of borrowers instead of the more general association rules that cover all the borrowers. It is also possible to compare association rules over time and by UDK. With constraints association rules can be discovered that otherwise might have been hard to discover. For example, interesting association rules can be difficult to discover when finding association rules for all the books. However, if time constraints are used, an association rule can be found within a given period. This could, e.g., be when Aalborg University starts a new semester that different association rules would occur instead of finding association rules for a whole year.

4.2 Finding Large Itemsets

When searching for association rules we need to find all sets of itemsets that occur frequently in the data. Such itemsets are also known as *large itemsets.* The sets of large itemsets form the foundation for mining association rules.

Large itemsets can be found by using different algorithms. The general idea of the algorithms is to scan the database for each set of candidate itemsets with a certain cardinality found in order to count their support.

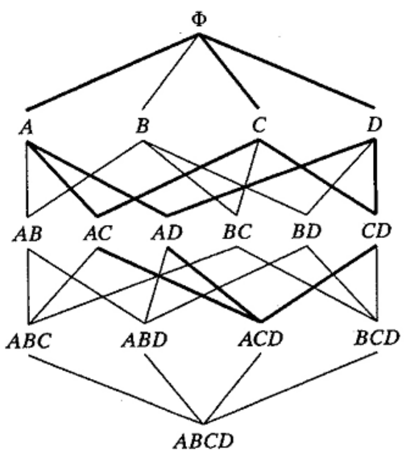


Figure 4.1: Any Large Itemsets must have Subsets that are also Large.

4.2.1 Apriori

The widely known algorithm for finding large itemsets is called Apriori, which was presented by Agrawal and Srikant [8]. Apriori does not count all itemsets that can be constructed from every single transaction in the database, instead it tries to reduce the number of itemsets to be counted. The algorithm is built on the fact that any large itemset must have subsets that are also large as illustrated in Figure 4.1. If the itemset ACD is large then all its subsets are also large. Reversely, it will suffice only to construct new itemsets from those already known to be large. These constructed itemsets are called *candidate itemsets*. Those candidate itemsets for which their subsets are not large can be discarded without counting their support in the database. This process is known as *pruning*. The rest of the candidate itemsets are potentially large, thus they need to be checked for being large by scanning the database once to verify whether the minimum support is met.

4.2.2 LIQ

In the process of generating association rules the most time-consuming part is finding the large itemsets. It is especially during the processes of counting the candidate itemsets for each transaction in order to find L_1 and L_2 that the time complexity for counting C_1 and, in particular, C_2 is very high. We have therefore investigated in retrieving L_1 and L_2 by querying the database, thus leaving the heavy job to the DBMS. We call our implementation for

Large Itemset Query (LIQ).

The bottleneck of Apriori is in particular the candidate generation followed by the scanning of candidate itemsets in C_2. For each transaction from the database every single candidate itemset must be matched in order to update their support count, which is a quite time consuming task. Here, the efficiency is limited by how many rows are fetched from the database at a time. LIQ circumvents this communication overhead of row fetching from the database to the Java application by by keeping the candidate generation and pruning task in the database such that only large itemsets are returned.

4.2.3 LIQ Hybrid

LIQ Hybrid combines the strengths of the Apriori algorithm and the LIQ algorithm. Our previous research results proved that the execution time for our sample data in pass 2 when running Apriori is very high. This is due to the large number of candidate itemsets in C_2 to be scanned as mentioned above. We found out that the execution time for Apriori is very high in the initial passes while the execution time of LIQ is very low in the beginning and increases for each pass. However, the execution time for LIQ increases for each pass. This is because each new pass introduces an extra join of data.

The LIQ Hybrid algorithm uses these facts to decrease the execution time. Therefore, LIQ Hybrid switches from LIQ to Apriori right after pass 2, because if L_2 is small then the pruning process of Apriori might reduce the number of candidate itemsets significantly yielding an execution time much less than the time it takes to join the data three times.

4.2.4 LIQ2

The biggest problem of LIQ is the multiple joins of data, which consists of all transactions. For each pass yet another join is introduced in the execution of LIQ, thus increasing the processing time significantly.

We have developed an algorithm LIQ2, which reduces the table size. This reduces the processing time of the multiple joins. Instead of joining a table containing all transactions with itself another approach is to gradually reduce the number of transactions in the table. The size of the tables used for the joins are reduced in size by creating new temporary tables containing all the transactions of a specific minimum size. New tables are created since it is faster than deleting from old tables. The approach finds L_1 from the original

TID	Item (Ordered)	Frequent Items
1	LR, HP, DZ	LR, HP
2	LR, HP	LR, HP
3	HP	HP
4	LR, HP	LR, HP

Figure 4.2: Example for AUB.

transaction database while, e.g., L_2 is found from joining a copy of the original transaction database containing all the transactions with minimum length 2. The temporary tables are reduced in size for each pass and the reduction is increased for each pass since a large amount of transactions have minimum length 2 and fewer transactions have minimum length 3 etc.

4.2.5 FPGrowth

We also wanted to test the current fastest algorithm for finding large itemsets against our algorithms. As mentioned in our previous report the FPGrowth algorithm was the fastest algorithm for finding large itemsets when analyzing large amounts of data. We therefore tested an implementation of the FPGrowth algorithm made by [22].

The FPGrowth algorithm from Han, Pei, and Yin [22] uses another technique than Apriori in order to retrieve the frequent itemsets from a database. FP-Growth avoids generating a huge set of candidates which is a major bottleneck as stated by [22]. Instead it makes use of an effective data structure to hold items having minimum support. FPGrowth makes use of a *Frequent Pattern Tree* (FP-tree), which is built from the data in the database.

There are three processes in the algorithm. These are shortly described here:

1. FPGrowth constructs a FP-tree by storing information about frequent patterns in the data. An example of data from AUB is shown in Figure 4.2.

   The FP-tree in Figure 4.3 shows the frequent patterns in the transaction data. It shows that LR occurs three times indicated by the nodes containing $LR : 3$. Furthermore, there are two sub nodes: one sub node where HP occurs three times with LR and another sub node where HP occurs alone.

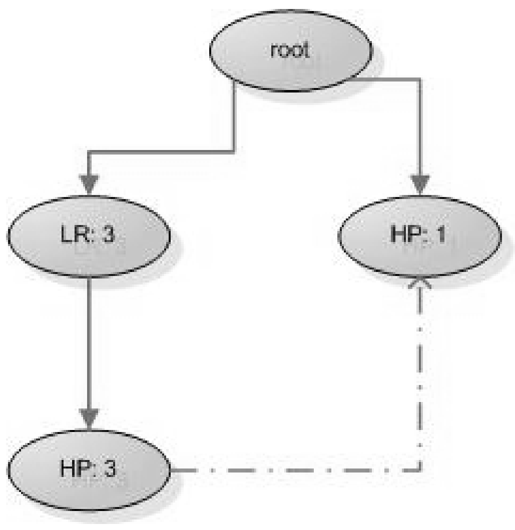


Figure 4.3: FP-Tree Example.

2. FPGrowth uses an FP-tree-based pattern mining method called ”FP-Growth”. The method starts from a frequent length-1 pattern, constructs its own conditional FP-tree and performs mining recursively with such a tree.

3. FPGrowth does the searching in a partitioning-based and divide-and-conquer method instead of the Apriori bottom-up generation of frequent itemsets.

The benefit of producing a frequent pattern tree is that we make a compressed copy of the patterns of the data in the database, and thereby avoids going through the whole database in order to find the frequent patterns. Instead, we only need to scan the database once in order to generate the frequent pattern tree, and afterwards we can use the FP-tree to find the frequent patterns we need given a specific support value.

In the following we point out the main weaknesses and strengths as explained in [22]. Interested readers are encouraged to read the article.

Weaknesses The FP-tree in the algorithm needs two scans of a transaction database and this may represent a nontrivial overhead. The first scan collects the set of frequent items, and the second constructs the FP-tree, which comprises the largest overhead. Furthermore, FP-tree needs to be in memory in order to scan it fast, though you could construct a disk-resident FP-tree. However, another disadvantage of using FPGrowth is that you cannot always fit the FP-tree into memory.

Strengths FPGrowth scales well because it only needs to generate a frequent pattern tree. The Apriori algorithm has to scan the whole database when finding frequent itemsets and it cannot avoid generating and testing candidates. The reason that FPGrowth scales very well is that when the support threshold goes down the number as well as the length of frequent itemsets increase dramatically. Apriori must handle large amounts of candidates and the processing of these candidates becomes very expensive. This also holds when the data amount increases, because then Apriori needs to generate more candidates, while FPGrowth only needs to update its FP-tree with frequent patterns counts. Other advantages of using FPGrowth are that the FP-tree is much smaller than the original database and thereby saves memory and database scanning. It also uses a pattern growth method where the major operations are prefix count adjustment, counting, and pattern fragment concatenation. Lastly, it makes uses of a divide-and-conquer technique that reduces the size of subsequent conditional pattern bases and the conditional FP-trees.

There are currently two approaches for implementing the FPGrowth algorithm. The first is to make use of a *loose-coupling* approach where the FP-Growth algorithm is run in, e.g., Java code that is connected with a database. The second is to implement a *SQL-based* FPGrowth algorithm, which runs in the DBMS and makes use of the query processing facilities. The implementation we use relies on the loose-coupling approach.

4.3 Implementation

The association rule mining uses the transactions defined in the data staging area, see Section 3.1.2. This leads to a minor complication since the same book can exist more than once in a single transaction. This is due to the fact that a borrower can borrow more than one edition of the same book in a single transaction. In order to avoid having duplicates the association rule mining creates a temporary table defined on the *loan* fact table and the *edition* dimension table. This way we focus on a higher level of abstraction.

Instead of finding association rules for the individual editions we go up one level to find association rules for books to avoid many association rules with low support. For instance we could have a number of association rules where several editions of one book associates with several editions of another book. Focusing on the book level will give us one association rule, which tells us that the first set of books associate with the second set of books. The support will also be higher because we have aggregated the editions.

Here is shown the general query for creating the temporary table containing transactions and books:

```
CREATE TABLE
    loanUnique
AS
    (
        SELECT
            edition.bookID,
            loan.transactionID
        FROM
            loan,
            edition
        WHERE
            loan.editionID = edition.ID
        GROUP BY
            loan.transactionID,
            edition.bookID
    )
```

Generalization- and Constraint-based mining

As stated above the temporary table is named *loanUnique* because each transaction contains distinct books due to the GROUP BY-clause. Here, the granularity is books but it is also possible to have an higher level of abstraction. Instead of grouping by *bookID*, we could group by *udkID*. Based on the found large itemsets, we will be able to mine association rules across UDKs instead of books. Yet another possibility is to mine association rules conditioned by a given status and type combination of the borrowers in order to see which books certain borrowers, e.g., psychologist students at the 5th semester, tend to borrow. Again, we can look at an higher level of abstraction and consider which classes of books, i.e. UDK, certain borrowers tend to borrow. These two additional variations calls for a total of three different kinds of temporary tables, which can be found in Appendix C. All variations have been implemented. In our description we focus on finding association rules for books. It is only a matter of replacing the above *loanUnique* with one of the other variations to find other kinds of association rules.

Section 4.3.1 explains how the large itemsets are found. Section 4.3.2 explains how LIQ may be improved by making a LIQ2 implementation that might improve the performance.

4.3.1 Large Itemset Query

We will in the following explain how our implementation of LIQ works. The next two sections explain how to retrieve L_1 and L_2 from the database followed by a section generalizing the queries such that LIQ can be used to retrieve any size of large itemsets.

Retrieving L_1 from database

LIQ query L_1 from the database. The query is as follows, where the *minSupport* is the specified minimum support:

```
SELECT
    l1.bookID AS item1,
    COUNT(l1.bookID) AS weight
FROM
    loanUnique AS l1
GROUP BY
    l1.bookID
HAVING
    COUNT(l1.bookID) >= minSupport
```

The query only selects those items, which occur at least the number of times specified by the minimum support.

Retrieving L_2 from database

The lower we specify the minimum support the more large itemsets can be found. It is especially computationally expensive when finding L_2 while the minimum support is set very low. This is because $|C_2|$ is huge. If e.g. $|L_1|$ is 500 then $|C_2|$ will be 124,750 ((500 * 499) / 2). In order to find L_2 using the classic Apriori algorithm, we need to store all the candidates in C_2 in the hash tree, scan the database and update the weights in the hash tree given each transaction. This yields a significant performance slowdown due to the communication overhead between the DBMS and our Java application.

Instead, we can query L_2 from the database. The query used is the following, where *minSupport* is the minimum support:

```
1  SELECT
2      l1.bookID AS item1,
3      l2.bookID AS item2,
4      COUNT(l2.bookID) AS weight
5  FROM
6      loanUnique AS l1,
7      loanUnique AS l2
8  WHERE
9      l1.transactionID = l2.transactionID AND
10     l1.bookID > l2.bookID
11 GROUP BY
12     l1.bookID,
13     l2.bookID
14 HAVING
15     COUNT(l2.bookID) >= minSupport
```

Line 2 - 4 select pairs of items and their number of co-occurrences. This is done by joining the view *loanUnique* with itself as indicated in line 6 and 7. The join is conditioned in line 9 - 10 such that two items should have the same *transactionID* and we only want the occurrence of two items to appear once in our selection. The rows are grouped by the *bookID* of the two items in line 12 - 13. In line 15 we specify that we only want those pairs of items that meet the minimum support.

Generalizing the retrieval of L_X

When matching the queries of L_1 and L_2 we can see a pattern emerge.

- In the *SELECT* clause we only select the number of items of interest and their number of occurrences.
- The *FROM* clause joins the view *loanUnique* with itself as many times as the number of items in the *SELECT* clause.
- In order to ensure that the selected items are different from each other we have the *WHERE* clause, which is divided into two parts.

    1. Ensure that the items occurs in the same transaction, thus the equality between *transactionID*s.
    2. Ensure that the set of items are only selected together once, thus the greater than sign between *bookID*s.

    The *WHERE* clause is only needed in order to ensure correct joins of the view *loanUnique* with itself, thus the query for L_1 contains no *WHERE* clause.

- The found rows are grouped by the *bookID* of the item(s).
- The *HAVING* clause ensures that we only get those itemsets that meet a given minimum support.

This pattern enables us to generalize the retrieval of L_1 and L_2 such that it is possible to retrieve large itemsets of any size, L_X, from the database without the previous overhead. The pseudo query is as follows, where X is the size of the large itemsets to be found:

```
SELECT
    l1.bookID AS item1,
    l2.bookID AS item2,
    ...,
    lX bookID AS itemX,
    COUNT(tX.bookID) AS weight
FROM
    loanUnique AS l1,
    loanUnique AS l2,
    ...,
    loanUnique lX
WHERE
    (l1.transactionID = l2.transactionID = ... = lX.transactionID) AND
    (l1.bookID > l2.bookID > ... > lX.bookID)
GROUP BY
    l1.bookID,
    l2.bookID,
    ...,
    lX.bookID
HAVING
    COUNT(lX.bookID) >= minSupport
```

The strengths and weaknesses for LIQ are the following:

Weaknesses Multiple joins are being made for each pass, and complete set of transactions is used for each join. Therefore, it does not scale well.

Strengths The DBMS efficiently generate and prune the candidate itemsets and returns large itemsets. The DBMS is used and we therefore make use of the powerful query processing facilities.

4.3.2 LIQ2

The biggest problem of LIQ is the multiple joins of *loanUnique*, which consists of all transactions. For each pass yet another join is introduced in the execution of LIQ, thus increasing the processing time significantly.

By reducing the size of *loanUnique* we might reduce the processing time of the multiple joins. Instead of joining a table containing all transactions with itself we gradually reduce the number of transactions. To be specific, in the first pass L_1 is found by querying *loanUnique*. Then the following L_X, $X > 1$, are found be querying temporary tables containing transactions with minimum length X. The process of table reduction continues until no more large itemsets are found.

The below pseudo query demonstrates how the temporary tables are created for each pass:

```
CREATE TABLE tempTableX AS
   SELECT
      tempTable(X-1).editionid,
      tempTable(X-1).transactionid
FROM
   (SELECT
      transactionid
   FROM
      tempTable(X-1)
   GROUP BY
      transactionid
   HAVING
      count(transactionid) >= X
   ) AS q,
   tempTable(X-1)
WHERE
   q.transactionid = tempTable(X-1).transactionid;
```

We insert the transactions from the prior transaction database having minimum length X into the new temporary table.

Generally, the size of the table containing the transactions is only reduced slightly after the first pass because many transactions contain more than 2 items. Already after the second pass the size is reduced because much fewer transaction contains 3 items. This kind of table reduction continues as we proceed through the pass numbers, i.e., searching for large itemsets of higher and higher cardinality.

The strengths and weaknesses for LIQ2 are the following:

Weaknesses Multiple joins are still being made for each pass, therefore it scales not so good. There is a overhead when creating the temporary tables, because of I/O time.

Strengths The DBMS efficiently generate and prune the candidate itemsets and returns large itemsets. The DBMS is used and we therefore make use of the powerful query processing facilities. We also reduce the number of transactions for each pass.

4.4 Performance

To compare the relative performance of the algorithms we carried out several test cases. The configuration of the test machine was the following:

- AMD Athlon, 32-bit processor, running at 1333 Mhz.
- 512 MB RAM.
- MS Windows 2000 Professional SP4 with 4 GB of swapfile.
- Java Virtual Machine version 1.4.
- The DBMS was running on the same machine and used the PostgreSQL DBMS using Cygwin.

It should be noted that PostgreSQL does not support Windows but UNIX-based systems, thus we used Cygwin which acts as a Linux emulation layer to access the PostgreSQL DBMS. This adds more processing time to the performance tests.

We compare the performance of Apriori, LIQ, and LIQ2 based on the test data set from 4.4 and introduce a new hybrid implementation in the following sections called LIQ Hybrid. LIQ Hybrid is a combination of Apriori and the best of LIQ and LIQ2.

The test data set from 4.5 is used to test the scalability of the found LIQ Hybrid against FPGrowth, which is a more efficient algorithm for finding large itemsets. We do this to see whether our implementation scales as well as FPGrowth, which is known to scale well with an increasing number of transactions [22].

The abbreviations of the table headers is explained in the following:

- |D|: Number of transactions.
- $|T_A|$: Average transactions size.
- $|L_A|$: Average large itemsets size.

|D|	$\|T_A\|$	$\|L_A\|$	**|I|**
25,000	3	3	12,500

Figure 4.4: Performance Test Data Set.

|D|	$\|T_A\|$	$\|L_A\|$	**|I|**
100,000	7	3	50,000
150,000	7	3	50,000
200,000	7	3	50,000
250,000	7	3	50,000
300,000	7	3	50,000
350,000	7	3	50,000
400,000	7	3	50,000

Figure 4.5: Scalability Test Data Set.

- |I|: Number of items.

We are using a synthetic data generator which is built by ARMiner [14]. It makes use of the algorithm for generating synthetic databases described by Agrawal and Srikant [8]. The algorithm starts with building a potential set of large itemsets, which is later used for generating the actual transactions using a correlation parameter and corruption parameter. The correlation is used for the large itemsets and the corruption is used for corrupting the large itemsets added to a transaction.

In our tests we choose to set the number of potential large itemsets to 100,000. Furthermore, we kept the default values for the correlation and corruption parameters.

4.4.1 Comparing Apriori, LIQ, and LIQ2

The execution time of Apriori and LIQ will be discussed in the following based on a minimum support set from 26 down to 20. The interval of minimum support is chosen because it gives a reasonable amount of large itemsets.

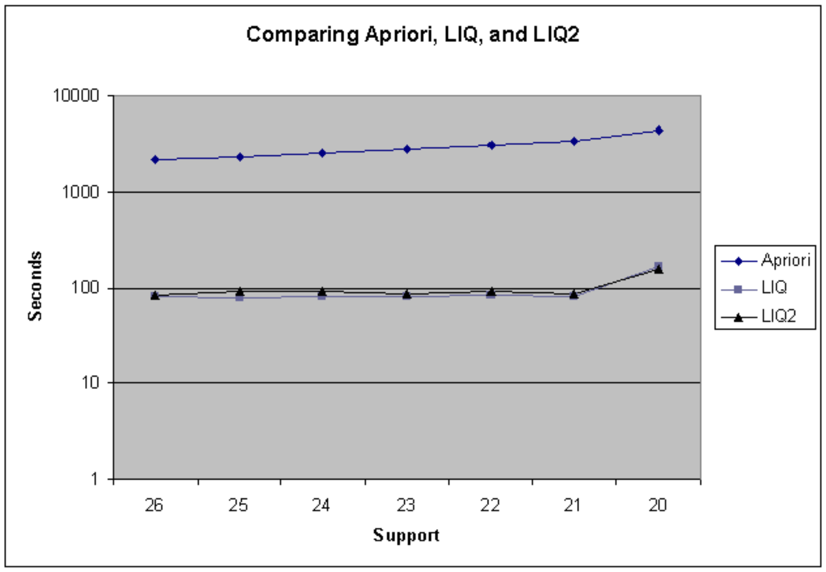


Figure 4.6: Execution Time of Apriori, LIQ, and LIQ2.

Apriori

The execution time of Apriori is proportional to the decrease in minimum support. The lower support the more candidate itemsets are generated which is reflected in the increase of execution time of Apriori in Figure 4.6. Note that a logarithmic scale has been used. With a lower minimum support the size of L_{k-1} gets larger, which results in an even larger C_k. Actually, the number of candidate itemsets before pruning is:

$$|C_k| = \frac{|L_{k-1}| * (|L_{k-1}| - 1)}{2} \tag{4.3}$$

In Equation 4.3, k is the pass number. The pruning yields a smaller C_k except when $k = 2$ because each subset of the candidate itemsets of size 2 is large.

LIQ

The bottleneck of Apriori is the candidate itemsets generation and the updating of the hash tree storing the candidate itemsets. LIQ does not spend time in candidate generation as it queries the DBMS for large itemsets. Its execution time is relatively stable primarily depending on the time it takes

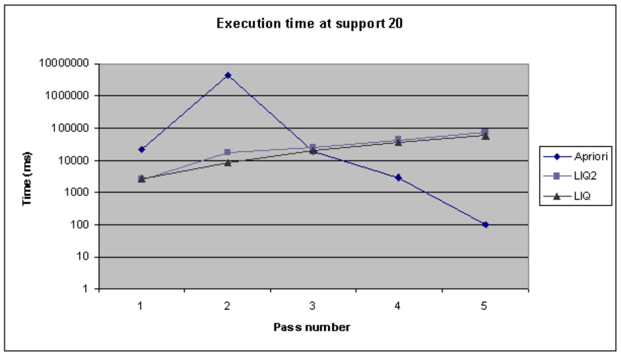


Figure 4.7: Execution Time of Apriori, LIQ and LIQ2 at Support 20.

to join k *loanUnique* views with itself. The stability in execution time is clearly expressed in Figure 4.6 using a logarithmic scale:

- Support 21-26 yield at most L_3 with a relatively constant execution time of approximately 80 seconds.
- Support 20 yields at most L_4. The execution time increases to approximately 170 seconds.

For each pass k, LIQ queries the DBMS to find L_k. It stops when the returned L_{k+1} is empty or the size of L_k is 1, thus LIQ does one more $(k + 1)$ pass than the cardinality of the largest large itemset.

LIQ2

We tried to increase the performance of LIQ by making use of temporary tables for finding large itemsets. However, LIQ2 as shown in Figure 4.6 does not perform any better than LIQ. Actually, LIQ2 performs worse than LIQ. The reason for the increase in execution time is the I/O time needed for creating the temporary tables in the DBMS.

4.4.2 Finding the Switching Point

In Figure 4.7 the execution time of Apriori, LIQ and LIQ2 is listed with logarithmic scale for each pass given a minimum support of 20. It would be interesting to see when it will pay off to switch from LIQ or LIQ2 to Apriori. Such a hybrid might keep the overall execution time low.

The execution time for pass 2 when running Apriori is very high. This is because there are 2,416 itemsets in L_1 yielding 2,917,320 candidate itemsets in C_2 according to Equation 4.3. All the candidate itemsets in C_2 have to be stored in the hash tree since pruning is not applicable. The hash tree is then updated for each transaction in the database during the scanning. The time it takes to update the hash tree depends on how many of the candidate itemsets also are in the transaction. The execution time after pass 2 decreases for Apriori since the number of large itemsets decreases. The number of large itemsets in L_2 is 50, which is significantly less than the number of itemsets in L_1.

The execution times for LIQ and LIQ2 increase for each pass. This is because each new pass introduces an extra join of the *loanUnique* as described in Section 4.3.1. Apriori is very time consuming in the initial passes while the execution time of LIQ is very low in the beginning and increases for each pass. LIQ2 takes a little more execution time because of the extra overhead for creating the tables in each pass. The overhead happens because of the I/O time the DBMS uses for copying the data, and it is not being sufficiently compensated by the joins of smaller tables, thus we have decided to make an hybrid out of LIQ and Apriori.

As illustrated in Figure 4.7 the execution time of Apriori and LIQ are fairly equal in pass 3, which suggests to either use Apriori or LIQ in this pass. We have chosen to make the switch from LIQ to Apriori right after pass 2, because if L_2 is small then the pruning process of Apriori might reduce the number of candidate itemsets significantly yielding an execution time much less than the time it takes to join *loanUnique* three times.

4.4.3 LIQ Hybrid

Below we have the algorithm for LIQ Hybrid:

```
$LIQHybrid(I, D, S)$
    $L = \emptyset$;

    Run LIQ to find $L_1$and $L_2$;
    $L = L \cup L_1$;
    $L = L \cup L_2$;

    **while** $|L_k| > 1$, where $k \geq 2$
        Run Apriori to find $L_{k+1}$;
        $L = L \cup L_{k+1}$;
        $k = k + 1$;

    **return** $L$;
```

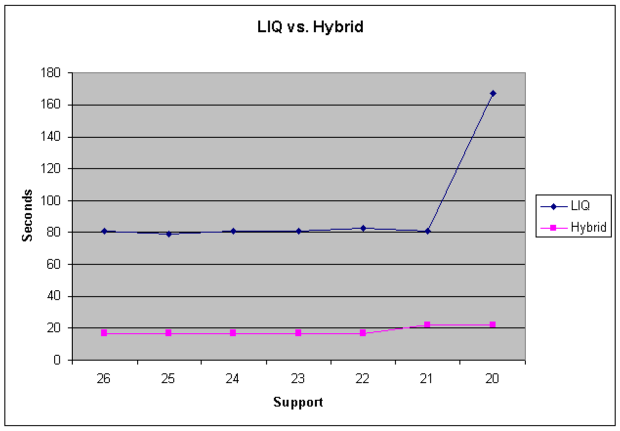


Figure 4.8: Execution Time of LIQ and LIQ Hybrid.

LIQ Hybrid achieves the fastest execution time, as it combines the strengths of the two algorithms. This is clearly depicted in Figure 4.8, where the performance of LIQ and LIQ Hybrid is compared. As the difference in execution time of LIQ and LIQ Hybrid is not as significant as it was the case with LIQ and Apriori, we have switched from a logarithmic scale to normal scale.

LIQ Hybrid outperforms LIQ for all supports from 26 down to 20. From support 26 down to 21 the largest large itemset has cardinality three. At support 20 the execution time of LIQ increases because now the cardinality of the largest large itemset has increased by one, which means an extra join of *loanUnique* view has been introduced. The execution time of LIQ Hybrid is kept constant from support 26 down to 22. Hereafter, the execution time increases with merely 5 seconds. Presumably, this is due to a small increase in the number of itemsets to be scanned.

4.4.4 Scalability of LIQ Hybrid vs. FPGrowth

As LIQ Hybrid outperforms LIQ it would be interesting to see how well LIQ Hybrid scales. For this purpose we have used the test data from Figure 4.5. Furthermore, we also utilized the FPGrowth algorithm using the source code from [14]. The source code was changed in order to be able to connect to our dataset in a database instead of using files. We wanted to test the scalability of the FPGrowth algorithm and compare it to LIQHybrid. However, the implementation of the FPGrowth algorithm from [14] is only using the loose-coupling approach.

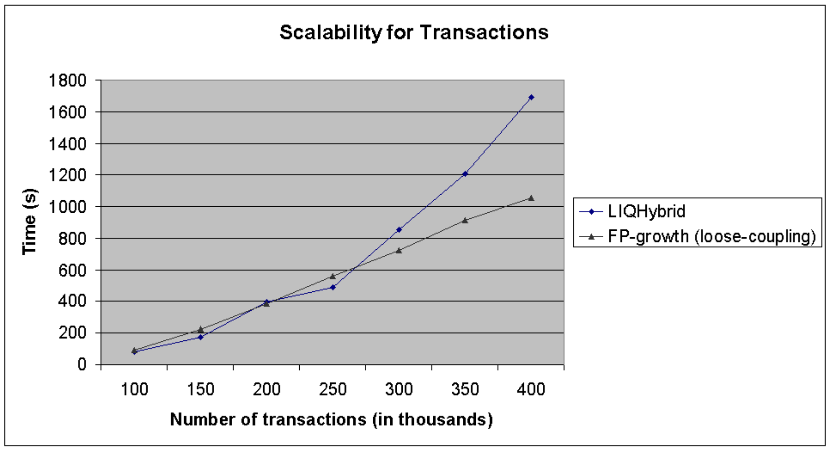


Figure 4.9: Scalability of LIQ Hybrid and FPGrowth.

The result of the scalability test is depicted on Figure 4.9. It seems that the execution time of LIQ Hybrid is faster than FPGrowth until about 225,000 transactions. However, as the number of transactions increases beyond 225,000 transactions the performance of LIQ Hybrid decreases and FP-Growth becomes faster than LIQ Hybrid.

In figure 4.10 the support scalability for LIQ Hybrid and FPGrowth is shown. The set of test data for these tests is the one in Figure 4.4. The support was decreased from 10 to 5 in order to show the strengths and weaknesses of both algorithm. LIQ Hybrid is faster than FPGrowth when the support value is high. However, as the support decreases LIQ Hybrid has to make joins of larger tables and possibly more joins if we get larger large itemsets. This will result in an increase in the amount of itemsets being returned to the Apriori algorithm leading to an increase in execution time. However, when using FPGrowth the execution time scales more linearly because FPGrowth builds its FP-tree on the number of transactions every time it is run. The extra overhead in the FPGrowth algorithm is because of the extraction of large itemsets from the FP-tree.

Therefore, the LIQ Hybrid algorithm is faster when computing large itemsets for small amounts of data, and FPGrowth is better for computing large itemsets for large amounts of data. With the current amount of loans per year from AUB LIQ Hybrid could be used for 6-7 years of loan data before FPGrowth is faster.

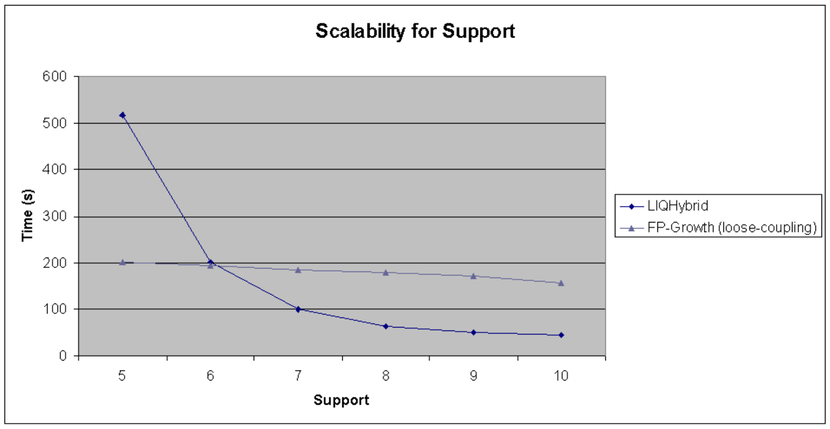


Figure 4.10: Support Scalability of LIQ Hybrid and FPGrowth.

4.5 Extension of FPGrowth

In this section, we will describe an extension of the FPGrowth algorithm for finding large itemsets. We will include a description of a SQL based FPGrowth approach that uses the same idea as LIQ such that performance can be increased when running the algorithm using the DBMS.

4.5.1 SQL Based Frequent Pattern Mining with FP-Growth

Another approach when implementing the FPGrowth algorithm is to make use of a DBMS as in [32], which describes how to get an efficient performance by the utilization of SQL based frequent pattern mining using the FPGrowth approach.

One weakness of the FPGrowth algorithm is the memory usage. This is because the algorithm has to keep an overview over the database in memory in order to find the Frequent Patterns. Another weakness is that every time the algorithm is run it has to scan all the transactions in order to find the itemsets that are usable given the minimum support.

These two weaknesses can be handled more efficiently when using a DBMS. The DBMS can select the transactions in an efficient way by making use of indices and efficient execution algorithms. Furthermore, it is possible to store the FP-tree, which is the overview over the data, in the DBMS. Two approaches are possible. The first is to insert all frequent itemsets into another table one by one. The second is to make use of a temporary table, which includes extra information about the itemsets in the transaction table. This makes it possible to avoid testing each frequent itemset and instead make use of the query processing capabilities of the DBMS.

The FPGrowth algorithm consists of two processes. First, a FP table is constructed. Second, the Frequent Patterns are extracted from the FP table. In the SQL based FPGrowth approach the implementation is done using three algorithms in the DBMS. The first algorithm creates a new table consisting of frequent transactions that are sorted in descending order by frequency. The second algorithm constructs the FP table using the FPGrowth algorithm. The third algorithm mines the FP table for frequent patterns also using the idea in the FPGrowth algorithm.

The performance of the SQL-based FPGrowth algorithm shows that it has better performance than the loose-coupling approach of the FPGrowth and the Apriori algorithm. Furthermore, the SQL-based FPGrowth algorithm can scale much better than the two other algorithms.

Weaknesses This algorithm still has the same weaknesses as the original FPGrowth algorithm. However, the algorithm can run more efficiently because of being executed in a DBMS and because of making use of the query processing of the DBMS.

Strengths The memory limitations are not a problem anymore because of using disk space for storing the FP-tree instead. Furthermore, when running the FPGrowth in a DBMS it is possible to make use of the powerful query processing capabilities in the DBMS. The DBMS also makes it possible to use memory buffers in order to increase the performance of the queries. The SQL-based FPGrowth algorithm has better performance and scalability than the loose-coupling approach of the original FPGrowth.

4.6 Future Work

In this section we will go through the different improvements that can be made to the algorithms for finding association rules.

4.6.1 New Hybrid between LIQ and FPGrowth

One way to increase the performance of LIQ and FPGrowth could be to develop a hybrid between the two algorithms. The hybrid could then make use of the strengths of LIQ when computing small amounts of data. When the amount of data increases a switch can be made to make use of the SQL based FPGrowth algorithm. Furthermore, one could develop a dynamic switching technique that makes use of different parameters for choosing the most optimal switching point. As an example the parameters could be the support value and amount of data being processed in order to choose whether LIQ or FPGrowth should be used.

The new hybrid would combine the strengths of the two algorithms. This hybrid algorithm could be used when the amount of data from AUB increases even more. As seen in the experiments, the FPGrowth algorithm is slower than LIQ Hybrid when the support value increases and this could be another reason for using the hybrid algorithm.

4.6.2 Privacy Preserving Mining of Association Rules

As we have written in Chapter 2, the Danish law of privacy must be obeyed. This means that we must protect a user's identity in the system in order to uphold anonymity. It is therefore important that we do privacy preserving mining of association rules. This makes it possible to still find association rules and at the same time makes it possible to protect the users' identities.

We have therefore looked at an approach proposed by [6] that proposes a technique where the data is randomized to preserve privacy of individual transactions. However, we have chosen not to implement the algorithm because we wanted to prioritize other things such as optimization of our existing algorithms higher.

[6] have analyzed the nature of privacy breaches and proposed a class of randomization operators that are more effective than uniform randomization in limiting the breaches. They derive a formulae for an unbiased support estimator and its variance, which allows them to recover itemset supports from randomized datasets. The technique proposed can be incorporated into our association rule mining algorithm. However, a deeper understanding of the technique is needed.

We will now give a short overview over the technique that [6] proposes:

The task is to find all frequent itemsets, while preserving the privacy of

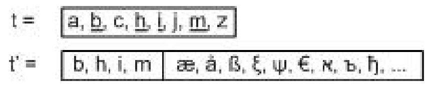


Figure 4.11: Privacy Preserving Association Rules Mining: Cut and Paste Randomization.

individual transaction. We solve the problem by:

- Inserting many false items into each transaction.
- Hiding true itemsets among false ones.

The technique can also be called "Cut and Paste Randomization". The general algorithm constructs a new transaction t' from a given transaction t of size m.

The algorithm does the following:

- Choose a number j between 0 and K_m (cutoff);
- Insert j items of t into t';
- Each randomized item is included into t' with probability p_m.

The choice of K_m and p_m is based on the desired level of privacy.

An example could be as the one shown on Figure 4.11.

Then, in order to recover the original support of an itemset, we need randomized supports of its subsets. The original support can then be estimate using a Partial Support formula and a Transition matrix as proposed in [6].

The last thing to do is to choose how many items that need to be added to be protect privacy enough. This can be calculated using a formula that relies on a privacy breach analysis.

It is then possible to preserve privacy and still find frequent itemsets. However, the technique proposed by [6] will give false positives and false drops. False positives are itemsets that are not true, and false drops are true itemsets that are removed by the technique. Though, this relies on the privacy breach level. A balance between the privacy breach level and the number of false drops and false positives must be chosen.

4.6.3 Sequential Patterns

Beside mining association rules it is also informative to mine *sequential patterns*. Association rules are intra-transaction patterns while the sequential patterns are *inter-transaction patterns* [30]. This means that the sequential patterns show the relations between the transactions instead of just the patterns within the transactions. A sequential pattern could, e.g., be "5% of the borrowers borrowed "Lord of the Ring", then "The Dead Zone", and then "Harry Potter". This information can help the librarian understand how the borrowers tend to borrow books not just within transactions but also over a longer periods covering several transactions.

In [30] an algorithm is described for finding sequential patterns. This algorithm can also be applied for the AUB data by ordering the transactions by time. Before applying an algorithm for sequential pattern mining more loan history must be added then is available in the sample data set from AUB. This is due to the fact that the sample data set only contains 10 months of loan history and therefore likely only contains few sequential patterns with high support.

5 Recommendation

In Chapter 4 we explored association rule mining, which concerned the books that are borrowed together in the same transactions based on a given minimum support and minimum confidence. This chapter further investigates the relationships between books based on ratings given by borrowers over time and the contents of the books. Before getting into too much detail, an example will be introduced, which the rest of this chapter refers to.

Imagine that we have the same borrower from Chapter 4, who would like to read some thrillers and decides to search for such books. The borrower finds the book *The Dead Zone*, which suits the borrower's taste. At this point it would be interesting to be able to offer him a list of books similar to *The Dead Zone*, which he has not borrowed before. This list could be ordered in such a way that those on top of the list are those which most likely suits his taste.

In order to provide such a service, some data analysis techniques have to be applied. These techniques are within the field of *Collaborative Filtering* (CF) [10]. In general, CF recommendation systems apply data analysis techniques in order to recommend relevant items for a particular user for instance in the shape of a top list. For AUB, items are books and users are borrowers. The basic idea is to provide book recommendations based on the opinions of other borrowers.

Seen from a business point of view, e.g. Amazon, the key motivation of using CF algorithms is to make more money. This is achieved by pushing relevant books to the customers, thus making it easier and more comfortable to find interesting books. With such a service the business organization can achieve customer loyalty and preferences, which increases the possibility that the customer will return to buy more books and increase the average number of books per order.

AUB is a free public research library and as such has no intention of making money out of their borrowers. However, the more loans they can accomplish the easier it will be to request for more funds. Nevertheless, their main objective is to provide better service by making it easier for borrowers to borrow more books and those books which are borrowed have a high relevancy. To support their goal we have implemented a recommendation system similar to those discussed in [10], which argues for the item-based approach rather than the user-based approach. We will not get into detail of either approaches. Interested readers are encouraged to read the article. However, the item-based approach needs large amount of data to work satisfactory. The sample data which we have been provided by AUB is not enough data for the item-based approach to work at best, thus we have implemented a content-based approach in order to compensate for this case.

The next section formally describes the techniques used, followed by a section regarding our implementations, performance and evaluation of these, a section about related work, and finally, a section about future works, where improvements are suggested.

5.1 Recommendation

In the following two subsections we will describe two approaches for finding similar books. The first approach is item-based and is based on the rating of books performed by the community of borrowers. The second approach is content-based and is based on the equality of the attributes describing the books.

5.1.1 Item-based Collaborative Filtering

The starting point is a borrower-book matrix as depicted in Figure 5.1, where we along the rows have a list of borrowers $U = \{$*Andrew, Betty, Chandler, Denise, Edward, Francise*$\}$ and along the columns have a list of books $I = \{$*DZ, LR, HP, HB, ED, It*$\}$, which are abbreviations for *The Dead Zone, Lord of the Ring, Harry Potter, Hound of the Baskervilles, Eyes of the Dragon* and *It*, respectively. A value in a cell expresses the rating given by a particular borrower on a particular book. The rating can be of either implicit or explicit nature. Explicit rating is when a borrower explicitly has rated a book for

	DZ	**LR**	**HP**	**HB**	**ED**	**It**
Andrew	2	3		1		
Betty	1	3	5			
Chandler						
Denise			1	5	1	1
Edward						
Francise			4	2	4	4

Figure 5.1: Borrower-book Matrix with Rating Scores.

	i_1	i_2	$\cdots$	i_n
u_1		3		5
u_2	5			
$\vdots$				
u_m	2	4		

Figure 5.2: User-item Matrix with Rating Scores.

instance by assigning a numeric score between 1 and 5, where 1 is very bad and 5 is very good. An implicit rating can be obtained by deducing a borrower's behavior for instance based on the borrower's browsing pattern telling which books the borrower has browsed through and the number of revisits of each page. As we can see from Figure 5.1 *Betty* has rated *DZ* with a rating score of *1* and *Francise* has rated *It* with a rating score of *4*.

The borrower-book matrix in Figure 5.1 can be formally expressed as in Figure 5.2, where we along the rows have a list of m users $U = u_1, u_2, \ldots, u_m$ and along the columns have a list of n items $I = i_1, i_2, \ldots, i_n$. Each user u_i has a list of items I_{u_i}, which the user has rated. Furthermore, $I_{u_i} \subseteq I$, and I_{u_i} can be empty, if the user has not rated any items.

The task of CF algorithms is to find item likeliness for an *active* user $u_a \in U$. The likeliness is expressed in two forms:

Prediction is a numerical value, $P_{a,j}$, expressing the predicted likeliness of item i_j not belonging to I_{u_a} for the active user u_a. This predicted value is within the same scale as the rating scores provided by u_a.

Recommendation is a list of N items, $I_r \subset I$, that the active user will probably like the most, where $I_r \cap I_{u_a} = \emptyset$.

	Most similar items
DZ	LR, HP, HB
LR	DZ, HP, HB
⋮	...

Figure 5.3: Conceptual View of Stored Similar Items.

In the following similarity and prediction computations are explained.

Similarity Computation

Prediction computation are based on item similarity values. We use the *adjusted cosine similarity* formula from [10], which basically computes the cosine value of the angle between two vectors, where each vector represents an item and the elements of the vector is the rating scores from each user. It is *adjusted* because the average rating is subtracted from each rating score. The closer the two vectors are to each other, i.e., high cosine value, the more similar they are. The *adjusted cosine similarity* from Equation 5.1 contains the following variables not previously explained. $R_{u,i}$ is the rating given by user u on item i. $R_{u,j}$ is the rating given by user u on item j. $\overline{R_u}$ is the average of the u-th user's ratings.

$$sim(i,j) = \frac{\sum_{u \in U}(R_{u,i} - \overline{R_u})(R_{u,j} - \overline{R_u})}{\sqrt{\sum_{u \in U}(R_{u,i} - \overline{R_u})^2}\sqrt{\sum_{u \in U}(R_{u,j} - \overline{R_u})^2}} \tag{5.1}$$

The similarity values are precomputed offline and for each item j its corresponding k most similar items are stored, where $k \ll n$. k is termed the *model size*. The k most similar items are used to do prediction computation. A conceptual view of the stored similar items is depicted in Figure 5.3.

It should be noted that when the user-item matrix is extremely sparse, say 99%, many of the similarity values are either 1, 0 or -1, which does not give a true picture of the relation between those implicated items. With such a sparse matrix, a similarity value of 0 occurs when users, who have rated both item i and j, have rated in such a way that the rating for either item i or j equals their average rating score $\overline{R_u}$. However, the probability for this to happen decreases as the matrix becomes more and more populated. Take for instance the similarity computation for items *DZ* and *LR* based on the formula from Equation 5.1 and the borrower-book matrix from Figure 5.1, where the borrowers are *Andrew* and *Betty*. As the result of the similarity computation is undefined ($\emptyset$) we regard this as being 0, see Formula 5.2. The closer the similarity value for item i and j approaches 0, the less correlated they are.

$$sim(DZ, LR) = \frac{(2-2)(3-2)+(3-3)(1-3)}{\sqrt{(2-2)^2+(3-3)^2}\sqrt{(3-2)^2+(1-3)^2}} = \emptyset \qquad (5.2)$$

For the similarity value to be 1 both item i and j must have received the same rating from each user, see Formula 5.3. It means that item i and j have a perfect correlation, i.e., they are perfectly alike. However, this is unlikely as the matrix becomes more and more populated but very likely with an extremely sparse matrix. Consider the similarity computation for items *ED* and *It*, where the only borrowers are *Denise* and *Francise* as can be seen from Figure 5.1.

$$sim(ED, It) = \frac{(1-2)(1-2)+(4-3.5)(4-3.5)}{\sqrt{(1-2)^2+(4-3.5)^2}\sqrt{(1-2)^2+(4-3.5)^2}} = 1 \qquad (5.3)$$

The opposite of a perfect correlation is a perfect negative correlation, i.e., whenever item i get a high rating item j will get the opposite low rating, see Formula 5.4. Consider the similarity computation for items *HP* and *HB*, where only *Denise* and *Francise* have borrowed both books.

$$sim(HP, HB) = \frac{(1-2)(5-2)+(4-3.5)(2-3.5)}{\sqrt{(1-2)^2+(4-3.5)^2}\sqrt{(5-2)^2+(2-3.5)^2}} = -1 \qquad (5.4)$$

The more user ratings backing up a similarity value the more trust we can place in the similarity value. Consequently, we ought to devalue our confidence in similarity value, which is only backed up by a few number of users. As recommended by [23] we use a significance weighting factor of $\frac{n}{50}$ to devalue the similarity value if it is only backed up by fewer than 50 users, where n is the number of users. The threshold is 50 and if n is larger than the threshold the similarity value is left untouched.

Prediction Computation

The prediction value of item i for user u, $P_{u,i}$, can be computed when similarity values have been computed. This is done by weighting each ratings by the corresponding similarity $s_{i,j}$ between item i and j based on items similar to i. The following formula from [10] is used:

$$P_{u,i} = \frac{\sum_{all\_similar\_items,N}(s_{i,N} * R_{u,N})}{\sum_{all\_similar\_items,N}(|s_{i,N}|)} \tag{5.5}$$

Here, *all_similar_items* is the set of items which are similar to item i.

A recommendation top list is generated by repetitively applying the prediction formula to those items that are similar to the one, which the user is currently looking at.

To be specific, imagine that a borrower has found the book *LR* at Amazon. In order to generate a recommendation top list the system retrieves precomputed similarity values of books that are similar to *LR* from the database. According to Figure 5.3 the following books are the similar ones: *DZ, HP, HB*. Those books for which the borrower has already rated is of no interest and should not occur in the recommendation top list, thus it is omitted from the prediction computation. The prediction formula is applied for each of the books for which the user has not yet rated.

The prediction values for *DZ, HP*, and *HB* are computed and ordered in descending order such that the book with the highest prediction value is on top hence forming a top list recommendation for the borrower. In order to compute the prediction value for, e.g., *DZ* we have to look up its similar books. According to Figure 5.3 we can see that these books are *LR, HP*, and *HB*. We apply Equation 5.5 to compute the prediction value of *DZ* for the particular borrower. We repeat the prediction computation for both *HP* and *HB* to find their prediction values. Once the prediction values have been found they are ordered in descending order to produce the top list recommendation.

5.1.2 Content-based Recommendation

When doing Collaborative Filtering it is important to be able to make recommendations for, e.g. all books. However, this is not always possible when only using item-based collaborative filtering. There exists two fundamental problems as proposed in [28] namely *the sparsity problem* and *the first-rater problem.*

Sparsity is a problem that occurs when users are not rating all the items in a database. For AUB it means that each borrower borrows much less than one percent of all the books because when calculating the sparsity of 6 years of transactions from AUB the sparsity is 99, 996%. This results in a sparse borrower-book matrix. We will later show how we calculated the sparsity (in Formula 5.8). The sparsity problem makes it impossible for the item-based CF approach to find items that are similar because of not being able to find any borrowers that have borrowed the books.

The other problem is the first-rater problem which says that an item cannot be recommended unless a user has rated it before. For AUB the problem can be seen when a new book is bought. It will not be recommended by the item-based approach because no transactions exist where the new book have been been borrowed.

A more serious problem is when users are borrowing books that have no relevance with each other, or if the majority of the users are borrowing books of low professional quality. This means that the recommendations made by the item-based approach cannot be used, because the recommendations made are of low quality and therefore useless.

Another problem is the self-perpetuating effect that raises when users are only borrowing books that are being recommended by the item-based approach. This means that the same books are being recommended by the item-based approach every time. Books that are not recommended by the item-based approach have lesser chances to appear in a recommendation. It is wise to combine the item-based approach with another approach that can handle the problems in a fairly good way. One such approach is the content-based approach. The content-based approach makes use of the data describing the items to find matches between the items and thereby making recommendations. However, one must not rely only on the content-based approach because the quality of the recommendations is only based on the description of each item. Therefore, it is ideal to make a hybrid approach between the item-based and content-based approach in order to make recommendations for all items.

Content-boosting

We have currently looked at two ways to implement the content-based approach and thereby making use of *content-boosting*. Content-boosting makes use of the item-based approach combined with a content-based approach that relies on classes of keywords that have been rated by different users. The classes of keywords are then used to rate each book. The first approach to do content-boosted collaborative filtering is described in [28]. The second way to do content-boosting is to do content-based book recommending using learning for text categorization as described by [27] and then combine the result with the item-based approach in some other way. The reader is advised to read the two articles in order to get an understanding of how the two approaches work.

However, there still exists one problem with the two proposed approaches, namely that user ratings of classes of words are needed in order to do the recommendation using the content-based approaches. This means that if no user ratings exist for the classes of words the approaches cannot be trained. Moreover, no algorithm such as, e.g., Naive Bayes which is used in [28], can be used for computing similarities for the items.

Naive CBPM

We therefore developed a new approach to make the content-based recommendations. It is inspired by the approach for making Text Categorization using the Naive Bayes algorithm. The Naive Bayes algorithm uses an approach that trains some classifiers on different kinds of data samples. Then some new data are tested by the model that has been built in order to find out how to categorize the unknown data. The Naive Bayes algorithm is also currently used, e.g., for finding spam emails like proposed in [24] and this is done with quite a good result as written in the article. However, our approach does not make use of the Bayes rule, because the number of classes that have to be trained would be equal to the number of books. The reason for this is that in order to be able to find the probability for a book being equal to another book, the book has to be compared with the class the other book is in. It would then not be possible to calculate the probability because the number of classes is too big for the Bayes formula. Instead it makes use of another idea in the algorithm namely that the placement and repetition of words are not important for classifying words.

We combine the idea of the Naive Bayes algorithm with information extraction and pattern matching, and thereby, create a new algorithm, called Naive CBPM, *Naive Content-Based Pattern Matching.* The content-based approach is shown in Figure 5.4. We will give a short description of the algorithm.

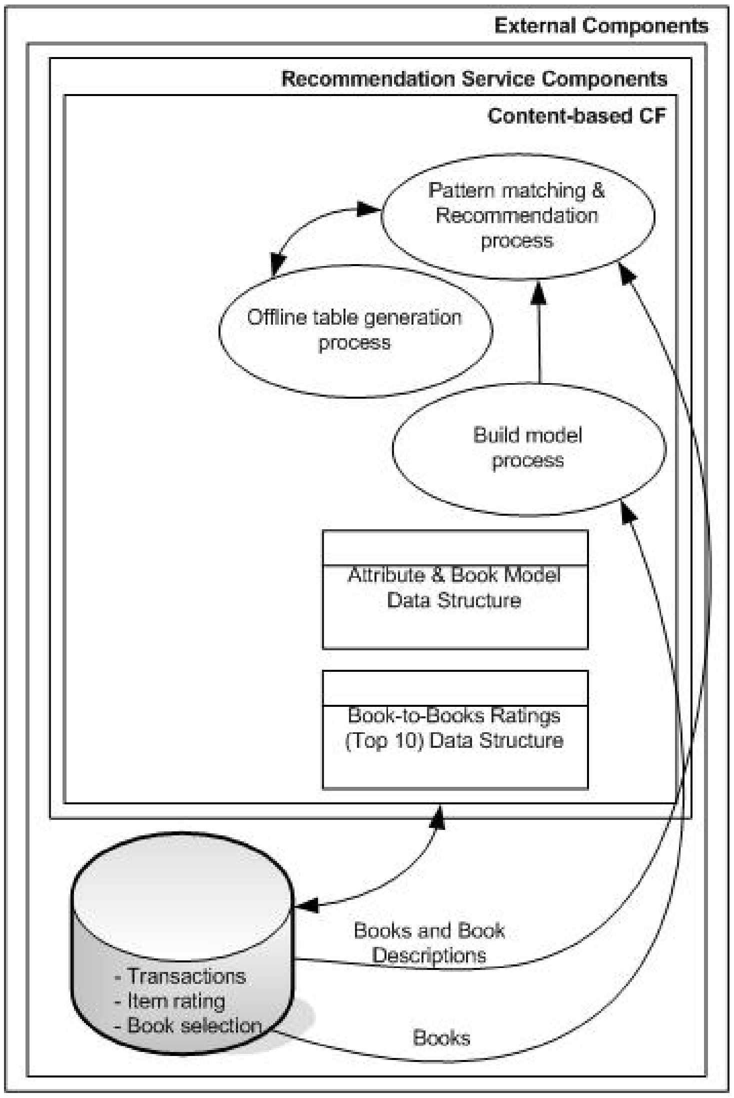

External Components
Recommendation Service Components
Content-based CF
Pattern matching & Recommendation process
Offline table generation process
Build model process
Attribute & Book Model Data Structure
Book-to-Books Ratings (Top 10) Data Structure
- Transactions
- Item rating
- Book selection
Books and Book Descriptions
Books

Figure 5.4: Illustration of AUB's Content-based Recommendation System.

Naive CBPM:

```
buildWordList()
   while (more books)
      for the current book
         extract all words from title
         extract all words from author
         remove words that are in the stopList
         store new words in wordSet
   return the whole wordSet or a top−N wordSet

getPattern(wordSet)
   for each word in wordSet
      put the word in a bitSet where corresponding bitIndex maps to a word
   return the bitSet ( General Pattern)

buildModel(bitSet)
   while (more books)
      for the current book
         compare the book with the bitSet
            store the bits that are set in data structure set of bookIDToPattern
            store the bits that are set in data structure set of attributeToBookIDs
   return the bookIDtoPattern and patternToBookIDs data structures

buildRecommendations(set of bookIDtoPattern and set of attributeToBookIDs)
   while (more books)
      for the currentBook
         get the pattern from the set of bookIDtoPattern
         for each attribute in the pattern
            get the bookIDs from the set of attributeToBookIDs
               for each bookID not equal to currentBook
                  check whether the book pair (currentBook, bookID) −
                   exists in item−based similarity table
                  if not true
                     increase the rating for bookID in the set of bookSimilarties
         sort the bookSimilarties
         store the top−N bookRatings in a datastructure or file
      continue to next book
```

The algorithm consists of three parts:

1. **The first part is the information extraction pass.**
   This consists of the *buildWordList()* and *getPattern(wordSet)* methods.

2. **The second part is the model building pass.**
   This consists of the *buildModel(bitSet)* method.

3. **The third part is the similarity or recommendation building pass.**

> This consists of the *buildRecommendations(set of bookIDtoPattern and set of attributeToBookIDs)* method.

As seen in the algorithm it makes a number of passes over the items and their descriptions. Furthermore, it makes the pattern matching of the items iterative. However, because the algorithm can be implemented to make use of HashMaps in memory, the performance is quite good.

Two parameters are available for adjusting the algorithm for both quality of the recommendations and the speed of the algorithm. The first parameter is the pattern size. If the size is increased, the quality of recommendations should become better because more words are included for comparing the items. However, the speed is decreased because of the size increase of the pattern. The second parameter is the number of items to be analyzed by the algorithm. This also have an impact on both the speed and quality of the algorithm. It is important here to mention that quality is equal to the number of matches of words between two items.

We have chosen to set the pattern size to include all words and the item size to include all books. This is done in order to be able to compare books that have the same words with each other, which means that we compare ”an apple with an apple”. A description of a book is currently only consisting of the title and authors of the book, because we do not have the keywords for the books.

The benefit of using the algorithm is that the computation of the content-based recommendation is fully automated and it is based on the description of the items that should include some semantics about the item. The argument for this is that the descriptions of each item has been made by users describing the item. Therefore, the idea is that two books are considered similar when the description of the two books has a high correlation.

5.2 Implementation

After having elaborated on the item-based and content-based approach for recommendation generation, the following sections focuses on the implementation of the approaches and the recommendation service. We have combined the content-based approach with the item-based approach in order to implement the recommendation service. This is illustrated in Figure 5.5. All the table definitions used in the implementation are listed in Appendix D.

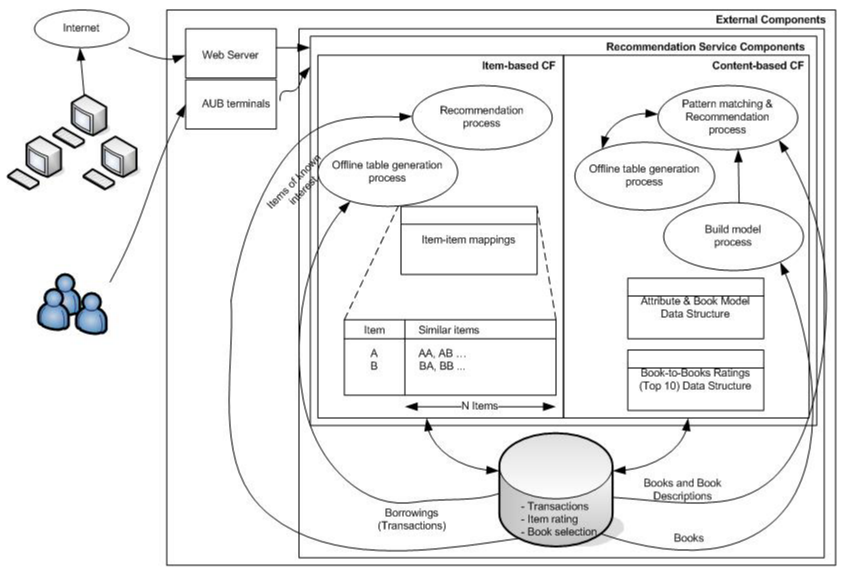

Internet
Web Server
AUB terminals
External Components
Recommendation Service Components
Item-based CF
Content-based CF
Recommendation process
Offline table generation process
Items of known interest
Item-item mappings
Item
Similar items
A
B
AA, AB ...
BA, BB ...
N Items
Pattern matching & Recommendation process
Offline table generation process
Build model process
Attribute & Book Model Data Structure
Book-to-Books Ratings (Top 10) Data Structure
- Transactions
- Item rating
- Book selection
Borrowings (Transactions)
Books and Book Descriptions
Books

Figure 5.5: Illustration of AUB's Recommendation Service.

5.2.1 Item-based CF

Item-based CF has been implemented in a recommendation service in order to enable AUB's web site to recommend books. The implementation requires that AUB registers the borrowers' ID when a book is borrowed. The borrower ID is needed in order to do the similarity computation as described in Section 5.1.1. Without the borrower IDs the item-based CF would not be possible to implement. Due to legal issues, the borrower IDs have been encrypted in order to not be able to identify a particular user but still be able to distinguish the users from each other.

An item-based CF implementation consists of the following parts as shown in Figure 5.5: A table mapping books to similar books is generated offline by using the loan history from the database. A recommendation process which fetches the similar books for a particular book and the books of known interest for a borrower, which the recommendation service must make a recommendation for. The recommendation service is accessed by the borrowers either through Auboline or at terminals at AUB. The books of known interest for a particular borrower is not used in the implementation for AUB. This is explained in more detail in Section 5.2.1, Prediction.

In order to carry out the similarity computation for each pairs of co-borrowed books we need some data from two tables. The first one is called *loanCount* and reflects how many times each borrower has borrowed a given book. The second one is called *borrower2Books* and allows us to conveniently retrieve a list of borrowers, which have borrowed the same two books. Once similarity values for pairs of co-borrowed books have been calculated they are stored in a *similarity* table along with the value.

If we have n books, we might have up to $\frac{n*(n-1)}{2}$ pairs of co-borrowed books. Each similarity computation for such a pair makes several queries to the DBMS for the needed data. For each query the query planner is invoked in order to optimize it. This amounts to a great number of query planner invocations, thus very time consuming. Instead, we have chosen to represent *loanCount* and *borrower2Books* by two data structures in memory. This way we can "query" for the data in-memory and avoid spending time on the query planner. The next section describes the two data structures representing the two tables.

In-memory Data Structures

The *loanCount* table is represented by the data structure shown in Figure 5.6. It consists of an outer HashMap, which maps each borrower ID to another HashMap. Each of the other HashMap consists of a set of book IDs representing the books, which the borrower has borrowed. Each of the book ID maps to an integer named *count* representing the number of times the borrower has borrowed this book.

The *borrower2Books* table is represented by the data structure shown in Figure 5.7. It consists of an outer HashMap, which maps a book ID to another HashMap. Each of the other HashMap consists of a list of book IDs representing the books, which have been borrowed together with the book in the outer HashMap. Each of the book ID maps to an ArrayList of borrower IDs. This way we can for each pair of co-borrowed books retrieve a list of borrowers.

Ratings

The item-based CF for AUB makes use of the implicit rating from the borrowers by keeping score of how many times each borrower has borrowed different books. The implicit rating is used in order not to depend on borrowers rating enough books before the system will be able to recommend any books. Instead the ratings are extracted from the *loanCount* data structure by first getting the number of loans a borrower has had of a specific book.

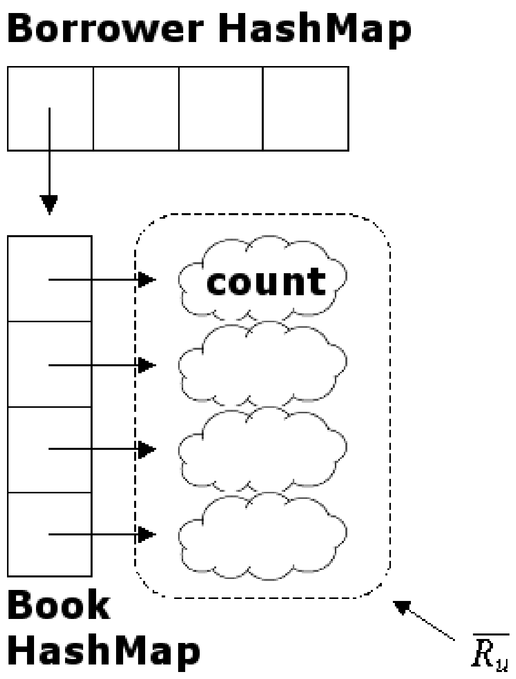


Figure 5.6: loanCount Data Structure.

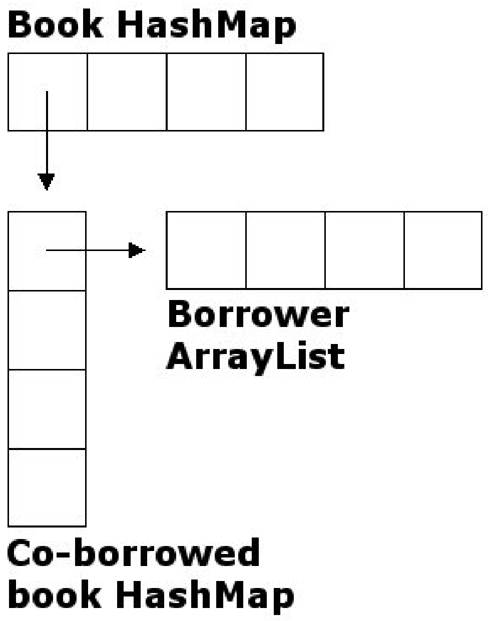


Figure 5.7: borrower2Books Data Structure.

This number is then taken to the ratio of the maximum number of loans of any book in the *loanCount* table. By doing this all the ratings are within the same scale and the books that have been frequently borrowed by the same borrower get a higher score. A formula describing the rating calculation is listed here:

$$R_{u,i} = \frac{count_{u,i}}{Max(count_{u \in U})} \tag{5.6}$$

If, e.g., a borrower has borrowed a book 3 times and the maximum number any books have been borrowed is 20 then the borrower's rating of the book is $\frac{3}{20} = 0.15$. In this case $count_{u,i}$ is 3, $Max(count_{u \in U})$ is 20, and $R_{u,i}$ is 0.15.

Similarity

The similarity is calculated for each pair of co-borrowed books. An overview of the similarity computation is given in the following pseudo code:

```
For each pair of co-borrowed books
    Calculate similarity value
```

The similarity value is calculated using the adjusted cosine formula from Equation 5.1 by doing the following steps:

1. **Finding the borrowers**
   By iterating over the *borrower2Books* data structure we are able to retrieve lists of borrowers who have borrowed pairs of co-borrowed books.

2. **Finding $R_{u,i}$ and $R_{u,j}$**
   We need to find the ratings of the co-borrowed books given by each of the borrowers. This is done by querying the *loanCount* data structure to find the number of times a given borrower has borrowed a particular book and then apply the rating formula from Equation 5.6 to calculate the implicit rating. This calculation is done for each borrower of each of the co-borrowed books.

3. **Finding $\overline{R_u}$**
   The average rating for each borrower is found the first time we query the *loanCount* data structure for that particular borrower. As we have the HashMap of books for that borrower, we iterate over the *count* values in order to calculate the average number of books this borrower borrows each time. The average number is then applied to the rating formula from Equation 5.6. This idea is shown in Figure 5.6.

4. **Calculating similarity**
   A similarity value can be computed when the above steps have been carried out.

The similarity values are all stored in a *similarity* table. The *similarity* table represents the similarity relationship between two books and contains the foreign keys of two books along with the relation attribute *value*, which is the similarity value of the two books. The foreign keys of the two books do not include references to the same book in a single row since similarity between the same book is not calculated.

The *borrower2Books* data structure is serialized at the end of the process of computing the similarity values. This is done because the content-based prediction process needs it to find recommendations for books, which is not already contained in *borrower2Books*. Furthermore, for each similarity computation two book IDs together with the corresponding similarity value is written to file such that the data can be copied into the *similarity* table at the end of all similarity computations.

Prediction

Prediction could have been used in order to do recommendation. The recommendation would have been done by doing the following:

1. Find the top-N similarity values for a specific book.
2. Compute the prediction value for each book associated with a specific similarity in the list of top-N similarity values.
3. Sort the prediction values in descending order.
4. Present the top-N books with the highest prediction values.

We have chosen not to use prediction in order to do recommendation. This is due to the fact that the borrower-book matrix is very sparse and because of the Danish privacy law prohibiting the direct tracking of user behaviors.

The sparsity [10] of the borrower-books matrix is calculated using the following formula:

$$sparsity = 1 - \frac{nonzeroentries}{totalentries} \tag{5.7}$$

If, e.g., we were to represent the borrower-book matrix with the loans from the sample data set from AUB A the sparsity would be:

$$1 - \frac{60,875}{37,022 * 8,758} = 99,996\% \tag{5.8}$$

The example in Formula 5.8 is a borrower-book matrix covering 10 months of loan history consisting of 64,371 loans, a total number of 8,758 borrowers, and 37,022 books. The number of ratings is smaller than the number of loans since a book can be borrowed more than once by the same borrower. A book borrowed several times by the same borrower is only represented with a single rating in the borrower-book matrix.

The high sparsity level is due to the fact that there are many books but each borrower often only borrows few books every year. This is a problem when calculating the prediction, since the similarity value is multiplied by the borrowers' ratings, but if a borrower has not rated any of the similar books then there are no applicable prediction values.

In order to do recommendation by using prediction the system also has to know for which borrower the recommendation is in order to fetch the borrower's ratings. This would require the borrower to login in order to identify the borrower. Due to legal issues AUB is not allowed to make a connection between a borrower login and the encrypted borrower ID in the borrower-book matrix.

By only using the similarity values the recommendation can be done without any need of borrower login and the recommendation can even be done for new borrowers without any ratings. Furthermore, we must obey the Danish law of privacy.

5.2.2 Content-based Recommendation

We have implemented the content-based recommendation approach using the algorithm we developed, and the approach has the system architecture shown on Figure 5.4. The implementation is done in Java and makes use of HashMaps in order to speed up the pattern matching. Furthermore, we fetch the books and the descriptions of each book from the DBMS. The implementation of the algorithm is illustrated in figures for each pass.

The first pass is information extraction and the second pass is building the model. This is implemented as shown in Figure 5.8. It shows how the algorithm is implemented in Java and how it makes use of HashMaps in order

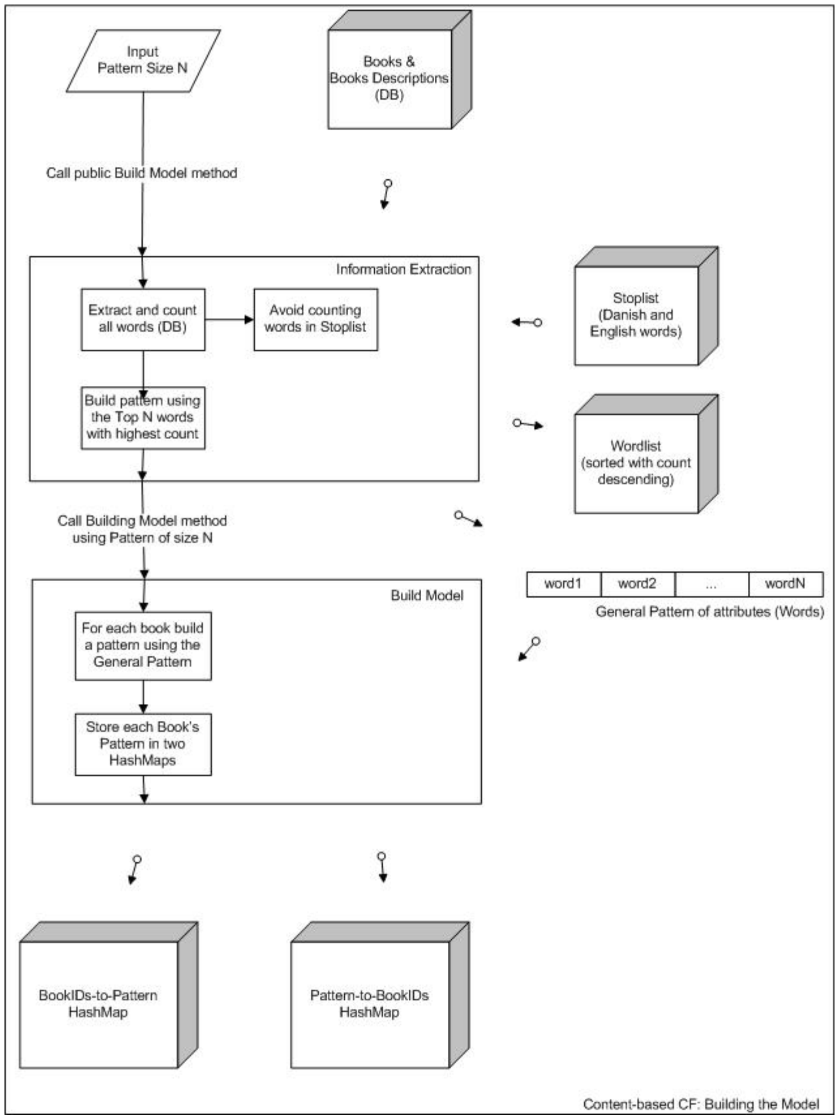


Figure 5.8: Information Extraction and Building the Model.

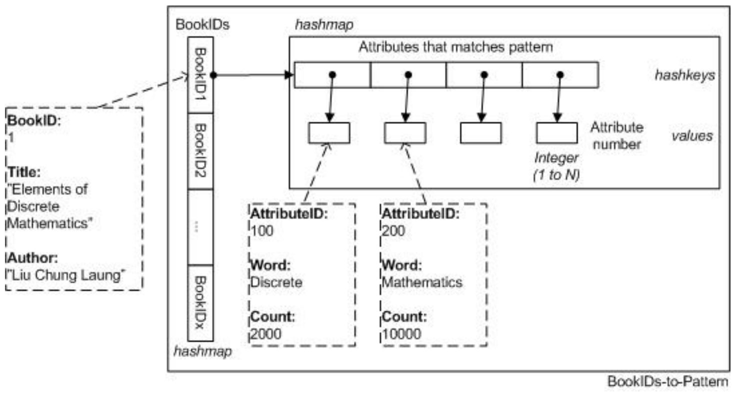


Figure 5.9: HashMap for BookIDs-to-Pattern.

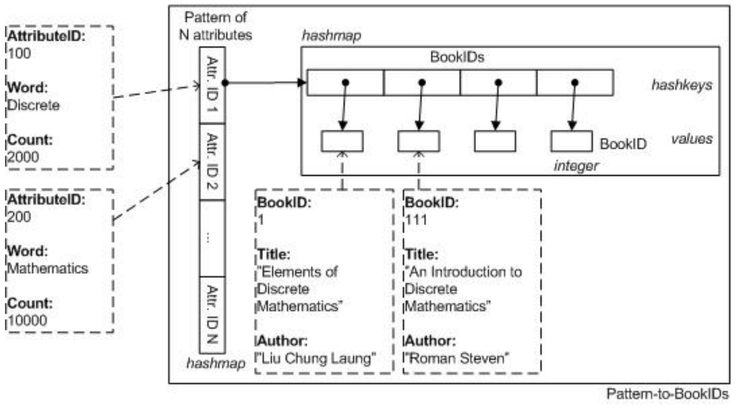


Figure 5.10: HashMap for Pattern-to-BookIDs.

to store *stoplist*, *wordlist*, the *bookIDs-to-pattern* HashMap and the *pattern-to-bookIDs* HashMap. The data structure of the two HashMaps are shown in Figure 5.9 and Figure 5.10. The *bookIDs-to-pattern* HashMap data structure stores the item mapping to the attributes that matches. The *pattern-to-bookIDs* HashMap data structure stores each attribute mapping to a series of items that match the attribute.

The third pass is the building of the similarities of the books that are stored in the *patternsim* table in the DBMS. This pass finds the similarities by counting the number of matches between the book that recommendations are wanted for and all the other books. This is shown in Figure 5.11. However, recommendations are only made for book pairs that are not in the item-based *similarity* table. This is done in order to avoid having duplicates of recommendations. As a side note the book with the highest number of matches is of course the book itself, because when comparing the book with itself it is equal.

The result of the algorithm is shown in Figure 5.12. It shows a HashMap consisting of all the books, where each book maps to a HashMap consisting of books sorted in highest rating order that matches the current book.

5.2.3 Recommendation

The recommendation can be done online by using the results from the offline computations for item-based CF and content-based recommendation. The results are the similarity mappings which are stored in the tables *similarity* and *patternsim*. How the two implementations are integrated is illustrated in Figure 5.13.

Having the *similarity* table it is only a matter of doing a lookup in order to do a recommendation given a specific book. This can be done by finding all the pairs that the specific book participates in, then sort by the similarity values, and list the, e.g., top 10 similarity values along with the author name and title for each book.

The recommendation might not find enough mappings in the *similarity* table to do a top 10 recommendation. The *patternsim* table is used in order to compensate for the missing mappings in the *similarity* table. The recommendation is implemented by first fetching maximum 10 mappings from the *similarity* table and the actual number of mapping is counted while writing the recommendation output. If the number of mappings is under 10 then the rest of the recommendation is fetched from the *patternsim* table.

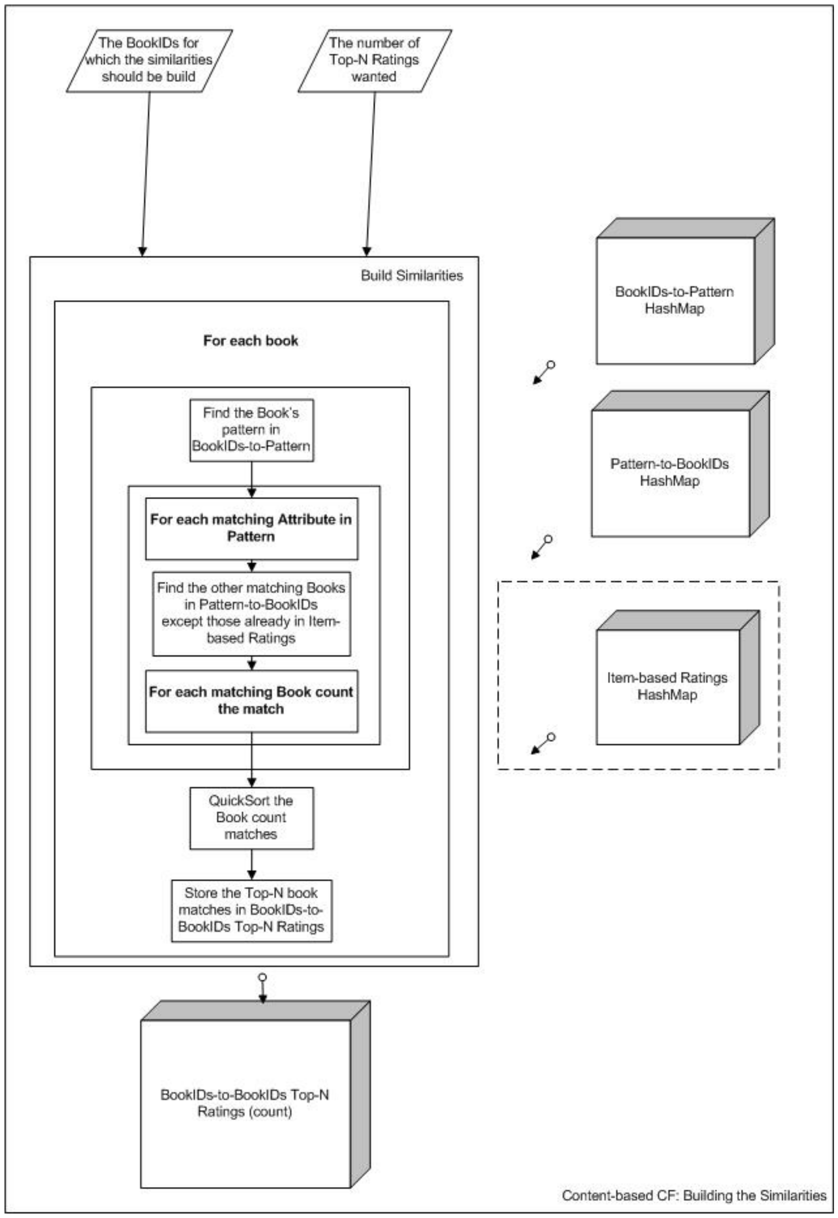

The BookIDs for which the similarities should be build
The number of Top-N Ratings wanted
Build Similarities
For each book
Find the Book's pattern in BookIDs-to-Pattern
For each matching Attribute in Pattern
Find the other matching Books in Pattern-to-BookIDs except those already in Item-based Ratings
For each matching Book count the match
QuickSort the Book count matches
Store the Top-N book matches in BookIDs-to-BookIDs Top-N Ratings
BookIDs-to-Pattern HashMap
Pattern-to-BookIDs HashMap
Item-based Ratings HashMap
BookIDs-to-BookIDs Top-N Ratings (count)
Content-based CF: Building the Similarities

Figure 5.11: Building the Content-Based Similarities (Recommendations) for each Book.

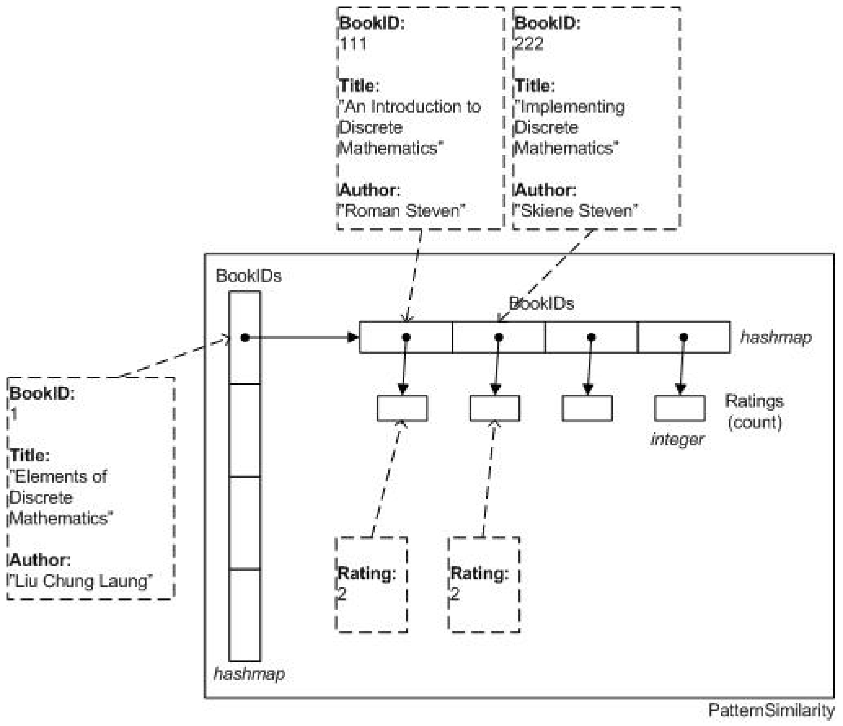


Figure 5.12: The Result of the Algorithm - the Book Recommendations.

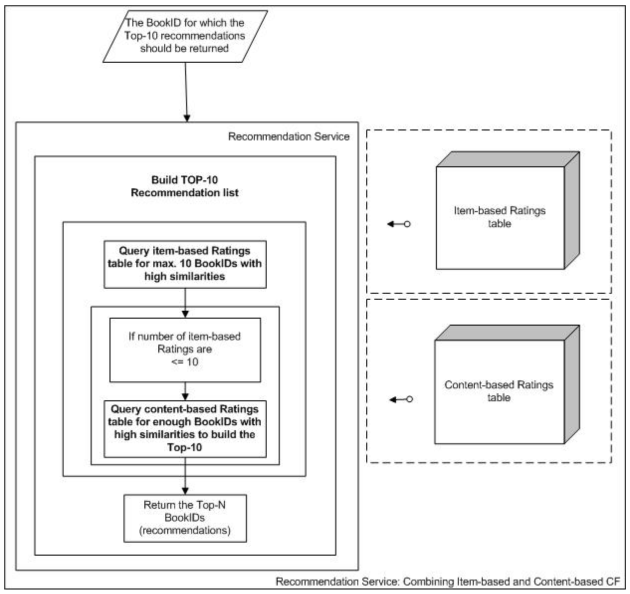

The BookID for which the Top-10 recommendations should be returned
Recommendation Service
Build TOP-10 Recommendation list
Query item-based Ratings table for max. 10 BookIDs with high similarities
If number of item-based Ratings are <= 10
Query content-based Ratings table for enough BookIDs with high similarities to build the Top-10
Return the Top-N BookIDs (recommendations)
Item-based Ratings table
Content-based Ratings table
Recommendation Service: Combining Item-based and Content-based CF

Figure 5.13: The Recommendation System.

The recommendation is produced by a recommendation service which outputs XML. The XML can easily be parsed at the Auboline web site, which already uses XML services. An example of the XML output is listed in Appendix D. An example of how AUB can implement the XML recommendation service is illustrated with screen shots in Chapter 6. Two results of the recommendation service are listed and evaluated in Section 5.4.

5.3 Performance

In this section the performance of item-based CF and content-based recommendation is examined by looking at the execution time and memory usage. The performance is examined in order to see how well the two implementations scale according to the number of similarity values computed. The performance of the online recommendation is also examined by looking at the query plan for the queries used in the online recommendation service.

5.3.1 Execution Time

The execution time of item-based CF and content-based recommendation is listed in Figure 5.14 and 5.15. Both approaches seem almost to scale linearly with the execution time and the similarity found. However, there seems to be an exception regarding the execution time for item-based CF as it takes relatively more time to process the last 4,373 transactions. One of the reasons is the need of increasing either the *loanCount* or the *borrower2Books* HashMap resulting in rehashing either of one or both of the HashMaps, which is a time consuming process [3]. Another reason is that the sum of PostgreSQL's and Java's memory usage exceeds the available main memory capacity. Consequently swap file is used to compensate for lack of memory.

5.3.2 Scalability

The memory usage of item-based CF and content-based recommendation is listed in Figure 5.16 and 5.17. In both implementations the size of the tables in the database containing the similarity mappings increase with the number of books or transactions. The item-based CF implementation uses more main memory as more similarity values are computed as illustrated in Figure 5.16. The increase in main memory usage is caused by the HashMaps data structures which increase in size since they contain more loan history,

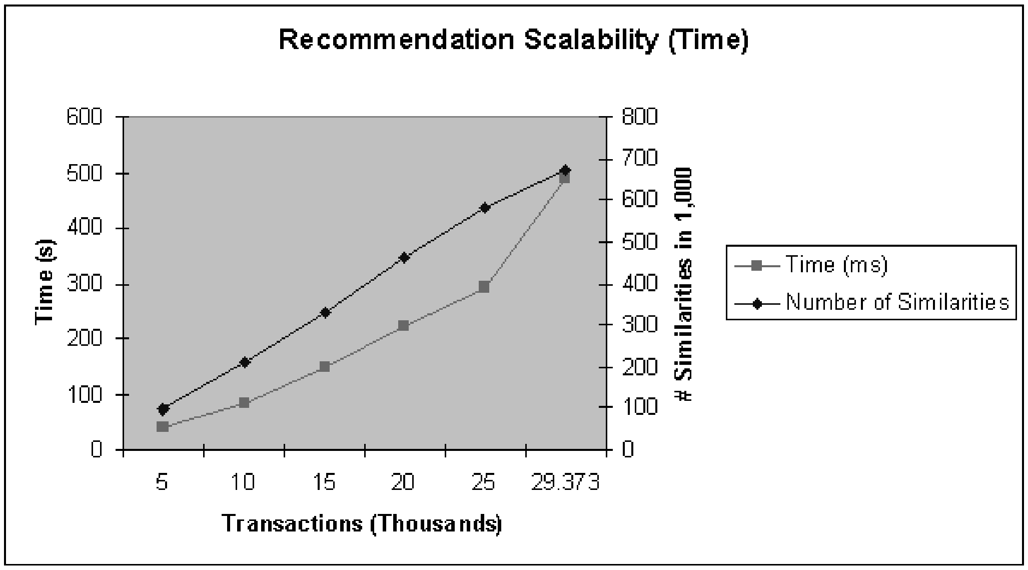


Figure 5.14: The Execution Time of the Item-based CF Implementation.

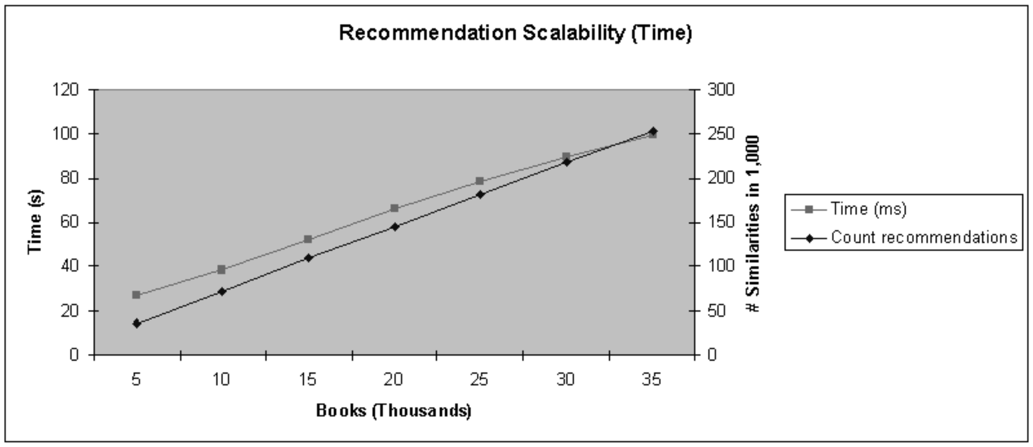


Figure 5.15: The Execution Time of the Content-based Recommendation Implementation.

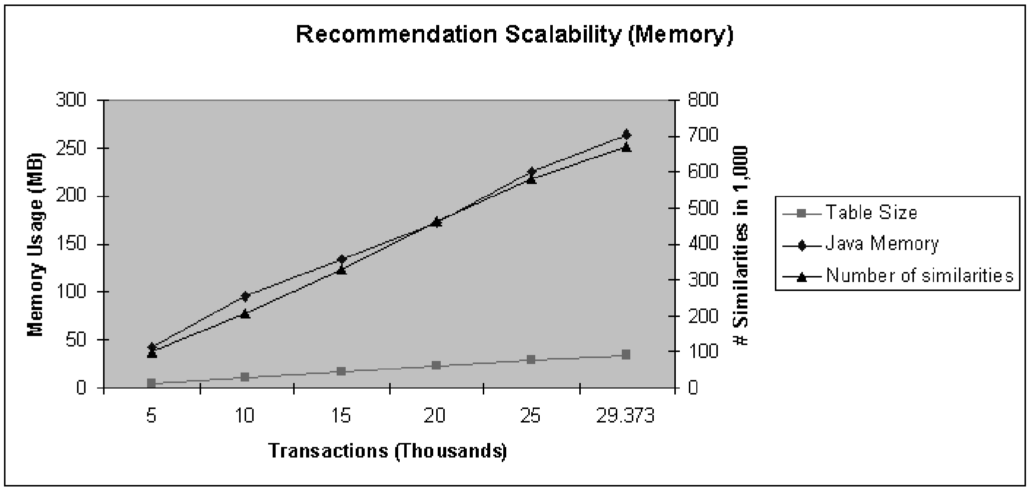


Figure 5.16: The Memory Usage of the Item-based CF Implementation.

which leads to extra similarity values. Furthermore, the computed similarity values are buffered in memory until the similarity computation is done. The main memory usage of the content-based recommendation implementation is constant as illustrated in Figure 5.17. The main memory usage is constant since the HashMaps are reused for each book computation. However, memory is also reserved for the model.

5.3.3 Online

The online recommendation consists of simple SQL queries to the DBMS that run through a XML Service. The XML Service receives the results from the queries and the web page is displayed using the Tomcat Web Server. We have therefore looked at the performance of the SQL queries in order to find out how much time it takes for the query to be executed by the DBMS.

In the PostgreSQL DBMS it is possible to get a query plan for the current query, an estimated execution time, and a real execution time of the current query. We have used these results to find out the time it takes to execute a query for one recommendation.

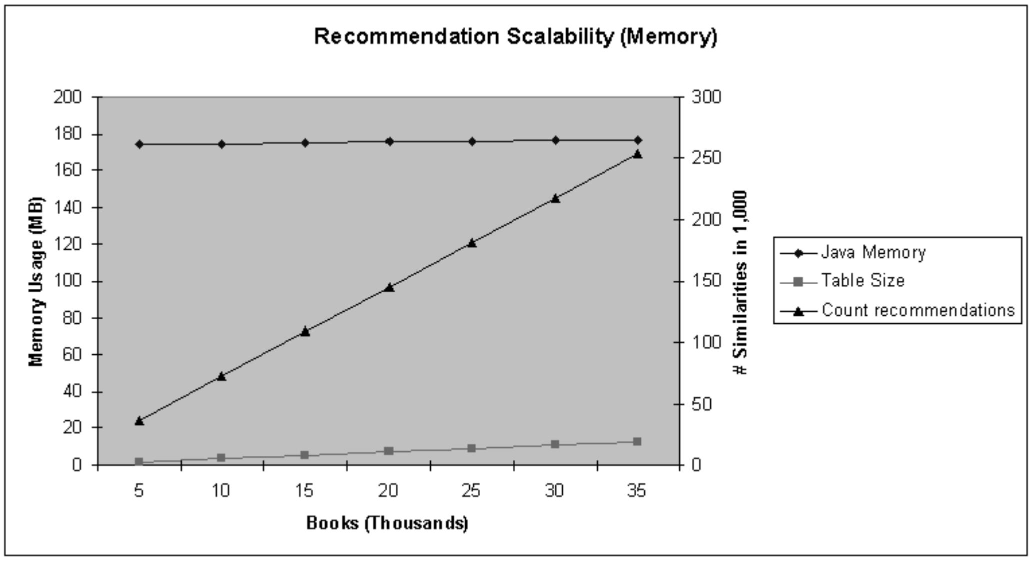


Figure 5.17: The Memory Usage of the Content-based Recommendation Implementation.

Item-based Query

The following query will retrieve books that are similar to a current book, which can be either *i* or *j* in each data row. Each book will have a similarity value that has been precomputed using the item-based CF approach. There are a total of 672,129 similarities that is made from 29,373 transactions.

The result of the query plan is:

The Query is:

```
SELECT
    i, j, value
FROM
    similarity
WHERE
    j = 148347 OR i = 148347
ORDER BY
    value DESC
LIMIT 10
```

The query plan and execution time returned from the DBMS:

```
                                    QUERY PLAN

Limit
(cost=471.20..471.22 rows=10 width=16)
(actual time=14.000..14.000 rows=10 loops=1)
  -> Sort
  (cost=471.20..471.51 rows=124 width=16)
  (actual time=14.000..14.000 rows=10 loops=1)
        Sort Key: value
        -> Index Scan using s_j_index, ps_i_index on similarity
        (cost=0.00..466.89 rows=124 width=16)
        (actual time=7.000..14.000 rows=26 loops=1)
              Index Cond: ((j = 148347) OR (i = 148347))

Total runtime: 15.000 ms
```

The query fetches the top 10 item-based similarity values for a book from the *similarity* table. Indices have been created for each attribute. The actual runtime is measured on a 1 GHz Intel 32-bit processor with 512 MB of RAM.

As seen in the actual runtime it takes about 1 ms in order to execute the query and return the results. Furthermore, if we take a look at the query plan, we can see the cost for executing the query. The cost is measured as the estimated statement execution cost. This is the planner's guess at how long it will take to run the statement (measured in units of disk page fetches). Actually two numbers are shown: the start-up time before the first row can be returned, and the total time to return all the rows.

If we take a look at the line beneath LIMIT we can find out what the total cost of the entire query is by looking at the start-up time and the total-time values of the line. This is because the LIMIT command is executed after all lines beneath it has been executed. Therefore, the number of disk page fetches is totally 471.22 to return all 10 rows of similarities from the DBMS.

By analyzing the whole query plan, we can find the sub query that takes the longest time is the Index Scan where the DBMS searches for the book with ID 148347 using Btree indexing. However, because indices are created, the DBMS makes use of these as seen in the query. Therefore, the number of disk page fetches is 466.89 because it needs to fetch 124 rows with a width of 16 from the table at the position in the index where i or j is equal to 148347. The next sub query that takes time to execute is the sorting of the ratings so that the book recommendations come in descending order. However, this only takes $471.51 - 466.89 = 4.62$ disk page fetches. The last sub query is to make the numbers i and j unique and the total number of disk fetches is therefore 471.22.

Content-based Query

The following query will retrieve all the books that are similar to the current book, which is bookid1. Each book has an ID (bookid2) and a similarity value (value). The books are sorted in the table so that books with high similarity come before books with lower similarity. There are a total of 271,621 similarities that is made from 37,022 books. More information about AUB Sample Data can be found in Appendix A.

Here is the query plan for fetching book recommendations from the content-based recommendation table:

The Query is:

```
SELECT
    bookid2, count
FROM
    patternsim
WHERE
    bookid1 = 148347
LIMIT 10
```

The Query Plan and Actual Execution Time returned from the DBMS:

```
                                    QUERY PLAN

Limit
(cost=0.00..2.86 rows=10 width=8)
(actual time=19.000..19.000 rows=1 loops=1)
  -> Index Scan using ps_bookid1_index on patternsim
  (cost=0.00..3.15 rows=11 width=8)
  (actual time=19.000..19.000 rows=1 loops=1)
        Index Cond: (bookid1 = 148347)

Total runtime: 11.000 ms
```

The query is run on a *patternsim* table that consists of the book recommendations for a given book with ID 148344. The estimated cost of executing this query is totally 2.86 disk page fetches.

On the basis of the analyzes of the query plans from the DBMS it is now possible to find out how many users can make use of the web site before the server is overloaded. Unfortunately, we do not have any technical details about the server that the system will reside on.

In order to get a good overview of the performance of the server and the online system real life tests must be done. A simple experiment can be setup by creating multiple clients running in threads on a single machine that retrieves data from the web site. This would make it possible to make a test that would emulate a real life scenario with multiple requests that are being sent to the DBMS. We could then find out what the throughput of the system is when it is run on a normal server that only needs to run our system.

5.4 Evaluation

The item-based CF and content-based recommendations are evaluated in this section. The evaluation is not a full evaluation due to lack of time. How a full evaluation should be done as described in the Future Work section.

We have used the item-based CF and content-based recommendation for doing recommendation using a sample set of loan history from AUB. The loan history consists of 64,371 loans from 10 months in the year 2003. The 64,371 loans cover the loans of 37,022 books so each book is borrowed around two times on average. More details of the loan history is listed in Appendix A.

5.4.1 Item-based CF

The item-based CF implementation run on the sample loan history results in a *similarity* table with 672,129 rows meaning that similarity is computed for 672,129 different book pairs. All the book pairs in the *similarity* table consist of 36,122 distinct books. This leads to 900 books for which there are no similar books. The lack of similar books is due to the fact that a book has to be borrowed by one borrower, which has also borrowed other books, in order to have a similarity computation and thereby map to similar books. Some of the books also only map to one similar book.

Most of the similarity values in the *similarity* table have the value 0. There are 216,577 similarity values of 0. The many 0 similarity values are due to the small amount of loan history. As described in Section 5.1.1 the similarity computation often gives 0 when the computation is based upon a very sparse borrower-book matrix. The 0 values is kept in the table because the similarity values are between -1 and 1 and 0 is not the same as not having any value.

All the recommendation can be done for each of the 36,122 books in the *similarity* table. An example of a recommendation is listed in Figure 5.18. The book used for recommendation is the book *Læring* written by Knud Illeris.

Author Name	Book Title	Similarity Value
Knud Illeris (red.)	Tekster om læring	1
Jens Bjerg (red.)	Pædagogik	1
Helle Alrø (red.)	Videoobservation	1
Helle Alrø	Personlig kommunikation og formidling	1
Oluf Danielsen	Læring og multimedier	1
Erik Damberg	Pædagogik og perspektiv	1
Else Hiim	Undervisningsplanlægning for faglærere	1
Poul Bitsch Olsen	Problemorienteret projektarbejde	0.67
Kjeld Fredens	Liv og læring	0.67
Niels Åkerstrøm Andersen	Diskursive analysestrategier	0.67

Figure 5.18: Item-based Recommendations for the Book *Læring*.

The books in the recommendation list from the example in Figure 5.18 seem to be very similar to the book *Læring*, which is a good indication for high relevancy. One of the books is by the same author and all within the same topic.

5.4.2 Content-based Recommendation

The content-based recommendation implementation run on the same sample loan history results in a *patternsim* table with 271,621 rows, containing 28,804 distinct books meaning that 8,218 books do not have any mappings.

When the content-based recommendation implementation is run with 10 months of book transactions and only using the title and author name of books as item descriptions it can produce recommendations such as in Figure 5.19 for the book called Elements of Discrete Mathematics by Liu Chung Laung

Other unwanted recommendations such as those shown in Figure 5.20 can also be produced, and this is because of the algorithm only having the title and author name to compare the books with each other.

TOP-N	Recommended Book Title	Rating
1.	An Introduction to Discrete Mathematics by Steven Roman	2
2.	Implementing Discrete Mathematics by Steven Skiene	2
3.	Handbook of Discrete and Combinatorial Mathematics by Kenneth H. Rosen (ed.)	2
4.	Discrete Mathematics with Graph Theory by Goodaire E.G.	1

Figure 5.19: Example for the Book *Recommendations for Discrete Mathematics* by Liu Chung Laung.

TOP-N	Recommended Book Title	Rating
5.	Estimation, Control, and the Discrete Kalman filter by Catlin Donald E.	1
6.	Chaos in Discrete Dynamic Systems by Ralph H. Abraham	1
7.	A first Cource in Discrete Dynamical Systems by Richard A. Holmgren	1

Figure 5.20: Example for Book *Recommendations for Discrete Mathematics* by Liu Chung Laung.

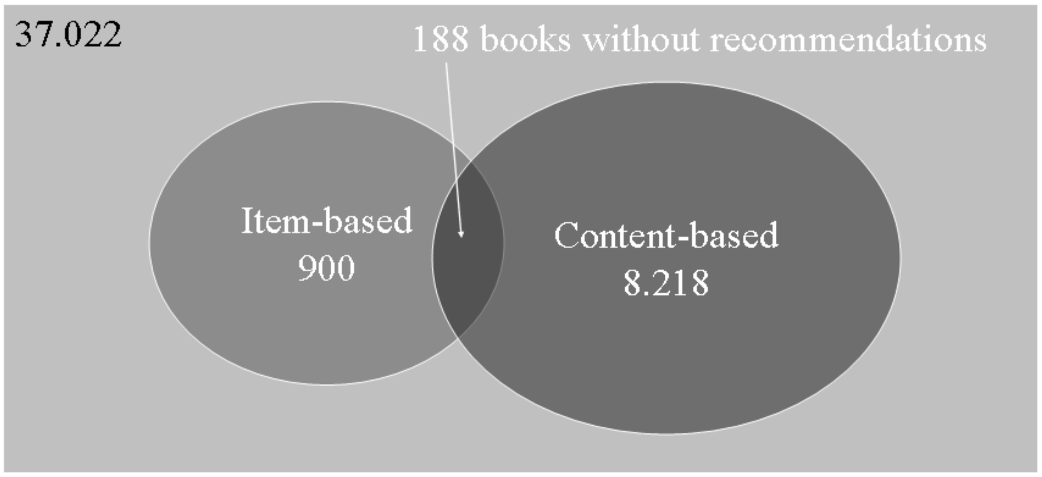


Figure 5.21: The Number of Books without Recommendations.

5.4.3 Improving the Quality

Combining the item-based CF and content-based recommendation leads to a recommendation service that only has 188 books that do not have any recommendations. These 188 books are not contained in either the *similarity* or *patternsim* table. The 188 books are the intersection between the books without recommendations in either item-based CF or content-based recommendation. This is illustrated in figure 5.21 where the 8.218 books without recommendations from content-based recommendation and 900 books without recommendations from item-based CF are represented along with the intersection consisting of 188 books.

The number of books in the *similarity* table can be increased by having more loan history for the similarity computation. This would also decrease the large amount of similarity values being 0.

The examples for content-based recommendation clearly illustrates that title and author name are very little text in order to find recommendations. The quality of the content-based recommendations can be improved by adding more text to each book. The extra text could, e.g., be keywords. This would most likely lead to a recommendation with higher quality and recommendations for more books.

With more loan history and keywords for the books the recommendation service should be able to do recommendations for all the books at AUB. This would most likely eliminate the last 188 books without recommendations.

5.5 Related work

In this section we will describe how the collaborative filtering can be done using the user-based method. We will also look at an article written by the employees at Amazon which describes their implementation of the item-based method.

5.5.1 User-based Collaborative Filtering

The goal of user-based CF [10] is the same as the goal of item-based CF; namely to suggest new books or to predict the utility of a certain book for a particular user based on the user's previous likings and the opinions of other like-minded users.

The task of the CF algorithm is to find an item's likeliness that can be of two forms as in item-based CF, namely *prediction* and *recommendation*.

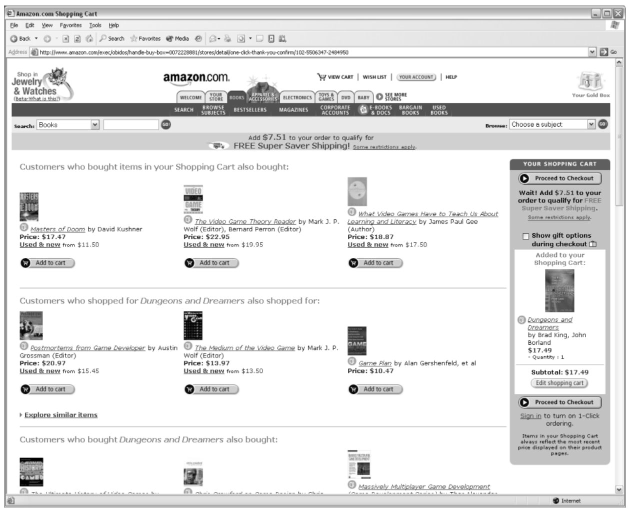
When comparing the user-based CF with the item-based CF, the main differences are that user-based CF has no model that is built offline and that user-based CF focuses on user-similarities rather than item-similarities.

The algorithms for user-based CF are called *memory-based*, because they make use of the entire user-item database in order to generate a prediction. The algorithms make use of techniques such as a nearest-neighbor algorithm in order to find a set of users that have historically similar preferences, which is also known as neighbors. The correlations could, e.g., consist of implicit or explicit ratings.

User-based CF has been used very much in the past [10]. However, it suffers from two major challenges. The first challenge is the sparsity, which is the problem of having only a small amount of ratings from users and a lot of items that have not been rated by any users. The second challenge is the scalability of the system because the nearest-neighbor algorithms require computation time and space that grows with the number of items and number of users.

5.5.2 Amazon's Collaborative Filtering System

Amazon [19] makes use of collaborative filtering system, which makes it possible to make recommendations for a customer and letting them rate books. Amazon uses a recommendation algorithm in order to personalize their online store for each customer. The store shows information based on the customer's previous purchases and the items in the shopping cart. We encourage the


Amazon.com Shopping Cart
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://www.amazon.com/exec/obidos/handle-buy-box=0072228881/stores/detail/one-click-thank-you-confirm/102-5506347-2484950
Go
Shop in Jewelry & Watches (Beta-What is this?)
amazon.com.
VIEW CART | WISH LIST | YOUR ACCOUNT | HELP
Your Gold Box
WELCOME YOUR STORE BOOKS APPAREL & ACCESSORIES ELECTRONICS TOYS & GAMES DVD BABY SEE MORE STORES
SEARCH BROWSE SUBJECTS BESTSELLERS MAGAZINES CORPORATE ACCOUNTS E-BOOKS & DOCS BARGAIN BOOKS USED BOOKS
Search: Books GO!
Browse: Choose a subject GO!
Add to cart
Proceed to Checkout
Edit shopping cart
Internet

Add $7.51 to your order to qualify for FREE Super Saver Shipping! Some restrictions apply.

Customers who bought items in your Shopping Cart also bought:

Masters of Doom by David Kushner
Price: $17.47
Used & new from $11.50

The Video Game Theory Reader by Mark J. P. Wolf (Editor), Bernard Perron (Editor)
Price: $22.95
Used & new from $19.95

What Video Games Have to Teach Us About Learning and Literacy by James Paul Gee (Author)
Price: $18.87
Used & new from $17.50

Customers who shopped for *Dungeons and Dreamers* also shopped for:

Postmortems from Game Developer by Austin Grossman (Editor)
Price: $20.97
Used & new from $15.45

The Medium of the Video Game by Mark J. P. Wolf (Editor)
Price: $13.97
Used & new from $13.50

Game Plan by Alan Gershenfeld, et al
Price: $10.47

Explore similar items

Customers who bought *Dungeons and Dreamers* also bought:

Massively Multiplayer Game Development

YOUR SHOPPING CART

Wait! Add $7.51 to your order to qualify for FREE Super Saver Shipping. Some restrictions apply.

Show gift options during checkout

Added to your Shopping Cart:

Dungeons and Dreamers by Brad King, John Borland
$17.49
· Quantity : 1

Subtotal: $17.49

Sign in to turn on 1-Click ordering.

Items in your Shopping Cart always reflect the most recent price displayed on their product pages.

Figure 5.22: The Shopping Cart in the Amazon.com Website.

reader to study the article written by Amazon [19] which describes how their algorithm is designed.

Amazon has developed their web site so that when a customer searches for, e.g., books they will get a recommendation for other books that have historically similar preferences. Each customer on the web site has a ”Your Recommendations” feature where the customer can sort recommendations and add new product ratings.

Furthermore, when a customer has items in the shopping cart, the system will add shopping cart recommendations based on the customer's cart and previous customer's alike shopping carts as shown in Figure 5.22. Amazon has about 29 million customers and a couple of million items. This means that they have to run the computation of the similar-items table offline. The lookup of the similar items for the customer's purchases and ratings can be run online. However, it is also very important that the system is able to scale in order to be let the customers be able to retrieve recommendations.

We will in the following section compare Amazon and AUB's algorithms.

Amazon versus AUB

When we compare the Amazon algorithm with the AUB algorithm we found a number of differences.

Implicit Ratings The main difference between the CF systems is that Amazon makes use of user ratings based on both the bought items and what the customer has rated for the items. In AUB's CF system it is not possible to rate for a particular book and get references based on the particular borrower's ratings.

In order to do a more precise recommendation for the customer, Amazon has developed a weighting formula. The formula calculates a weight based on indicia of the customer's affinity for, or current interest in, the corresponding items of known interest.

AUB cannot make full use of the weighting formula, because the formula uses the borrower's explicit ratings for a given item and the purchase date of the item, if the values exist. It could be possible for AUB to make partly use of the formula by only giving the borrower's implicit ratings, loan date and how many borrowers have borrowed it in the last couple of weeks. Though this would decrease the usability of the formula.

The recommendations in the Amazon CF system are generated using the following:

- A table mapping items to a list of similar items.
- The items which the customer has recently purchased.
- The customer's purchase histories.
- The content-based similarities extracted from item descriptions.

Shopping Cart Amazon makes use of a shopping cart function that stores what the customer has currently selected and historical data about the customer's previous purchases. Based on this, Amazon's CF system finds similar items for the particular customer. In AUB's current system we can only show similar items based on a selected item. We do not have the possibility to base the similar items on the current borrower's historical data, because the borrower is not known to the system in order to uphold the Danish privacy law.

A useful feature of Amazon's CF system could be the possibility of having multiple shopping carts. In AUB's CF system a borrower could have an option that, e.g., specifies that only children books are shown on the recommendation list.

It could also be a possibility to define different search queries in AUB's system and then base the recommendations on the items found using the search queries.

It could be usable for AUB to base the recommendations on a borrower's interest. However, the process of identifying popular items can be used and is used in AUB's CF system. Although Amazon's formula for calculating the similarity between items is not used, instead the adjusted cosine similarity formula has been used.

Explicit Ratings Figure 5.5 on page 98 shows how AUB's CF system is designed as in Chapter 2. The main difference between Amazon's and AUB's CF system is that in AUB's CF system you do not have any BookMatcher service which uses the users explicit ratings. The BookMatcher is described in detail in U.S. Appl. No. 09/040,171 filed March 17, 1998. The service allows users to interactively rate individual books on a scale of 1-5 to create personal item ratings profiles, and applies collaborative filtering techniques to these profiles to generate personal recommendations. The ratings data collected by the BookMatcher service and similar services is incorporated into the recommendation process of Amazon's invention.

Data Removal In AUB's CF system a borrower is never removed from the similar items table, meaning that the calculations over time will be based on both old and new data. Furthermore, the size of the table will increase because of new borrowers being added to the table.

Regarding Amazon's CF system it is assumed that when a customer is deleted in the main system, the customer is also removed from the CF system and not used in the calculations for building the similar items table anymore. This is done in order to save space in the database and computation time when building the similar items table. Unfortunately, AUB does not have as much data as Amazon, therefore the removal of data may not be usable for AUB's CF system.

5.6 Future Work

As listed in the Section 5.4, *keywords* must be included in the content-based recommendation for each book. This way the content-based recommendation is based on more text which leads to more recommendations and recommendations of higher quality. Without the keywords the basis for content-based recommendation is too small.

One could question the quality of the present way to present recommendations, as it implies that the quality of the item-based recommendations always are superior to the content-based recommendations. Another approach is to group the recommendation top list such that books with 90% word-similarities should appear at the top. Item-based recommendations surpassing a minimum similarity value should appear next followed by books having between 70% and 90% word-similarity. Word-similarity is expressed as the number of attributes of the recommended book matching the one in question:

$$word\_similarity = \frac{recommended\_book\_attributes}{book\_attributes} \tag{5.9}$$

The three threshold values could be determined by imperical runs on a large set of loan history. A third approach is to convert the word-similarity percentages into similarity values like those used in the item-based approach. It would then be possible to do a merge of book recommendations between both approaches.

A full evaluation should also be done by trying to compare the item-based CF approach with the content-based recommendation approach in order to find out what the quality of each approach is. Furthermore, it would be interesting to find out if the recommendations are the same in both approaches and to see what the quality of the recommendations is. One question to raise is whether the recommendations actually are usable for the users of the library. It is also important to involve the librarians and borrowers in the evaluation. The librarians can explain whether the recommendations seems reasonable while the borrowers can tell whether the recommendations actually helped.

The implementation of the item-based recommendation approach holds all the data in memory, which poses a limitation. Once the loan history data surpasses the size of the available physical memory, swapping will be utilized which significantly slows down the performance. Once this threshold is reached, it would be advisable to take a DBMS approach like our first implementation of the item-based recommendation [33]. Instead of multiple queries to the DBMS for each similarity calculation, it might be possible to merge these set of queries into one or more large queries. This way we can significantly reduce the number of query planner invocations resulting in a much more efficient execution of the queries.

The implementation of the content-based recommendation approach can be optimized further by using a HashSet instead of a HashMap in order to fetch bookIDs for a given attribute in the pattern. When using a HashSet instead of a HashMap the amount of memory used is decreased because the HashSet can be used as an iterator to retrieve the values in the HashSet. In a HashMap it is not possible to iterate through each HashKey instead each HashKey much have an associated object which should be the bookid Integer. It is then possible to iterate through the values of every HashKey in the HashMap, however memory is used for storing the bookIDs both as HashKeys and Integers.

Another type of improvement to add is recommendation using the ”shopping cart”, which is implemented in AUB's web site. The ”shopping cart” is used for adding books that a borrower might want to reserve later on. The books in the ”shopping cart” can be used for making a recommendation which is based on a collection of books. This is simply done by fetching all the similarity values for all the books, sort the values in descending order, and list the top-N books. Word-similarity values from the content-based approach need to be converted into similarity values like those used in the item-based approach as mentioned earlier.

6 The Data Access Tools

The librarian service can be used for retrieving information about the data warehouse and association rules. Furthermore, it is possible to try out the book recommendations. The librarian service is available to the librarians.

The borrowers will use the recommendation service, which should be implemented in the Auboline system. The recommendation service makes it possible for the borrowers to see recommendations inside the Auboline system.

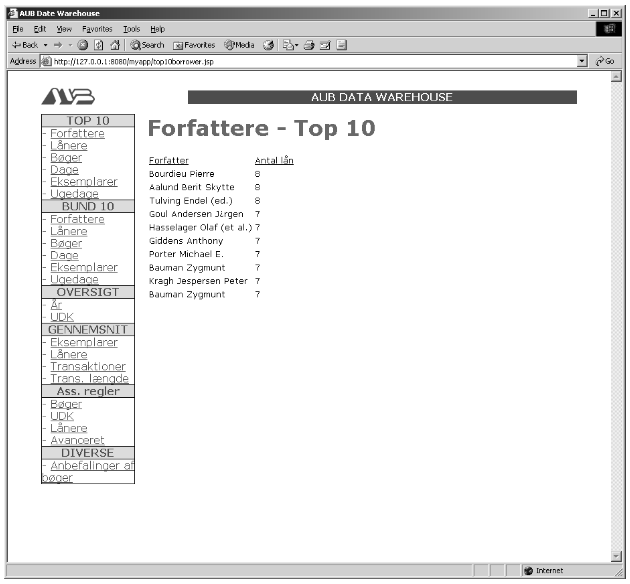
We will now go through the following three parts:

- Statistics (the librarian service)
- Association Rules (the librarian service)
- Recommendations (the recommendation service)

6.1 The Librarian Service

The statistics for the data is divided into four groups, which the following four sections elaborate on. The first part regards extracting statistics about the data.

Top-10 presents the most popular books, most active days, most popular authors and the most active borrowers.



Forfattere - Top 10

Forfatter	Antal lån
Bourdieu Pierre	8
Aalund Berit Skytte	8
Tulving Endel (ed.)	8
Goul Andersen J¿rgen	7
Hasselager Olaf (et al.)	7
Giddens Anthony	7
Porter Michael E.	7
Bauman Zygmunt	7
Kragh Jespersen Peter	7
Bauman Zygmunt	7

Figure 6.1: The Most Popular Authors.

Top-10

Forfattere This page shows the most popular authors. See Figure 6.1.

Lånere This page shows the borrowers that borrow the largest number of books.

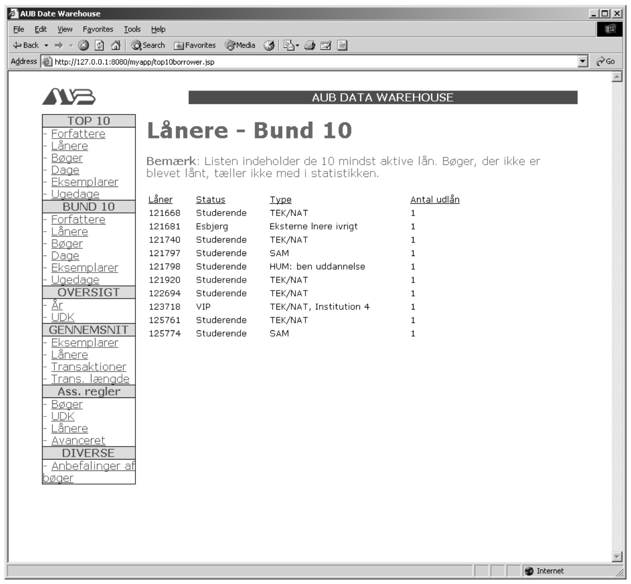
Bøger This page shows which books are borrowed the most.

Dage This page shows the 10 days where the largest number of books have been borrowed.

Eksemplarer This page shows the editions that have been borrowed the most.

Ugedage This page shows which weekdays the books are being borrowed the most.

Bund-10 shows an inverse view of the Top-10 pages. It is important here to mention that the Bund-10 statistics are based upon the books that have been borrowed. This is because of the limited amount of data we have got from AUB and because the data was based on loans. In order to get correct statistics we need to have all the book data from AUB thereby including books that have not been borrowed at all. See the Appendix A for more information about the data sample from AUB.



Lånere - Bund 10

Bemærk: Listen indeholder de 10 mindst aktive lån. Bøger, der ikke er blevet lånt, tæller ikke med i statistikken.

Låner	Status	Type	Antal udlån
121668	Studerende	TEK/NAT	1
121681	Esbjerg	Eksterne lnere ivrigt	1
121740	Studerende	TEK/NAT	1
121797	Studerende	SAM	1
121798	Studerende	HUM: ben uddannelse	1
121920	Studerende	TEK/NAT	1
122694	Studerende	TEK/NAT	1
123718	VIP	TEK/NAT, Institution 4	1
125761	Studerende	TEK/NAT	1
125774	Studerende	SAM	1

Figure 6.2: The Borrowers Borrowing the Smallest Number of Books.

Bund-10

Forfattere This page shows which authors are least borrowed.

Lånere This page shows which borrowers borrow the least number of books a year etc. See Figure 6.2.

Bøger This page shows which books the borrowers borrow the least number of times.

Dage This page shows the 10 days where the least number of books have been borrowed.

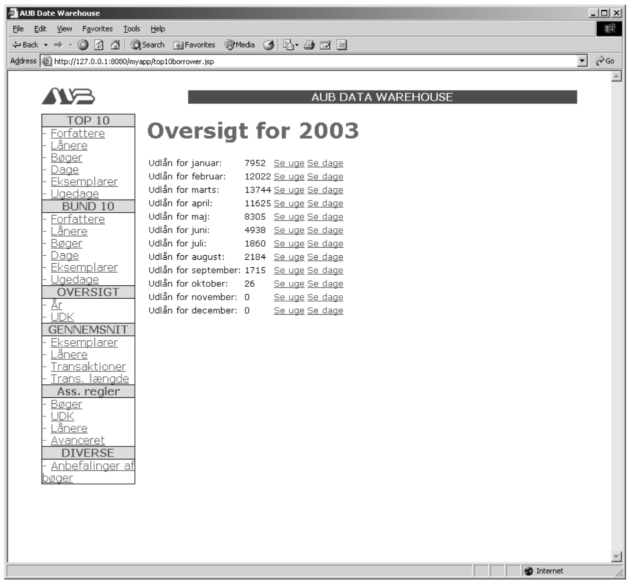
Eksemplarer This page shows the editions that have been borrowed the least.

Ugedage This page shows which weekdays the books are being borrowed the least.

The *Lån Oversigt* pages show the number of loans on each day, week, month, and year. The overview pages also show the number of loans per UDK level. It is possible to drill up and down on UDK number level-wise.

Lån Oversigt

År It is possible to drill down to a specific month, say January, and see the total number of loans that month, see Figure 6.3. The



Oversigt for 2003

Udlån for januar:	7952	Se uge Se dage
Udlån for februar:	12022	Se uge Se dage
Udlån for marts:	13744	Se uge Se dage
Udlån for april:	11625	Se uge Se dage
Udlån for maj:	8305	Se uge Se dage
Udlån for juni:	4938	Se uge Se dage
Udlån for juli:	1860	Se uge Se dage
Udlån for august:	2184	Se uge Se dage
Udlån for september:	1715	Se uge Se dage
Udlån for oktober:	26	Se uge Se dage
Udlån for november:	0	Se uge Se dage
Udlån for december:	0	Se uge Se dage

Figure 6.3: An Overview of Loan by Months.

total number of loans per week in the current month and the total number of loans for each day in the current month can be seen by further drilling down.

UDK Level 1 This overview makes it possible to see the number of borrowers per UDK on the top level. A UDK consists of several levels and it is possible in the GUI to drill down to level 2, explore the sub-topics on this level, and further drill down to level 3 for a given sub-topic. You can see an example of a UDK overview in Figure 6.4.

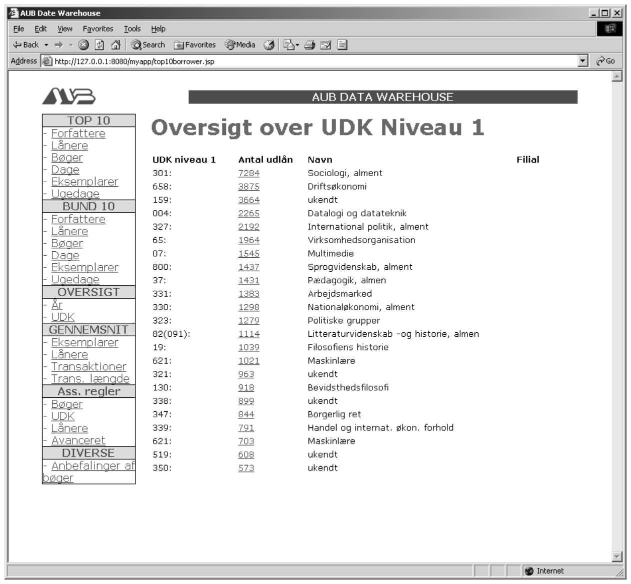
The final statistics shows the following:

Gennemsnit

Eksemplarer This page shows the average number of editions per month, per week, and per day.

Lånere This page shows the number of borrowers per month, per week, and per day, see Figure 6.5.

Transaktioner This page makes it possible to retrieve information about how many transactions in average have been registered per month, per week, and per day.



Oversigt over UDK Niveau 1

UDK niveau 1	Antal udlån	Navn	Filial
301:	7284	Sociologi, alment	
658:	3875	Driftsøkonomi	
159:	3664	ukendt	
004:	2265	Datalogi og datateknik	
327:	2192	International politik, alment	
65:	1964	Virksomhedsorganisation	
07:	1545	Multimedie	
800:	1437	Sprogvidenskab, alment	
37:	1431	Pædagogik, almen	
331:	1383	Arbejdsmarked	
330:	1298	Nationaløkonomi, alment	
323:	1279	Politiske grupper	
82(091):	1114	Litteraturvidenskab -og historie, almen	
19:	1039	Filosofiens historie	
621:	1021	Maskinlære	
321:	963	ukendt	
130:	918	Bevidsthedsfilosofi	
338:	899	ukendt	
347:	844	Borgerlig ret	
339:	791	Handel og internat. økon. forhold	
621:	703	Maskinlære	
519:	608	ukendt	
350:	573	ukendt	

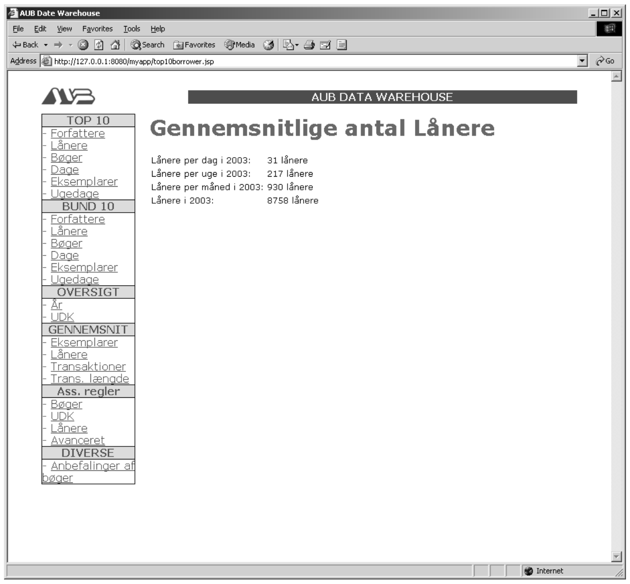
Figure 6.4: An Overview of Loan by UDK.

Transaktionslængde This page shows information about the average transaction length by calculating the number of books per number of transactions a year.

We have made aggregate tables in order to speed up the displaying of the statistics. The following web pages make use of aggregate tables:

- Overview for Loan Days
- Overview for Loan Months
- Overview for Loan Weeks
- Overview for Loan Year
- Overview for UDK Level 1
- Overview for UDK Level 2
- Overview for UDK Level 3

The second part of the librarian service displays association rules that the system has generated based on the current data from the data warehouse, see the Appendix A. LIQ Hybrid runs offline in order to compute the association



Gennemsnitlige antal Lånere

```
Lånere per dag i 2003:     31 lånere
Lånere per uge i 2003:     217 lånere
Lånere per måned i 2003: 930 lånere
Lånere i 2003:             8758 lånere
```

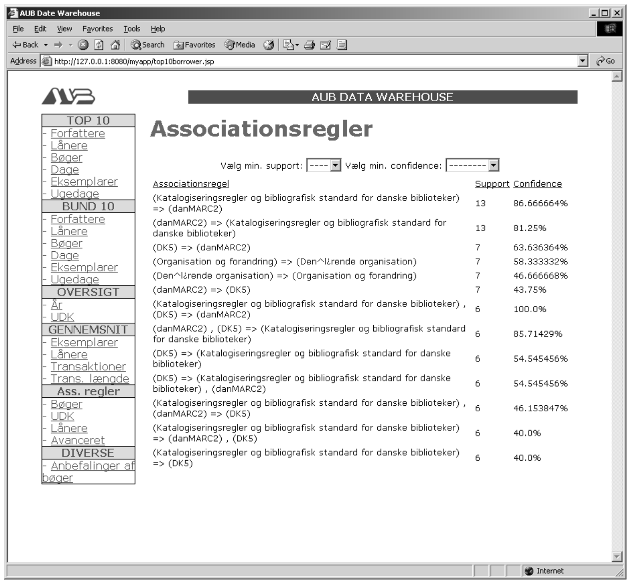
Figure 6.5: The Average Number of Borrowers.

rules and save the results in a table. When the librarian chooses to see the association rules the corresponding table is queried and the results are shown on the web site.

It is also possible to display association rules using constraints and generalizations. As an example the user can choose an UDK level or a StatusType in order to display association rules within an UDK level or an StatusType. An advanced edition of the association rules can also be run through the web interface. The advanced edition makes it possible to choose both an UDK and StatusType and to run the association rule mining algorithm online.

The illustration on Figure 6.6 shows an example of retrieved association rules from the librarian service. The business user of the librarian service can choose to set the minimum support and confidence values on the JSP page in order to get the corresponding association rules.

We have implemented a privacy constraint on the association rules when the rules are displayed on the web page. This means that we make use of end-suppression in order to remove association rules for books that have not got enough borrowers. The reason for this is that if an association rule has too few borrowers, it is possible for an user of the web site to find out who the borrowers are, and this must not be possible according to the Danish law of privacy.



Associationsregler

Vælg min. support: ---- Vælg min. confidence: --------

Associationsregel	Support	Confidence
(Katalogiseringsregler og bibliografisk standard for danske biblioteker) => (danMARC2)	13	86.666664%
(danMARC2) => (Katalogiseringsregler og bibliografisk standard for danske biblioteker)	13	81.25%
(DK5) => (danMARC2)	7	63.636364%
(Organisation og forandring) => (Den^l¿rende organisation)	7	58.333332%
(Den^l¿rende organisation) => (Organisation og forandring)	7	46.666668%
(danMARC2) => (DK5)	7	43.75%
(Katalogiseringsregler og bibliografisk standard for danske biblioteker) , (DK5) => (danMARC2)	6	100.0%
(danMARC2) , (DK5) => (Katalogiseringsregler og bibliografisk standard for danske biblioteker)	6	85.71429%
(DK5) => (Katalogiseringsregler og bibliografisk standard for danske biblioteker)	6	54.545456%
(DK5) => (Katalogiseringsregler og bibliografisk standard for danske biblioteker) , (danMARC2)	6	54.545456%
(Katalogiseringsregler og bibliografisk standard for danske biblioteker) , (danMARC2) => (DK5)	6	46.153847%
(Katalogiseringsregler og bibliografisk standard for danske biblioteker) => (danMARC2) , (DK5)	6	40.0%
(Katalogiseringsregler og bibliografisk standard for danske biblioteker) => (DK5)	6	40.0%

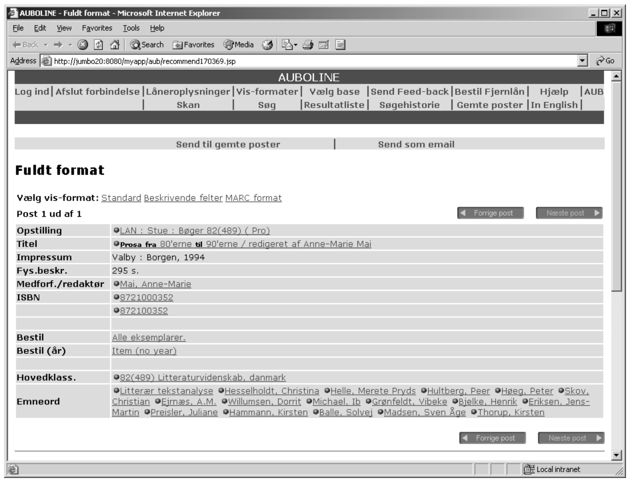
Figure 6.6: Association Rules.

6.2 Recommendation Service

The recommendation part is available through the recommendation service. It enables the borrower to see the results from the recommendation implementation. A borrower can choose any book available in the Auboline system and see what recommendations are available for the book.

The recommendation service must be implemented in Auboline's system so that it is possible to see recommendations about a book when choosing a book in Auboline. On Figure 6.7 it is shown how Auboline's system looks like and Figure 6.8 shows how the recommendation service could be implemented inside Auboline in order to show the recommendations. See the Appendix A for details on the data sample from AUB.

It is possible to implement the recommendation service inside Auboline because the recommendation service makes the recommendation data available to Auboline using XML. See the Appendix D for an example of the XML data.


AUBOLINE - Fuldt format - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://jumbo20:8080/myapp/aub/recommend170369.jsp
Go
AUBOLINE
Log ind | Afslut forbindelse | Låneroplysninger | Vis-formater | Vælg base | Send Feed-back | Bestil Fjernlån | Hjælp | AUB
Skan | Søg | Resultatliste | Søgehistorie | Gemte poster | In English
Send til gemte poster | Send som email
Forrige post
Næste post
Local intranet

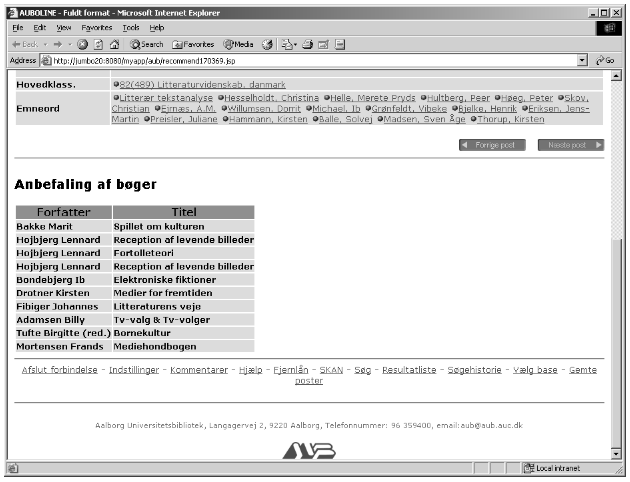
Fuldt format

Vælg vis-format: Standard Beskrivende felter MARC format

Post 1 ud af 1

Opstilling	●LAN : Stue : Bøger 82(489) (Pro)
Titel	●Prosa fra 80'erne til 90'erne / redigeret af Anne-Marie Mai
Impressum	Valby : Borgen, 1994
Fys.beskr.	295 s.
Medforf./redaktør	●Mai, Anne-Marie
ISBN	●8721000352
	●872100352
Bestil	Alle eksemplarer.
Bestil (år)	Item (no year)
Hovedklass.	●82(489) Litteraturvidenskab, danmark
Emneord	●Litterær tekstanalyse ●Hesselholdt, Christina ●Helle, Merete Pryds ●Hultberg, Peer ●Høeg, Peter ●Skov, Christian ●Ejrnæs, A.M. ●Willumsen, Dorrit ●Michael, Ib ●Grønfeldt, Vibeke ●Bjelke, Henrik ●Eriksen, Jens-Martin ●Preisler, Juliane ●Hammann, Kirsten ●Balle, Solvej ●Madsen, Sven Åge ●Thorup, Kirsten

Figure 6.7: Auboline's Web page for a Book.


AUBOLINE - Fuldt format - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://jumbo20:8080/myapp/aub/recommend170369.jsp
Go
Forrige post
Næste post
Afslut forbindelse - Indstillinger - Kommentarer - Hjælp - Fjernlån - SKAN - Søg - Resultatliste - Søgehistorie - Vælg base - Gemte poster
Local intranet

Hovedklass.	●82(489) Litteraturvidenskab, danmark
Emneord	●Litterær tekstanalyse ●Hesselholdt, Christina ●Helle, Merete Pryds ●Hultberg, Peer ●Høeg, Peter ●Skov, Christian ●Ejrnæs, A.M. ●Willumsen, Dorrit ●Michael, Ib ●Grønfeldt, Vibeke ●Bjelke, Henrik ●Eriksen, Jens-Martin ●Preisler, Juliane ●Hammann, Kirsten ●Balle, Solvej ●Madsen, Sven Åge ●Thorup, Kirsten

Anbefaling af bøger

Forfatter	Titel
Bakke Marit	Spillet om kulturen
Hojbjerg Lennard	Reception af levende billeder
Hojbjerg Lennard	Fortolleteori
Hojbjerg Lennard	Reception af levende billeder
Bondebjerg Ib	Elektroniske fiktioner
Drotner Kirsten	Medier for fremtiden
Fibiger Johannes	Litteraturens veje
Adamsen Billy	Tv-valg & Tv-volger
Tufte Birgitte (red.)	Bornekultur
Mortensen Frands	Mediehondbogen

Aalborg Universitetsbibliotek, Langagervej 2, 9220 Aalborg, Telefonnummer: 96 359400, email:aub@aub.auc.dk

Figure 6.8: Auboline with Book Recommendations.

6.3 Using the System

The data access tools are only prototypes for showing the possibilities available in the recommendation service and librarian service we have developed. It has been presented for AUB in order to discuss the features available in business intelligence and in order to find out what they would like to make use of.

The idea with the data access tools is that statistics, the association rules, and recommendations should be available for the librarians. The reason for this is that AUB can use this kind of data to increase their knowledge about their users. Furthermore, it is possible for AUB to, e.g., make book collections available for borrowing that are made from the association rules. These book collections could have different main subjects, e.g., fiction books, and the books could be chosen by finding the association rules that support these choices. The recommendations from the data access tool enable the librarians to check the quality.

The idea with the recommendation service is to let it be available to borrowers on AUB's web site. This makes it possible for borrowers to choose a book on the web site and see what recommendations there are for other books. The recommendations will be based on the community of users borrowing books.

6.4 Feedback

We have presented the data access tools to AUB and at a conference. We showed them how they can get statistics for the books, borrowers and authors available in the database, get overviews over the UDK-levels, and the number of loans per UDK. We also showed them how they could get overviews over the loans per day, week, month, and year. Furthermore, we presented the librarian service, which they liked very much.

At the conference it was said that a more serious problem with the system is when users are borrowing books that have no relevance with each other, or if the majority of the users are borrowing books of low professional quality. Another problem is the self-perpetuating effect that raises when users are only borrowing books that are being recommended by the item-based approach. It was said at the conference that the recommendations are a better alternative to the searching capabilities of the current system. Currently, many users cannot find books that are relevant to them and they often do not use more than one word when searching for books. With the recommendation system books that are linked to each other are shown to the user, this was considered to be more usable to the users.

We also presented the recommendation system both to AUB and to the conference. We showed them examples on books, e.g., *Organisation og forandring* and which other books that was borrowed together with the current book, e.g., *Den lærende organisation.* AUB and people at the conference were very interested in this and they could see the possibilities in using this feature in their existing web site. They therefore asked us to expand the system with a recommendation service that makes it possible to exchange data using XML, so that their web site can retrieve data from our system using XML. They also wanted us to setup a test-system, thereby making it possible for their librarians to test the data access tools.

Regarding personalized recommendations they expressed that it would probably be ages before it is possible to implement predictions in the system. This would make it possible to see recommendations for a book based on the current user's loan history. The reason for this is the Danish privacy law, that makes user monitoring illegal.

The last feedback we were given from them, were that all information about their current system must be held confidential. It is therefore not possible for us to include the table description in our appendix.

In all it looks like the library is very favorable for the system extension we have developed.

6.5 Future Work

In this section we will go through the possible improvements of the data access tools. Some of the improvements can, e.g., be choosing an OLAP application for analyzing the data, making better graphical overviews on the web site, and add help and support to the web site.

We will now show some examples of the improvements that can be made and also try to suggest what kind of inspections would have to be made when improving the system.

6.5.1 OLAP

Online analytical processing (OLAP), are being used by organizations to discover valuable business trends from data marts and data warehouses. OLAP provides an online statistical overview of data, which can, e.g., be a historical view of loans.

One could instead of using the librarian service find an already existing OLAP application in order to make the statistics part.

When finding an OLAP application there are a number of things we need to consider. Some of them are listed here:

Platform Choice The first step is to make a list of OLAP vendors that have software that are compatible with the platform that AUB is using. In this case it is UNIX. However, some vendors tend to be only Microsoft centric or have products that are better supported on a particular platform. So this must be analyzed first. We would also need to find out if there are any free and open source OLAP applications available.

Vendor History There are many new OLAP vendors that have popped up since the mid-nineties. Some of the newer companies have rewarding and rich OLAP tools that can be cost effective. It is a good idea to find out the reputation of a vendor.

Consulting It is wise to find out how much consulting would be required to install the product. Also, if changes are needed after installation, it is wise to find out if it will require the need of external consultants.

Integration to Database We should find out how tightly integrated the OLAP tool is to our database.

Price Finally, it is often challenging to gauge the additional hidden cost from the actual purchase price. Some of the points above can help make a good financial forecast. We need to consider what tool is the best given the number of features and the price for it.

The OLAP application has to be compatible with PostgreSQL because the system is using this DBMS.

Here is given a sample list of OLAP vendors: Oracle, Targit, Microsoft, SAS, IBM, Business Object and Hyperion. For more information see the web site[1].

[1] http://www.olapreport.com

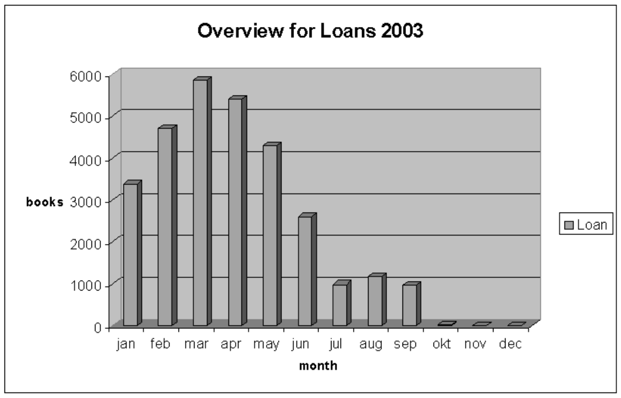


Figure 6.9: Example of Diagram for Loan Per Month Overview.

6.5.2 Graphical overviews

The librarian service can be expanded with the possibility of showing the statistics in different graphical overviews, such as diagrams.

In Figure 6.9 is shown an example of how the overview for loan per month could be made graphically and thereby give a much clearer view over what the data is meaning. The figure shows that we have a lot of loans in the first half year, however because we are missing data from the last half year, the last bars in the diagram are empty.

6.5.3 Help and Support

The librarian service could be improved by implementing help and support features. This would make it possible for the business users to find help within the web site and to search for guides, e.g., regarding how to use the association rules or to find out what support or confidence means.

The help and support could be used to guide the business users to use the web site. In order to develop a help system we could make a test of the system together with different business users in order to find out what information the business users want to have in the help system and also find out how the general user interface should look like.

6.5.4 Aggregate Navigator

Another optimization is directed at efficiently providing results based on SQL queries for aggregated data. If for instance we want to query for the number of books borrowed by computer science students in March 2003, it would be much faster to get a result from an aggregated table containing number of loans by month, year, and status as opposed to aggregating a large fact table containing millions of rows every time such a query was wanted. We all know that we need to "cheat" and calculate some values in advance, such as totals, and store them in the database in order to improve performance. What we suggest is to build aggregate tables that can be used with the most queries in the system. Every time business users send SQL queries there must be an intermediate application to parse them and find out whether one or more of the aggregate tables can be applied in order to increase performance. Such an intermediate application is known as an *Aggregate Navigator* [1]. The Aggregate Navigator rewrites the query if possible and uses the aggregated tables in the rewrited queries.

7 Conclusion

AUB wants to increase their level of service to their borrowers, which should be accomplished by extending their current system with the possibilities of recommending books. This should lead to more loans due to relevant recommendations of books. Furthermore, the librarians should also be aided in the form of a decision support system, which enables them to make more efficient business decisions. The bottom line is to cut costs, get more funds, and increase level of service.

We have built a BI system, which is based upon a data warehouse. The ETL processes have the responsibility of populating the data warehouse. Previously this exercise took 4-5 hours to complete for 10 months of loan data. It now only takes 2 minutes. The librarian and recommendation service use the data warehouse.

The librarian service is aimed towards the business users. Through a web site they are given access to association rules and statistics.

Regarding the association rule mining we have implemented Apriori, LIQ, and LIQ Hybrid in order to find out both the frequency and tendency of books being borrowed together. We have carried out a performance test on all three implementations resulting with LIQ Hybrid as the one with the lowest execution time. Afterwards, a scalability test has been carried out between LIQ Hybrid and FP-Growth. It seems that LIQ Hybrid is faster while we have less than 225,000 transactions or 3-4 years of loan history. Once we surpass this threshold, FP-Growth is superior to LIQ Hybrid. Beside being able to find associations between all books it is also possible to constrain the search on borrower status and type. Furthermore, association rules can be mined across UDKs.

The recommendation service is aimed towards borrowers and produces XML data that can be used by AUB's current system, Auboline. The XML data contains recommendation of books based on books which are similar to the one the borrower has selected. A content-based approach has been implemented in order to compensate the lack of recommendations from the item-based approach and improve the quality of the recommendations. The lack of recommendations in the item-based approach is due to cold-start problems. The performance for both implementations is also very satisfactory as each of them only take a couple of minutes to be executed offline for 10 months of loan data. The actual recommendations can then be done online. Compared with the previous implementation of the item-based approach, we have achieved a significant improvement on its offline execution time as it previously took 2 hours to complete the same amount of loan data.

The requirements stated in Section 2.6 has been fulfilled except from being able to synchronize seamlessly with the source system and providing online help. The synchronization process is not possible as we are not allowed to access the source data directly but will receive them in flat files instead. We have built a BI system that can run on AUB's existing platforms and hardware. Another requirement was to use open source applications in order to save license costs. This has been met as we use Tomcat as a web and application server. Furthermore, the data warehouse uses the PostgreSQL DBMS. Both applications are open source. Finally, the Danish law of privacy prohibits any information that can lead to an identification of a particular user. As we only have access to the borrower ID, which is encrypted, and no other personal information, we believe to have uphold the law. This has been done by suppressing the association rules from the data access web tool, which are below a certain threshold.

Future work has to be done in order to get a integrated BI system that AUB can use. The ETL processes have to extract data directly from the original source tables. The librarian service should be extended with graphical overviews or an existing OLAP tool should be integrated. Online help should also be available in order to ease the understanding of the librarian service.

AUB and other libraries see big possibilities in using the recommendation service and association rule mining. Therefore, they would like to have a full implementation of the system. In order to accomplish this the data should be extracted directly from the source system of the library into the data warehouse. Furthermore the recommendation service needs to be implemented in the web site service of the library.

AUB wants to test the librarian service by doing different acceptance tests in order to make sure it meets the librarians' needs. The full implementation will make it possible for the borrowers to receive recommendations from the Auboline system and for the business users to find useful association rules and statistics.

To conclude, it looks like AUB and other libraries are very favorable for the system extension that we have developed.

Appendices

A Sample Data

The development of the BI system for AUB was based on an extraction of data from the operational source system at AUB. The extraction caused several limitations because of the limited amount of data and access to data.

The extraction only consists of loan data from January to October 2003. There are a total of 64,371 loans in the extraction and it is only book loans meaning that loans of articles, audio CDs etc. have been excluded. The attributes that are covered by the extraction are listed in Figure A.1. Information regarding the loan data covered by the extraction is listed in Figure A.2. A transaction corresponds to loans grouped by the same user on the same day.

The limitations of the sample data are that we only get the books that are borrowed and the borrowers that borrow books. This fact makes it impossible for the business users to get a view of the books that have not been borrowed nor a view of the borrowers who have not borrowed any books. The number of attributes are also very limited. It could, e.g., have been interesting to have had a loan time in order to try different transaction definitions and see the different number of transactions. In addition, if key words describing the books were included the quality of the content-based recommendation could be improved. The extraction also limited the implementation of the ETL since the extraction could not be done from the base tables in the operational source system.

Attribute	Source Attribute	Description
A	Z36H-REC-KEY	An administrative system number consisting of a DOC-NUMBER and ITEM-SEQUENCE.
B	Z13-DOC-NUMBER	A foreign key referring to the Z13 table.
C	Z36H-ID	The ID of the borrower.
D	Z36-LOAN-DATE	The date of the loan.
E	Z36-RETURNED-DATE	The date the book was returned.
F	Z36H-BOR-STATUS	The status of the borrower.
G	Z36H-BOR-TYPE	The type of the borrower.
H	Z36H-NO-RENEWAL	Number of renewals.
I	Z36H-TIME	A timestamp of when this row was added.
J	Z30-ITEM-STATUS	The status of the material.
K	Z30-OPEN-DATE	The date of creation of the edition.
L	Z30-CALL-NO	The UDK for the edition.
M	Z13-AUTHOR	The author of the book.
N	Z13-TITLE	The title of the book.
O	Z13-YEAR	When the book was published.

Figure A.1: Attributes from Extraction.

Description	Number
Loans	64,371
Transactions	29,373
Books	37,022
Authors	28,392
Borrowers	8,758
Borrower Types	51
Borrower Status	10

Figure A.2: The Extraction Data Set.

B Data Warehouse

This appendix covers the definitions for tables used for the dimension tables, fact table, and the aggregate tables. Furthermore, indices have been created on attributes used for selection in the librarian service.

B.1 Dimensions and Fact Table Definitions

The following is the schemas for the dimensions and fact tables:

```
CREATE TABLE edition
(
  id int4,
  open_date int4,
  issue_date int4,
  editionnumber varchar(6),
  title   varchar(200),
  bookid int4,
  authorname varchar(100),
  CONSTRAINT edition_pkey PRIMARY KEY (id)
)
```

There is created an index in the *Edition* dimension table for the id attributes.

```
CREATE TABLE statustype
(
  id int4,
  status varchar(200),
  type varchar(200),
  statusname varchar(50),
  typename varchar(50)
)
```

There is created an index in the *Status Type* dimension table for ID attribute.

```
CREATE TABLE udk
(
  id int4,
  udklevel1 varchar(30),
  udklevel1name varchar(200),
  udklevel2 varchar(30),
  udklevel2name varchar(200),
  udklevel3 varchar(30),
  udklevel3name varchar(200),
  udkdepartment varchar(200)
)
```

There are created indices in the *UDK* dimension table for each of the following attributes: id, udklevel1, udklevel2, and udklevel3.

```
CREATE TABLE time
(
  id int4 NOT NULL,
  hour varchar(2),
  minute varchar(2),
  CONSTRAINT time_pkey PRIMARY KEY (id)
)
```

There is created an index for the id attribute in the *Time* dimension table.

```
CREATE TABLE date
(
  id int4 NOT NULL,
  year int4,
  month varchar(10),
  day int4,
  weekday varchar(10),
  quarter int4,
  daymonthyear varchar(10),
  daymonth varchar(5),
  monthyear varchar(7),
  yearquarter varchar(8),
  yearmonthday varchar(8),
  daynumber int4,
  weeknumber int4,
  monthnumber int4,
  schoolyear varchar(9),
  period varchar(100),
  weekdaynumber int4,
  CONSTRAINT date_pkey PRIMARY KEY (id)
)
```

There is created an index for the id attribute in the *Date* dimension table.

```
CREATE TABLE loan
(
  id int4 NOT NULL,
  transactionid int4,
  borrowerid int4,
  loandate int4,
  loantime int4,
  returndate int4,
  returntime int4,
  duedate int4,
  duetime int4,
  editionid int4,
  udkid int4,
  statustypeid int4,
  CONSTRAINT loan_pkey PRIMARY KEY (id)
)
```

There are indices on every attribute as they are used to join between the dimension tables.

B.2 Table Definitions for Aggregate Tables

The following are the schemas for the aggregate tables:

```
CREATE TABLE loanoverviewday
(
  year int4,
  monthnumber int4,
  day int4,
  weekday varchar(10),
  weeknumber int4,
  count int8
)
```

There are created composite indices in the *loanoverviewday* aggregate table for each of the following attributes: (year, monthnumber) and (year, weeknumber).

```
CREATE TABLE loanoverviewmonth
(
  year int4,
  month varchar(10),
  monthnumber int4,
  count int8
)
```

There is created an index in the *loanoverviewmonth* aggregate table for attribute year.

```
CREATE TABLE loanoverviewweek
(
  year int4,
  monthnumber int4,
  weeknumber int4,
  count int8
)
```

There is created a composite index in the *loanoverviewweek* aggregate table for the following attributes: (year, monthnumber).

```
CREATE TABLE loanoverviewyear
(
  year int4,
  count int8
)
```

There has been created no index in the *loanoverviewyear* aggregate table.

```
CREATE TABLE udk1overview
(
  count int8,
  udklevel1 varchar(30),
  udklevel1name varchar(200),
  udkdepartment varchar(200)
)
```

There are created no index in the *udk1overview* aggregate table.

```
CREATE TABLE udk2overview
(
  count int8,
  udklevel1 varchar(30),
  udklevel2 varchar(30),
  udklevel2name varchar(200),
  udkdepartment varchar(200)
)
```

There is created an index in the *udk2overview* aggregate table for attribute udklevel1.

```
CREATE TABLE udk3overview
(
  count int8,
  udklevel1 varchar(30),
  udklevel2 varchar(30),
  udklevel3 varchar(30),
  udklevel3name varchar(200),
  udkdepartment varchar(200)
)
```

There is created a composite index in the *udk3overview* aggregate table covering the following attributes: (udklevel1, udklevel2).

C Association Rules

The tables used for association rules mining in LIQ and Apriori are listed in the following.

C.1 Loan Table

The following query is used to create the table containing transactions and books. This table is used for mining association rules across books.

```
CREATE TABLE
    loanUnique
AS
    (
        SELECT
            loan.transactionID
            edition.bookID,
        FROM
            loan,
            edition
        WHERE
            loan.editionID = edition.ID
        GROUP BY
            loan.transactionID,
            edition.bookID
    )
```

C.2 UDK Table

The following query is used to create the table containing transactions and UDKs. This table is used for mining association rules across UDKs.

```
CREATE TABLE
    loanUnique_udk
AS
    (
        SELECT
            loan.transactionID
            loan.udkID,
        FROM
            loan
        GROUP BY
            loan.transactionID,
            loan.udkID
    )
```

C.3 StatusType Table

The following query is used to create the table containing transactions and status types. This table is used for mining association rules across the borrowers' status and type.

```
CREATE TABLE
    loanUnique_statustype
AS
    (
        SELECT
            loan.transactionID
            loan.statustypeID
        FROM
            loan
        GROUP BY
            loan.transactionID,
            loan.statustypeID
    )
```

D Recommendation

The following covers the tables that are used in the implementation of item-based CF and content-based recommendation. Lastly, there is listed an example of the XML output from the recommendation service.

The table used for fetching a borrower's number of loans of a book:

```
CREATE TABLE loanCount AS
(
    SELECT
        l.borrowerid AS borrowerid,
        e.bookid AS bookid,
        COUNT(e.id) AS count
    FROM
        loan AS l,
        edition  AS e
    WHERE
        l.editionID = e.id
    GROUP BY
        l.borrowerID,
        e.bookID
)
```

There are indices in the *loanCount* table for the following attributes: borrowerid and bookid.

The table used for finding all the borrowers who have borrowed a specific pair of books:

```
CREATE TABLE borrower2Books AS
(
   SELECT
      lc1.borrowerid AS borrowerid,
      lc1.bookid AS bookid,
      lc2.bookid AS bookid2
   FROM
      loanCount AS lc1,
      loanCount AS lc2
   WHERE
      lc1.borrowerid = lc2.borrowerid AND
      lc1.bookid != lc2.bookid AND
      lc1.bookid > lc2.bookid
)
```

There is created a composite index in the *borrower2Books* table for the following attributes: (bookid, bookid2).

The table used for storing the computed item-based similarity values:

```
CREATE TABLE similarity
(
   i  INTEGER,
   j  INTEGER,
   value  float8
)
```

There are created indices in the *similarity* table for the following attributes: i and j.

The table used for storing the computed content-based similarity values:

```
CREATE TABLE patternsim
(
  bookid1 int4,
  bookid2 int4,
  count int4
)
```

There is created an index in the *patternsim* table for attribute bookid1.

The recommendation service outputs XML that can be parsed at the Aubo-line web site at AUB. An example of the XML containing recommendations for the book *Læring* written by Knud Illeris is listed here:

```
<?xml version=”1.0” encoding=”utf−8” ?>
<Recommendation xmlns:xsd=”http://www.w3.org/2001/XMLSchema”
xmlns:xsi=”http://www.w3.org/2001/XMLSchema−instance”
xmlns=”http://www.aub.auc.dk/webservices/”>
<Book>
   <Author>Knud Illeris (red.)</Author>
   <Title>Tekster om ælring</Title>
</Book>
<Book>
   <Author>Jens Bjerg (red.)</Author>
   <Title>æPdagogik</Title>
</Book>
<Book>
   <Author>Helle øAlr (red.)</Author>
   <Title>Videoobservation</Title>
</Book>
<Book>
   <Author>Helle øAlr</Author>
   <Title>Personlig kommunikation og formidling</Title>
</Book>
<Book>
   <Author>Oluf Danielsen</Author>
   <Title>æLring og multimedier</Title>
</Book>
<Book>
   <Author>Erik Damberg</Author>
   <Title>æPdagogik og perspektiv</Title>
</Book>
<Book>
   <Author>Else Hiim</Author>
   <Title>æUndervisningsplanlgning for æfaglrere</Title>
</Book>
<Book>
   <Author>Poul Bitsch Olsen</Author>
   <Title>Problemorienteret projektarbejde</Title>
</Book>
<Book>
   <Author>Kjeld Fredens</Author>
   <Title>Liv og ælring</Title>
</Book>
<Book>
   <Author>NielsÅ økerstrm Andersen</Author>
   <Title>Diskursive analysestrategier</Title>
</Book>
</Recommendation>
```

E Bibliography

Bibliography

[1] The aggregate navigator.
http://www.fortunecity.com/skyscraper/oracle/699/orahtml/dbmsmag/9511d05.html, 1995.

[2] Crc32 - java api.
http://java.sun.com/j2se/1.4.2/docs/api/java/util/zip/CRC32.html, 2004.

[3] Hashmap - java api.
http://java.sun.com/j2se/1.4.2/docs/api/java/util/HashMap.html, 2004.

[4] Routine database maintenance tasks.
http://www.postgresql.org/docs/7.4/interactive/maintenance.html, 2004.

[5] J. Gehrke A. Evfimievski and R. Srikant. Limiting privacy breaches in privacy preserving data mining. 2003.

[6] R. Agrawal A. Evfimievski, R. Srikant and J. Gehrke. Privacy preserving mining of association rules. 2002.

[7] R. Agrawal and R. Srikant. Privacy-preserving data mining. 2000.

[8] Rakesh Agrawal and Ramakrishnan Srikant. Fast algorithms for mining association rules. 1994.

[9] Rakesh Agrawal and Ramakrishnan Srikant. Mining generalized association rules. 1995.

[10] Joseph Konstan Badrul, George Karypis and John Riedl. Item-based collaborative filtering recommendation algorithms. 2001.

[11] Prof. Michael Stonebraker U. C. Berkeley. *POSTGRES QUERy Language.* http://s2k-ftp.cs.berkeley.edu:8000/postgres/postgres.html, 1994.

[12] Prof. Michael Stonebraker U. C. Berkeley. *University POSTGRES 4.2.* http://s2k-ftp.cs.berkeley.edu:8000/postgres/postgres.html, 1994.

[13] Aaron Brazell. Introducing cron. http://www.sitepoint.com/article/1196/1, 2003.

[14] Laurentiu Cristofor. Arminer project. http://www.cs.umb.edu/ laur/ARMiner/, 2000.

[15] Apache Software Foundation. *The Jakarta Site - Apache Tomcat.* http://jakarta.apache.org/tomcat/, 01-11-03.

[16] Apache Software Foundation. *Jakarta Apache.* http://jakarta.apache.org/, 2003.

[17] The Apache Software Foundation. *Apache.* http://www.apache.org/licenses, 2003.

[18] Alan Gilchrist. *The case for taxonomies in Information Architecture.* http://www.cs.auc.dk/ jacobm/aub/Taxonomies.ppt, 09-03-2004.

[19] Brent Smith Greg Linden and Jeremy York. Amazon.com recommendations - item-to-item collaborative filtering. 2003.

[20] The PostgreSQL Global Development Group. *PostgreSQL 7.3.3 Documentation.* http://www.us.postgresql.org/postgresql-7.3.3/, 2003.

[21] Marty Hall. *Using Tomcat 4.* http://www.moreservlets.com/Using-Tomcat-4.html, 2003.

[22] Jian Pei Jiawei Han and Yiwen Yin. Mining frequent patterns without candidate generation. 2000.

[23] Al Borchers Jonathan L. Herlocker, Joseph A. Konstan and John Riedl. An algorithmic framework for performing collaborative filtering. 1999.

[24] David Heckermann Mehran Sahami, Susan Dumais and Eric Horvitz. A bayesian approach to filtering junk e-mail. 1998.

[25] Sun Microsystems. *Java Virtual Machine.* http://www.java.com, 2004.

[26] Bruce Momjian. *PostgreSQL FAQ*. http://www.postgresql.org/docs/faqs/FAQ.html, 04-11-2003.

[27] Raymond J. Mooney and Loriene Roy. Content-based book recommending using learning for text categorization. 2000.

[28] Raymond J. Mooney Prem Melville and Ramadass Nagarajan. Content-boosted collaborative filtering. 2001.

[29] Margy Ross Ralph Kimball. *The Complete Guide to Dimensional Modeling*. Wiley Computer Publishing, 2002. ISBN: 0-471-20024-7.

[30] Rakesh Agrawal Ramakrishnan Srikant. Mining sequential patterns. 1995.

[31] Pierangela Samarati and Latanya Sweeney. Protecting privacy when disclosing information: k-anonymity and its enforcement through generalization and supression. 1998.

[32] Sattler Kai-Uwe Shang Xuequn and Geist Ingolf. Sql based frequent pattern mining with fp-growth. 2004.

[33] Kristian Skouboe Reesen Trien Huy Ly, Jacob Mogensen. *Building a Business Intelligent System for AUB*. 18-12-2003.