WEB - BASED AUCTION SYSTEM

GRADUATE PROJECT

BY

MARIA BONDARENKO

MAY 2001

COMMITTEE MEMBERS

MICHELLE MOORE,
Chairperson

NANCY LEE CAMERON,
Member

DULAL CHANDRA KAR,
Member
Web-Based Auction System

By

Maria Aleksandrovna Bondarenko, M.A.

Graduate Project

Presented to the Faculty of the Graduate School
of the Texas A&M University – Corpus Christi
in Partial Fulfillment
of the Requirements
For the Degree of

Master of Science

Texas A&M University – Corpus Christi
May 2001
This graduate project is dedicated to my mother, for all her love.
Acknowledgements

My sincere thanks to Dr. Michelle Moore for her assistance and constructive recommendations and support, Dr. Dulal Chandra Kar for his advice and his instructions, and Dr. Nancy Lee Cameron for her suggestions and her reading.

Also the ideas and help of Thomas Riedmueller and Kurt Miller from Dell Computer Corporation in Austin, TX are greatly appreciated. Also I owe a debt of thanks to Roy Spencer from IBM Corporation in Austin, TX for assistance in technical writing and proof reading. Special thanks go to Alexander Medvedev for his inspiration, patience, and support during this research project.

It is also very important that I acknowledge all my friends who thoroughly tested the application and made valuable comments and useful recommendations.

May 2001
Abstract

This graduate project is the design and implementation of a Web-based auction system for selling textbooks. The system can be used on any campus to help lower the cost of books for students by using supply and demand to establish the selling prices.

This application allows students to register themselves (each establish a user name and password) and submit entries for the books they would like to offer for sale or would like to buy. The system provides an option to view all books offered for sale by other users and to submit bids to buy these or other textbooks. It allows a user to check the books that are in demand and to submit bids to sell these or other textbooks. Furthermore, a student can request a search for any specific book to check if it has been offered for sale or if somebody wants to buy it. The system maintains a database of information about textbooks, submitted bids, and completed trades. In addition the system calculates the auction statistics.
Table of Contents

Acknowledgements ........................................................................................................ v

Abstract ........................................................................................................................ vi

List of Tables .................................................................................................................. viii

List of Figures ................................................................................................................ ix

Chapter 1: Introduction and Background ....................................................................... 1

Chapter 2: The Web-Based Auction System ................................................................. 4
  2.1 Terms and Definitions ........................................................................................... 4
  2.2 Operational Features ........................................................................................... 7

Chapter 3: System Design ............................................................................................. 28
  3.1 Environment ........................................................................................................ 28
  3.2 Database Structure ............................................................................................. 30
  3.3 Procedural Design ............................................................................................... 36

Chapter 4: Results .......................................................................................................... 52

Bibliography .................................................................................................................. 54

Appendix A: Code Listing in Alphabetical Order ......................................................... 55
List of Tables

Table 3.1   Layers of Code in an Oracle Application 30
Table 3.2   Table Structure with Attributes 34
List of Figures

Figure 2.1  Welcome Screen ................................................................. 7
Figure 2.2  User Interface Diagram ...................................................... 8
Figure 2.3  Auction Policies ............................................................... 9
Figure 2.4  New User Registration Form .............................................. 11
Figure 2.5  Login Screen ................................................................. 12
Figure 2.6  Main Homepage .............................................................. 13
Figure 2.7  Search Form ................................................................. 14
Figure 2.8  Bid Submission Form ....................................................... 15
Figure 2.9  Textbook Information Screen ......................................... 17
Figure 2.10 Textbooks in Demand ..................................................... 19
Figure 2.11 Textbooks for Sale ......................................................... 20
Figure 2.12 Check Bids ................................................................. 21
Figure 2.13 Update Bid Form ........................................................... 22
Figure 2.14 Delete Bid Form ............................................................ 23
Figure 2.15 Check Trades ............................................................... 24
Figure 2.16 Auction Statistics ......................................................... 25
Figure 2.17 Account Information ..................................................... 26
Figure 2.18 Logout Screen .............................................................. 27
Figure 3.3  Relational Schema .......................................................... 35
Figure 3.4  Procedural Structure ...................................................... 37
Chapter 1: Introduction and Background

Online auctions have become a very dynamic area of activity lately. There are thousands of auction sites on the Internet offering computers, books, CDs, antiques, collectibles, toys, etc. The most prominent auction sites are ebay.com and Onsale.com. Many believe that online auctions will become more popular than posted-price transactions in the future, and this belief has led to very active research in the field of online auctions.

Automated auctions have been determining the real-time price of commodities in the NASDAQ financial market for more than twenty years, and during this time technology has been evolving tremendously. Several factors have contributed to the rise of even more online auctions, such as the development of secure transactions, the spread of Web browsers and point-and-click technology, the growth of the Internet population, and the advantages of doing business online.

An auction is an economic mechanism for determining the price of an item. It requires a determined methodology, an item for sale, bidders who would like to buy the item, and the sale of the item to the highest bidder. Many researches conclude that the online auction has great potential as an automated price-negotiation tool that can be used for virtually any kind of goods and services. Both Web and Internet technology are relatively new, thus the online auction companies are new also. Only about half of those companies providing online auctions have been in business for over four years (C. Beam, A. Segev, 1998).
There are different traditional types of auctions depending on the number and frequency of market-clearing events, restrictions on bidding, disclosure of intermediate information, and the policies for determining prices and matching buyers with sellers. However, the online auctions do not always follow the traditional types, mostly because of the newly enabled experimentation with auction rules. The most popular auction is the English auction. The reasons for this are ease of understanding the rules, bidder interaction and competition, and asynchronous bidding (suitable for auctions longer than a few minutes). Another type of auction is Continuous Double Auction (CDA), which is similar to the NASDAQ procedures. Buyers and sellers continuously view market and prices, make real-time offers, which clear at the market price. The CDA matches bids in the order received. When a new buy bid is processed, the auction checks whether the offered price would match the lowest existing sell bid, and vice versa. On detection of a match, the auction clears at the price of the existing bid. The CDA is a common mechanism for organized exchanges, such as stock and commodity markets.

Although the mediums change from situation to situation, the basic concept of commerce is the idea of uniting buyers and sellers. It is the premise that drove ancient marketplaces, and it is the same force behind the huge expansions in modern day e-commerce. The purpose of this work is development of another medium that serves the same purpose of uniting buyers and sellers. However, this project takes the basic interaction of trade and tries to make a mutually beneficial situation for both buyer and seller by making the exchange as optimal as possible for both parties.

For example, when a student is in need of a textbook for a given term there is but one logical choice; i.e., go to the campus bookstore and purchase the desired text. The
situation exists that the price the student (the buyer) pays is dictated by the bookstore (the seller). The student is forced to pay the price the seller dictates, because the only medium available for acquisition of the needed text is the bookstore. The inverse is then true when the student no longer needs the text. The student is then the seller and the only buyer for the text then becomes the bookstore, or more appropriately the company facilitating the repurchase of the textbooks at a fraction of their original cost. This is a monopolistic business practice that this project attempts to bypass by utilizing *Continuous Double Auction* (CDA) to develop a system that offers students an alternative medium for both buying and selling textbooks.

The author of this project has substantial background in Economics (MS degree) as well as both professional and advanced academic experience in business (1.5 years in an MBA Program). The purpose of this work is to develop software for the specific applications area – Economics, using Computer Science knowledge and skills.

This project implements a Web-based CDA for selling textbooks on a university campus or in a department. It can be easily maintained by computer personnel and save hundreds of dollars to students per semester on textbooks. This type of an auction system requires very little maintenance and administration. The whole process is automated and execution of trades is triggered by entering bids.
Chapter 2: The Web-Based Auction System

This project is a development of an online auction system for the redistribution of textbooks among students. As was discussed in the first chapter, there are a variety of different types of auctions determined by the unique characteristics of each type. These characteristics include attributes, such as the number and frequency of market-clearing events, the restrictions on bidding, the disclosure of intermediate information, and the policies for determining prices to match buyers and sellers. The process of uniting a buyer and a seller in an easy-to-use and equally beneficial medium is the primary focus of this work.

Since the project relies heavily on CDA, the following section describes how the attributes of CDA are applied to this work.

2.1 Terms and Definitions

For the purpose of this project, an auction is defined as described by McAfee and McMillan (1987) as “a market institution with an explicit set of rules determining resource allocation and prices based on the basis of bids from the market participants”. This is the method for reaching the optimal exchange situation discussed earlier for both buyer and seller. The seller places an item into the marketplace with a minimum acceptable exchange rate that a buyer will pay for the good. In a CDA format, the seller takes the first offered exchange that is equal to or greater than the minimum exchange
rate. The buyer who is the first to satisfy the seller’s exchange rate, gains the seller’s merchandise. This is how both parties obtain the optimal exchange.

The auction makes this happen by allowing multiple buyers to offer “bids” on what they think a seller’s good or service is worth. A bid is defined in this work as “the terms offered by the bidder to buy or sell an auctioned good”. In this project the merchandise up for bid is a textbook. The seller has a preconceived notion as to the minimal bid that will be acceptable for, in this case, a textbook they are selling. However, the multiple buyers in the marketplace may have no idea what this preconceived notion might be, and here is where the mutually optimal situation arises.

In general, bids specify a quantity and price of the good, or even a set of quantities and their corresponding prices. Many auctions, however, accept bids only for a single unit of the good. In this case, the bidder specifies whether the bid is to buy or to sell, and at what price. The seller essentially places a bid on the good or service that they are offering by placing a minimal acceptable exchange rate. In doing this, they create a starting point for the bidding. Anything over this minimal bid can be viewed as an added benefit, while anything less can be rejected. The process of monitoring whether the seller obtains at least the minimal desired exchange is dealt with by the auction policies.

In establishing a successful auction, the very definition given to us by McAfee and McMillan dictates that there must be rules and policies in place regarding how the auction functions. This process is known as clearing, or matching a seller’s desired price to a buyer’s willingness to pay. In the case of an online textbook auction, when a user logs in and bids to purchase some specific textbook for $20, the auction system checks whether the offered price is higher than the lowest existing sell price. On
detection of a match, the auction clears at the clearing price. So clearing, in this case, is
the system taking bids from the respective buyers, searching the database for matches to
the specified criteria, and uniting buyers and sellers when an agreeable scenario is
discovered. Clearing, in the same regard, is the process of the system looking for a buyer
who meets the criteria of each seller. At anytime when this combination of both buyer
and seller criteria is either met or exceeded, both parties experience a mutually beneficial
exchange.

So what exactly is the “clearing price”? This is the price at which both buyer and
seller experience some benefit. This project uses the following equation to describe the
clearing price:

\[
\text{buy offer price} \geq \text{clearing price} \geq \text{sell offer price}.
\]

In using this relation, the clearing price is by definition a variable that is determined by
the range between the maximum value a buyer is willing to grant and the lowest value the
seller is willing to accept. For example, if X wants to buy textbook A at $150 and another
person Y wants to sell at $140 then the clearing price can be an average ($145), or the
lowest sell price ($140), or the highest buy price ($150). In this project, the clearing price
is determined as an average of the buyer and seller prices. However, if the buy offer is
less than the sell offer, then a trade cannot occur. In case there is “no match”, the system
returns a message telling the user that there were no matches to the bid. Matched bidders
are notified of the results of the transaction, their bids are removed from the auction, and
the auction continues.

In case different quantities are submitted for trade by buyer and seller, the system
determines the clearing quantity using the lower of the two values.
Chapter 1: Introduction and Background

Online auctions have become a very dynamic area of activity lately. There are thousands of auction sites on the Internet offering computers, books, CDs, antiques, collectibles, toys, etc. The most prominent auction sites are ebay.com and Onsale.com. Many believe that online auctions will become more popular than posted-price transactions in the future, and this belief has led to very active research in the field of online auctions.

Automated auctions have been determining the real-time price of commodities in the NASDAQ financial market for more than twenty years, and during this time technology has been evolving tremendously. Several factors have contributed to the rise of even more online auctions, such as the development of secure transactions, the spread of Web browsers and point-and-click technology, the growth of the Internet population, and the advantages of doing business online.

An auction is an economic mechanism for determining the price of an item. It requires a determined methodology, an item for sale, bidders who would like to buy the item, and the sale of the item to the highest bidder. Many researches conclude that the online auction has great potential as an automated price-negotiation tool that can be used for virtually any kind of goods and services. Both Web and Internet technology are relatively new, thus the online auction companies are new also. Only about half of those companies providing online auctions have been in business for over four years (C. Beam, A. Segev, 1998).
There are different traditional types of auctions depending on the number and frequency of market-clearing events, restrictions on bidding, disclosure of intermediate information, and the policies for determining prices and matching buyers with sellers. However, the online auctions do not always follow the traditional types, mostly because of the newly enabled experimentation with auction rules. The most popular auction is the English auction. The reasons for this are ease of understanding the rules, bidder interaction and competition, and asynchronous bidding (suitable for auctions longer than a few minutes). Another type of auction is Continuous Double Auction (CDA), which is similar to the NASDAQ procedures. Buyers and sellers continuously view market and prices, make real-time offers, which clear at the market price. The CDA matches bids in the order received. When a new buy bid is processed, the auction checks whether the offered price would match the lowest existing sell bid, and vice versa. On detection of a match, the auction clears at the price of the existing bid. The CDA is a common mechanism for organized exchanges, such as stock and commodity markets.

Although the mediums change from situation to situation, the basic concept of commerce is the idea of uniting buyers and sellers. It is the premise that drove ancient marketplaces, and it is the same force behind the huge expansions in modern day e-commerce. The purpose of this work is development of another medium that serves the same purpose of uniting buyers and sellers. However, this project takes the basic interaction of trade and tries to make a mutually beneficial situation for both buyer and seller by making the exchange as optimal as possible for both parties.

For example, when a student is in need of a textbook for a given term there is but one logical choice; i.e., go to the campus bookstore and purchase the desired text. The
situation exists that the price the student (the buyer) pays is dictated by the bookstore (the seller). The student is forced to pay the price the seller dictates, because the only medium available for acquisition of the needed text is the bookstore. The inverse is then true when the student no longer needs the text. The student is then the seller and the only buyer for the text then becomes the bookstore, or more appropriately the company facilitating the repurchase of the textbooks at a fraction of their original cost. This is a monopolistic business practice that this project attempts to bypass by utilizing *Continuous Double Auction* (CDA) to develop a system that offers students an alternative medium for both buying and selling textbooks.

The author of this project has substantial background in Economics (MS degree) as well as both professional and advanced academic experience in business (1.5 years in an MBA Program). The purpose of this work is to develop software for the specific applications area – Economics, using Computer Science knowledge and skills.

This project implements a Web-based CDA for selling textbooks on a university campus or in a department. It can be easily maintained by computer personnel and save hundreds of dollars to students per semester on textbooks. This type of an auction system requires very little maintenance and administration. The whole process is automated and execution of trades is triggered by entering bids.
Chapter 2: The Web-Based Auction System

This project is a development of an online auction system for the redistribution of textbooks among students. As was discussed in the first chapter, there are a variety of different types of auctions determined by the unique characteristics of each type. These characteristics include attributes, such as the number and frequency of market-clearing events, the restrictions on bidding, the disclosure of intermediate information, and the policies for determining prices to match buyers and sellers. The process of uniting a buyer and a seller in an easy-to-use and equally beneficial medium is the primary focus of this work.

Since the project relies heavily on CDA, the following section describes how the attributes of CDA are applied to this work.

2.1 Terms and Definitions

For the purpose of this project, an auction is defined as described by McAfee and McMillan (1987) as “a market institution with an explicit set of rules determining resource allocation and prices based on the basis of bids from the market participants”. This is the method for reaching the optimal exchange situation discussed earlier for both buyer and seller. The seller places an item into the marketplace with a minimum acceptable exchange rate that a buyer will pay for the good. In a CDA format, the seller takes the first offered exchange that is equal to or greater than the minimum exchange
rate. The buyer who is the first to satisfy the seller’s exchange rate, gains the seller’s merchandise. This is how both parties obtain the optimal exchange.

The auction makes this happen by allowing multiple buyers to offer “bids” on what they think a seller’s good or service is worth. A bid is defined in this work as “the terms offered by the bidder to buy or sell an auctioned good”. In this project the merchandise up for bid is a textbook. The seller has a preconceived notion as to the minimal bid that will be acceptable for, in this case, a textbook they are selling. However, the multiple buyers in the marketplace may have no idea what this preconceived notion might be, and here is where the mutually optimal situation arises.

In general, bids specify a quantity and price of the good, or even a set of quantities and their corresponding prices. Many auctions, however, accept bids only for a single unit of the good. In this case, the bidder specifies whether the bid is to buy or to sell, and at what price. The seller essentially places a bid on the good or service that they are offering by placing a minimal acceptable exchange rate. In doing this, they create a starting point for the bidding. Anything over this minimal bid can be viewed as an added benefit, while anything less can be rejected. The process of monitoring whether the seller obtains at least the minimal desired exchange is dealt with by the auction policies.

In establishing a successful auction, the very definition given to us by McAfee and McMillan dictates that there must be rules and policies in place regarding how the auction functions. This process is known as clearing, or matching a seller’s desired price to a buyer’s willingness to pay. In the case of an online textbook auction, when a user logs in and bids to purchase some specific textbook for $20, the auction system checks whether the offered price is higher than the lowest existing sell price. On
detection of a match, the auction clears at the clearing price. So clearing, in this case, is
the system taking bids from the respective buyers, searching the database for matches to
the specified criteria, and uniting buyers and sellers when an agreeable scenario is
discovered. Clearing, in the same regard, is the process of the system looking for a buyer
who meets the criteria of each seller. At anytime when this combination of both buyer
and seller criteria is either met or exceeded, both parties experience a mutually beneficial
exchange.

So what exactly is the “clearing price”? This is the price at which both buyer and
seller experience some benefit. This project uses the following equation to describe the
clearing price:

\[ \text{buy offer price} \geq \text{clearing price} \geq \text{sell offer price}. \]

In using this relation, the clearing price is by definition a variable that is determined by
the range between the maximum value a buyer is willing to grant and the lowest value the
seller is willing to accept. For example, if X wants to buy textbook A at $150 and another
person Y wants to sell at $140 then the clearing price can be an average ($145), or the
lowest sell price ($140), or the highest buy price ($150). In this project, the clearing price
is determined as an average of the buyer and seller prices. However, if the buy offer is
less than the sell offer, then a trade cannot occur. In case there is “no match”, the system
returns a message telling the user that there were no matches to the bid. Matched bidders
are notified of the results of the transaction, their bids are removed from the auction, and
the auction continues.

In case different quantities are submitted for trade by buyer and seller, the system
determines the clearing quantity using the lower of the two values.
2.2 Operational Features

The first page of the online auction system that a student can see is a welcome page to the CDA (Figure 2.1). This page congratulates a person for finding the site and explains the purpose and organization of the auction system. It also shows two links to choose from; one for registered users and the other one for new users.

![Welcome Screen](image)

Figure 2.1 Welcome Screen

The user interface diagram is shown on Figure 2.2. Each rectangle displays the name of the screen (for example, ‘Welcome page’) and available options (links or buttons) from that page. Arrows lead to the screens that are displayed when each particular option gets selected.
A student who is accessing the system for the first time needs to register. After activating the link for new users to register, a page with the Continuous Double Auction rules and regulations is displayed (Figure 2.3). It contains several paragraphs describing how the system works and how a student can get the most out of it.

Figure 2.3. Auction Policies

Topics discussed include:

- **Membership Eligibility.** A disclaimer that students are the only ones who are allowed to use the system, to prevent book resellers from using the site.

- **Type of Auction.** Continuous Double Auction type is defined.

- **How the Auction Operates.** CDA attributes are explained in detail.

- **Clearing Price.** The term is described and its calculation is shown.
• **Actual Book Exchange.** This paragraph mentions that the site is not involved in any actual exchanges, its purpose is to match buyers and sellers when there is possibility of a trade. It also states that there is no control over the quality of textbooks and the accuracy of the listings, and there is no guarantee that a trader completes a transaction.

• **Trading Partners.** This part of the rules tells the student about the responsibility to complete a transaction in case the submitted bid results in trade. It also explains when a bid can be changed or deleted.

• **Limited Responsibility.** The last part of the rules states that in case of any dispute the site owners/operators are released from all claims. It also mentions that there is no information control on the site, so common sense should be used while placing bids. Furthermore, there are no warranties.

From that page there are two option buttons: one to accept the rules and the other one to decline. Any person declining the rules is directed to the last page, which thanks the user for accessing the CDA system. However, if a person accepts the rules, a registration form is displayed.

The registration form consists of two parts (Figure 2.4). The first part asks for private information about the user such as name, address, and phone number. All information entered here can not be seen by other users of the system and its sole purpose is to uniquely identify a student. The second part of the form collects public data that is shown to other users of the system in all transactions. There are two fields in this section: e-mail address and trader name. After completing the registration form and selecting the submit button, users are taken to the next page that displays the results of the registration.
If any required field is left blank, a message is displayed for the student to go back and add all missing fields. If a student is already registered, an appropriate message is displayed and a link to the system login page is provided. The entered trader name is also checked against all existing trader names in the system. The trader name is supposed to uniquely identify a person in all transactions. If the same trader name already exist in the system, the user is asked to choose another name and try to register again. When the user completes and submits the registration information a unique username and a password are generated. This username and password must be entered every time the user wants to access the system, however they can be changed later. The registration results page shows the assigned username and the password for the student, and a link is provided to the system login page.
It is necessary to **login** in order to enter the system (see Figure 2.5). The login screen prompts a student for a username and the password. After entering these values, the user needs to push the submit button.

![Login Screen](image)

**Figure 2.5. Login Screen**

The system first checks if the username entered is in the database. In case no match is found, a message is displayed stating that the entered username is not in the database, and a link is provided to the auction’s rules and policies page for the student to register. If the username is found in the database, then the system checks to determine if the password entered matches the one stored for that user. In case they do not match, a message is displayed stating that an incorrect password has been entered and a link back to the login screen is provided. If both the username and the password are correct, a
message is shown that the login was successful and the link to the user’s main homepage is provided.

The system dynamically customizes the **homepage** for each user based on their profile. Dynamically means that there is a basic template that is the same for all users, however, this template is changed by the system itself based on each user’s profile. The system maintains a database of information about all submitted bids and completed trades, calculates daily auction statistics, and checks logs. All of these details are available for all users as options from their homepages (Figure 2.6).

![Figure 2.6. Main Homepage](image)

There are ten options that the user can select. These options are described in detail below.
1. The first option is to search for a specific textbook. Selecting this option takes users to the search form where they can enter one or more search criteria, including book identification number, author, title, and book description (Figure 2.7). If more than one criteria is entered the logical operator ‘or’ is used to combine all of them together for the search. After selecting the submit button, a screen with all books that match one or more criteria is displayed.

Figure 2.7. Search Form

At least one search criteria needs to be entered, otherwise an error message is displayed. Found textbooks are divided according to book categories. Information displayed includes book identification number, author and title. A link back to the search form is provided. The user can click on the book identification number of every book and check all submitted bids on that particular textbook.
The bids to buy and to sell are displayed separately. Bid information includes the date the bid was submitted, the number of books for sale, and a description of the textbook. The last column shows if the user has bids on that textbook himself. From this page a user can go back to the search form to enter new search criteria. In case no bids are found on that textbook a corresponding message is displayed and a link to the search form is provided. If the book being searched for is found, the user can note the book identification number and select the second option from the main page to submit a bid.

2. The next option from the main page is to place a bid. After choosing this link the user needs to fill out a bid submission form (Figure 2.8).

![Bid Submission Form](image)

**Figure 2.8. Bid Submission Form**

When providing this information, the user indicates whether the bid is to buy or to sell and inputs either the book identification number (if known) or the book category,
author, and title. A specific category or subject can be selected from a list, which consists of arts, biology, business, computer science, mathematics, etc. An optional description of the textbook can then be entered, along with the minimal acceptable price, and the number of books for sale. The description is not used in matching bids; it is only displayed when users are searching through books. This field can be entered in both cases, while selling and buying textbooks. After selecting the submit button, a message with the results of the bid is displayed on the screen.

If all required fields are not entered, a message is displayed for the student to go back and add all missing data. If there is a problem with entered information (for example, price or quantity are negative values or zeroes, or not numeric), the system shows an error message and suggests what needs to be corrected.

When a user enters a category, author and title, or book identification number the system first checks if the textbook can be found in the database. If the author and title can not be found in the database, the book is registered, and a message is displayed to the user about successful bid submission. If the book identification number can not be found in the database, an error message is displayed asking the user to correct the book identification number or use category, author and title to register the bid. When the book exists in the database, the system checks if the user already has any bids on that textbook. To avoid the possibility of people checking for the existing acceptable prices set by other users, bidders are not allowed to submit multiple bids for the same textbook. In case a bid is found, the system displays a message reminding the student that only one bid can be submitted per book.
When the entered author and title of the book is found, a message is displayed to inform the user of the book identification number (Figure 2.9).

The student is then asked to confirm that the book is the one the student would like to bid on. The confirmation can be done by checking bids submitted by other users and reading their descriptions (the link to existing bids is provided). The student may then go back to the submission form and enter the book identification number instead of the category, author and title. Students are encouraged to use the existing book identification number when no other bids on that textbook exist. In some cases, the book, the student wants to bid on, might be only slightly different from the one that exists in the database. For example, the descriptions of the existing bids can say that the book is new, but the student wants to buy or sell a used copy. In this case, the student is instructed to

**Figure 2.9. Textbook Information Screen**
modify the title of the book (perhaps by adding the word “used”) and register it as a
different entry with a different identification number. This will prevent the new book
from being matched with the existing bids.

When the book identification number is entered and it can be found in the
database, a message about successful submission is displayed, the bid is submitted, and
the system checks to see if a trade is possible. In other words, if the bid to sell is
submitted, the system immediately checks to determine if there are any existing bids to
buy this book with a price equal to or higher than the minimal acceptable selling price
given by the seller. If a bid to buy is entered, the system searches the database to
determine if there are any existing bids to sell this book with the price equal to or lower
than the maximum acceptable buying price given by the buyer. In case of a match, the
trade is automatically completed and added to the user’s completed trades. The user gets
a message of congratulations on the trade and a link is provided to the completed trades
page.

However, if the bid does not result in trade, a message is displayed about the
successful bid submission and a link to place a new bid is provided, along with a link for
the user to check all his outstanding bids.
3. Another option from users’ homepages is to check textbooks that other students want to buy – *books in demand* (Figure 2.10). Books are divided into categories and the search can be done based on a specific category or on all textbooks.

**Figure 2.10. Textbooks in Demand**

After selecting a category, the system finds all bids submitted in that category. For each bid found, the information displayed includes the book identification number, author, and title of the textbook, as well as the description, date the bid was submitted, and quantity. Potential sellers are not able to see what prices the buyers are willing to pay for the textbooks. For each book, the system also shows whether the student performing the search has any outstanding bids. If a user finds a book that he wants to sell, he needs to note the book identification number and select the option for placing a bid from the main page.
4. Another option from users’ homepages is to check textbooks available for sale. The student can view all bids that were submitted to sell textbooks based on book category (Figure 2.11).

![Figure 2.11. Textbooks for Sale]

Similar to the ‘books in demand’ option, the information provided here consists of book identification number, author, title of each textbook, along with book description, date and quantity. Buyers can not see what the minimal selling prices are. If they find a book they would like to buy, they need to take a note of that book’s identification number and select the option to place a bid.
5. The next option from the homepage is to check outstanding bids (Figure 2.12). Students are able to come back after a period of time, login into the system, and check the status of their bids. If the user’s bid has been matched, then the corresponding data is deleted from the outstanding bids and added to the completed trades. Only bids to sell and to buy books that have not resulted in trades are shown on the outstanding bids page.

![Image of outstanding bids](image)

**Figure 2.12. Check Bids**

The bids are divided by book category. For each bid, information displayed includes the date the bid was submitted, type of bid (to sell or to buy), book identification number, author, title and description of the textbook, price and quantity entered. In addition, there are two different kinds of flags related to submitted bids. One flag appears
when a new bid has been added for one of the books being displayed. The other flag shows bids that have been modified since the user’s most recent access.

It is possible for the students to modify their bids, however, there are some conditions that have to be met. For example, in order to update price, quantity and/or book description, the bid must be at least two hours old. In case a student wants to delete a specific bid, the bid must be at least one day old. If the time conditions are met, a student can select the links to update a bid or to delete a bid. Corresponding forms are displayed allowing the user to enter the information.

In the update bid form it is required to specify book identification number, and at least one of the three fields: book description, price or quantity (Figure 2.13).
After selecting submit button, the system checks if the book identification number can be found in the database and if the user has a bid on it. In case the bid with the specified book identification number does not exist, the user gets an error message and is asked to check and correct the entered information. If the bid exists then that bid is updated. In case the price was modified, the system checks for the possibility of trade. A message is displayed to the user saying whether the updated bid has resulted in trade or not, and a link to check trades or to check outstanding bids is provided.

In the delete bid form, the student only needs to enter the book identification number (Figure 2.14). The system checks to see if the bid exists and then deletes it.

![Figure 2.14. Delete Bid Form](image-url)
A message about successful bid deletion is then displayed. In case the book identification number entered is not found in the database, an error message is displayed and the user is told to correct the book identification number.

6. Selecting an option to check trades results in a page showing the details of all trades for that user (Figure 2.15). Completed trades are divided into textbooks bought and textbooks sold. Each completed trade is shown with information consisting of date of the trade, book identification number, author and title, clearing price, and quantity.

![Figure 2.15. Check Trades](image)

The other trader’s name is also provided, as well as the e-mail address to arrange the actual book exchange. To double-check the textbook data, there is an option to have both the current user and the other trader’s description shown for each trade. The new trades that a user has not seen are flagged for easy checking.
7. The next option that is offered on the main homepage is to check auction statistics (Figure 2.16).

![Auction Statistics Table]

**Figure 2.16. Auction Statistics**

After each trade is completed, the system updates information in the database to reflect the new number of trades for each category/subject of the textbooks, total quantity sold, and total value of the completed transactions. Total value is the quantity exchanged in each transaction multiplied by the clearing price. This information is available for all users of the system on a daily basis.
One other option that a user can select from the main page is to **update account information** (Figure 2.17).

Figure 2.17. Account Information

Both public and private information entered by the user during registration process is displayed. The student can modify part of the private information, such as username and password. In case a user needs to update name, address or phone number, an e-mail request to change this information needs to be sent to the owners/operators of the site. Both of the public fields, e-mail address and trader name can be updated on the site. Selecting the corresponding link results in a page that prompts the user to enter either a new username, or a new password, or a new e-mail address, or a new trader name.
9. *Action rules* are shown as another option. This page is similar to the one shown when a student registers. This option is added to give users a chance to access the rules and regulations of the auction system whenever they have questions or need to clarify anything. A link to e-mail the owners/operators of the site is provided on that page.

10. The last option from each user’s homepage is to *logout* from the auction system (Figure 2.18).

![Figure 2.18. Logout Screen](image)

After activating this link the user session is closed and the last page with the “thank you for using the system” note is displayed. At that point, if a user wants to initiate another transaction, the system prompts for login credentials. It continues only after they are entered correctly.
Chapter 3: System Design

3.1 Environment

Oracle database management system (DBMS) is used as a backend for the developed application. It can be installed on either a UNIX or Windows platform. In order to run Web-based database applications, the Oracle Web Application Server (WAS) must be installed (See Oracle Technology Network website at http://technet.oracle.com/, section Documentation, for step by step instructions on how to perform install and how to configure the Server). A key component of WAS is the Web Listener (basically a HTTP daemon) which accepts URL requests from a browser. For dynamic files, the URL specification must include a module, PL/SQL Agent, which allows the WebServer to execute PL/SQL stored procedures. It also uses utilities like htp and owa_util packages to generate dynamic HTML documents. PL/SQL includes several component libraries, for example, HyperText Procedures (HTP) and HyperText Functions (HTF). These libraries contain routines that are useful in generating HTML.

The programming languages that are used in the implementation of the auction system are PL/SQL, which is Oracle Corporation’s procedural language extension of the SQL relational database language, and HTML. PL/SQL also includes a number of Oracle development tools and other tools, mentioned earlier, that execute Oracle stored procedures. To implement the logic of the online auction system, 50 stored procedures were written, as well as four functions. Some of the procedures include forms for the user to enter data in the Web environment (such as bid submission or student registration.
forms). After submitting these forms, all input parameters are passed to the procedures that have the business logic to call other procedures depending on the user input.

As mentioned in the first chapter, the CDA system is completely automated (event-driven model). Which means that all the processes, such as matching the bids, completing the trades, and calculating auction statistics, do not require any user intervention. The only input that is needed for the system to begin functioning is for students to register and begin entering bids. These actions trigger the execution of the corresponding stored procedures.

The rest of the chapter describes how the online auction system, explained in the previous chapters, is implemented. As it was stated in the previous chapter, the project is based on the Oracle DBMS. As Steven Feuerstein mentioned in his work, “…The whole idea of ‘programming’ in Oracle-based applications has a somewhat different meaning from what it used to in a third-generation, or even a wholly procedural fourth-generation, language”. While building such an application, different technologies and programming styles are usually blended together. Conceptually, there can be as many as five layers of “code” in a single application. Each of these different layers of code requires a different approach to programming. Table 3.1 shows layers of code with the corresponding roles that these layers play in the application.

Thus, the first part of the CDA system was the design of the database tables, relying mostly on the declarative SQL language. After the table structures with appropriate attributes had been finalized, the procedural design was done. Database procedures and triggers were created, and then coded in PL/SQL to implement business rules at a database level. From there, the screens and forms that comprise the user
Table 3.1. Layers of Code in an ORACLE Application

<table>
<thead>
<tr>
<th>Code Level</th>
<th>Role of Code in Application</th>
</tr>
</thead>
<tbody>
<tr>
<td>Graphical objects</td>
<td>Accessed directly by the user</td>
</tr>
<tr>
<td>Event triggers</td>
<td>Triggered by user activity; triggers are usually attached to events associated with a specific graphical object.</td>
</tr>
<tr>
<td>Form-based PL/SQL</td>
<td>Executed by a trigger; implements nondeclarative, non-default functions in an application.</td>
</tr>
<tr>
<td>RDBMS-based PL/SQL</td>
<td>Stored procedures and database triggers executed in response to actions initiated in the form.</td>
</tr>
<tr>
<td>SQL</td>
<td>The data definition and data manipulation statements used to create and maintain the data in the Oracle Server.</td>
</tr>
</tbody>
</table>

interface were designed. Once the graphical objects were ready, PL/SQL procedural code was associated with those objects. The event triggers, such as the press of the submit button, were attached to the appropriate objects. When the user triggers such an event, that trigger fires and executes the PL/SQL code (calls the corresponding procedure).

3.2 Database Structure

The auction system needs several tables to store information related to such major entities as students (buyers and sellers), textbooks, bids, and trades (or matched bids). However, some intermediary tables are needed as well to simplify the auction matching
process, and keep log data and statistics. Thus, the CDA is operating with the following set of eight tables:

1. **STUDENT** – for storing information related to each student, such as student name, address, and phone number, e-mail address, username and password. Each entry in this table is assigned a system-generated student identification number. The student_ids are sequentially generated integers that are used later for all transactions of that user.

2. **LOG_CHECK** – for saving all login attempts of each user for creating user sessions. The table stores student identification number and timestamp. The system makes an entry into this table each time the login credentials are entered on the web site or the user logs off the system. The result field of the table records whether the login was successful or not, and the logged off status.

3. **TEXTBOOK** – for keeping information related to textbooks offered for trade, such as author, title, and the book category for easy searching based on the subject. Similar to the **STUDENT** table, each entry in this table is assigned a system-generated book identification number. These book_ids are sequentially generated integers that are used for referencing this particular textbook in all transactions throughout the site.

4. **BID** – which contains information related to each bid, such as the student identification number of the user who placed the bid, the book identification number and optional description of the textbook that the bid is placed on, the type of bid (whether the student wants to sell or to buy the book), as well as the price and quantity of the book that is offered for sale. Based on the CDA rule that all bids are matched in the order received, each entry in the **BID** table is time-stamped and numbered for accuracy of transaction
processing. The table also has two columns to keep new and update flags. These flags are used when a student selects ‘your bid’ option on the main page.

5. **POSSIBLE** – an intermediary table. In order to make bid matching easy, it is beneficial to keep track of the highest price among all buy bids and the lowest price among all sell bids for each textbook. This is accomplished by updating these values after each bid is submitted into the system or modified. The table stores the book identification number, the lowest price to sell, the quantity and the student identification number of the user who wants to sell the book for that price. In addition, it stores the highest price to buy with the corresponding quantity and student identification number of the user who placed the bid to buy this textbook for that price.

6. **TRADE** – for saving information related to the matched bids. After updating table **POSSIBLE**, the two prices are compared. According to the auction rules discussed in the first chapter of the project, a trade is possible when the highest buy price is greater than or equal to the lowest sell price. In that case, a new entry with all relevant information is inserted into the **TRADE** table. Such information includes the book identification number of the textbook exchanged, the student identification numbers both for the buyer and the seller, the quantity sold based on the lowest of the two quantities specified in sell and buy bids, and the clearing price (which is determined as an average of the prices offered by the buyer and the seller). The table also saves the book descriptions entered by buyer and seller. Each trade is assigned a unique identification number for tracking purposes. The last two fields in the table contains new flags to show when a trade has not been seen by a buyer or a seller.
7. **AUCTION_DETAIL** – for capturing statistics of the trades based on the book category. Every time a trade is completed this table is updated such that the number of trades is incremented, the quantity sold in that particular trade is added to the total quantity sold for that category, and the total value (calculated by multiplication of the clearing price on the quantity) is added to the grand total for that category.

8. **LOG_TABLE** – for recording all information related to the submitted bids. It is important to save each bid’s data, because after a trade is executed the bids are deleted from the BID table. The bid can be deleted by a user or by the system in case the total quantity of the bid has been matched. However, if the trade quantity is different from the bid quantity, the bid is updated to reflect the remaining quantity. A new entry is written into this table each time a new buy or sell bid is submitted (the change type is insert), or the BID table is updated as a result of a completed trade (the change type is update).

Table 3.2 shows the tables described with corresponding attributes in tabular form. Figure 3.3 represents the relational schema for the database. In addition to the tables described, there are five views that are needed for easy accessing of the data from more than one table at once. The first view is the **STUDENTBIDVIEW** that retrieves data from both the BID and TEXTBOOK tables to combine the student identification number with textbook information for all submitted bids of that user. The other two views are the **BUYBIDVIEW** and the **SELLBIDVIEW** that show data from the BID and TEXTBOOK tables related to all bids submitted to buy and to sell textbooks correspondingly. The data includes the book identification number with author, title, description, and book_category, as well as the quantity offered for sale, timestamp, and student_id. The fourth view is the **TRADEVIEW1**, which combines the data from three tables:
Table 3.2. Table Structure with Attributes

<table>
<thead>
<tr>
<th>Table</th>
<th>Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>STUDENT</td>
<td>Student_ID, First_Name, Last_Name, Street_Address, City, State, Zip, Phone_Number, E_Mail, Username, Password, Trader_Name</td>
</tr>
<tr>
<td>LOG_CHECK</td>
<td>Student_ID, Timestamp, Result</td>
</tr>
<tr>
<td>TEXTBOOK</td>
<td>Book_ID, Author, Title, Book_Category</td>
</tr>
<tr>
<td>BID</td>
<td>Bid_ID, Book_ID, Student_ID, Bid_Type, Bid_Price, Quantity, Timestamp, Book_Description, Updated_Flag</td>
</tr>
<tr>
<td>POSSIBLE</td>
<td>Book_ID, Student_ID_Sell, Low_Sell, Qty_Sell, Student_ID_Buy, High_Buy, Qty_Buy</td>
</tr>
<tr>
<td>TRADE</td>
<td>Trade_ID, Book_ID, Student_ID_Sell, Student_ID_Buy, Qty, Clearing_Price, Timestamp, Bookdesc_Sell, Bookdesc_Buy, New_Flag_S, New_Flag_B</td>
</tr>
<tr>
<td>AUCTION_DETAIL</td>
<td>Book_Category, Number_of_Trades, Total_Qty, Total_Value</td>
</tr>
<tr>
<td>LOG_TABLE</td>
<td>Change_Type, Timestamp, StudentID, BookID, BidType, Old_BidPrice, Old_Qty, New_BidPrice, New_Qty</td>
</tr>
</tbody>
</table>

TEXTBOOK, TRADE and STUDENT. This view is used for displaying information related to all trades in which a specified student was on the selling side of the completed trade. The view provides the student identification number of the student who sold a textbook, the book identification number, category, author and title of the textbook, as well as both buyer and seller book descriptions, the quantity sold, clearing price, date, and identification number of the trade, and the contact information of the buyer (trader name and e-mail address). The last view that is created is TRADEVIEW2 that is similar to
the TRADEVIEW1. The view contains all the details of the trades in which the specific student was not on the selling side (as in TRADEVIEW1) but on the buying side of the transaction.

3.3 Procedural Design

The second step in the implementation was the design of the procedures corresponding to the operational features described in detail in the second chapter of the project. The CDA system is an internet-based system. Thus in order to access it, a student needs to type its URL in a browser window (Windows Explorer is recommended). This triggers the call to the first procedure Welcome, which displays the welcome screen with two options (Figure 3.4). The first option is to login for the registered users of the system. Selecting this option results in the call to the corresponding procedure CheckID with the login screen. The second option is to register for the new users. Choosing this option calls the procedure Rules, which lists all auction rules and regulations and has two option-links at the bottom: to accept and decline the rules. Declining the rules results in the last page of the system with a thank you note. When a student accepts the rules the AddUser procedure is called. This procedure contains a registration form with a submit button. After submitting all information on the registration page, the procedure AddForm is called. The data entered by a student is passed as input parameters. This procedure performs all the checks discussed in detail on page 11 of chapter 2.2. First, all required fields are checked for null values. If any of those fields were left blank, the user is asked to go back and add missing entries. Otherwise, the STUDENT table is accessed to
determine whether a student has been registered already using the name and address
fields. If an existing student record is found in the table, a message is displayed to the
student, and a link to the login page is provided. If no student record is found, the
procedure inserts the entered data in the \textit{STUDENT} table, automatically generates a
unique username and a password, and displays a message about the successful
registration and generated login credentials. The \textit{CheckID} procedure with login prompt
can then be called.

After entering the login credentials on the login screen and selecting the submit
button, the next procedure \textit{CheckNow} is called with those parameters passed to it. This
procedure first verifies that both fields are filled out. Then it verifies that the entered
username exists in the \textit{STUDENT} table. If no record can be found in the database, a
message with a link to the registration page is displayed. In case the username is found in
the table, the password is compared to the one entered and a new record is inserted into
the \textit{LOG_CHECK} table. If the passwords match, the Student\_ID number is retrieved to
keep track of the user’s session. The successful login result gets written into the
\textit{LOG_CHECK} table. If the entered password is incorrect, then a message with a link to
try the login again is shown and the unsuccessful login attempt is recorded in a result
field of the \textit{LOG_CHECK} table.

After a successful login, the procedure \textit{Main}, that displays the main user’s homepage,
can be called with the student identification number passed as a parameter. This
procedure has a title with the trader’s name, a menu bar on the left of the page, which
lists ten different options to choose from, and a frame where all the data is displayed. All
options of the menu bar are described below. Selecting each of the options triggers a call

to the corresponding procedure and passes the student identification number to it. At the beginning of each procedure a simple check is done to make sure the student is logged in (the table \textit{LOG\_CHECK} is queried and the result field is analyzed).

1. **Perform a search for a specific textbook.** Selecting this option calls the procedure \textit{SearchForm}. The user can enter at least one search criteria and select the submit button. This triggers a call to the procedure \textit{SearchNow}. This procedure performs the search using entered criteria in the table \textit{TEXTBOOK}. In case more than one is entered, the logical operator ‘or’ is used to combine all of them together. If no textbook matches the criteria, a message is displayed to the user and a link to the search form is provided. If some matches are found, the procedure prints out tables with book identification numbers, authors and titles of textbooks that match the criteria. One table is printed for each category where at least one book is a match. Book identification numbers are printed as links to the procedure \textit{SearchBookID}, which checks for submitted bids on that textbook.

   Selecting any one of the book identification numbers results in a call to that procedure. If no bids of other users of the system exist in the \textit{STUDENTBIDVIEW} for the book identification number a message is displayed to the student. If at least one bid is found the procedure \textit{PrintBookBids} is called with student and book identification numbers as input parameters.

   This procedure prints out information about existing bids from \textit{STUDENTBIDVIEW}. Only bids submitted by other users of the system are shown.

2. **Place a bid to buy or to sell a textbook.** This option of the user’s homepage calls the procedure \textit{AddBid} which displays a bid submission form with a submit button. After
submitting all information on the registration page, the next procedure \textit{AddNewBid} is called with the data entered passed to it. This procedure first verifies that all required fields are entered, it also confirms that no non-numeric or negative entries are entered in the numeric fields, like price and quantity. Otherwise a user is asked to go back and add or correct missing data.

If all required data is entered, then the \textit{TEXTBOOK} table is checked to determine whether this book has already been entered in the database using the book identification number, or author and title. If not, all the book data is inserted into the \textit{TEXTBOOK} table and a new book identification number is generated for keeping track of the textbook. In case the book identification number is entered and it can not be found in the database, an error message is displayed asking the user to correct the book identification number or use category, author and title to register the bid. If a “match” is found for the book, the \textit{BID} table is queried for the outstanding bids of this user for that particular textbook. One of the rules of the auction is that a user cannot submit multiple bids for the same book. Hence, the system needs to make sure that a student does not have any outstanding bids for the textbook he is trying to bid on. If the user already has outstanding bids for a book, a message with the CDA rules is displayed. If no outstanding bids are found, execution of the procedure continues.

If the student has entered author and title and the book is found in the database, the user is asked to confirm submission of the bid with an existing book identification number. Confirmation can be done by viewing the description section of other students’ bids. Another option the user has is to register a book as a new entry (for example, with
slight modifications in title), thus obtaining a new identification number for the book and no match with existing bids.

If a book identification number has been entered and it exists in the database, then the procedure inserts a new record with the entered price and quantity into the BID table and updates the LOG_TABLE. After the new bid is entered into the system, the procedure UpdatePossible is called with several parameters passed to it. These parameters are the student identification number, book identification number, price, quantity and type of bid.

The procedure UpdatePossible checks the table POSSIBLE for the records on the textbook submitted for the bid. In case the record is not found, a new entry is inserted into the table and control is passed back to the calling procedure. A message is displayed to the user about a successful bid submission (in this case the bid cannot result in trade, because it is the first bid on this textbook). The entry into the table POSSIBLE contains the book identification number, student identification number, price and quantity. However, if the textbook already has an entry in the table POSSIBLE, in the case of the bid to sell, the entry is compared with the lowest sell price if there is one stored. If the new value is lower than the existing one, the POSSIBLE table is updated with the new student identification number, the price and the quantity. In the case of a bid to buy, the entry is compared with the highest buy bid for that textbook if there is one stored. If the new value is greater than the existing one, the table POSSIBLE is updated with the new values for the new student identification number, the highest price and the quantity offered to buy. In case no changes are made to the table, nothing else is done; control is passed back to the calling procedure and a message is displayed about the successful bid submission (the bid is submitted without any problems, but has not resulted in trade). If
any update is done to the table POSSIBLE, the next procedure UpdateTrade is called with the book identification number and the student identification number passed as input parameters.

The procedure UpdateTrade selects the prices for both bids to buy and bids to sell for a given textbook from the table POSSIBLE and compares them. If the highest price to buy is less than or equal to the lowest price to sell, then a trade is possible and a procedure AddTrade gets called with the book and student identification numbers.

When the control returns to the procedure UpdateTrade, quantities are compared to make necessary updates to the table POSSIBLE. In case the quantity to sell is less than the quantity to buy, the trade is done using the quantity to sell. Thus, the bid to sell the textbook was completely satisfied and all information about it should be removed from the table POSSIBLE (the bid itself is deleted in the procedure AddTrade). After that, the table BID is searched for any remaining bids to sell the textbook. If at least one bid to sell exists, the table POSSIBLE gets updated with the new lowest price to sell that book. When there are two bids with the same lowest price, the one that was submitted earlier gets priority. If the table POSSIBLE is modified with the new minimum price to sell the textbook, the procedure recursively calls itself to check for the possibility of trade.

In case the quantity to sell is greater than the quantity to buy, it can be inferred that the trade was done using buy quantity. Similarly to the previous case, all information about the bid that was completely satisfied (bid to buy in this case) is removed from the table POSSIBLE. Then the table BID is searched for the remaining bids to buy the textbook. If at least one bid to buy is found, the table POSSIBLE is modified to reflect the new highest price to buy the book. When there are two bids with the same highest price,
the one that was entered earlier gets processed first. If the table \textit{POSSIBLE} is modified with the new maximum price to buy the textbook, the procedure recursively calls itself to check for the possibility of trade.

The last case is when the quantities are equal. In this case, the \textit{BID} table is searched for any other bids for that textbook. If no more records are found, all fields are set to null for that textbook in the table \textit{POSSIBLE}. However, if there is at least one more bid for that textbook, the entry in the \textit{POSSIBLE} table is updated to reflect the student identification numbers with the existing lowest sell price and/or the highest buy price, and quantity.

The procedure \textit{AddTrade} selects all the buyer and the seller information related to a textbook from the table \textit{POSSIBLE}. Then the clearing price is calculated as an average of the prices of the buyer and the seller. The quantity that can be exchanged in the transaction is determined as the lower of the two. After that, a new entry is inserted into the table \textit{TRADE} and the procedure \textit{CheckStatistics} is called with two input parameters (the book identification number and the trade identification number). After the statistics are updated, the bid in which the total quantity is satisfied in the trade is deleted from the \textit{BID} table, while the bid with excess quantity is updated to reflect the remaining amount. For example, if the quantity being offered to sell is less than the quantity being purchased the sell bid is deleted and the quantity in the bid to buy is updated (and vice versa). In case the quantities are equal, both corresponding bids are deleted from the table \textit{BID}. Any change to the table \textit{BID} results in a new record being entered to the \textit{LOG\_TABLE}. At the end of this procedure, a message of congratulation is displayed to the user. A link to check completed trades is provided.
The procedure \textit{CheckStatistics} first selects the book category from the table \texttt{TEXTBOOK} corresponding to the book identification number that just has been sold. After that, the details about the trade that has just been performed are selected using the trade identification number passed to this procedure, and the table \texttt{AUCTION\_DETAIL} is updated. Which means, the number of trades for that particular category of books is incremented, the total quantity is recalculated using the quantity of the last transaction, and the total value is updated to reflect the value (price multiplied by quantity) of the last trade.

3. \textbf{Check textbooks in demand} (the books that other users of the system would like to buy). Selecting this option calls the procedure \textit{OtherBuyBids}, which lists all categories of textbooks as links to the procedure \textit{CheckOtherBuys} with specific book category passed as input parameter along with student identification number.

The procedure \textit{CheckOtherBuys} checks if there is at least one textbook in the category selected in the \texttt{BUYBIDVIEW}. If no bids are found, the appropriate message is displayed to the user. When some bids to buy are found, the procedure prints the category name as a header for the table and calls the procedure \textit{PrintBuyBids} to display all textbooks from the \texttt{BUYBIDVIEW} in the selected category.

The procedure \textit{PrintBuyBids} selects all books that students want to buy in a specific category by any student other than the one who is performing the search from \texttt{BUYBIDVIEW}. Then, all the information about these textbooks is displayed in a table for sellers to view. Sellers may note the book identification numbers of the books they might be interested in selling.
4. **Check textbooks available for sale.** Selecting this option calls the procedure `OtherSellBids`, which lists all categories of textbooks as links to the procedure `CheckOtherSells` with specific book category passed as input parameter along with the student identification number.

   The procedure `CheckOtherSells` checks if there is at least one textbook in the category selected in the `SELLBIDVIEW`. If no bids are found, the appropriate message is displayed to the user. When some bids to buy are found, the procedure prints category name as a header for the table and calls the procedure `PrintSellBids` to display all textbooks from the `SELLBIDVIEW` in the selected category.

   The procedure `PrintSellBids` selects all books for sale in a specific category by any student other than the one who is performing the search from `SELLBIDVIEW`. Then, all the information about these textbooks is displayed in a table for buyers to view. Buyers may make notes of the book identification numbers of the books they might be interested in buying.

5. **Check your outstanding bids.** Selecting this option from the user’s homepage calls the procedure `CheckBids` and passes the student identification number as an input parameter. The procedure first checks if there is at least one bid in `STUDENTBIDVIEW` of that student. If no bids are found, the appropriate message is displayed to the user. If some bids are found, the procedure `PrintBids` is called. There are two links provided in this procedure, one to update a bid, which calls procedure `UpdateBid`, and the other one to delete a bid – procedure `DeleteBid` gets called.

   Procedure `PrintBids` selects all records from `STUDENTBIDVIEW` for that student. Then it prints a table with information about student’s outstanding bids. After the table is
printed, table $BID$ is updated so that the updated flag for all bids of that user is set to ‘1’, meaning that the user has seen the bid.

Procedure $UpdateBid$ contains an update bid form with a submit button. After submitting information, the procedure $UpdateBidNow$ is called with the data entered by a student passed as input parameters. The procedure first checks if all required fields are entered, and if not, displays a message to the user asking him to add missing information. Then, the table $BID$ is queried to check if there is a bid of that student for the book identification number specified. If no bid is found, a message is displayed to the student to check the book identification number and a link to go back to the bid’s page is provided. When the bid is found, the time when the bid was submitted is checked. By the auction rules students can modify their bids only if they were submitted at least two hours ago. If the time constraint is not met, a message with the auction rules is displayed to the student and a link to go back to the outstanding bids is provided. If the bid can be updated, the table $BID$ is modified to reflect the changes the user has entered in the form. Corresponding records are added to the table $LOG_TABLE$. In case only the description of the book is modified, nothing else is done in the procedure except informing the user about successful update and providing the link back to the bids. When price or quantity is updated, the procedure checks whether the table $POSSIBLE$ needs to be modified. This is accomplished by searching the table $BID$ for the lowest price of all bids to sell (if the sell bid is being modified), or for the highest price of all buy bids (if the bid to buy is being modified). After finding those values, the table $POSSIBLE$ is updated and procedure $UpdateTrade$ is called to check for the possibilities of trade.
The procedure **DeleteBid** contains a delete bid form with a submit button. After submitting the book identification number, the procedure **DeleteBidNow** is called. The procedure first checks to see if the book identification number is entered, and if not, displays a message to the user. Then the table **BID** is queried to check if there is a bid by that student for the book identification number specified. If no bid is found, a message is displayed to the student to check the book identification number and try to delete again. When the bid is found, the time when the bid was submitted is checked. By the auction rules students can delete their bids only when they were submitted at least one day ago. If the time constraint is not met, a message with the auction rules is displayed to the student and a link back to the outstanding bids is provided. If the bid has been entered or modified more than a day ago, the bid is deleted from the table **BID**. A record is added to the table **LOG_TABLE**. After that, the procedure makes necessary changes to the table **POSSIBLE**. The table **BID** is queried for the lowest price of all bids to sell (if the sell bid has just been deleted), or for the highest price of all buy bids (if the bid to buy has been deleted). After finding those values, the table **POSSIBLE** gets updated and the procedure **UpdateTrade** is called to check for the possibilities of trade.

6. **Check your completed trades.** This option calls the procedure **CheckTrade** that prints information about all completed trades of that user. First it checks to determine if there is at least one trade with the specified student identification number in the table **TRADE** where the student was a buyer or a seller. The appropriate messages are displayed in case no records are found (for example, if the student does not have an entry in the table **TRADE** in the role of a buyer, then the message says that the student has not bought any textbooks). In case the student has completed trades, procedures
*PrintTrades1* and *PrintTrades2* are called. After control returns back from the printing procedures, table *TRADE* is updated so that the new flag for all trades of that user is set to ‘1’ meaning that the user has seen the trade.

Both procedures, *PrintTrades1* and *PrintTrades2*, are very similar. The only difference is that they are printing the data from different views. One is using *TRADEVIEW1* which contains all information for trades where specified student was selling textbooks (in other words, was on the selling side of the transaction). For detailed explanation of the views see page 33. The other procedure gets its data from *TRADEVIEW2*. This view contains all the details of the trades in which the specific student was on the buying side of the transaction.

7. **Check auction statistics.** This option calls the procedure *Statistics*, which prints out the title of the page and calls procedure *PrintStats* to print out the table with all the records from the *AUCTION_DETAIL* table.

8. **Update your account information.** Selecting this option from the main page results in a call to the procedure *UpdateInfo*, which prints out all the fields of the table *STUDENT* for the specified student identification number. There are four links from that page:

- **Change username** – call to the procedure *UsernameChange*. The procedure lets the student enter a new username and select the submit button. After that, the procedure *UsernameNow* is called with the entered value as an input parameter along with the student identification number. In case the user has left the name field blank, the procedure asks the student to reenter the new username. If a username has been entered, the table *STUDENT* is queried to check that no other user of the system has the same
username. If the duplicate exists, the student is informed about that and asked to modify the name choice. If the entered name is unique, the table $STUDENT$ is updated to reflect the new username for the specified student identification number and the student gets a message about successful update. An option is provided to go back to the account’s information page.

- **Change password** – call to the procedure $PwdChange$. The procedure asks for three values: old password, new password, and re-typed new password for confirmation. Selecting the submit button calls the procedure $PwdNow$ with all three entered variables as input parameters in addition to the student identification number. In case the user has left any of the fields blank, a message is displayed that asks the student to go back and enter all three values. After all fields have been filled out, the old password entered is matched with the one in the table $STUDENT$ for the specified student identification number. If they match, both new password fields are compared between themselves. In case they match too, the table $STUDENT$ is updated to reflect the new password and the student gets a message about successful update. An option is also provided to go back to the account’s information page. If any of the checks fail, the student is informed and asked to reenter information.

- **Change e-mail address** – call to the procedure $EmailChange$. The procedure asks the user to enter a new e-mail address twice for confirmation. Selecting the submit button results in a call to the procedure $EmailNow$ with the entered values send as input parameters along with the student identification number. In case any of the fields has been left blank, the procedure asks the user to go back and fill out both fields. When both fields have been entered, the fields are compared between themselves. If two fields do not
match, the user gets an appropriate message. In case of a match, the table [STUDENT] is updated to reflect the new e-mail address for the specified student identification number and the student gets a message about the successful update. A link is provided from that page to go back to the account information.

- **Change trader name** – call to the procedure [TraderChange]. The procedure lets the student enter a new trader name and select the submit button. After that, the procedure [TraderNow] is called with the entered value as an input parameter along with the student identification number. In case the user has left the name field blank, the procedure asks the student to reenter the new trader name. If a trader name has been entered, the table [STUDENT] is queried to check that no other user of the system has the same trader name. If a duplicate exists, the student is informed and asked to modify the name choice. If the entered name is unique, the table [STUDENT] is updated to reflect the new trader name for the specified student identification number and the student gets a message about successful update. There is a link to the account’s information from that page.

9. **View auction rules and regulations.** Selecting this option calls the procedure [Rules], which lists all auction rules and regulations as described on the page 9 of the second chapter.

10. **Logout from the system.** Selecting this option results in the call to [Logoff] procedure, which displays a “thank you for using the CDA system” message, and terminates the session for that student. An entry is added to the [LOG_CHECK] table with the logged out result for that student’s identification number. The link to the welcome page of the auction is also provided in case the user wants to go back into the system.
Along with the procedures previously described, there are four other functions which are called from different procedures to perform routine tasks and return values needed for the execution of those procedures. One function *CheckAuction* checks the *AUCTION_DETAIL* table for existence of a particular book category. The function returns ‘1’ or true if the record is found. Otherwise it returns ‘0’ or false. The other two functions are *CheckBidBuy* and *CheckBidSell*. These functions check for the existence of bids to buy and to sell for the particular textbook in the *BID* table. They return ‘1’ in case a record is found and ‘0’ if the record is not found.
Chapter 4: Results

The result of the project is a Web-based Continuous Double Auction system. The system can be used as an online mechanism for selling and buying textbooks on any university campus. It offers students an alternative to regular university bookstores. The system makes the process of searching for a desired textbook more convenient and allows students to save hundreds of dollars on textbooks each semester. This auction system requires very little maintenance and administration as a result of its sophisticated event driven nature.

Every participant entering the site for the first time needs to register. A unique username and password combination is generated for every user. All subsequent logins into the system can be done using these credentials. Each registered student can offer textbooks for sale or place a request for a specific textbook. Each user can also submit bids for textbooks suggested by other registered users of the system. Submission of a bid triggers execution of the auction system’s logic for matching buyers and sellers and completing trades.

Certain customized features are added to this Web application. For example, users are allowed to achieve individuality by selecting trader names and writing an optional description of the book they are offering for trade. The system also dynamically changes the user's homepages to display information related only to that particular student based on their profile. This lets users keep track of all their submitted bids and completed trades. They also are able to check all textbooks available for sale or all textbooks in
demand. Furthermore, each student can perform a search on a particular category of books or a specific textbook.

The auction system could be expanded further to allow posting a variety of other college goods such as financial and scientific calculators, art supplies and even memorabilia. The system can also track users based on profiles, problem users or users who failed to live up to obligations.
Bibliography


Appendix A. Code Listing in Alphabetical Order

Tables
1. Auction_Detail .......................................................... 57
2. Bid ........................................................................ 57
3. Log_Check .................................................................. 57
4. Log_Table .................................................................... 57
5. Possible ....................................................................... 57
6. Student ....................................................................... 57
7. Textbook ..................................................................... 57
8. Trade .......................................................................... 58

Sequences
1. Bid_ID_Seq .................................................................. 58
2. Book_ID_Seq ............................................................... 58
3. Student_ID_Seq ............................................................ 58
4. Trade_ID_Seq ............................................................... 58

Views
1. BuyBidView .................................................................. 58
2. SellBidView .................................................................. 58
3. StudentBidView ............................................................. 58
4. TradeView1 ................................................................... 58
5. TradeView2 ................................................................... 58

Procedures
1. AddBid ......................................................................... 59
2. AddForm ....................................................................... 60
3. AddNewBid .................................................................... 61
4. AddTrade ....................................................................... 63
5. AddUser ......................................................................... 65
6. CheckBids ..................................................................... 65
7. CheckID ........................................................................ 66
8. CheckNow ...................................................................... 66
9. CheckOtherBuys ............................................................ 67
10. CheckOtherSells ............................................................. 68
11. CheckStatistics ............................................................. 68
12. CheckTrade .................................................................... 69
13. DeleteBid ....................................................................... 70
14. DeleteBidNow .................................................................. 71
15. EmailChange ................................................................. 73
16. EmailNow ...................................................................... 73
17. ErrorPage ...................................................................... 74
18. Footer .......................................................................... 74
19. Logoff .......................................................................... 75
20. Main ............................................................................. 75
21. NotLogged ..................................................................... 76
22. OtherBuyBids ............................................................... 76
23. OtherSellBids ............................................................... 77
24. PrintBids ...................................................................... 79
25. PrintBookBids .............................................................. 80
26. PrintBuyBids .................................................................. 81
27. PrintSellBids .................................................................. 83
28. PrintStats ...................................................................... 85
29. PrintTrades1 .................................................................. 86
30. PrintTrades2 .................................................................. 87
31. PwdChange .................................................................... 89
32. PwdNow ........................................................................ 89
33. Rules ............................................................................ 90
Functions

1. CheckAuction ................................................................. 108
2. CheckBidBuy .............................................................. 108
3. CheckBidSell .............................................................. 108
4. CheckPoss ................................................................. 108
Tables

create table Auction_Detail
(book_category varchar2(30) constraint pk_auction primary key,
 number_of_trades number,
 total_qty number,
 total_value number);

create table Bid
(bid_id number(5) constraint pk_bid primary key,
 book_id number(5) constraint fk_bid_book references textbook(book_id) on delete cascade,
 student_id number(5) constraint fk_bid_student references student(student_id) on delete cascade,
 bid_type varchar2(4),
 bid_price number(7, 2),
 quantity number(5),
 timestamp date,
 book_description varchar2(300),
 updated_flag number(1));

create table Log_Check
(student_id number(5),
 timestamp date,
 result varchar2(1));

create table Log_Table
(change_type varchar2(8),
 timestamp date,
 studentid number(5),
 bookid number(5),
 bidtype varchar2(4),
 old_bidprice number(7, 2),
 old_qty number(5),
 new_bidprice number(7, 2),
 new_qty number(5));

create table Possible
(book_id number(5) not null constraint fk_poss_book references textbook(book_id) on delete cascade,
 studentid_sell number(5) constraint fk_poss_stsell references student(student_id),
 low_sell number(7, 2),
 qty_sell number(5),
 studentid_buy number(5) constraint fk_poss_stbuy references student(student_id),
 high_buy number(7, 2),
 qty_buy number(5),
 constraint pk_poss primary key (book_id));

create table Student
(student_id number(5) constraint pk_student primary key,
 first_name varchar2(50),
 last_name varchar2(50),
 street_address varchar2(50),
 city varchar2(50),
 state varchar2(50),
 zip varchar2(50),
 phone_number varchar2(50),
 e_mail varchar2(50),
 username varchar2(50),
 password varchar2(50),
 trader_name varchar2(50));

create table Textbook
(book_id number(5) constraint pk_textbook primary key,
 author varchar2(100),
 title varchar2(100),
 book_category varchar2(30));

57
create table Trade
(trade_id number(5) constraint pk_trade primary key,
  book_id number(5),
  studentid_sell number(5),
  studentid_buy number(5),
  qty number(5),
  clearing_price number(7, 2),
  timestamp date,
  bookdesc_sell varchar2(300),
  bookdesc_buy varchar2(300),
  new_flag number(1),
  new_flag_b number(1));

Sequences

create sequence Bid_ID_Seq
increment by 1 start with 1

create sequence Book_ID_Seq
increment by 1 start with 1

create sequence Student_ID_Seq
increment by 1 start with 1

create sequence Trade_ID_Seq
increment by 1 start with 1

Views

create or replace view BuyBidView
(book_id, author, title, book_category, description, quantity, student_id, timestamp) AS
select t.book_id, t.author, t.title, t.book_category,
  b.book_description, b.quantity, b.student_id, b.timestamp
from textbook t, bid b
where t.book_id = b.book_id and b.bid_type = 'buy'

create or replace view SellBidView
(book_id, author, title, book_category, description, quantity, student_id, timestamp) AS
select t.book_id, t.author, t.title, t.book_category,
  b.book_description, b.quantity, b.student_id, b.timestamp
from textbook t, bid b
where t.book_id = b.book_id and b.bid_type = 'sell'

create or replace view StudentBidView
(book_id, author, title, book_category, bid_id, student_id, bid_type, bid_price, quantity, timestamp,
  book_description, updated_flag) AS
select t.book_id, t.author, t.title, t.book_category, b.bid_id, b.student_id, b.bid_type,
  b.bid_price, b.quantity, b.timestamp, b.book_description, b.updated_flag
from textbook t, bid b
where b.book_id = t.book_id

create or replace view TradeView1
(studentid_sell, book_id, author, title, qty, clearing_price, trader_name, timestamp, e-mail, new_flag,
  book_category, trade_id, bookdesc_sell, bookdesc_buy) AS
select t.studentid_sell, b.book_id, b.author, b.title, t.qty, t.clearing_price, s.trader_name, t.timestamp,
  s.e_mail, t.new_flag, b.book_category, t.trade_id, t.bookdesc_sell, t.bookdesc_buy
from textbook b, trade t, student s
where t.studentid_sell = b.book_id and s.student_id = t.studentid_sell

create or replace view TradeView2
(studentid_sell, book_id, author, title, qty, clearing_price, trader_name, timestamp, e-mail, new_flag,
  book_category, trade_id, bookdesc_sell, bookdesc_buy) AS
select t.studentid_sell, b.book_id, b.author, b.title, t.qty, t.clearing_price, s.trader_name, t.timestamp,
  s.e_mail, t.new_flag, b.book_category, t.trade_id, t.bookdesc_sell, t.bookdesc_buy
from textbook b, trade t, student s
where t.studentid_sell = b.book_id and s.student_id = t.studentid_sell
Procedures

create or replace procedure AddBid
(p_studentid student.student_id%type :=null) as
  p_status log_check.result%type;
begin
  /*check if the student is logged in*/
  select result into p_status
  from log_check
  where student_id = p_studentid and timestamp =
    (select max(timestamp) from log_check where student_id = p_studentid);
  if p_status <> 'S' then notlogged;
  else /*the student is logged in*/
    begin
      style;
      htp.p('<style>table{font-family:verdana;font-size:12px}.small
        {font-size:10px;color:black}</style>');</n      htp.centerOpen;
      htp.header(2,'Bid Submittion Form');
      htp.centerClose;
      htp.formOpen('addnewbid');
      htp.preOpen;
      htp.formhidden('p_studentid',p_studentid);
      htp.formselectopen('newbidtype','<b>BID TYPE:              </b>');
      htp.formselectoption('choose');
      htp.formselectoption('sell');
      htp.formselectoption('buy');
      htp.formselectclose;
      htp.bold('YOU NEED TO ENTER <i><u>EITHER</u></i> BOOK ID (if available) <i><u>OR</u></i> CATEGORY, AUTHOR, and TITLE');
      htp.p('<b>OPTION 1: Book ID:     </b>'|| htf.formText('bookid') || '<i> Format: NUMERIC</i>');
      htp.formselectopen('bookcategory','<b>OPTION 2: Category:    </b>');
      htp.formselectoption('choose');
      htp.formselectoption('Arts');
      htp.formselectoption('Biology');
      htp.formselectoption('Business');
      htp.formselectoption('Computer Science');
      htp.formselectoption('Education');
      htp.formselectoption('Mathematics');
      htp.formselectoption('Natural Sciences');
      htp.formselectoption('Physics');
      htp.formselectoption('Other');
      htp.formselectclose;
      htp.p('<b>          Author:      </b>'|| htf.formText('newauthor') || '<i> Format: Last_Name, F.I., Last_Name, F.I., ...</i>');
      htp.p('<b>          Title:       </b>'|| htf.formText('newtitle')||'<i> Format: please NO quotation marks and abbreviations</i>');
      htp.p('<i>NOTE: Description is an optional field, it is <b>NOT</b> used in matching bids. It is only displayed when people are searching through books.</i>');
      htp.p('</td></tr></table>');
      htp.p('<b>DESCRIPTION:           </b><textarea name="newdescription" rows="3" cols="17">''
        Example: edition, publisher, ISBN, book condition (new/used)</textarea><i><a class=small>optional!</a></i>');
      htp.p('<b>PRICE:                $</b>'|| htf.formText('newprice')||'<i> Format: 1000.00 ONLY DIGITS with ONE DOT</i>');
      htp.p('<b>QUANTITY:              </b>'|| htf.formText('newquantity')||'<i> Format: NUMERIC</i>');
      htp.p('</b></tr></td></table>');
    end;
  end;
end;
create or replace procedure AddForm
(  myfirstname student.first_name%type := null,
  mylastname student.last_name%type := null,
  mystreet student.street_address%type := null,
  mycity student.city%type := null,
  mystate student.state%type := null,
  myzip student.zip%type := null,
  myphone student.phone_number%type := null,
  myemail student.e_mail%type := null,
  mytradername student.trader_name%type := null) as
  templastname student.last_name%type;
  tempname student.username%TYPE;
  tempid student.student_id%type;
  p_trader student.trader_name%type;
begin
  style;
  /*check for null values in the input fields*/
  if myfirstname is null or mylastname is null or mystreet is null or mycity is null or mystate is null or myzip is null or myemail is null or mytradername is null then
    begin
      htp.header(4,'Sorry, you have to do this only once ;-) But you have to fill out ALL the fields!');
      htp.header(4,'Please go back in your browser and add missing entries.');
      htp.header(4,'Thank you for understanding!');
      htp.br;
    end;
  else /*all input values were not nulls*/
    begin
      /*check if the student is already registered based on name and address entered*/
      select last_name into templastname
      from student
      where first_name = myfirstname and last_name = mylastname and street_address = mystreet;
      begin
        htp.header(4,'Sorry, you are already registered');
        htp.anchor('checkid','Please click here to login to the system');
      end;
      EXCEPTION when no_data_found then /*name and address is not in the database*/
        begin
          /*check if the trader name entered is already taken by some other user*/
          select trader_name into p_trader
          from student
          where trader_name = mytradername;
          begin
            htp.header(4,'The trader name you entered is already used by some other user.');?>
            htp.header(4,'Please go back in your browser and enter a different trader name.');
            htp.header(4,'Thank you!');
          end;
          EXCEPTION when no_data_found then /*trader name is not taken by any user*/
            begin
              /*get student id from the new student from sequence*/
              select student_id_seq.nextval into tempid from dual;
              /*create username by combining first name and student id values*/
              select concat(myfirstname,tempid) into tempname from dual;
              /*create new student record*/
              insert into student values (tempid, myfirstname, mylastname, mystreet, mycity,
              mystate, myzip, myphone, myemail, tempname, mycity, mytradername);
            end;
      end;
    end;
end;
htp.header(4, myfirstname || ' ' || 'you have been registered!');
htp.header(5, 'Please take a note of your username and password. You will need them to login to the system!!');
htp.header(5, 'You will be able to change them later ');
htp.header(4, 'Username: ' || tempname || '');
htp.header(4, 'Password: ' || mycity || '');
htp.anchor('checkid', 'Please click here to login to the system');
end;
end;
end if;
footer;
htp.bodyClose;
htp.htmlClose;
end;
create or replace procedure AddNewBid
( p_studentid student.student_id%type := null,
newbidtype bid.bid_type%type := null,
bookid varchar2 := null,
bookcategory textbook.book_category%type := null,
newauthor textbook.author%type := null,
newtitle textbook.title%type := null,
newdescription bid.book_description%type := null,
newprice varchar2 := null,
newquantity varchar2 := null) as
p_status log_check.result%type;
p_bookid textbook.book_id%type;
p_author textbook.author%TYPE;
p_title textbook.title%type;
p_category textbook.book_category%type;
p_bidid bid.bid_id%type;
all_ok number := 0;
begin
select result into p_status
from log_check
where student_id = p_studentid and timestamp =
(select max(timestamp) from log_check where student_id = p_studentid);
if p_status <> 'S' then notlogged;
else /*the student is logged in*/
begin
style;
/*check for null values in the input fields*/
if p_studentid is null or newbidtype = 'choose' or (bookid is null and
(newauthor is null or newtitle is null or bookcategory = 'choose')) or
newprice is null or newprice < 0.01 or newquantity is null or newquantity < 1
then
begin
htp.bold('Sorry, but in order to have your bid submitted you
have to fill out ALL the fields except book id or author, title, and category! <br>');
htp.bold('<i>NOTE: price and quantity can NOT be zeroes or negative values. </i><br>');
htp.bold('Please go back in your browser and correct the registration form. <br>');
htp.bold('Thank you!');
end;
else /*all required values are entered*/
begin
if bookid is not null and (newauthor is not null or newtitle is not null or bookcategory <> 'choose')
then begin
htp.bold('Sorry, but in order to have your bid submitted you need to fill EITHER book id OR
author, title, and category. <br> NOT BOTH OPTIONS!</br>');</htp.bold('Please go back in your browser and correct the registration form. <br>');
htp.bold('Thank you!');
end;
else
begin

/*check if book id is entered*/
if bookid is not null then
begin
/*check if the bookid entered exists in the database*/
select book_id into p_bookid
from textbook
where book_id = bookid;
begin
/*check if the student already has bids on this book id*/
select bid_id into p_bidid
from bid
where book_id = p_bookid and student_id = p_studentid;
begin
htp.bold('<br>Sorry, you can NOT submit multiple bids on the same textbook.<br>');
end;
/*the student does not have any bids on this book id*/
exception when no_data_found then then all_ok := 1;
when others then errorpage;
end;
exception when no_data_found then
/*book id value entered is not in the database*/
begin
htp.bold('Sorry, but the BOOK ID you entered is not in the database. You can: <br>');
htp.bold('(1) check book ID and try again OR <br>');
htp.bold('(2) use category, author, and title fields to register a new book.<br>');
htp.bold('Thank you!');
end;
when others then errorpage;
end;
else /*book id is null */
begin
select initcap(newauthor) into p_author from dual;
select initcap(newtitle) into p_title from dual;
/*check if the book is already registered using author and title*/
select book_id, book_category into p_bookid, p_category
from textbook
where author = p_author and title = p_title;
begin
/*check if the student already has bids on this book id*/
select bid_id into p_bidid
from bid
where book_id = p_bookid and student_id = p_studentid;
begin
htp.bold('<br>Sorry, you can NOT submit multiple bids on the same textbook. <br>');
end;
exception when no_data_found then /*the student doesn’t have bids on the book id*/
begin
htp.header(4,'The Author and Title you entered already exist in the database:);
htp.header(3,'<i>BOOK ID '||p_bookid||'</i> in the <i>''||p_category||'' category</i>.');
htp.header(4,'You have two options:');
htp.bold('1) Use BOOK ID '||p_bookid||' to submit your bid. ');
htp.p('"<i>Note: in this case, you first need to </i>"');
htp ancor('searchbookid?p_studentid='||p_studentid||'&bookid='||p_bookid||'
'&showbids=0','<b>CHECK</b>');
htp.p('other users’ bids on that textbook, to make sure you are describing the same book –
edition, year, publisher). If there are no submitted bids you can use it for your book without
any doubts :-)<</i> </b>');
htp.p('"<b>After you have '</b>');
htp ancor('searchbookid?p_studentid='||p_studentid||'&bookid='||p_bookid||'
'&showbids=0','<b>checked</b>');
htp.p('other user’s bids and are sure about identity of your textbook, you can go back
to the Bid Submittion Form and in Option 1 add value ''||p_bookid||'' for Book ID, then
remove entries in Option 2 for Category, Author and Title fields. <br>');
htp.bold('2) Change your entry (for example, by adding a year into the title) and submit
your book as different from existing textbooks. It will be assigned a new BOOK ID.>');
htp.p('"<i>Note: in this case, your bid will NOT be matched with bids for BOOK ID
''||p_bookid||'.</i> »</b>');
end;
end;
end;
end;
You are encouraged to use existing BOOK ID when there are no submitted bids. In case there are bids submitted for this BOOK ID, you can still use it if you have read the descriptions and they DO match your book. If the descriptions of the submitted bids do NOT match your book, you need to use option 2 and submit your book (after minor modifications in the title field) as a new textbook.

Thank you!

begin
    select book_id_seq.nextval into p_bookid from dual;
    insert into textbook values (p_bookid, p_author, p_title, bookcategory);
    all_ok := 1;
    when others then errorpage;
    end;
end;
end;

create or replace procedure AddTrade
/*after a new trade record is added into the table trade, table BID is updated to reflect new quantities, also bids that has been satisfied are deleted from table BID*/
(p_bookid textbook.book_id%TYPE,
 p_studentid student.student_id%type,
 p_studentcheck number) AS
r_sidsell possible.studentid_sell%type;
 r_lowsell possible.low_sell%type;
 r_qtysell possible.qty_sell%type;
 r_sidbuy possible.studentid_buy%type;
 r_highbuy possible.high_buy%type;
 r_qtubuy possible.qty_buy%type;
 r_trade trade.trade_id%type;
 r_qty possible.qty_buy%type;
 r_bidprice bid.bid_price%type;
 r_descsell trade.bookdesc_sell%type;
 r_descbuy trade.bookdesc_buy%type;
begin
    begin
        /*get all information about traders from table possible*/
        select studentid_sell,low_sell,qty_sell,studentid_buy,high_buy,qty_buy
        into r_sidsell,r_lowsell,r_qtysell,r_sidbuy,r_highbuy,r_qtubuy
        from possible
        where book_id = p_bookid;
        exception when no_data_found then errorpage;
        end;
    end;
select book_description into r_descsell
from bid
where book_id = p_bookid and student_id = r_sidsell;
exception when others then r_descsell := null;
end;
begin
select book_description into r_descbuy
from bid
where book_id = p_bookid and student_id = r_sidbuy;
exception when others then r_descbuy := null;
end;
select trade_id_seq.nextval into r_trade from dual;
if r_qtysell < r_qtybuy then /*quantity to sell is less than quantity to buy*/
begin
  r_qty := r_qtysell; /*trade quantity will be equal to quantity to sell*/
  r_bidprice := (r_lowsell + r_highbuy) / 2;
  insert into trade values (r_trade,p_bookid,r_sidsell,r_sidbuy,r_qty,
    r_bidprice,sysdate,r_descsell,r_descbuy, 0,0);
  checkstatistics(p_bookid,r_trade);
  insert into log_table values ('D',sysdate,r_sidsell,p_bookid,'sell',r_lowsell,r_qtysell,null,null);
  delete from bid where book_id = p_bookid and student_id = r_sidsell;
  r_qty := r_qtybuy - r_qty;
  insert into log_table values ('U',sysdate,r_sidbuy,p_bookid,'buy',r_highbuy,r_qtybuy,r_highbuy,r_qty);
  update bid set quantity = r_qty, updated_flag = 2
  where book_id = p_bookid and student_id = r_sidbuy;
end;
else /*quantities to sell and to buy are equal*/
begin
  r_qty := r_qtybuy; /*trade quantity will be equal to quantity to buy*/
  r_bidprice := (r_lowsell + r_highbuy) / 2;
  insert into trade values (r_trade,p_bookid,r_sidsell,r_sidbuy,r_qty,
    r_bidprice,sysdate,r_descsell,r_descbuy,0,0);
  checkstatistics(p_bookid,r_trade);
  insert into log_table values ('D',sysdate,r_sidbuy,p_bookid,'buy',r_highbuy,r_qtybuy,null,null);
  delete from bid where book_id = p_bookid and student_id = r_sidbuy;
  r_qty := r_qtysell - r_qty;
  insert into log_table values ('U',sysdate,r_sidsell,p_bookid,'sell',r_lowsell,r_qtysell,r_lowsell,r_qty);
  update bid set quantity = r_qty, updated_flag = 2
  where book_id = p_bookid and (student_id = r_sidsell or student_id = r_sidbuy);
end;
else
begin
  r_qty := r_qtysell - r_qtybuy;
  r_bidprice := (r_lowsell + r_highbuy) / 2;
  insert into trade values (r_trade,p_bookid,r_sidsell,r_sidbuy,r_qty,
    r_bidprice,sysdate,r_descsell,r_descbuy,0,0);
  checkstatistics(p_bookid,r_trade);
  insert into log_table values ('D',sysdate,r_sidsell,p_bookid,'sell',r_lowsell,r_qtysell,null,null);
  delete from bid where book_id = p_bookid and student_id = r_sidsell;
  r_qty := r_qtybuy - r_qty;
  insert into log_table values ('U',sysdate,r_sidbuy,p_bookid,'buy',r_highbuy,r_qtybuy,r_highbuy,r_qty);
  update bid set quantity = r_qty, updated_flag = 2
  where book_id = p_bookid and student_id = r_sidbuy;
end;
end if;
end if;
st style;
if p_studentcheck = 0 then
  /*print only if student id matches the student id of the user logged in*/
begin
  htp.bold('Your Bid was successfully submitted and resulted in TRADE!!!');
  htp.bold('<br>You can');
  htp.print('<a target="theFrame"');
  htp.anchor('checktrade?studentid='||p_studentid||'&show=0','check your trades');
  htp.print('</a>');
  htp.bold(' to see all information related to this bid.');
  htp.bold('<br><br>Our warmest congratulations on your purchase :-) Good luck');
end;
create or replace procedure AddUser as
begin
  style;
  htp.print('"a name="f_adduser">');
  htp.centerOpen;
  htp.htitle('New User Registration Form');
  htp.centerClose;
  htp.bold('* required field');
  htp.formOpen('addform');
  htp.preOpen;
  htp.header(3,'PRIVATE DATA: (can NOT be seen by other users of the system)');
  htp.p('<b>FIRST NAME*:     </b>'|| htf.formText('myfirstname'));
  htp.p('<b>LAST NAME*:      </b>'|| htf.formText('mylastname'));
  htp.p('<b>STREET ADDRESS*: </b>'|| htf.formText('mystreet'));
  htp.p('<b>CITY*:           </b>'|| htf.formText('mycity'));
  htp.p('<b>STATE*:          </b>'|| htf.formText('mystate'));
  htp.p('<b>ZIP CODE*:       </b>'|| htf.formText('myzip'));
  htp.p('<b>PHONE NUMBER:    </b>'|| htf.formText('myphone'));
  htp.header(3,'<br>PUBLIC DATA: (is shown to other users of the system in all your transactions.)');
  htp.p('<b>E-MAIL*:         </b>'|| htf.formText('myemail'));
  htp.p('<b>TRADER NAME*:    </b>'|| htf.formText('mytradername'));
  htp.preClose;
  htp.paragraph;
  htp.formSubmit;
  htp.formClose;
end;
create or replace procedure CheckBids
(p_studentid student.student_id%type ) as
  p_check student.student_id%type;
  p_status log_check.result%type;
  p_check2 number(1) := 0;
begin
  /*check if the student is logged in*/
  select result into p_status
  from log_check
  where student_id = p_studentid and timestamp =
    (select max(timestamp) from log_check where student_id = p_studentid);
  if p_status <> 'S' then notlogged;
  else /*the student is logged in*/
    style;
    htp.centerOpen;
    htp.header(2,'Your Outstanding Bids');
    htp.centerClose;
    /*check if there is at least one bid*/
    select student_id into p_check from studentbidview
    where student_id = p_studentid and rownum = 1;
    begin
      printbids(p_studentid);
    end;
    exception when no_data_found then /*the student does not have any bids*/
      begin
        p_check2 := 1;
        htp.bold('There are no bids');
      end;
end;
end;
if p_check2 = 0 then /*print only when there is at least one bid*/
begin
    htp("<br><br><i>NOTE: You can UPDATE price, quantity and/or description only of the
bids that were submitted/modified at least 2 hours ago.<br>*/
    htp.p("You can DELETE only the bids that were submitted/modified at least 24 hours ago.<br>");
    htp.bold("<br>To modify a bid: (1) note BOOK ID, (2) click on one of the links below.<br>");
    htp.listitem(htf.anchor('updatebid?p_studentid='||p_studentid||'','Update a bid'));
    htp.br;
    htp.listitem(htf.anchor('deletebid?p_studentid='||p_studentid||'','Delete a bid'));
end;
end if;
end;
end if;
end;
end;
create or replace procedure CheckID as
begin
    style;
    htp.print('<a name="f_checkid">');</n    htp.centerOpen;
    htp.htitle ('Login Screen');
    htp.formOpen('checknow');
    htp.preOpen;
    htp.p(htf.header(4,'USERNAME: '|| htf.formText('acceptname')));
    htp.p(htf.header(4,'PASSWORD: '|| htf.formPassword('acceptpwd')));
    htp.preClose;
    htp.paragraph;
    htp.formSubmit;
    htp.formClose;
    htp.centerClose;
    footer;
    htp.bodyclose;
    htp.htmlClose;
end;
create or replace procedure CheckNow
(accepnname student.username%type :=Null,
acceptpwd student.password%type :=Null) AS
p_password student.password%type;
p_studentid student.student_id%type;
BEGIN
    style;
    /*get student id and password from student records based on the username entered*/
    select student_id, password into p_studentid, p_password
    from student
    where username = acceptname;
    /*check if password entered match the one in the database*/
    if p_password = acceptpwd then
        begin
            /*logon the user*/
            insert into log_check values (p_studentid,sysdate,'S');
            htp.header(4,'Login Successful ');
            htp.header(5,'Username: '||acceptname||' has been successfully logged in.');
            htp.anchor('main?p_studentid='||p_studentid ||'', 'Please click here to go to your main page');
            footer;
            htp.bodyclose;
            htp.htmlClose;
        end;
    else /*password do not match*/
        begin
            htp.header(4,'Sorry incorrect password!');
            htp.anchor('checkid','Please click here to try logging again');
        end;
end;
insert into log_check values (p_studentid,sysdate,'F');
footer;
htp.bodyclose;
htp.htmlclose;
end;
end if;
EXCEPTION when no_data_found then /*username is not in the database*/
begin
htp.header(4,'Sorry, you are not in the database!');
htp.anchor('rules','Please click here to register!');
footer;
htp.bodyclose;
htp.htmlclose;
end;
END;

create or replace procedure CheckOtherBuys
(p_studentid student.student_id%type := null,
bookcategory textbook.book_category%type := null ) as
p_status log_check.result%type;
p_check textbook.book_category%type;
p_print number := 0;
begin
select result into p_status
from log_check
where student_id = p_studentid and timestamp =
(select max(timestamp) from log_check where student_id = p_studentid);
if p_status <> 'S' then notlogged;
else /*the student is logged in*/
begin
style;
if bookcategory = 'all' then
begin
select book_category into p_check from buybidview
where student_id <> p_studentid and rownum = 1;
p_print := 1;
exception when no_data_found then htp.p('<br>There are no textbooks in demand<br>');
end;
else
begin
select book_category into p_check from buybidview
where book_category = bookcategory and student_id <> p_studentid and rownum = 1;
p_print := 1;
exception when no_data_found then
htp.p('<br>There are no textbooks in demand in the <i><b>'||bookcategory||'</b></i> category<br>');
end;
end if;
if p_print = 1 then
begin
if bookcategory <> 'all' then
begin
htp.print('<i>');
htp.header(3,bookcategory);
htp.print('</i>');
end;
end if;
printbuybids(bookcategory,p_studentid);
end;
else
htp.bodyclose;
htp.htmlclose;
end;
end if;
end;
create or replace procedure CheckOtherSells
(p_studentid student.student_id%type := null,
bookcategory textbook.book_category%type := null ) as
p_status log_check.result%type;
p_check textbook.book_category%type;
p_print number := 0;
begin
  select result into p_status
  from log_check
  where student_id = p_studentid and timestamp =
    (select max(timestamp) from log_check where student_id = p_studentid);
if p_status <> 'S' then notlogged;
else /*the student is logged in*/
  begin
    style;
    if bookcategory = 'all' then
      begin
        select book_category into p_check from sellbidview
        where student_id <> p_studentid and rownum = 1;
        p_print := 1;
        exception when no_data_found then
          htp.p('<br>There are no textbooks for sale<br>');
      end;
    else
      begin
        select book_category into p_check from sellbidview
        where book_category = bookcategory and student_id <> p_studentid and rownum = 1;
        p_print := 1;
        exception when no_data_found then
          htp.p('<br>There are no textbooks for sale in the <i><b>'||
            bookcategory||'</b></i> category<br>');
      end;
    end if;
    if p_print = 1 then
      begin
        if bookcategory <> 'all' then
          begin
            htp.print('<i>');
            htp.header(3,bookcategory);
            htp.print('</i>');
          end;
          printsellbids(bookcategory,p_studentid);
        htp.br;
        htp.bold('PLEASE NOTE BOOK_ID OF THE TEXTBOOK YOU WISH TO PURCHASE');
      end;
    end if;
  end if;
end;
end if;
end;
end;
create or replace procedure CheckStatistics
(p_bookid textbook.book_id%type,
p_tradeid trade.trade_id%type) as
t_book_category varchar(20);
t_qty number;
t_price number;
t_value number;
begin
  begin
    select book_category into t_book_category
    from textbook
    where book_id = p_bookid;
  end;
begin
  begin
    select book_category into t_book_category
    from sellbidview
    where student_id <> p_studentid and rownum = 1;
    t_print := 1;
    exception when no_data_found then
      htp.p('<br>There are no textbooks for sale in the <i><b>'||
        bookcategory||'</b></i> category<br>');
  end;
  if t_print = 1 then
    begin
      if bookcategory <> 'all' then
        begin
          htp.print('<i>');
          htp.header(3,bookcategory);
          htp.print('</i>');
        end;
        printsellbids(bookcategory,p_studentid);
      htp.br;
      htp.bold('PLEASE NOTE BOOK_ID OF THE TEXTBOOK YOU WISH TO PURCHASE');
    end;
  end if;
end.
end;

create or replace procedure CheckStatistics
(p_bookid textbook.book_id%type,
p_tradeid trade.trade_id%type) as
t_book_category varchar(20);
t_qty number;
t_price number;
t_value number;
begin
  begin
    select book_category into t_book_category
    from textbook
    where book_id = p_bookid;
  end;
begin
  begin
    select book_category into t_book_category
    from sellbidview
    where student_id <> p_studentid and rownum = 1;
    t_print := 1;
    exception when no_data_found then
      htp.p('<br>There are no textbooks for sale in the <i><b>'||
        bookcategory||'</b></i> category<br>');
  end;
  if t_print = 1 then
    begin
      if bookcategory <> 'all' then
        begin
          htp.print('<i>');
          htp.header(3,bookcategory);
          htp.print('</i>');
        end;
        printsellbids(bookcategory,p_studentid);
      htp.br;
      htp.bold('PLEASE NOTE BOOK_ID OF THE TEXTBOOK YOU WISH TO PURCHASE');
    end;
  end if;
end.
end;
exception when others then errorpage;
end;

begin
select qty, clearing_price
into t_qty, t_price
from trade
where trade_id = p_tradeid;
exception when others then errorpage;
end;

if checkauction(t_book_category) = 1 then
/*category already exists in the table auction_detail*/
begin
update auction_detail
set number_of_trades = number_of_trades + 1, total_qty = total_qty + t_qty,
total_value = total_value + (t_price * t_qty)
where book_category = t_book_category;
end;
else /*category does not exist in the table auction_detail*/
begin
    t_value := t_price * t_qty;
    insert into auction_detail values (t_book_category,1,t_qty,t_value);
end;
end if;
end;

create or replace procedure CheckTrade
(p_studentid student.student_id%type,
show number) as
p_check trade.trade_id%type;
p_status log_check.result%type;
p_check2 number(1) := 0;
p_check3 number(1) := 0;

begin
/*check if the student is logged in*/
select result into p_status
from log_check
where student_id = p_studentid and timestamp =
(select max(timestamp) from log_check where student_id = p_studentid);
if p_status <> 'S' then notlogged;
else
begin /*the student is logged in*/
    style;
    begin
    /*check if the student have bought any books*/
    select trade_id into p_check from trade
    where studentid_buy = p_studentid and rownum = 1;
    begin
    htp.centerOpen;
    htp.header(2,'Textbooks Bought');
    if show = 0 then
        htp.anchor('checktrade?p_studentid='||p_studentid||'&show=1','SHOW descriptions');
    else
        htp.anchor('checktrade?p_studentid='||p_studentid||'&show=0','HIDE descriptions');
    end if;
    htp.centerClose;
    htp.br;
    printtrades2(p_studentid, show);
    exception when no_data_found then /*no textbooks bought*/
    begin
    p_check2 := 1;
    htp.bold('<br><br>There were no textbooks bought<br>');
    end;
    end;
    begin
/*check if the student have sold any books*/
select trade_id into p_check from trade
where studentid_sell = p_studentid and rownum =1;
begin
  htp.centerOpen;
  htp.centerClose;
end;

begin
  htp.centerOpen;
  htp.centerClose;
end;

exception when no_data_found then /*no textbooks sold*/
begin
  p_check3 := 1;
  htp.bold('<br><br>There were no textbooks sold<br>');
end;
end;

if p_check2 = 0 or p_check3 = 0 then /*print only if there was at least one trade*/
begin
  htp.bold('<br><br><i>NOTE: You are encouraged to contact the other trader(s) and to
  check the details about the book(s). You need to e-mail the person to arrange the exchange
  of the textbook.</i>');
end;
end if;

update trade set new_flag=1 where studentid_sell = p_studentid;
update trade set new_flag_b=1 where studentid_buy = p_studentid;
htp.bodyclose;
htp.htmlclose;
end;
end;

create or replace procedure DeleteBid
(p_studentid student.student_id%type :=null) as
  p_status log_check.result%type;
begin
/*check if the student is logged in*/
select result into p_status
from log_check
where student_id = p_studentid and timestamp =
  (select max(timestamp) from log_check where student_id = p_studentid);
if p_status <> 'S' then notlogged;
else begin /*the student is logged in*/
style;
  htp.p('<style>table{font-family:verdana;font-size:12px}</style>');
  htp.centerOpen;
  htp.header(2,'Delete Bid Form');
  htp.centerClose;
  htp.formOpen('deletebidnow');
  htp.preOpen;
  htp.formhidden('p_studentid',p_studentid);
  htp.p('<table width="90%"><tr><td>');
  htp.p('<i>NOTE: Only bids submitted/modified more that 24 hours ago can be deleted!</i>');
  htp.p('</td></tr>');
  htp.p('BOOK ID:        </b>'|| htf.formText('bookid')||'<i> Format: NUMERIC</i>');
  htp.preClose;
  htp.formSubmit;
  htp.formClose;
  htp.bodyclose;
  htp.htmlClose;
except
create or replace procedure DeleteBidNow
(p_studentid student.student_id%type := null,
bookid varchar2 := null) as
begin
/*check if the student is logged in*/
select result into p_status
from log_check
where student_id = p_studentid and timestamp =
(select max(timestamp) from log_check where student_id = p_studentid);
if p_status <> 'S' then notlogged;
else /*the student is logged in*/
begin
/*check for null input values */
if p_studentid is null or bookid is null then
begin
htp.bold('Please go back in your browser and enter book identification number.');
htp.bold('<br>Thank you!');
end;
else /*all values entered are not nulls*/
begin
/*check if bid exists */
select timestamp, bid_type, bid_price, quantity
into p_datecheck, p_bidtype, p_price, p_qty
from bid
where book_id = bookid and student_id = p_studentid;
begin
/*check that the bid was submitted more than 24 hours ago*/
select sysdate into p_date from dual;
if (p_date - p_datecheck) < 1 then
begin
htp.bold('<br>Sorry, you can not delete bids submitted less than 24 hours ago<br>');
htp.br;
htp.br;
htp.anchor('checkbids?p_studentid='||p_studentid||'','Go back to your bids');
end;
else /*the bid was submitted more than 24 hours ago*/
begin
delete from bid
where book_id = bookid and student_id = p_studentid;
insert into log_table values ('D',sysdate, p_studentid, bookid, p_bidtype, p_price, p_qty, null, null);
if p_bidtype = 'sell' then
begin
if checkbidsell(bookid) = 1 then /*there are more sell bids for this book id*/
begin
select min(bid_price) into t_lowsell
from bid
where book_id = bookid and bid_type = 'sell';
end;
end;
end;
end;
end;
end;
end if;
end;
end;
begin
select quantity, student_id into t_qtysell, t_studentid
from bid
where book_id = bookid and bid_price = t_lowsell and bid_type = 'sell' and
    timestamp = (select min(timestamp) from bid where book_id = bookid and
    bid_price = t_lowsell and bid_type = 'sell');
end;
begin
update possible set qty_sell = t_qtysell, low_sell = t_lowsell, studentid_sell = t_studentid
where book_id = bookid;
end;
if p_studentid <> t_studentid then t_studentcheck := 1; end if;
updatetrade(bookid, t_studentid, t_studentcheck);
end;
else /* no more sell bids*/
begin
update possible set low_sell = null, qty_sell = null, studentid_sell = null
where book_id = bookid and studentid_sell = p_studentid;
end;
end if;
end;
else /*bid type is "buy" */
begin
if checkbidbuy(bookid) = 1 then /*there are more buy bids on this book id*/
begin
begin
select max(bid_price) into t_highbuy
from bid
where book_id = bookid and bid_type = 'buy';
end;
begin
select quantity, student_id into t_qtybuy, t_studentid
from bid
where book_id = bookid and bid_price = t_highbuy and bid_type = 'buy' and
    timestamp = (select min(timestamp) from bid where book_id = bookid and
    bid_price = t_highbuy and bid_type = 'buy');
end;
begin
update possible set qty_buy = t_qtybuy, high_buy = t_highbuy, studentid_buy = t_studentid
where book_id = bookid;
end;
if p_studentid <> t_studentid then t_studentcheck := 1; end if;
updatetrade(bookid, t_studentid, t_studentcheck);
end;
else /*no more buy bids exist*/
begin
update possible set high_buy = null, qty_buy = null, studentid_buy = null
where book_id = bookid;
end;
end if;
end if;
if t_studentcheck = 0 then
begin
htp.bold('Your bid was succesfully deleted.<br>');
htp.bold('Good luck with future bids :) Thank you!');
htp.br;
htp.br;
htp.anchor('checkbids?p_studentid='||p_studentid||'','Go back to your bids');
end;
end if; /* if (p_date - p_datecheck) < 1*/
end;
exception when no_data_found then /*bid id entered does not exist in the database*/
create or replace procedure EmailChange
(p_studentid student.student_id%type := null) as
  p_status log_check.result%type;

begin
  /*check if the student is logged in*/
  select result into p_status
  from log_check
  where student_id = p_studentid and timestamp =
    (select max(timestamp) from log_check where student_id = p_studentid);
  if p_status <> 'S' then notlogged;
  else begin
    style;
    htp.formOpen('emailnow');
    htp.preOpen;
    htp.formhidden('p_studentid', p_studentid);
    htp.p('<b>NEW E-MAIL ADDRESS:      </b>' || htf.formText('newemail1'));
    htp.p('<b>RE-ENTER E-MAIL ADDRESS: </b>' || htf.formText('newemail2'));
    htp.preClose;
    htp.paragraph;
    htp.formSubmit;
    htp.formClose;
    htp.htmlClose;
  end;
  end if;
end;

create or replace procedure EmailNow
(p_studentid student.student_id%type := null,
 newemail1 student.e_mail%type := null,
 newemail2 student.e_mail%type := null) as
  p_status log_check.result%type;

begin
  /*check if the student is logged in*/
  select result into p_status
  from log_check
  where student_id = p_studentid and timestamp =
    (select max(timestamp) from log_check where student_id = p_studentid);
  if p_status <> 'S' then notlogged;
  else
    begin
    style;
    if newemail1 is null or newemail2 is null then
      begin
        htp.bold('Please go back in your browser and enter your new e-mail address in both fields.');
        htp.bold('<br>Thank you!');
      end;
    else /*both fields have been entered */
      begin
        if newemail1 <> newemail2 then
          begin
            htp.bold('Two e-mail address fields do not match!<br>');
          end;
        else
          begin
            /*...*/
          end;
        end;
      end;
    end;
  end;
create or replace procedure ErrorPage as
begin
  style;
  htp.centeropen;
  htp.header(5,'There was an ERROR submitting your request.');
  htp.centerclose;
  htp.bold('Please check your input and try again.<br>);
  htp.bold('NOTE: Possible cause of error might be entering characters in the numeric fields like
    bid ID, price, quantity; or trying to access incorrect URL.<br><br>
  htp.bold('If the problem persists, please contact the system administrator. Sorry for the
    inconvenience.');
  footer;
  htp.bodyclose;
  htp.htmlclose;
end;

create or replace procedure Footer as
begin
  htp.htmlopen;
  htp.print('<body bgcolor = aliceblue style="font-family:verdana;font-size:10px">');
  htp.centeropen;
  owa_util.signature;
  htp.p('Created by: ');
  htp.p(htf.italic('Maria Bondarenko'));
  htp.br;
  htp.mailto('mab89411@batman.tamucc.edu','Comments or problems');
  htp.centerclose;
  htp.bodyclose;
  htp.htmlclose;
end;

create or replace procedure Logoff
(p_studentid student.student_id%type) as
  p_status log_check.result%type;
begin
  /*check if the student is logged in*/
  select result into p_status
  from log_check
  where student_id = p_studentid and timestamp =
    (select max(timestamp) from log_check where student_id = p_studentid);
  if p_status <> 'S' then notlogged;
  else /*student is logged in*/
    begin
      style;
      htp.centeropen;

create or replace procedure Main
(p_studentid student.student_id%type := null) as
  p_tradername student.trader_name%type;
  p_status varchar2(1);
  p_check number := 0;
begin
  /*check if the user is logged in*/
  begin
    select result into p_status
    from log_check
    where student_id = p_studentid and timestamp =
      (select max(timestamp) from log_check where student_id = p_studentid);
    begin
      if p_status <> 'S' then notlogged;
    else
      /*user is logged into the system*/
      select trader_name into p_tradername
      from student
      where student_id = p_studentid;
      begin
        htp.htmlOpen;
        htp.headOpen;
        htp.htitle(p_tradername||''||'`s HOMEPAGE');
        htp.header(4,'Select an option');
        htp.print('<style>body{font-family:verdana}table{font-family:verdana;
        font-size:12px;color:Navy}body{background-color:#003399;color:aliceblue}
        a{Color:white;}a:hover{color:white;font-weight:bold}
        a.nav{Color:aliceblue;text-decoration:none;font-weight:bold;width:100%;
        border:1px outset #009933;padding:3px;background-color:navy}
        a:hover.nav{color:navy;background-color:aliceblue;border:1px inset Beige}
        </style>');
        htp.headClose;
        htp.print('<body bottommargin="0" topmargin=1px rightmargin=3px leftmargin=1px>);
        htp.print('<tr><td valign="top" width="100px" style="">');
        htp.print('<a class=nav target="theFrame"');
        htp.anchor('searchform?p_studentid='||p_studentid||'','Search');
        htp.print('</a><a class=nav target="theFrame"');
        htp.anchor('addbid?p_studentid='||p_studentid||'','Place a Bid');
        htp.print('</a><a class=nav target="theFrame"');
        htp.anchor('otherbuybids?p_studentid='||p_studentid||'','Books in Demand');
        htp.print('</a><a class=nav target="theFrame"');
        htp.anchor('othersellbids?p_studentid='||p_studentid||'','Available Books');
        htp.print('</a><a class=nav target="theFrame"');
        htp.anchor('checkbids?p_studentid='||p_studentid||'','Your Bids');
        htp.print('</a><a class=nav target="theFrame"');
        htp.anchor('checktrade?p_studentid='||p_studentid||'&show=0','Your Trades');
        htp.print('</a><a class=nav target="theFrame"');
        htp.anchor('statistics?p_studentid='||p_studentid||'','Auction Statistics');
        htp.print('</a><a class=nav target="theFrame"');
        htp.anchor('updateinfo?p_studentid='||p_studentid||'','Update Account Info');
        htp.print('</a><a class=nav target="theFrame"');
    </body>');
create or replace procedure NotLogged as
begin
  htp.centeropen;
  htp.header(5,'You have to login in order to use the system');
  htp.centerclose;
  htp.bold('<br>Please go to the welcome page of the Auction');
  htp.bodyclose;
  htp.htmlclose;
end;

create or replace procedure OtherBuyBids
(p_studentid student.student_id%type := null ) as
  p_status log_check.result%type;
begin
  /*check if the student is logged in*/
  select result into p_status
  from log_check
  where student_id = p_studentid and timestamp =
    (select max(timestamp) from log_check where student_id = p_studentid);
  if p_status <> 'S' then notlogged;
  else
    /*the student is logged in*/
    htp.htmlopen;
    htp.headopen;
    htp.print('<style>body{font-family:verdana}table{font-family:verdana;
    font-size:12px;color:Navy;text-align:center}
    body{background-color:aliceblue;color:navy}a{Color:navy;}
    a:hover{font-weight:bold}
    a.nav{Color:aliceblue;text-decoration:none;font-weight:bold;
    width:100%;border:1px outset #009933;padding:3px;background-color:#003399}
    a:hover.nav{color:navy;background-color:aliceblue;border:1px inset Beige}
</style>');
    htp.headClose;
    htp.print('<body bottommargin=1px topmargin=13px rightmargin=1px leftmargin=1px>');
    htp.print('<table height=100% width=100% cellpadding=2px cellspacing=0px>');
    htp.print('<tr><td colspan=10>');
    htp.header(2,'Textbooks in Demand');
    htp.print('</td></tr><tr><td colspan=3>');
    htp.bold('Click on a category or select ');
    htp.print('<a target="innerFrame"');
    htp.anchor('checkotherbuys?p_studentid='||p_studentid||'&bookcategory=all','ALL');
    htp.print('</a>');
    htp.print('</td><td colspan=7>');
    htp.bold('Column "YOUR BIDS" shows if you have any bids for that textbook.');
    htp.bodyClose;
    htp.htmlClose;
end;
create or replace procedure OtherSellBids
(p_studentid student.student_id%type := null) as

begin
/*check if the student is logged in*/
select result into p_status
from log_check
where student_id = p_studentid and timestamp = (select max(timestamp) from log_check where student_id = p_studentid);
if p_status <> 'S' then notlogged;
else /*the student is logged in*/
begin

### Textbooks for Sale

Click on a category or select 'ALL'.

<table>
<thead>
<tr>
<th>Column</th>
<th>YOUR BIDS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arts</td>
<td></td>
</tr>
<tr>
<td>Biology</td>
<td></td>
</tr>
<tr>
<td>Business</td>
<td></td>
</tr>
<tr>
<td>Chemistry</td>
<td></td>
</tr>
<tr>
<td>Computer Science</td>
<td></td>
</tr>
<tr>
<td>Education</td>
<td></td>
</tr>
<tr>
<td>Mathematics</td>
<td></td>
</tr>
<tr>
<td>Natural Sciences</td>
<td></td>
</tr>
<tr>
<td>Physics</td>
<td></td>
</tr>
<tr>
<td>Other</td>
<td></td>
</tr>
</tbody>
</table>
create or replace procedure PrintBids
(p_studentid student.student_id%type)

begin
   p_lastcategory := 'new';
   htp.print('new means that the bid has NOT been modified and you
   have NOT looked at it after submission');
   htp.br;

   for v_print in print loop
   begin
      if p_lastcategory = 'new' or p_lastcategory <> v_print.book_category then
         begin
            htp.print('<i>');
            htp.header(3,v_print.book_category);
            htp.print('</i>');
            p_lastcategory := v_print.book_category;
            begin
               if v_print.updated_flag = 0 then htp.tabledata('new');
               elsif v_print.updated_flag = 2 then htp.tabledata('update');
               else htp.tabledata('-'); end if;
               htp.tabledata(to_char(v_print.timestamp,'mm/dd/yy, hh:mi am'));
               htp.tabledata(v_print.bid_type);
               htp.tabledata(v_print.book_id);
               htp.tabledata(v_print.author);
               htp.tabledata(v_print.title);
               htp.tabledata(v_print.bid_price);
               htp.tabledata(v_print.quantity);
               if v_print.book_description is null then htp.tabledata('N/A');
               else htp.tabledata(v_print.book_description); end if;
   end if;
end;
begin
    p_lasttype := 'new';
    htp.print('<style>
    Table{border:0px;font-family:verdana;font-size:12px}
    .flag{font-size:10px;color:red}
    .TD{Border:1px inset Blue}
    .TH{Border:1px inset Blue}
    </style>');
    for v_print in print loop
        begin
            if p_lasttype = 'new' or p_lasttype <> v_print.bid_type then
                begin
                    if p_lasttype <> 'new' then htp.tableclose; end if;
                    htp.print('<i>
                    if v_print.bid_type = 'sell' then htp.header(4,'Sell Bids');
                    else htp.header(4,'Buy Bids'); end if;
                    htp.print('</i>');
                    p_lasttype := v_print.bid_type;
                    htp.print('<table cellpadding=2px cellspacing=0px>
                    <tableheader<Date>
                    <tableheader<Quantity>
                    htp.tableheader('Book Description');
                    if showbids = 1 then htp.tableheader('Your Bids'); end if;
                    htp.tablerowclose;
                    htp.tablerowopen;
                    htp.tabledata(to_char(v_print.timestamp,'mm/dd/yy, hh:mi am'));
                    if v_print.book_description is null then htp.tabledata('N/A');
                    else htp.tabledata(v_print.book_description); end if;
                    htp.tableclose;
                end if;
            end if;
        end loop;
    end;
end;
else htp.tabledata(v_print.book_description); end if;
if showbids = 1 then
begin
begin
select bid_type into p_bidtype
from bid
where book_id = bookid and student_id = p_studentid;
exception when no_data_found then p_bidtype := null;
end;
if p_bidtype is null then htp.tabledata('-');
else htp.tabledata('<a class=flag>'||p_bidtype||'</a>'); end if;
end if;
end if;
htp.tablerowclose;
end;
else /*bid type is the same*/
begin
htp.tablerowopen;
htp.tabledata(to_char(v_print.timestamp,'mm/dd/yy, hh:mi am'));
htp.tabledata(v_print.quantity);
if v_print.book_description is null then htp.tabledata('N/A');
else htp.tabledata(v_print.book_description); end if;
if showbids = 1 then
begin
begin
select bid_type into p_bidtype
from bid
where book_id = bookid and student_id = p_studentid;
exception when no_data_found then p_bidtype := null;
end;
if p_bidtype is null then htp.tabledata('-');
else htp.tabledata('<a class=flag>'||p_bidtype||'</a>'); end if;
end if;
htp.tablerowclose;
end;
end loop;
htp.tableclose;
htp.p('<i>NOTE: Column "YOUR BIDS" shows if you have any bids for that textbook.
'); end;

create or replace procedure PrintBuyBids
(category textbook.book_category%type,
p_studentid student.student_id%type ) as
p_lastcategory textbook.book_category%type;
p_bidtype bid.bid_type%type := null;
cursor print1 is
select book_id, author, title, quantity, description, timestamp, book_category
from buybidview
where student_id <> p_studentid
order by book_category, timestamp, book_id;
cursor print2 is
select book_id, author, title, quantity, description, timestamp
from buybidview
where book_category = category and student_id <> p_studentid
order by timestamp, book_id;
begin
p_lastcategory := 'new';
htp.print('<style> Table{border:0px;font-family:verdana;font-size:12px}
.flagbuy{font-size:10px;color:blue}
.flagsell{font-size:10px;color:red}');
if category = 'all' then
begin
for v_print in print1 loop
if p_lastcategory = 'new' or p_lastcategory <> v_print.book_category then
begin
if p_lastcategory <> 'new' then htp.tableclose; end if;
htp.print('
');
htp.header(3,v_print.book_category);
htp.print('
');
p_lastcategory := v_print.book_category;
htp.print('<table cellpadding=1px cellspacing=0px>');
htp.tablerowopen;
htp.tableheader('Date');
htp.tableheader('Book ID');
htp.tableheader('Author');
htp.tableheader('Title');
htp.tableheader('Quantity');
htp.tableheader('Description');
htp.tableheader('Your Bids');
htp.tablerowclose;
htp.tablerowopen;
htp.tablerowopen;
htp.tabledata(to_char(v_print.timestamp,'mm/dd/yy, hh:mm am'));
htp.tablerowopen;
htp.tabledata(v_print.book_id);
htp.tablerowopen;
htp.tabledata(v_print.author);
htp.tablerowopen;
htp.tabledata(v_print.title);
htp.tablerowopen;
htp.tabledata(v_print.quantity);
if v_print.description is null then htp.tabledata('N/A');
else htp.tabledata(v_print.description); end if;
begin
select bid_type into p_bidtype
from bid
where book_id = v_print.book_id and student_id = p_studentid;
exception when no_data_found then p_bidtype := null;
end;
if p_bidtype is null then htp.tabledata('•');
else
if p_bidtype = 'sell' then htp.tabledata('<a class=flagsell>sell</a>');
else htp.tabledata('<a class=flagbuy>buy</a>');
end if;
end if;
htp.tablerowclose;
end;
else /*category is the same*/
begin
htp.tablerowopen;
htp.tablerowopen;
htp.tablerowopen;
htp.tablerowopen;
htp.tabledata(to_char(v_print.timestamp,'mm/dd/yy, hh:mm am'));
htp.tablerowopen;
htp.tabledata(v_print.book_id);
htp.tablerowopen;
htp.tabledata(v_print.author);
htp.tablerowopen;
htp.tabledata(v_print.title);
htp.tablerowopen;
htp.tabledata(v_print.quantity);
if v_print.description is null then htp.tabledata('N/A');
else htp.tabledata(v_print.description); end if;
begin
select bid_type into p_bidtype
from bid
where book_id = v_print.book_id and student_id = p_studentid;
exception when no_data_found then p_bidtype := null;
end;
if p_bidtype is null then htp.tabledata('•');
else
if p_bidtype = 'sell' then htp.tabledata('<a class=flagsell>sell</a>');
else htp.tabledata('<a class=flagbuy>buy</a>');
end if;
end if;
end;
end if;
htp.tablerowclose;
end;
end if;
end loop;
end if;
else /*specific category was entered*/
begin
htp.print('<table cellpadding=1px cellspacing=0px>');</
htp.tableheader('Date');
htp.tableheader('Book ID');
htp.tableheader('Author');
htp.tableheader('Title');
htp.tableheader('Quantity');
htp.tableheader('Description');
htp.tableheader('Your Bids');
htp.tableheader('Your Bids');
htp.tablerowclose;
for v_print in print2 loop
begin
htp.tablerowopen;
htp.tabledata(to_char(v_print.timestamp,'mm/dd/yy, hh:mi am'));
htp.tabledata(v_print.book_id);
htp.tabledata(v_print.author);
htp.tabledata(v_print.title);
htp.tabledata(v_print.quantity);
if v_print.description is null then htp.tabledata('N/A');
else htp.tabledata(v_print.description); end if;
begin
select bid_type into p_bidtype
from bid
where book_id = v_print.book_id and student_id = p_studentid;
exception when no_data_found then p_bidtype := null;
end;
if p_bidtype is null then htp.tabledata('-');
else
if p_bidtype = 'sell' then htp.tabledata('<a class=flagsell>sell</a>');
else htp.tabledata('<a class=flagbuy>buy</a>');
end if;
end if;
htp.tablerowclose;
end;
end loop;
end if;
htp.tableclose;
end;

create or replace procedure PrintSellBids
(category textbook.book_category%type,
p_studentid student.student_id%type ) as
p_lastcategory textbook.book_category%type;
p_bidtype bid.bid_type%type := null;
cursor print1 is
select book_id, author, title, quantity, description, timestamp, book_category
from sellbidview
where student_id <> p_studentid
order by book_category, timestamp, book_id;
cursor print2 is
select book_id, author, title, quantity, description, timestamp
from sellbidview
where book_category = category and student_id <> p_studentid
order by timestamp, book_id;
begin
p_lastcategory := 'new';
```sql
begin
    for v_print in print1 loop
        if p_lastcategory = 'new' or p_lastcategory <> v_print.book_category then
            if p_lastcategory <> 'new' then htp.tableclose; end if;
            htp.print('<i>');
            htp.header(3, v_print.book_category);
            htp.print('</i>');
            p_lastcategory := v_print.book_category;
            htp.print('<table cellpadding=1px cellspacing=0px>');
            htp.tablerowopen;
            htp.tableheader('Date');
            htp.tableheader('Book ID');
            htp.tableheader('Author');
            htp.tableheader('Title');
            htp.tableheader('Quantity');
            htp.tableheader('Description');
            htp.tableheader('Your Bids');
            htp.tablerowclose;
            htp.tablerowopen;
            htp.tabledata(to_char(v_print.timestamp,'mm/dd/yy, hh:mi am'));
            htp.tabledata(v_print.book_id);
            htp.tabledata(v_print.author);
            htp.tabledata(v_print.title);
            htp.tabledata(v_print.quantity);
            if v_print.description is null then htp.tabledata('N/A');
            else htp.tabledata(v_print.description); end if;
            begin
                select bid_type into p_bidtype
                from bid
                where book_id = v_print.book_id and student_id = p_studentid;
                exception when no_data_found then p_bidtype := null;
            end;
            if p_bidtype is null then htp.tabledata('-');
            else
                if p_bidtype = 'sell' then htp.tabledata('<a class=flagsell>sell</a>');
                else htp.tabledata('<a class=flagbuy>buy</a>');
            end if;
        end if;
        htp.tablerowclose;
    end;
else /*category is the same*/
begin
    htp.tablerowopen;
    htp.tabledata(to_char(v_print.timestamp,'mm/dd/yy, hh:mi am'));
    htp.tabledata(v_print.book_id);
    htp.tabledata(v_print.author);
    htp.tabledata(v_print.title);
    htp.tabledata(v_print.quantity);
    if v_print.description is null then htp.tabledata('N/A');
    else htp.tabledata(v_print.description); end if;
    begin
        select bid_type into p_bidtype
        from bid
        where book_id = v_print.book_id and student_id = p_studentid;
        exception when no_data_found then p_bidtype := null;
    end;
    if p_bidtype is null then htp.tabledata('-');
    else
        if p_bidtype = 'sell' then htp.tabledata('<a class=flagsell>sell</a>');
        else htp.tabledata('<a class=flagbuy>buy</a>');
    end if;
end
```

84
else htp.tabledata('&lt;a class=flagbuy&gt;buy&lt;/a&gt;');
        end if;
      end if;
      htp.tablerowclose;
    end;
  end loop;
else /*specific category was entered*/
  begin
    htp.print('&lt;table cellpadding=1px cellspacing=0px&gt;');
    htp.tablerowopen;
    htp.tableheader('Date');
    htp.tableheader('Book ID');
    htp.tableheader('Author');
    htp.tableheader('Title');
    htp.tableheader('Quantity');
    htp.tableheader('Description');
    htp.tableheader('Your Bids');
    htp.tablerowclose;
    for v_print in print2 loop
      begin
        htp.tablerowopen;
        htp.tabledata(to_char(v_print.timestamp,'mm/dd/yy, hh:mi am'));
        htp.tabledata(v_print.book_id);
        htp.tabledata(v_print.author);
        htp.tabledata(v_print.title);
        htp.tabledata(v_print.quantity);
        if v_print.description is null then htp.tabledata('N/A');
        else htp.tabledata(v_print.description); end if;
        begin
          select bid_type into p_bidtype
            from bid
            where book_id = v_print.book_id and student_id = p_studentid;
          exception when no_data_found then p_bidtype := null;
        end;
        if p_bidtype is null then htp.tabledata('-');
        else
          if p_bidtype = 'sell' then htp.tabledata('&lt;a class=flagsell&gt;sell&lt;/a&gt;');
          else htp.tabledata('&lt;a class=flagbuy&gt;buy&lt;/a&gt;');
        end if;
        end if;
        htp.tablerowclose;
      end loop;
  end if;
  htp.tableclose;
end;

create or replace procedure PrintStats
as
  cursor print is
    select book_category, number_of_trades, total_qty, total_value
    from auction_detail
    order by book_category;
begin
  htp.print('&lt;style&gt; Table{border:0px;font-family:verdana;font-size:12px}');
  htp.print('&lt;TD{Border:1px inset Blue}');
  htp.print('&lt;TH{Border:1px inset Blue}');
  htp.print('&lt;/style&gt;');
  htp.print('&lt;table cellpadding=2px cellspacing=0px&gt;');
  htp.tablerowopen;
  htp.tableheader('Book Category');
  htp.tableheader('Number of Trades');
create or replace procedure PrintTrades1
(p_studentid student.student_id%type,
show number) as
  p_lastcategory textbook.book_category%type;
cursor print is
  select timestamp, book_id, author, title, clearing_price, qty,
  trader_name, e_mail, new_flag, book_category, bookdesc_sell, bookdesc_buy
  from tradeview1
  where studentid_sell = p_studentid
  order by book_category, timestamp, book_id;
begin
  p_lastcategory := 'new';
  htp.print('<style> Table{border:1px;font-family:verdana;font-size:12px}
 .flag{font-size:10px;color:red}');
  htp.print('  TD{Border:1px inset Blue}');
  htp.print('  TH{Border:1px inset Blue}');
  htp.print('</style>');
  htp.print('<a class=flag>new</a> <i> flag means that you have NOT seen this trade</i>');
  for v_print in print loop
    if p_lastcategory = 'new' or p_lastcategory <> v_print.book_category then
      begin
      if p_lastcategory <> 'new' then htp.tableclose; end if;
      htp.print('<i>');
      htp.header(3,v_print.book_category);
      htp.print('</i>');
      p_lastcategory := v_print.book_category;
      htp.print('<table cellpadding=4px cellspacing=0px>');
      htp.tablerowopen;
      htp.tableheader('Flag');
      htp.tableheader('Date');
      htp.tableheader('Book ID');
      htp.tableheader('Author');
      htp.tableheader('Title');
      htp.tableheader('Price');
      htp.tableheader('Quantity');
      htp.tableheader('Trader Name');
      htp.tableheader('E-mail');
      if show = 1 then
        begin
        htp.tableheader('Your Description');
        htp.tableheader('Trader Description');
        end;
      end if;
      end if;
    end loop;
  htp.tableclose;
  end;
create or replace procedure PrintTrades2
(p_studentid student.student_id%type,
show number) as
p_lastcategory textbook.book_category%type;
cursor print is
select timestamp, book_id, author, title, clearing_price, qty,
trader_name, e_mail, new_flag, book_category, bookdesc_sell, bookdesc_buy
from tradeview2
where studentid_buy = p_studentid
order by book_category, timestamp, book_id;
begin
p_lastcategory := 'new';
htp.print('<style> Table{border:1px;font-family:verdana;font-size:12px}
.flag{font-size:10px;color:red}');
htp.print(' TD{Border:1px inset Blue}');
htp.print(' TH{Border:1px inset Blue}');
htp.print('</style>');
htp.print('<a class=flag>new</a> <i> flag means that you have NOT seen this trade</i>');
for v_print in print loop
if p_lastcategory = 'new' or p_lastcategory <> v_print.book_category then
begin
if p_lastcategory <> 'new' then htp.tableclose; end if;
htp.print('<i>');
```sql
htp.header(3, v_print.book_category);
htp.print('</i>');
p_lastcategory := v_print.book_category;
htp.print('<table cellpadding=4px cellspacing=0px>');</
htp.tablerowopen;
htp.tableheader('Flag');
htp.tableheader('Date');
htp.tableheader('Book ID');
htp.tableheader('Author');
htp.tableheader('Title');
htp.tableheader('Price');
htp.tableheader('Quantity');
htp.tableheader('Trader Name');
htp.tableheader('E-mail');
if show = 1 then
begin
  htp.tableheader('Your Description');
  htp.tableheader('Trader Description');
end;
end if;
htp.tablerowclose;
htp.tablerowopen;
if v_print.new_flag = 0 then htp.tabledata('<a class=flag>new</a>'); else htp.tabledata('-'); end if;
htp.tabledata(to_char(v_print.timestamp,'mm/dd/yy, hh:mi am'));
htp.tabledata(v_print.book_id);
htp.tabledata(v_print.author);
htp.tabledata(v_print.title);
htp.tabledata(v_print.clearing_price);
htp.tabledata(v_print.qty);
htp.tabledata(v_print.trader_name);
htp.tabledata(htf.mailto(v_print.e_mail,v_print.e_mail));
if show = 1 then
begin
  if v_print.bookdesc_buy is null then htp.tabledata('N/A'); else htp.tabledata(v_print.bookdesc_buy); end if;
  if v_print.bookdesc_sell is null then htp.tabledata('N/A'); else htp.tabledata(v_print.bookdesc_sell); end if;
end;
end if;
htp.tablerowclose;
end; else begin
htp.tablerowopen;
if v_print.new_flag = 0 then htp.tabledata('<a class=flag>new</a>'); else htp.tabledata('-'); end if;
htp.tabledata(to_char(v_print.timestamp,'mm/dd/yy, hh:mi am'));
htp.tabledata(v_print.book_id);
htp.tabledata(v_print.title);
htp.tabledata(v_print.clearing_price);
htp.tabledata(v_print.qty);
htp.tabledata(v_print.trader_name);
htp.tabledata(htf.mailto(v_print.e_mail,v_print.e_mail));
if show = 1 then
begin
  if v_print.bookdesc_buy is null then htp.tabledata('N/A'); else htp.tabledata(v_print.bookdesc_buy); end if;
  if v_print.bookdesc_sell is null then htp.tabledata('N/A'); else htp.tabledata(v_print.bookdesc_sell); end if;
end;
end if;
htp.tablerowclose;
end; end if;
```
create or replace procedure PwdChange
(p_studentid student.student_id%type := null) as
p_status log_check.result%type;

begin
/*check if the student is logged in*/
select result into p_status
from log_check
where student_id = p_studentid and timestamp =
(select max(timestamp) from log_check where student_id = p_studentid);
if p_status <> 'S' then
notlogged;
else /*student is logged in*/
begin
style;
htp.p('<i>NOTE: Passwords are case sensitive.</i>');
htp.formOpen('pwdnow');
htp.preOpen;
htp.formhidden('p_studentid', p_studentid);
htp.p('<b>OLD PASSWORD:      </b>' || htf.formPassword('oldpwd'));
htp.p('<b>NEW PASSWORD:      </b>' || htf.formPassword('newpwd1'));
htp.p('<b>RE-ENTER PASSWORD: </b>' || htf.formPassword('newpwd2'));
htp.preClose;
htp.paragraph;
htp.formSubmit;
htp.formClose;
htp.bodyclose;
htp.htmlClose;
end;
end if;
end;
end;

create or replace procedure PwdNow
(p_studentid student.student_id%type := null,
oldpwd student.password%type := null,
newpwd1 student.password%type := null,
newpwd2 student.password%type := null) as
p_check student.password%type;
p_status log_check.result%type;

begin
/*check if the student is logged in*/
select result into p_status
from log_check
where student_id = p_studentid and timestamp =
(select max(timestamp) from log_check where student_id = p_studentid);
if p_status <> 'S' then
notlogged;
else /*the student is logged in*/
begin
style;
if oldpwd is null or newpwd1 is null or newpwd2 is null then
begin
htp.bold('Please go back in your browser and enter your old password
and your new password in two fields. <br>');
htp.bold('Thank you!');
end;
else /*all three fields have been entered*/
begin
select password into p_check
from student
where student_id = p_studentid;
if p_check = oldpwd then

begin
if newpwd1 <> newpwd2 then
begin
  htp.bold('Two new password fields do not match!<br>
  Please go back in your browser and try again.<br>
  Thank you!');
  end;
else /*passwords match*/
begin
  update student set password = newpwd1
  where student_id = p_studentid;
  htp.bold('Your password has been succesfully updated.<br>
  Please use your new password for all future logins.<br>
  Thank you!');
  htp.br;
  htp.br;
  htp.anchor('updateinfo?p_studentid='||p_studentid||'','Return to the Account
Information');
  end;
end if;
else /*old password entered is incorrect*/
begin
  htp.bold('Your old password is incorrect.<br>
  Please go back in your browser and try again.<br>
  Thank you!');
  end;
end if;
end;
end if;
end;
end if;
end if;
end;
end if;
end if;
end;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
end if;
commodities.<br><br>
htp.print('<b>Clearing Price</b>');
htp.p('<br>In this auction system, the clearing price or price at which a textbook will be sold, is
determined as an average of the buyer and seller prices. Which means, that if you are a seller, you will
sell your book to the first person who is offering more than your minimum acceptable price and at a
price equal to average of your two exchange rates. In case you are a buyer, you will buy a book from
the first person who is willing to sell it at a lower rate than your maximum price and at a clearing price
which equals to the average of these two values.<br><br>');
htp.print('<b>How the Auction Operates</b>');
htp.p('<br>The seller places textbook information (like author, title, description, available quantity)
along with the minimum acceptable price. All potential buyers can see all books offered for sale
(without prices) and can bid on them. The auction system accepts bids in the order they are received.
The first person who bids higher than the minimum price of the seller will get the book.<br>);
htp.p('The opposite situation is also acceptable. The buyer places textbook information (like author, title,
description, quantity needed) along with the maximum price. All users can see what books are in
demand (of course, without prices) and can sell their books. Again, the first person who bids lower than
the maximum price the buyer is willing to pay for the book will sell it.<br>);
htp.p('Neither buyers nor sellers may manipulate the price of any book nor may they interfere with other
user's information or bids. Each user can submit only one bid for a particular textbook (it can be either
sell or buy bid)<br>');</
htp.print('<b>Actual Book Exchange</b>');
htp.p('<br>The Auction system will only provide you with information about other user's intentions to
buy or to sell some books. The site acts as the venue for sellers to list textbooks and buyers to bid on
them. The system is not involved in the actual transaction between buyers and sellers. It is entirely
your responsibility to contact the person who agreed to buy or sell book (an e-mail address will be
provided for this purpose) and to agree on how you can arrange the exchange. You are encouraged to
communicate directly with potential trading partners. The site does not deal with financial transactions
at all (no credit card or similar data is collected, no address or identity verification is done, no media to
arrange shipments or other delivery is provided. It will be the whole responsibility of the parties to do
the actual book exchange. As a result, there is no control over the quality of the books offered, the
accuracy of the listings, the ability of sellers to sell books or the ability of buyers to buy books. There is
no guarantee that a buyer or seller will actually complete a transaction.<br>');</
htp.print('<b>Trading Partners</b>');
htp.p('<br>If your bid results in trade (conditions you offered were acceptable by a buyer or a seller),
you are obligated to complete the transaction with the trading partner. By placing a bid on a book you
agree to be bound by the conditions of sale included in the book's description. Bids that have resulted
in trade are not retractable except in exceptional circumstances such as when the seller materially
changes the item's description after you bid, a clear typographical error is made, or you cannot
authenticate the partner's identity.<br>');</
htp.print('<b>Limited Responsibility</b>');
htp.p('<br>Because the system is not involved in the actual transaction between buyers and sellers, in
the event that there is a dispute with one or more users, the system owners/operators are released
from claims, demands and damages (actual and consequential) of every kind and nature, known and
unknown, suspected and unsuspected, disclosed and undisclosed, arising out of or in any way
connected with such disputes.<br>');</
htp.p('There is no information control on the site. All data is displayed the way users have entered it into
the system. This information might be offensive, harmful, inaccurate, or deceptive. Please use caution,
common sense, and practice safe trading when using this site. Please note that there are also risks of
dealing with people acting under false pretense.<br>');</
htp.p('The web site is provided "AS IS" and without any warranty or condition, express, implied or
statutory. Any implied warranties are specifically disclaimed. In no event will the system
owners/operators be liable for lost profits or any special, incidental or consequential damages arising
out of or in connection with this site, or this agreement (however arising, including negligence).');
if p_studentid = 0 then /*new user*/
begin
htp.print('</td></tr><tr><td><center>');
htp.anchor('adduser',htf.header(5,'Accept'));
htp.print('</center></td><td><center>');
htp.anchor('thankyou',htf.header(5,'Decline'));
htp.print('</center></td></tr>');
end;
else htp.print('</td></tr>');
end if;
htp.tableclose;
htp.centerClose;
if p_studentid = 0 then footer; end if;
create or replace procedure SearchBookID
(p_studentid student.student_id%type,
bookid textbook.book_id%type,
showbids number) as
p_author textbook.author%type;
p_title textbook.title%type;
p_category textbook.book_category%type;
p_status log_check.result%type;
begin
/*check if the student is logged in*/
select result into p_status
from log_check
where student_id = p_studentid and timestamp =
(select max(timestamp) from log_check where student_id = p_studentid);
if p_status <> 'S' then notlogged;
else /*the student is logged in*/
begin
begin
select author, title, book_category
into p_author, p_title, p_category
from studentbidview
where book_id = bookid and rownum = 1;
begin
htp.header(4,'Category: '||p_category);
htp.header(3,'<i>BOOK ID: '||bookid||' ""||p_title||'" by '||p_author||'</i>');
printbookbids(p_studentid,bookid,showbids);
end;
exception when no_data_found then
htp.bold('There are no bids submitted by other users of the system for a textbook with
BOOK ID <i><b>'||bookid||'</b></i>.');
end;
htp.bodyclose;
htp.htmlclose;
end if;
end;
end;
end;
create or replace procedure SearchForm
(p_studentid student.student_id%type :=null) as
p_status log_check.result%type;
begin
/*check if the student is logged in*/
select result into p_status
from log_check
where student_id = p_studentid and timestamp =
(select max(timestamp) from log_check where student_id = p_studentid);
if p_status <> 'S' then notlogged;
else /*the student is logged in*/
begin
begin
htp.p('<style>table{font-family:verdana;font-size:12px} .small{font-size:10px;color:black}</style>');}
htp.centerOpen;
htp.header(2,'Search Form');
htp.centerClose;
htp.formOpen('searchnow');
htp.preOpen;
htp.formhidden('p_studentid',p_studentid);
htp.bold('PLEASE ENTER AT LEAST ONE SEARCH CRITERIA');
htp.p('<b>BOOK ID:          </b>'|| htf.formText('bookid')||'<i> Format: NUMERIC</i>');
create or replace procedure SearchNow
(p_studentid student.student_id%type := null,
bookid varchar2 := null,
author textbook.author%type := null,
title textbook.title%type := null,
description bid.book_description%type := null) as
p_status log_check.result%type;
p_check number := 0;
p_author textbook.author%type;
p_title textbook.title%type;
p_desc bid.book_description%type;
p_lastcategory textbook.book_category%type;
cursor got_BATD is
select book_id, author, title, book_category
from textbook
where book_id = bookid or author like p_author or title like p_title or book_id IN
(select book_id from bid where book_description like p_desc)
order by book_category, book_id;
begin
/*check if the student is logged in*/
select result into p_status
from log_check
where student_id = p_studentid and timestamp =
(select max(timestamp) from log_check where student_id = p_studentid);
if p_status <> 'S' then notlogged;
else /*the student is logged in*/
begin
style;
/*check for null values in the input fields*/
if p_studentid is null or (bookid is null and author is null and title is null and description is null)
then begin
htp.bold('Sorry, but in order to search you need to enter at least one search criteria.

Please go back in your browser and correct the form.

Thank you!');
end;
else /*all required values are entered*/
begin
if author is not null then /*prepare author entry*/
begin
select initcap(author) into p_author from dual;
select concat('%',p_author) into p_author from dual;
select concat(p_author,'%') into p_author from dual;
end;
end if;
if title is not null then /*prepare title entry*/
begin
select initcap(title) into p_title from dual;
select concat('%',p_title) into p_title from dual;
end;
end if;
end;
cursor got_BATD is
select book_id, author, title, book_category
from textbook
where book_id = bookid or author like p_author or title like p_title or book_id IN
(select book_id from bid where book_description like p_desc)
order by book_category, book_id;
end;
select concat(p_title,'%') into p_title from dual;
end;
end if;
if description is not null then /*prepare description entry*/
begin
select concat('%',description) into p_desc from dual;
select concat(p_desc,'%') into p_desc from dual;
end;
end if;
select count(*) into p_check from textbook
where book_id = bookid or author like p_author or title like p_title or book_id IN
(select book_id from bid where book_description like p_desc);
if p_check <> 0 then /*some matches found*/
begin
  p_lastcategory := 'new';
  htp.print('<style> Table{border:0px;font-family:verdana;font-size:12px}');</
  htp.print('  TD{Border:1px inset Blue}');</
  htp.print('  TH{Border:1px inset Blue}');</
  htp.print('</style>');
  htp.bold('Click on BOOK ID to check for submitted bids on that particular textbook.<br>');
  for v_print in got_BATD loop
    begin
      if p_lastcategory = 'new' or p_lastcategory <> v_print.book_category then
        begin
          if p_lastcategory <> 'new' then htp.tableclose; end if;
          htp.header(3,v_print.book_category);
          htp.print('</i>');
          p_lastcategory := v_print.book_category;
        htp.print('<table cellpadding=1px cellspacing=0px>');
        htp.tablerowopen;
        htp.tableheader('Book ID');
        htp.tableheader('Author');
        htp.tableheader('Title');
        htp.tablerowclose;
        htp.tablerowopen;
        htp.tabledata(htf.anchor('searchbookid?p_studentid='||p_studentid||'
          &bookid='||v_print.book_id||'&showbids=1',v_print.book_id));
        htp.tabledata(v_print.author);
        htp.tabledata(v_print.title);
        htp.tablerowclose;
      end;
      else /*category is the same*/
        begin
          htp.tabledata(htf.anchor('searchbookid?p_studentid='||p_studentid||'
            &bookid='||v_print.book_id||'&showbids=1',v_print.book_id));
          htp.tabledata(v_print.author);
          htp.tabledata(v_print.title);
          htp.tablerowclose;
        end;
    end if;
  end loop;
else /*there are no matches*/
begin
  htp.bold('Sorry, no books were found that match your search criteria.<br>');
end;
end if;
htp.tableclose;
end if;
htp.br;
htp.anchor('searchform?p_studentid='||p_studentid||'
,Enter a new Search');
htp.bodyclose;
create or replace procedure ShowLog
/*this procedure can be used for testing purposes. It is NOT called from anywhere on the site*/
(p_studentid student.student_id%type) as
p_check textbook.book_id%type;
table_name varchar2(20) := 'log_table';
ignore boolean;
p_status log_check.result%type;

begin
/*check if the student is logged in*/
select result into p_status
from log_check
where student_id = p_studentid and timestamp =
(select max(timestamp) from log_check where student_id = p_studentid);
if p_status <> 'S' then notlogged;
else /*the student is logged in*/
begin
style;
htp.centerOpen;
htp.header(2,'Your Log Table');
htp.centerClose;
begin
select bookid into p_check
from log_table
where studentid = p_studentid and rownum=1;
exception when no_data_found then errorpage;
end;
ignore := owa_util.tableprint(table_name,'BORDER', owa_util.html_table, 'change_type, timestamp,
bookid, bidtype, old_bidprice, old_qty, new_bidprice, new_qty', 'where studentid='||p_studentid||'');
htp.bodyclose;
htp.htmlclose;
end;
end if;
end;
end;
create or replace procedure Statistics
(p_studentid student.student_id%type) as
p_check student.student_id%type;
p_status log_check.result%type;

begin
/*check if the student is logged in*/
select result into p_status
from log_check
where student_id = p_studentid and timestamp =
(select max(timestamp) from log_check where student_id = p_studentid);
if p_status <> 'S' then notlogged;
else /*the student is logged in*/
begin
style;
htp.centerOpen;
htp.header(2,'Auction Statistics');
htp.centerClose;
printstats;
htp.bodyclose;
htp.htmlclose;
end if;
end;
end;

95
create or replace procedure Style as
begin
  htp.htmlOpen;
  htp.headOpen;
  htp.print('<style>');</
  htp.print('body {background-color:aliceblue;font-family:verdana; font-size:12px;color:navy;}
  a{Color:navy;} a:hover{color:navy;font-weight:bold}');</
  htp.print('</style>');</
  htp.headClose;
  htp.bodyOpen;
end;

create or replace procedure ThankYou as
begin
  style;
  htp.header(4,'Thank you very much for visiting Textbooks Auction System website.');</
  htp.header(4,'Hope you will join us oneday ;-)');</
  footer;
  htp.bodyClose;
  htp.htmlClose;
end;

create or replace procedure TraderChange
(p_studentid student.student_id%type := null) as
  p_status log_check.result%type;
begin
  /*check if the student is logged in*/
  select result into p_status
    from log_check
    where student_id = p_studentid and timestamp =
      (select max(timestamp) from log_check where student_id = p_studentid);
  if p_status <> 'S' then notLogged;
  else /*the student is logged in*/
    begin
      style;
      htp.p('<i>NOTE: Trader name is the name under which all your
transactions will be listed to other users.</i>');
      htp.formOpen('tradernow');
      htp.preOpen;
      htp.formHidden('p_studentid',p_studentid);
      htp.p('<b>NEW TRADER NAME: </b>'|| htf.formText('newtrader'));
      htp.preClose;
      htp.paragraph;
      htp.formSubmit;
      htp.formClose;
    end;
  end if;
end;

create or replace procedure TraderNow
(p_studentid student.student_id%type := null,
 newtrader student.trader_name%type := null) as
  p_trader student.trader_name%type;
  p_status log_check.result%type;
begin
  /*check if the student is logged in*/
  select result into p_status
    from log_check
    where student_id = p_studentid and timestamp =
      (select max(timestamp) from log_check where student_id = p_studentid);
  if p_status <> 'S' then notLogged;
  else /*the student is logged in*/
    begin
      style;
      htp.p('<i>NOTE: Trader name is the name under which all your
transactions will be listed to other users.</i>');
      htp.formOpen('tradernow');
      htp.preOpen;
      htp.formHidden('p_studentid',p_studentid);
      htp.p('<b>NEW TRADER NAME: </b>'|| htf.formText('newtrader'));
      htp.preClose;
      htp.paragraph;
      htp.formSubmit;
      htp.formClose;
      htp.htmlClose;
    end;
end;
begin
style;
if newtrader is null then
begin
htp.bold('Please go back in your browser and enter your new trader name.');
htp.bold('<br>Thank you!');
end;
else
begin
select trader_name into p_trader
from student
where trader_name = newtrader and student_id <> p_studentid;
begin
htp.bold('The trader name you entered is already used by some other user.');
htp.bold('<br>Please go back in your browser and enter a different trader name.');
htp.bold('<br>Thank you!');
end;
exception when no_data_found then
begin
update student set trader_name = newtrader
where student_id = p_studentid;
htp.bold('Your trader name has been succesfully updated');
htp.bold('<br>Thank you!');
htp.br;
htp.br;
htp.anchor('updateinfo?p_studentid='||p_studentid||'','Return to the Account Information');
end;
end if;
htp.bodyclose;
htp.htmlClose;
end;
end if;
end;
end if;
create or replace procedure UpdateBid
(p_studentid student.student_id%type :=null) as
  p_status log_check.result%type;
begin
/*check if the student is logged in*/
select result into p_status
from log_check
where student_id = p_studentid and timestamp =
  (select max(timestamp) from log_check where student_id = p_studentid);
if p_status <> 'S' then
  notlogged;
else /*the student is logged in*/
begin
style;
htp.p('<style>table{font-family:verdana;font-size:12px}</style>');
htp.centerOpen;
htp.header(2,'Update Bid Form');
htp.centerClose;
htp.formOpen('updatebidnow');
htp.preOpen;
htp.formhidden('p_studentid',p_studentid);
htp.p('Please enter Book Identification Number (you can get it from the previous page)');
htp.p('<b>BOOK ID:        </b>'|| htf.formText('bookid')||'<i> Format: NUMERIC</i>');
htp.p('Please note, that even if you ONLY update the description now, you will not
begin
style;
htp.p('<table{font-family:verdana;font-size:12px}</style>');
htp.centerOpen;
htp.header(2,'Update Bid Form');
htp.centerClose;
htp.formOpen('updatebidnow');
htp.preOpen;
htp.formhidden('p_studentid',p_studentid);
htp.p('NOTE: Only bids submitted/modified more that 2 hours ago can be modified!</i>');
htp.p('</td></tr>');
htp.bold('Please enter Book Identification Number (you can get it from the previous page)');
htp.p('BOOK ID:        </b>'|| htf.formText('bookid')||'<i> Format: NUMERIC</i>');
htp.p('NOTE: In description you can include edition, year, publisher, ISBN (if available).
You can also mention book condition (new/used). Please remember that description is an
optional field, it is NOT used in matching bids.');
htp.p('Please note, that even if you ONLY update the description now, you will not
be able to modify your bid again for 2 hours.

```sql
create or replace procedure UpdateBidNow
(p_studentid student.student_id%type := null,
 bookid varchar2 := null,
 newdescription bid.book_description%type := null,
 newprice varchar2 := null,
 newquantity varchar2 := null) as
 p_status log_check.result%type;
 p_datecheck date;
 p_date date;
 p_bidtype bid.bid_type%type;
 p_price bid.bid_price%type;
 p_qty bid.quantity%type;
 p_check number := 0;
 t_lowsell possible.low_sell%type;
 t_highbuy possible.high_buy%type;
 t_qtysell possible.qty_sell%type;
 t_studentid student.student_id%type;
 t_studentcheck number := 0;

begin

/*check if the student is logged in*/
select result into p_status
from log_check
where student_id = p_studentid and timestamp =
(select max(timestamp) from log_check where student_id = p_studentid);
if p_status <> 'S' then notlogged;
else /*the student is logged in*/
begin

style;
/*check for null values in the input fields*/
if p_studentid is null or bookid is null or
(newdescription is null and
(newprice is null or newprice < 0.01) and
(newquantity is null or newquantity < 1)) then
begin
htp.bold('Please go back in your browser and enter book identification number
and at least one field from description, price or quantity.');
htp.bold('<i>NOTE: price and quantity can NOT be zeroes or negative values.</i>');
htp.bold('<br>Thank you!');
end;
else /*all values entered were not nulls*/
begin

/*check if the bid id entered exists in the database*/
select timestamp, bid_type, bid_price, quantity
into p_datecheck, p_bidtype, p_price, p_qty
from bid
where book_id = bookid and student_id = p_studentid;
begin
/*check that the bid was submitted at least 2 hours ago*/
select sysdate into p_date from dual;
if (p_date - p_datecheck) < 1/12 then
end;
end if;
end;```
Sorry, you can not modify bids submitted/modified less than 2 hours ago.

Go back to your bids.

**the bid was submitted more than 2 hours ago**

begin
  if newdescription is not null then /*description is modified*/
  begin
    update bid set book_description = newdescription, timestamp = sysdate, updated_flag=2
    where book_id = bookid and student_id = p_studentid;
    end;
  end if;
  if newprice is not null then /*price is modified*/
  begin
    update bid set bid_price = newprice, timestamp = sysdate, updated_flag=2
    where book_id = bookid and student_id = p_studentid;
    insert into log_table values ('U',sysdate,p_studentid,bookid,p_bidtype, p_price,p_qty,newprice,p_qty);
    p_check := 1;
    end;
  end if;
  if newquantity is not null then /*quantity is modified*/
  begin
    update bid set quantity = newquantity, timestamp = sysdate, updated_flag=2
    where book_id = bookid and student_id = p_studentid;
    insert into log_table values ('U',sysdate,p_studentid,bookid,p_bidtype, p_price,p_qty,p_price,newquantity);
    p_check := 1;
    end;
  end if;
  if p_check = 1 then /* price and/or quantity has been modified */
  begin
    if p_bidtype = 'sell' then
      begin
        /*find the lowest sell bid*/
        begin
          select min(bid_price) into t_lowsell
          from bid
          where book_id = bookid and bid_type = 'sell';
          end;
        begin
          select quantity, student_id into t_qtysell, t_studentid
          from bid
          where book_id = bookid and bid_price = t_lowsell
          and bid_type = 'sell' and timestamp = (select min(timestamp) from bid
          where book_id = bookid and bid_price = t_lowsell and bid_type = 'sell');
          end;
        update possible set qty_sell = t_qtysell, low_sell = t_lowsell, studentid_sell = t_studentid
        where book_id = bookid;
        if p_studentid <> t_studentid then t_studentcheck := 1; end if;
      end;
    else /*bid type is "buy" */
    begin
      /*find the highest buy bid*/
      select max(bid_price) into t_highbuy
      from bid
      where book_id = bookid and bid_type = 'buy';
      end;
    begin
      select quantity, student_id into t_qtybuy, t_studentid
      from bid
CREATE OR REPLACE PROCEDURE UpdateInfo
(p_studentid student.student_id%type := null)
AS
BEGIN
/*check if the student is logged in*/
select result into p_status
from log_check
where student_id = p_studentid and timestamp =
(select max(timestamp) from log_check where student_id = p_studentid);
if p_status <> 'S' then notlogged;
else /*the student is logged in*/
BEGIN
style;
htp.p('<style>table{font-family:verdana;font-size:12px}</style>');</
htp.centeropen;
htp.header(2,'Account Info');
htp.centerclose;
select first_name, last_name, street_address, city, state, zip, phone_number, e_mail, username,
trader_name into p_firstname, p_lastname, p_street, p_city, p_state, p_zip, p_phone, p_email,
CREATE OR REPLACE PROCEDURE UpdatePossible
(p_studentid student.student_id%type,
p_bookid textbook.book_id%type,
p_bidprice bid.bid_price%type,
p_quantity bid.quantity%type,
p_bidtype bid.bid_type%type,
p_studentcheck number) as
t_price bid.bid_price%type;
begin
style;
if checkposs(p_bookid) = 2 then  /*no record for the book id exists in table possible*/
begin

/*add a new record into table possible*/
if p_bidtype = 'buy' then
  insert into possible (book_id,studentid_buy,high_buy,qty_buy)
  values (p_bookid, p_studentid, p_bidprice, p_quantity);
else /*bid type is sell*/
  insert into possible (book_id,studentid_sell,low_sell,qty_sell)
  values (p_bookid, p_studentid, p_bidprice, p_quantity);
end if;
if p_studentcheck = 0 then
  /*print only if student id matches the student id of the user logged in*/
  begin
    htp.bold('Your bid was successfully submitted but has NOT resulted in trade');
    htp.bold('<br>Good luck with future bids :) THANK YOU');
  end;
end if;
else /* checkposs(p_bookid) = 1 the record for the book id already exists in table possible*/
begin
  if p_bidtype = 'buy' then
    begin
    begin
      /*compare new price with the existing highest buy price*/
      select nvl(high_buy,0) into t_price
      from possible
      where book_id = p_bookid;
      exception when no_data_found then errorpage;
    end;
    if t_price = 0 or t_price < p_bidprice then /*new price is the highest buy price*/
      begin
        update possible
        set qty_buy = p_quantity, high_buy = p_bidprice, studentid_buy = p_studentid
        where book_id = p_bookid;
        /*table possible has been updated, check for possibility of a trade*/
        updatetrade(p_bookid, p_studentid, p_studentcheck);
      end;
    else /*new buy price is not the highest, table possible has not been changed*/
      begin
        if p_studentcheck = 0 then
          /*print only if student id matches the student id of the user logged in*/
          begin
            htp.bold('Your bid was successfully submitted but has NOT resulted in trade');
            htp.bold('<br>Good luck with future bids :) THANK YOU');
          end;
        end if;
      end;
    end;
  else /* p_bidtype = 'sell' */
    begin
    begin
      /*compare new price with the existing lowest sell price*/
      select nvl(low_sell,0) into t_price
      from possible
      where book_id = p_bookid;
      exception when no_data_found then errorpage;
    end;
    if t_price = 0 or t_price > p_bidprice then /*new price is the lowest sell price*/
      begin
        update possible
        set qty_sell = p_quantity, low_sell = p_bidprice, studentid_sell = p_studentid
        where book_id = p_bookid;
        /*table possible has been updated, check for possibility of a trade*/
        updatetrade(p_bookid, p_studentid, p_studentcheck);
      end;
    else /*new sell price is not the lowest, table possible has not been changed*/
      begin

if p_studentcheck = 0 then
/*print only if student id matches the student id of the user logged in*/
begin
htp.bold('Your bid was successfully submitted but has NOT resulted in trade');
htp.bold('<br>Good luck with future bids :) THANK YOU');
end;
end if;
end;
end;
end if;
end;
end if;
end if;
end;
end if;
htp.bodyclose;
htp.htmlclose;
end;

create or replace procedure UpdateTrade
/*after procedure AddTrade has been called, table POSSIBLE is updated with new quantities, also to reflect new lowest sell and highest buy bids*/
(p_bookid textbook.book_id%type,
p_studentid student.student_id%type,
p_studentcheck number) as
t_lowsell possible.low_sell%type;
t_highbuy possible.high_buy%type;
t_qtysell possible.qty_sell%type;
t_qtysell possible.qty_sell%type;
t_qtysell possible.qty_buy%type;
t_qtysell possible.qty_sell%type;
t_studentid student.student_id%type;
begin
style;
/*select the lowest sell price from table possible for the book id*/
select nvl(low_sell,0) into t_lowsell
from possible
where book_id = p_bookid;
exception when no_data_found then errorpage;
end;
/*select the highest buy price from table possible for the book id*/
select nvl(high_buy,0) into t_highbuy
from possible
where book_id = p_bookid;
exception when no_data_found then errorpage;
end;
if t_lowsell <> 0 and t_highbuy <> 0 then /*if both sell and buy bids exist*/
begin
if t_lowsell <= t_highbuy then /* trade is possible */
begin
addtrade(p_bookid, p_studentid, p_studentcheck);
begin
select qty_sell, qty_buy into t_qtysell, t_qtysell
from possible
where book_id = p_bookid;
exception when no_data_found then errorpage;
end;
if t_qtysell < t_qtysell then /*quantity to sell is less than quantity to buy*/
begin
if t_qtysell <= t_qtysell then /* trade quantity will be equal to quantity to sell*/
update possible set low_sell = null, qty_buy = qty_buy - t_qty
where book_id = p_bookid;
if checkbidsell(p_bookid) = 1 then /*there are more sell bids for this book id*/
begin
select min(bid_price) into t_lowsell
end;
from bid
where book_id = p_bookid and bid_type = 'sell';
exception when no_data_found then errorpage;
end;
begin
select quantity, student_id into t_qtysell, t_studentid
from bid
where book_id = p_bookid and bid_price = t_lowsell
and bid_type = 'sell' and timestamp = (select min(timestamp) from bid
where book_id = p_bookid and bid_price = t_lowsell
and bid_type = 'sell');
exception when no_data_found then errorpage;
end;
update possible set qty_sell = t_qtysell, low_sell = t_lowsell,
studentid_sell = t_studentid
where book_id = p_bookid;
updatetrade(p_bookid, t_studentid, 1);
end;
else /* no more sell bids*/
begin
update possible set qty_sell = null, studentid_sell = null
where book_id = p_bookid;
end;
end if;
end; /*quantity to sell is greater than quantity to buy*/
begin
if t_qtysell > t_qtybuy then
begin
t_qty := t_qtybuy; /*trade quantity will be equal to quantity to buy*/
update possible set high_buy = null, qty_sell = qty_sell - t_qty
where book_id = p_bookid;
if checkbidbuy(p_bookid) = 1 then /*there are more buy bids on this book id*/
begin
begin
select max(bid_price) into t_highbuy
from bid
where book_id = p_bookid and bid_type = 'buy';
exception when no_data_found then errorpage;
end;
begin
select quantity, student_id into t_qtybuy, t_studentid
from bid
where book_id = p_bookid and bid_price = t_highbuy
and bid_type = 'buy' and timestamp = (select min(timestamp) from bid
where book_id = p_bookid and bid_price = t_highbuy
and bid_type = 'buy');
exception when no_data_found then errorpage;
end;
update possible set qty_buy = t_qtybuy, high_buy = t_highbuy,
studentid_buy = t_studentid
where book_id = p_bookid;
updatetrade(p_bookid, t_studentid, 1);
end;
else /*no more buy bids exist*/
begin
update possible set qty_buy = null, studentid_buy = null
where book_id = p_bookid;
end;
end;
end if;
end;
else /*quantities to sell and to buy are equal*/
begin
update possible set high_buy = null, low_sell = null
where book_id = p_bookid;
if checkbidbuy(p_bookid) = 1 then /*there are more buy bids on this book id*/
begin
    begin
        select max(bid_price) into t_highbuy
        from bid
        where book_id = p_bookid and bid_type = 'buy';
        exception when no_data_found then errorpage;
    end;
    begin
        select quantity, student_id into t_qtybuy, t_studentid
        from bid
        where book_id = p_bookid and bid_price = t_highbuy
        and bid_type = 'buy' and timestamp = (select min(timestamp) from bid
        where book_id = p_bookid and bid_price = t_highbuy
        and bid_type = 'buy');
        exception when no_data_found then errorpage;
    end;
    update possible set qty_buy = t_qtybuy, high_buy = t_highbuy,
    studentid_buy = t_studentid
    where book_id = p_bookid;
    end;
else /*no more buy bids exist*/
    begin
        update possible set qty_buy = null, studentid_buy = null
        where book_id = p_bookid;
    end; end if;
if checkbidsell(p_bookid) = 1 then /*there are more sell bids for this book id*/
    begin
        select min(bid_price) into t_lowsell
        from bid
        where book_id = p_bookid and bid_type = 'sell';
        exception when no_data_found then errorpage;
    end;
    begin
        select quantity, student_id into t_qtysell, t_studentid
        from bid
        where book_id = p_bookid and bid_price = t_lowsell
        and bid_type = 'sell' and timestamp = (select min(timestamp) from bid
        where book_id = p_bookid and bid_price = t_lowsell
        and bid_type = 'sell');
        exception when no_data_found then errorpage;
    end;
    update possible set qty_sell = t_qtysell, low_sell = t_lowsell,
    studentid_sell = t_studentid
    where book_id = p_bookid;
    updatetrade(p_bookid, t_studentid, 1);
    end;
else /* no more sell bids*/
    begin
        update possible set qty_sell = null, studentid_sell = null
        where book_id = p_bookid;
    end; end if;
end;
else /*trade is not possible*/
    begin
        if p_studentcheck = 0 then
            /*print only if student id matches the student id of the user logged in*/
            begin
                htp.bold('Your bid was successfully submitted but has NOT resulted in trade');
                htp.bold('<br>Good luck with future bids :) THANK YOU');
end;
end if;
end;
end if;
end;
else /*there are only buy OR sell bids for the book id*/
begin
if p_studentcheck = 0 then
/*print only if student id matches the student id of the user logged in* /
begin
htp.bold('Your bid was successfully submitted but has NOT resulted in trade');
htp.bold('<br>Good luck with future bids :) THANK YOU');
end;
end if;
end if;
end;

create or replace procedure UsernameChange
(p_studentid student.student_id%type := null) as
p_username student.username%type;
p_status log_check.result%type;
begin
/*check if the student is logged in*/
select result into p_status
from log_check
where student_id = p_studentid and timestamp =
(select max(timestamp) from log_check where student_id = p_studentid);
if p_status <> 'S' then notlogged;
else /*the student is logged in*/
begin
/*check if the student is logged in*/
select result into p_status
from log_check
where student_id = p_studentid and timestamp =
(select max(timestamp) from log_check where student_id = p_studentid);
if p_status <> 'S' then notlogged;
else /*the student is logged in*/
begin
/*create or replace procedure UsernameNow*/
(p_studentid student.student_id%type := null,
newusername student.username%type := null) as
p_username student.username%type;
p_status log_check.result%type;
begin
/*check if the student is logged in*/
select result into p_status
from log_check
where student_id = p_studentid and timestamp =
(select max(timestamp) from log_check where student_id = p_studentid);
if p_status <> 'S' then notlogged;
else /*the student is logged in*/
begin
style;
if newusername is null then
106
begin
  htp.bold('Please go back in your browser and enter your new username.');
  htp.bold('<br>Thank you!');
  end;
else
  begin
    select username into p_username
    from student
    where username = newusername and student_id <> p_studentid;
    begin
      htp.bold('The username you entered is already used by some other user.');
      htp.bold('<br>Please go back in your browser and enter a different username.');
      htp.bold('<br>Thank you!');
    end;
    exception when no_data_found then
      begin
        update student set username = newusername
        where student_id = p_studentid;
        htp.bold('Your username has been successfully updated');
        htp.bold('<br>Please use your new username for all future logins');
        htp.bold('<br>Thank you!<br><br>');
        htp.anchor('updateinfo?p_studentid='||p_studentid||'','Return to the Account Information');
      end;
    end if;
  end if;
end;
end if;
end;
end;

create or replace procedure Welcome as
begin
  style;
  htp.print('<style> Table{border:0px;font-family:verdana;font-size:14px}</style>');
  htp.centerOpen;
  htp.htitle('Welcome to the Textbooks Auction system');
  htp.print('<i><b>');
  htp.p('Congratulations on finding this site ;-)');
  htp.p('<br>/i><br>');
  htp.print('<table width="70%" cellpadding=1px cellspacing=0px>');
  htp.print('<tr><td colspan=2>');
  htp.p('This Online Auction System has been designed for students to help lower the cost of textbooks. It works as any other auction that establishes selling prices using supply and demand mechanisms. In other words, there are no sponsors to this site, no advertising, and nobody is making any money off of you! The whole purpose is to provide a media for students to communicate effectively with each other and exchange their textbooks. So now, instead of sending an e-mail to the science club and asking if anybody would be interested in buying some book from you (or if somebody has the book you need), you can just come here and post information about your book on the site. Anybody accessing the site will be able to see it and bid on it. If they bid more than you are asking for, both of you will be notified about the possibility of a deal. After that, it is up to you two to agree on how the book will actually be exchanged/shipped. You will not even need to negotiate the price! It will already be established using the price that the buyer is willing to pay for your textbook!<br>');
  htp.p('<br>Just a quick note: please do NOT try to hack or destroy the system. Find some other target for that inquisitive mind of yours! The intention of this system is to help the student community. All work was voluntary, please be thankful and considerate!');
  htp.bold('THANKS ;-)');
  htp.print('</td>');
  htp.print('<td width="50%" colspan=2><center>');
  htp.bold('<br>Please click on one of the options below to access the site');
  htp.print('</center>');
  htp.print('<tr><td width="50%">');
  htp.listItem(htf.anchor('checkid','REGISTERED USERS'));
  htp.print('</td>');
  htp.print('<td width="50%">');
  htp.listItem(htf.anchor('rules?p_studentid=0','NEW USERS'));
end;
Functions

create or replace function CheckAuction
(f_book_cat textbook.book_category%TYPE)
RETURN number IS
  v_found number(1) := 2;
begin
  select 1 into v_found
  from auction_detail
  where book_category = f_book_cat;
  return v_found;
exception when no_data_found then
  return v_found;
end;

create or replace function CheckBidBuy
(f_bookid textbook.book_id%type)
RETURN number IS
  v_found number(1) := 2;
begin
  select 1 into v_found from bid
  where book_id = f_bookid and bid_type='buy' and rownum = 1;
  return v_found;
exception when no_data_found then
  return v_found;
end;

create or replace function CheckBidSell
(f_bookid textbook.book_id%type)
RETURN number IS
  v_found number(1) := 2;
begin
  select 1 into v_found from bid
  where book_id = f_bookid and bid_type='sell' and rownum = 1;
  return v_found;
exception when no_data_found then
  return v_found;
end;

create or replace function CheckPoss
(f_book_id bid.book_id%TYPE)
RETURN number IS
  v_returnvalue number;
  v_book_id number;
begin
  v_returnvalue := 1;
  begin
    select book_id into v_book_id
    from possible
    where book_id = f_book_id;
    exception when no_data_found then
      v_returnvalue := 2;
  end;
  return v_returnvalue;
end;