CS 6400 DATABASE SYSTEMS CONCEPTS AND DESIGN

TradePlaza - Project Phase 1 Report

Team 29

Jane Agim
Elaine Bentivegna
Nathan Gardner
Michael Lukacsko
ChinPo Tsai

TradePlaza Table of Contents

Key

- *Form* italic
- **Task Name** bolded
- **Table Name** underlined and bolded
- ***Button Name*** bolded and italic
- EntityType is displayed in blue
- Entity.attribute standard text where the entity name is specified first, separated by a dot and then attribute name is then specified
- *User.Sessiondata* italic and green after a blue entity type, for information about the current user from the HTTP Session/Cookie (e.g. *user.email*)
- $UserInputInformation, a dollar sign goes in front of user-input information (e.g. $UserPassword in a registration form)

TradePlaza Tables and Data Types

User

Attribute	Data Type	Nullable
email	string	Not null
password	string	Not null
first_name	string	Not Null
last_name	string	Not Null
nickname	string	Not Null
postal_code	string	Not Null

Item

Attribute	Data Type	Nullable
lister_email	string	Not null
title	string	Not null
item_no	int	Not null
game_type	string	Not null
number_of_cards	int	Null

platform	string	Null
media	string	Null
condition	string	Not Null
description	string	Null
listing_url	string	Not Null

Game_Platform_Map

Attribute	**Data Type**	**Nullable**
game_type	string	Not null
platform	string	Not null

Trade

Attribute	**Data Type**	**Nullable**
proposer_email	string	Not null
counterparty_email	string	Not null
proposer_item_no	int	Not null
counterparty_item_no	int	Not Null
proposed_date	date	Not Null
accept_reject_date	date	Null
status	string	Not Null
trade_history_link	string	Not Null

Location_Lookup (note: not in EER diagram)

Attribute	**Data Type**	**Nullable**
postal_code	string	Not null
city	string	Not null

state	string	Not null
latitude	float	Not Null
longitude	float	Not Null

Response_color_lookup (note: not in EER diagram)

Attribute	**Data Type**	**Nullable**
response_lower_range	float	Not null
response_upper_range	float	Not null
text_color	string	Not null

Rank_lookup (note: not in EER diagram)

Attribute	**Data Type**	**Nullable**
trade_lower_range	int	Not null
trade_upper_range	int	Not null
rank_label	string	Not null

Trade Plaza Business Constraints

Users

- Users are self-registered
- Users must provide:
    - Email
    - Password
    - First name
    - Last name
    - Nickname
    - Postal Code
- Postal codes, with city, state, and their central latitude & longitude will be used to validate input.
- Email addresses will be used to identify users in the system
- Nicknames will be unique to a single user

The TradePlaza System

- All items available to trade will include:
    - A title or name of the item
    - The items game type
    - The items condition
- Optionally, the user can include a description

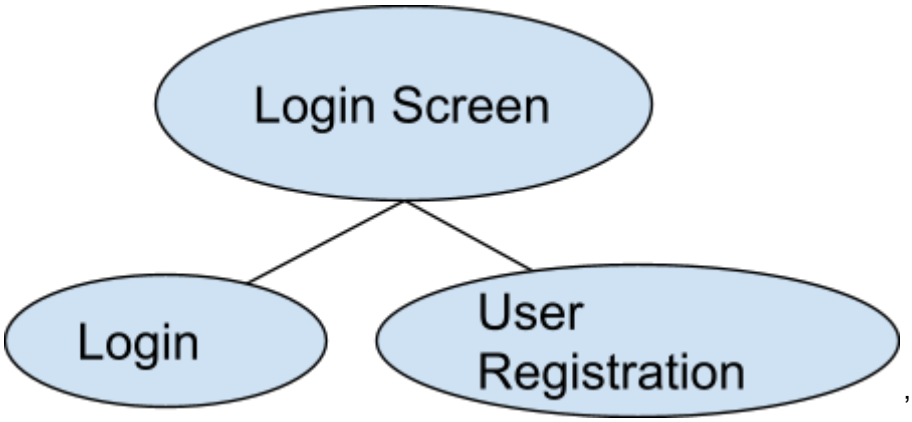
Trading Items

- Items listed to trade are assigned an item number
- Users that have listed items will be able to trade items with each other.
- Items associated with an unaccepted trade (a proposed trade not yet accepted or rejected) are not available for trading.
- Users cannot trade items with themselves
- Users with no listed items may browse items but cannot trade.
- The user proposing a trade is called the "**proposer**"
- The user receiving the proposed trade is called the "**counterparty**".
- A proposer will request a trade with a proposed item for one of the counterparty's items (called the "**desired item**").
- The date the proposal is made will be stored.
- The counterparty will then accept or reject the trade, with the date of acceptance or rejection also stored.
- Rejected Trades
    - Each of the items in the rejected trade can participate in a new proposed trade, however the specific item-for-item trade that was rejected cannot be proposed again with the same proposed item and desired item.

- Accepted Trades
    - Upon accepting, the counterparty will see the contact information for the proposer.
- Completed Trade
    - Trades are considered completed once they are accepted.
    - Items that have been traded cannot be traded again,
    - A user can enter the item into the system as a new item listing (which may have different/new information, such as an updated condition or description) for another trade.

TradePlaza Task Decomposition With Abstract Code

Login



Task Decomp

Lock Types: Read-only lookup on **User**
Number of Locks: Single, just need to read **User**
Enabling Conditions: None
Frequency: High
Consistency (ACID): not critical, order is not critical.
Subtasks: Mother Task of displaying the login screen is needed to split between registration and login.

Abstract Code

- Show “Sign In” panel accompanied by ***Login*** button and “New User” panel accompanied by ***Register*** button
- If user hits ***Register***
    - Jump to the **User Registration** task
- If user hits ***Login***
    - If user-entered *email/nickname* (‘$Email/Nickname’) is not empty and found in **User**
        - If user-entered *password* (‘$Password’) is blank
            - Display an error message of “Please enter your password”
        - If user-entered password is not blank, lookup the password associated with this user in **User** (user.password).
            - If user-entered *password* (‘$Password’)= user.password
                - Jump to **Main menu** task

            - If user-entered *password* ('$Password') is not equal to user.password
                - Display an error message of "Password incorrect"
    - If user-entered *email/nickname* ('$Email/Nickname') is empty
        - Display an error message of "Please enter your email/nickname"
    - If user-entered *email/nickname* ('$Email/Nickname') is not empty but not found in **User**
        - Display an error message of "User email/nickname ('$Email/Nickname') not found"

User Registration

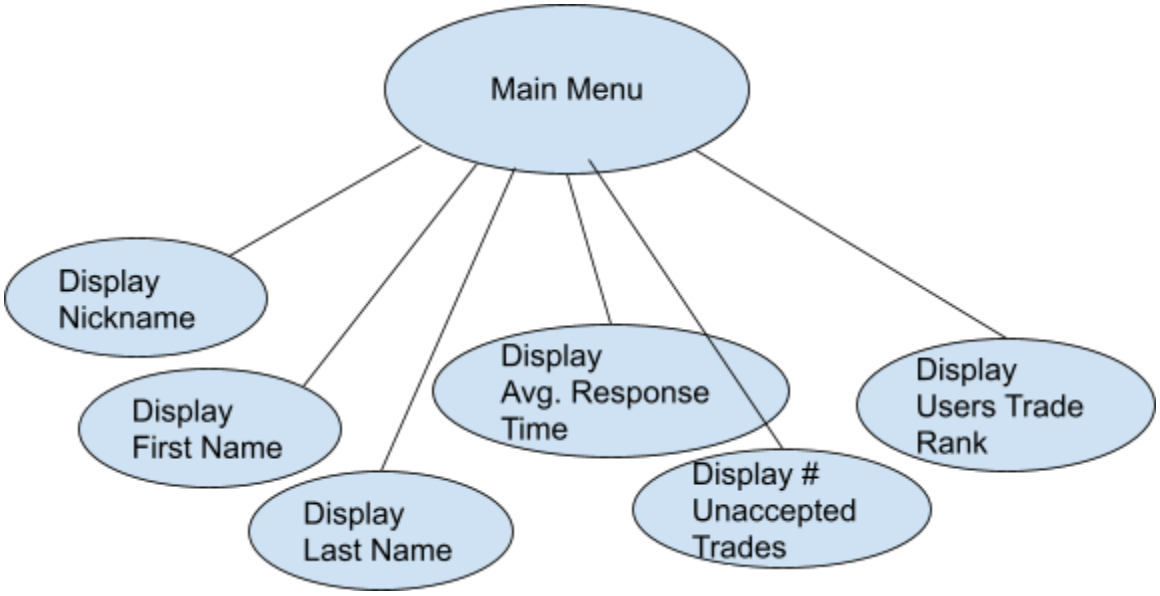
Task Decomp

Locktypes: 1 insert to the **User** table; 1 lookup in **Location_Lookup** table
Number of Locks: 2
Enabling Conditions: Enabled by clicking the ***Register*** button
Frequency: 1 (only once for every new user)
Consistency (ACID): not critical
Subtasks: none

Abstract Code

- Click ***Register*** button and enters the *registration* form.
- Input the following info: (1) $Email (2) $Password (3) $Nickname (4) $First Name (5) $Last Name (6) $Postal Code.
- Some checks have to be done before we proceed:
    - Are all fields input? If not, show a popup error message
    - Is the $Email unique across the database? If not, show a popup error message.
    - Is the $Nickname unique across the database? If not, show popup an error message.
    - Does the $Postal Code exist in **Location_Lookup** table? If not, show a popup error message.
- After passing the above checks, add the newly-registered info into **User** table.

Main Menu



Task Decomp

Locktypes: 2 Read Only lookups on **User** and **Trade** table
Number of Locks:
Enabling Conditions: User must be registered and logged in
Frequency: High
Consistency (ACID): Non-critical.
Subtasks: Mother task of registering and logging in required. No other subtasks

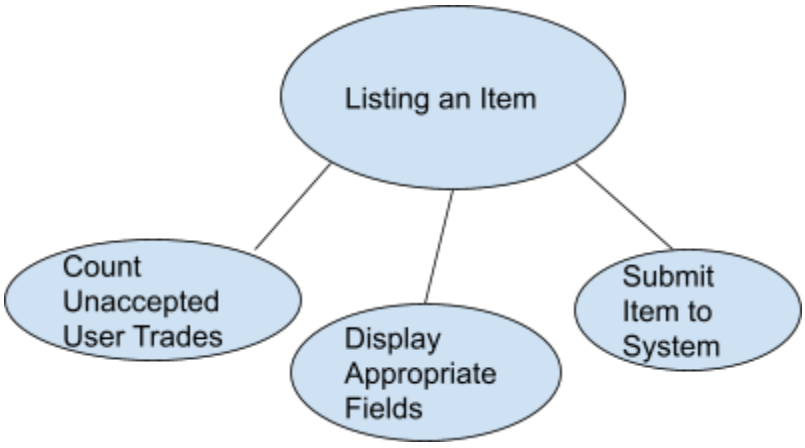
Abstract Code

- Display “Welcome” Message accompanied by the current user’s First Name, Last Name, and, in parenthesis, the current user’s Nickname
    - Lookup first_name, last_name, and nickname from **User** table
- Display the number of unaccepted trade
    - Lookup status = unaccepted in **Trade** table to get count
    - If greater than zero, links to accept/reject trades
- Display the current user’s average response time
    - Count total instance where a lookup of trade propsed_date and accept_reject_date are present where status = accepted or rejected
    - Map the count to a response range and text color.
        - Use the **Response_color_lookup** table to find where the user’s response time is greater than or equal to the response_lower_range and

less than the response_upper_range. Return text_color associated with the record.

- Display the current user's current trader rank
    - Count total instances where status = 'Accepted'
    - Map this count to a rank
        - Use the **rank_lookup** table to find the record where the trade count is greater than or equal to the trade_lower_range and less than the trade_upper_range. Return the rank_label associated with this record.
- Display link to "List Item"
    - Upon clicking, navigates the user to **List Item**
- Display link to "My items"
    - Upon clicking, navigates the user to **My Items**
- Display link to "Search items"
    - Upon clicking, navigates the user to **Search**
- Display link to "Trade history"
    - Upon clicking, navigates the user to **Trade History**
- Display link to "Logout"
    - Upon clicking, the user will be logged out and brought back to **Login**

Listing an Item



Task Decomp

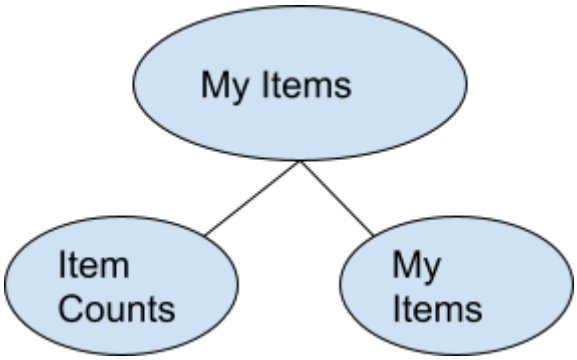
Lock Types: 1 lookup on **Trade**, 1 insert in **Item**
Number of Locks: 2 Locks
Enabling Conditions: User must be logged in, Current user must not have more than 2 unaccepted trades where they are the counterparty

Frequency: High
Consistency (ACID): not critical, the appropriate fields should be displayed first based on the Game Type selection, then number of unaccepted trades must be counted, and then
Subtasks: Mother Task “Listing an Item”, subtasks for counting the number of unaccepted user trades, displaying appropriate fields based on game type, and submitting an item to the system

Abstract Code

- User clicked on ***List Item*** from **Main Menu**
- Display $Game Type, $Title, $Condition, and $Description input boxes/radio buttons
- If Video Game is selected as $Game Type, also display $Platform and $Media
    - Limit Platform choices to Nintendo, Playstation, and Xbox
    - Limit Media choices to Optical Disc, Game Card, and Cartridge
- Else If Computer Game is selected as Game Type, also display $Platform
    - Limit choices to Linux, macOS, and Windows
- When User clicks ***Submit***:
    - Validate that no fields except Description are left blank
        - If there are blanks, display error message
    - Lookup the count of unaccepted trades where user.email = *user.email*
        - If the number of unaccepted trades where the user is the counterparty is greater than or equal to 2, display an error message
    - If the input validation and number of unaccepted trades are acceptable, insert a record into the **Items** table and display a success message with the recently created Item Number (auto-incremented) displayed

My Items



Task Decomp

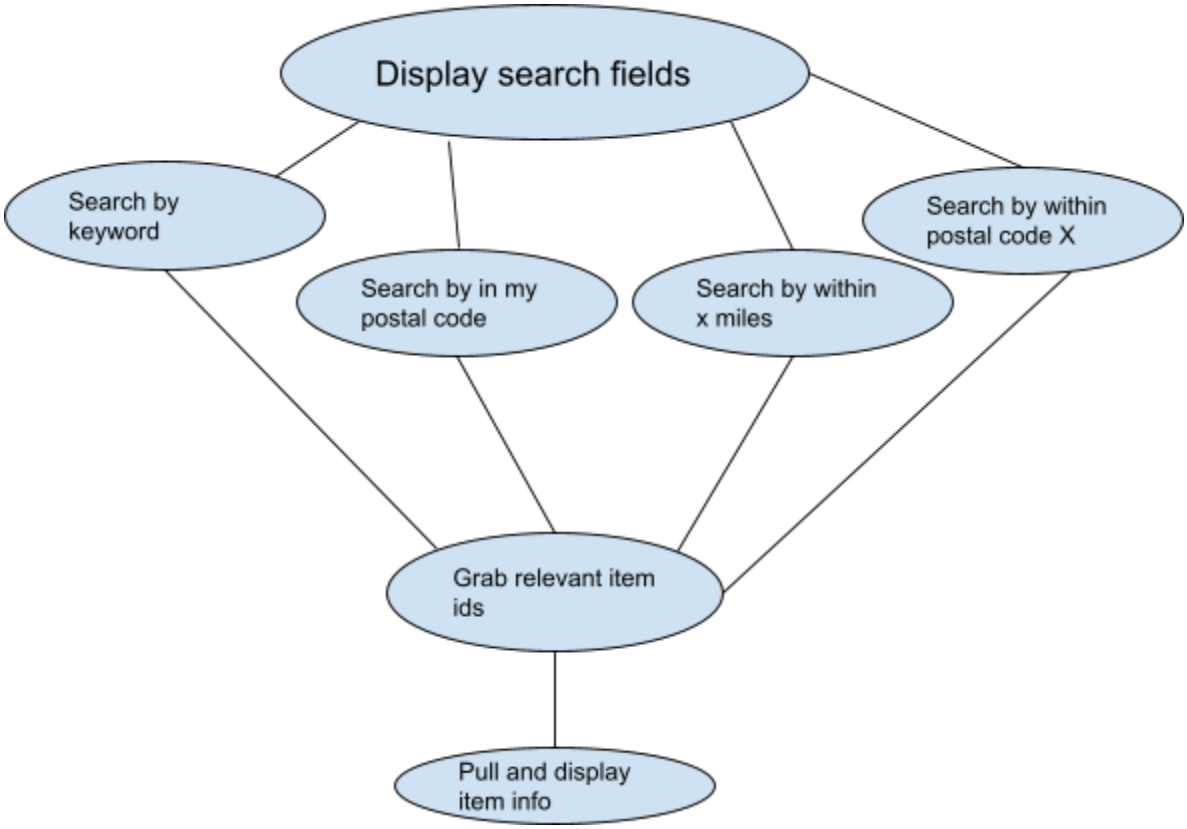
Locktypes: 1 lookup on **User** table, all read-only

Number of Locks: Single
Enabling Conditions: User must be logged in
Frequency: Low
Consistency (ACID): not critical, order is not critical
Subtasks: Mother task “my items” split into subtasks “item counts” and “my Items”

Abstract Code

- The user clicks the ***my items*** button in the main menu and enters the **My Items** task
- If user is logged in,
    - Find the current user in the **user** table using *user.email*
- Show *item counts* and *my items* tables
    - Under *item counts*
        - Using the lister_email attribute, search the **item** table, count and display number of listed:
            - Board games
            - Playing card games
            - Computer games
            - Collectible card games
            - Video Games
        - Add up total number of listed games and display in table
    - Under *my items*
        - For each listed item, use the item_no attribute to find in the **item** table, and display in table (sorted by ascending item number) :
            - Item number
            - Game type
            - Title
            - Condition
            - Description (first 100 characters, use ellipses for descriptions longer than 100 characters)
            - ***Detail*** button
                - Upon clicking the “detail” button on an item, jump to **View Item** task
        - If a user has no listed items, this table should still be visible, albeit empty.

Searching for Items



Task Decomp

Locktypes: 3 lookups on **User**, **Item** and **Location_lookup**
Number of Locks: Several different schema constructs are needed
Enabling Conditions: Display search fields is enabled by user login, display of search results is enabled by a user-initiated search
Frequency: High
Consistency (ACID): not critical, it's unlikely the information changes between a search being submitted and displayed
Subtasks: Mother task of displaying the search prompt, sub-tasks of searching and displaying search results required

Abstract Code

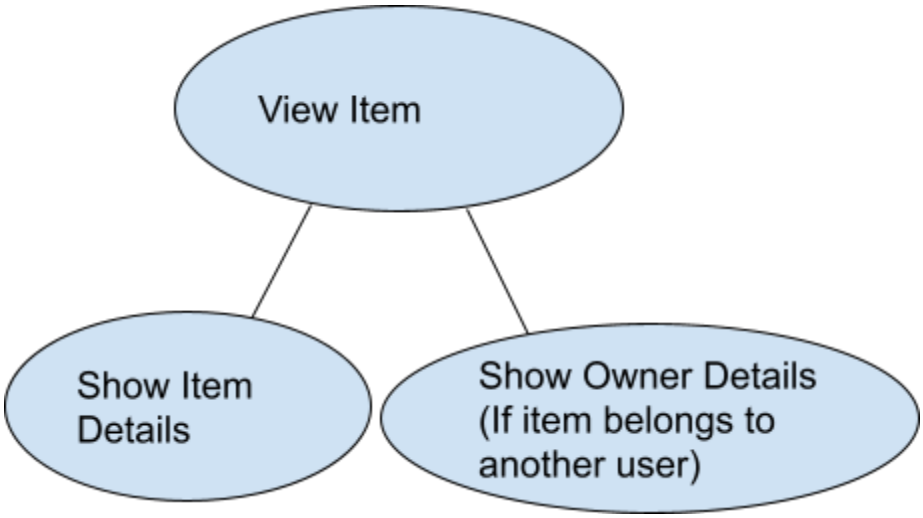
- Display Search panel with search options
    - User could chose not to search, nothing happens
    - If user chooses to search, return the item numbers produced by the search

        - If user searches by 'Keyword', user selects radio button next to "by keyword" and inputs $KeywordSearchTerm
            - Query **Item** for Description contains '$KeyWordSearchTerm' or Title contains '$KeyWordSearchTerm%'
            - Query **Trade** for the item numbers and return a list of item numbers where status='Unaccepted'
        - If user searches by 'In my postal code', user selects radio button next to "in my postal code"
            - Query **User** for the emails of users in the searcher's postal code, *user.postalcode*
            - Query **Item** for items for which the lister_email is in the list above
            - Query **Trade** for the item numbers and return a list of item numbers where status='Unaccepted'
        - If user searches by 'Within 'X' miles of me', user selects radio button next to "Within 'X' miles of me' and inputs the number of miles to search within, $MilesWithin
            - Perform that Haversine formula between *user.postalcode* and each postal code in **Location_lookup**
            - Create a list of postal codes for which the above formula yields a value<=$MilesWithin
            - Query **User** for all user emails whose postal codes are within the list above
            - Query **Item** for the item numbers for which the item.lister_email is in the list above
            - Query **Trade** for the item numbers and return a list of item numbers where trade.status='Unaccepted'
        - If user searches by 'In postal code', user selects radio button next to "In postal code:" and inputs the postal code they want to search within $PostalCodeSearch
            - If the postal code does not exist in **Location_lookup** display an error message of "Postal code invalid"
            - If the postal code exists in **Location_lookup**
                - Query **User** for the emails of users whose postal_code=$PostalCodeSearch
                - Query **Item** for item numbers belonging to the list of emails found above
                - Query **Trade** for the item numbers and return a list of item numbers where trade.status='Unaccepted'
    - If user searched, use the returns item numbers to find relevant information
        - Find the Item number, Game type, Title, Condition, link to listing and description of relevant item numbers
            - Query **Item** for title, game_type, condition, listing_url, description returned for the list of item numbers
        - Find the seller response time, rank and distance

- Query **Item** for lister_email associated with each item number
    - Find the lister response time
        - Use lister_email to query **Trade** to find all trades where trade.status='Accepted' and the trade.lister_email=counterparty_email
        - If there are no such records, return "None"
        - If there is at least one such record, find the average of trade.accept_reject_date-trade.proposed_date
    - Find the lister rank
        - Count trades the lister has successfully completed
            - Use lister_email address to count how many Trade instances there are with status='Accepted' and the lister's email address as either proposer_email or counterparty_email.
        - Map this count to a rank
            - Use **Rank_Lookup** to find the record where the trade count is greater than or equal to the trade_lower_range and less than the trade_upper_range. Return the rank_label associated with this record.
    - Find the lister distance from searchee
        - If user searched by "In my postal code", this value should be 0.0
        - If user searched by "In postal code"
            - Lookup *user.postalcode* in **Location_lookup** table to find searcher's latitude and longitude, lookup $PostalCodeSearch in **Location_lookup** to find seller's latitude and longitude, compute distance between these two locations with haversine formula
        - Else if user searched by keyword or 'Within X miles of me'
            - Lookup lister's postal code by querying **User** where *user.email*=item.lister_email and finding the postal code
            - Lookup the lister's postal code in **Location_lookup** to find the lister's latitude and longitude, lookup *user.postalcode* in **Location_lookup** table to find searcher's latitude and longitude, compute distance between these two locations with the haversine formula

- Display search information in tabular form
  - Display a row for each item number returned
    - Display each item number in a column called “Item #”
  - Display game type, title, condition
    - Map the game_type, title, and condition info fetched above into table columns “Game type”, “Title”, “Condition”
  - Display truncated description for relevant item numbers
    - If description <=100 characters, display full description in a column “Description”
    - Else if description >100 characters, display first 100 characters and place an ellipsis (...) at the end to indicate the description is truncated. Display in column “Description”
  - Display response time
    - Display response time rounded to the nearest tenth decimal place
    - Query **Response_color_lookup** to map the response time to the color it should be mapped to
  - Display rank
    - Display Rank information fetched above in a column “Rank”
  - Display distance
    - Display distance information fetched above in a column “Distance” rounded to the nearest hundreds place
  - Display link
    - In a column with no name, create a hyperlink called “Detail” that links to the item.listing_url fetched for the item number
    - If user clicks this link
      - Jump to **View Item** task

View Item



Task Decomp

Locktypes: Read-only in **Item** table; Read-only in **User** table
Number of Locks: 2
Enabling Conditions: Enabled by (1) clicking ***Detail*** button in *My items* form (2) clicking **Detail** button in *Search Result* form
Frequency: Not specified (but should be quite often)
Consistency (ACID): Critical
Subtasks:
This task can be separated into 2 tasks:

- Show item details, including (1) Item.item_no (2) Item.title (3) Item.game_type (4) Item.platform (5) etc.
- Show owner details (only shows when viewing other user's items)

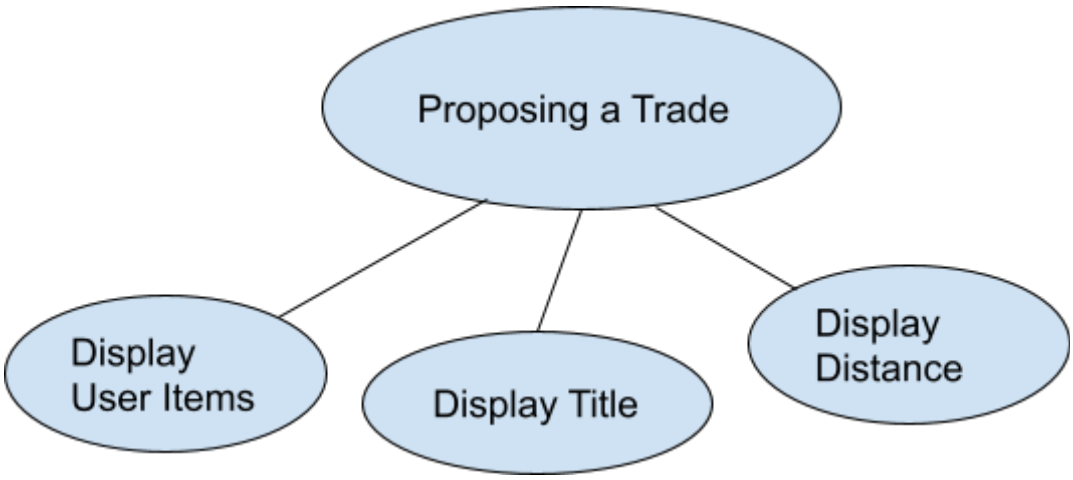
Abstract Code

- Enter the *View Item* form after clicking ***My Items*** button and ***Detail*** button
    - Show item details, including (1) Item.item_no (2) Item.title (3) Item.game_type (4) Item.media (if Item.game_type is video game), (5) Item.platform (if Item.game_type is video game or computer game), (6) Item.no_cards (if Item.game_type is "Collectible Card Game"), (7) Item.condition (8) Item.description (if Item.description is not null)
    - Note: Owner info doesn't have to show up here since user is viewing his/her own items.

- Enter the *View Item* form after clicking ***Detail*** button in the *Search Result* form.
    - Find Item.lister_email by querying **Item** table on Item.item_no.
        - If item.lister_email=*user.email* (i.e, the user is viewing his/her own item)
            - Show item details, including (1) Item.item_no (2) Item.title (3) Item.game_type (4) Item.media (if Item.game_type is video game), (5) Item.platform (if Item.game_type is video game or computer game), (6) Item.no_cards (if Item.game_type is “Collectible Card Game”), (7) Item.condition (8) Item.description (if Item.description is not null)
            - No need to show user info.
        - If item.lister_email does not equal *user.email* (i.e, the user is viewing other user’s item)
            - Show item details, including (1) Item.item_no (2) Item.title (3) Item.game_type (4) Item.media (if Item.game_type is video game), (5) Item.platform (if Item.game_type is video game or computer game), (6) Item.no_cards (if Item.game_type is “Collectible Card Game”), (7) Item.condition (8) Item.description (if Item.description is not null)
            - Use Item.lister_email and lookup **User** table, and show (1) User.first_name and User.last_name (2) Location (by using User.postal_code as key to search in **Location_Lookup** table (3) Response Time (4) Rank (5) Distance.
                - To show response time:
                    - Use **Trade** table, and find the instances with the following condition and calculate the average response time (rounded to tenths):
                        - Trade.status = accepted or rejected
                        - Trade.counterparty_email = User.email
                    - Map the number to a color by using **Response_color_lookup**
                - To show rank:
                    - Count total numbers of completed trades of the User either as a proposer or as a counter party.
                    - In other words, use **Trade** table and count the number of instances with the condition:
                        - Trade.status = ‘Accepted’
                        - (Trade.proposer_email = User.email) OR (Trade.counterparty_email = User.email)
                    - Map this count to a **Rank_lookup** table.
                - To show distance:
                    - From Item.lister_email, search in **User** table and get User.postal_code, and use it as the key to search in **Location_Lookup** table to get latitude and longitude coordinates.

- Use *User.email* to get User.postal_code and lookup the latitude and longitude associated with the User in the **Location_Lookup** table
- Use the haversine distance formula to calculate the distance between the two users. Round the figure to the nearest hundredth.
    - If the distance is 0 (i.e, the item owner has the same postal_code as the current user), then don't show the distance.
    - If distance is less than 25miles, present it with a GREEN background
    - If distance is 25~50 miles, present it with YELLOW background
    - If distance is 50~100 miles, present it with ORANGE background
    - If distance is over 100 miles, present it with RED background.

Proposing a Trade



Task Decomp

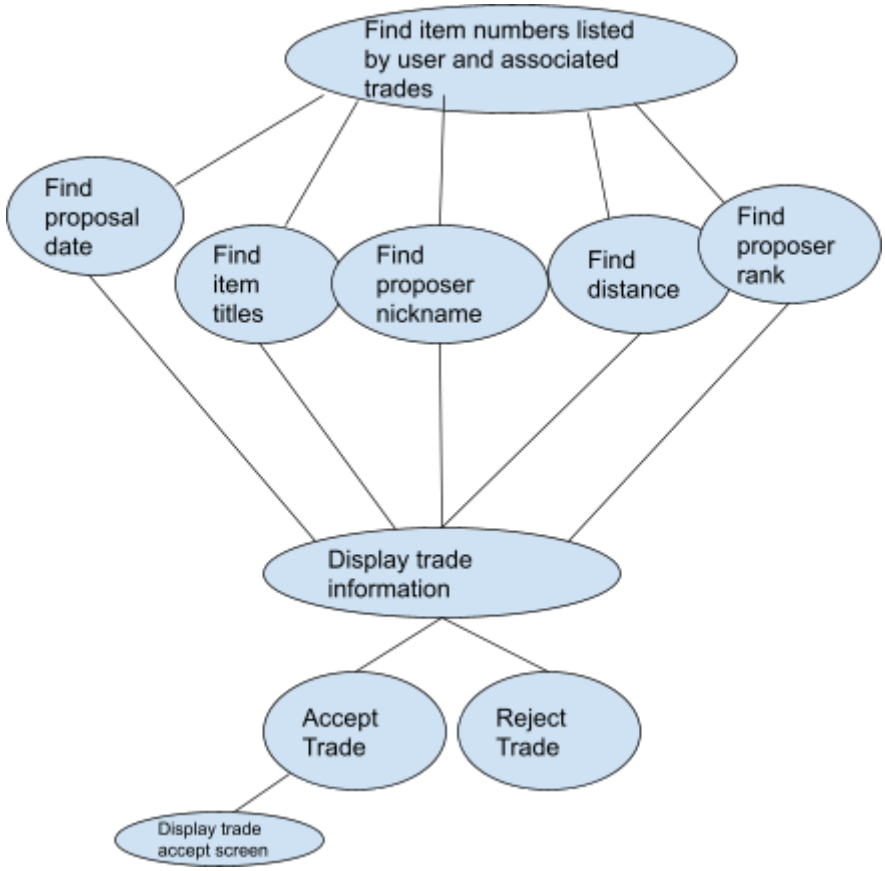
Locktypes: Read only lookup on **Item** and **Location_Lookup**, One insert into **Trade**
Number of Locks: 3 locks
Enabling Conditions: Current user must not have more than 2 unaccepted trades where they are the counterparty
Frequency:
Consistency (ACID):

Subtasks: Mother task of finding items listed by the current user, subtasks of displaying relevant information based on item numbers. Confirming a trade is also a subtask because it will insert a new proposed trade into the database

Abstract Code

- Display counterparty distance if the counterparty distance is greater than or equal to 100.0 miles
    - Lookup the postal code associated with the current user
        - In the **User** table, lookup the postal_code associated where email=*user.email*
    - Lookup the postal code associated with the proposer email in the **User** table
        - Lookup the latitude and longitude coordinates associated with the proposer's postal code in the **Location_lookup** table
        - Lookup the latitude and longitude associated with *user.postalcode* in the **Location_lookup** table
        - Use the haversine formula to calculate the distance between the two users. Round the figure to the nearest hundredth.
            - If the counterparty distance is greater than or equal to 100.0 miles, a warning message containing that distance is displayed at the top of the form.
            - Else, the distance is not displayed
- Display text "You are proposing a trade for" along with the item title if selected to be traded
- Display a listing with selector to select an item and propose a trade
    - Display item list
        - Select item_no, game_type, title, and condition from **Item** table where lister_email=*user.email*
        - Order results by item_no
    - Display item selector
        - Display a radio button to select items from item list
- Display "Confirm" link
    - Upon click, a new record with proposer_email, counterparty_email, proposer_item_no, counterparty_item_no, proposed_date, and status will be written to the **Trade** table
    - Display confirmation message
        - User will have the option to return to the main menu

Accept and Reject Trades



Task Decomp

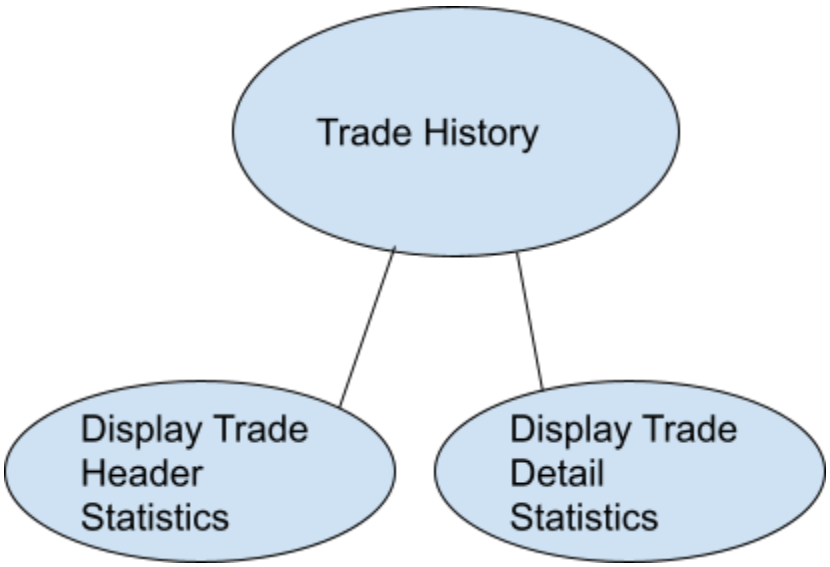
Locktypes: Lookups on **Trade, User**, **Item**, **Location_lookup**, **Rank_lookup,** 1 write to **Trade**
Number of Locks: Several different schema constructs are needed
Enabling Conditions: Landing on the page requires the user navigating from the main menu (which requires being logged in) and having unaccepted trades so that the link to accept/reject trades exists
Frequency: Medium
Consistency (ACID): Important - you don't want to display a trade that involves an item that is no longer available. However, if this happened, you could just display an error message indicating that the item became unavailable.
Subtasks: Mother task of finding items listed by the user, subtasks of displaying relevant information based on item numbers. Accepting and rejecting the trade are also sub-tasks because they require writing to the database (all other subtasks just require reading)

Abstract Code

- Do a lookup in the **Item** table to find all item numbers associated where lister_email=*user.email*
- Find trade information
    - Using item numbers found above, in **Trade** find all trades associated with the item numbers.
- Display trade date
    - Display the trade.proposal_date associated with a trade in **Trade**
- Display Desired Item
    - Lookup the item number in the **Items** table. Display the title associated with the item number
- Display proposer nickname
    - Find the proposer email associated with the trade
        - Lookup the nickname associated with the proposer email in **User** and display the nickname
- Display proposer rank
    - Count trades the proposer has successfully completed
        - Use proposer email address to count how many Trade instances there are with trade.status='Accepted' and the proposer's email address as either trade.proposer_email or trade.counterparty_email.
    - Map this count to a rank
        - Use the **Rank_lookup** to find the record where the trade count is greater than or equal to the trade_lower_range and less than the trade_upper_range. Return the rank_label associated with this record.
- Display proposer distance
    - Lookup the postal code associated with the current user
        - In the **User** table, lookup the postal_code associated where email=*user.email*
    - Lookup the postal code associated with the proposer email in the **User** table
        - Lookup the latitude and longitude coordinates associated with the proposer's postal code in the **Location_lookup** table
        - Lookup the latitude and longitude associated with *user.postalcode* in the **Location_lookup** table
        - Use the haversine formula to calculate the distance between the two users. Round the figure to the nearest tenth. Display the figure.
- Display the Proposed Item
    - Display the proposed item associated with a trade
- It's possible the user clicks nothing, in which case, nothing happens
    - If the user clicks accept
        - Display a dialogue with the proposer's email and firstname
            - Display the proposer email associated with the trade
            - In the **User** table, look up the first name associated with the proposer's email

        - In the **Trade** table update the status of the trade to ‘Accepted’
        - After this, re-query **Trade** for trades where proposer’s email address is either proposer_email or counterparty_email and status=’Accepted’.
            - If trades meet this criteria, repeat the process outlined above to display these trades and their information
            - If no trades meet this criteria, return this user to the main menu
    - If the user clicks reject
        - In **Trade**, update the status of the trade to ‘Rejected’
        - After this, re-query **Trade** for trades where proposer’s email address is either proposer_email or counterparty_email and status=’Accepted’.
            - If trades meet this criteria, repeat the process outlined above to display these trades and their information
            - If no trades meet this criteria, return this user to the main menu

Trade History



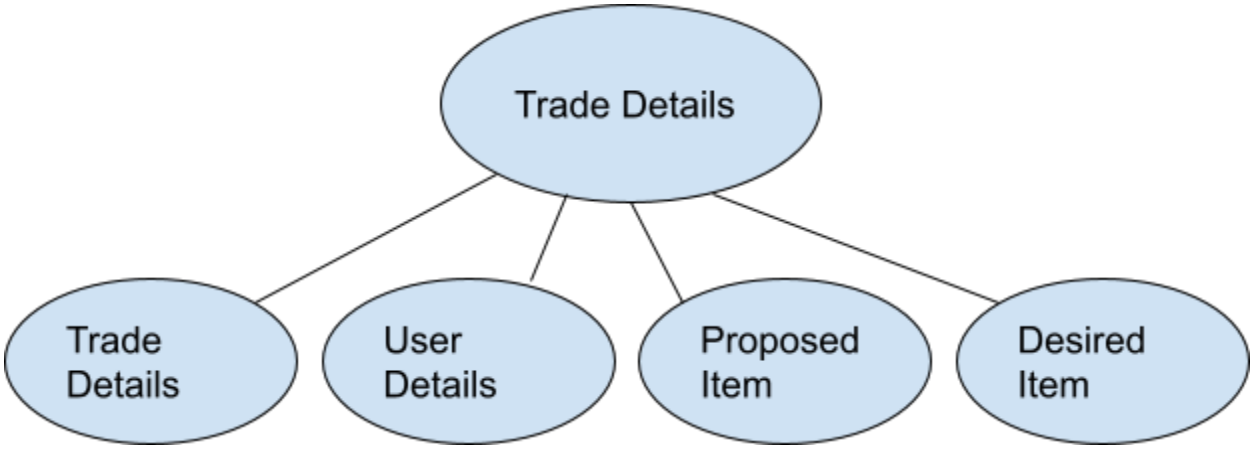
Task Decomp

Lock Types: 2 read-only lookups on **Trade** table
Number of Locks: 2 Locks
Enabling Conditions: User must be logged in, user must have clicked “Trade History” button from main menu
Frequency: Low
Consistency (ACID): not critical, even if trade is being accepted or rejected while a user is looking at it. Order is not critical
Subtasks: Display Trade Header Statistics, Display Trade Detail Statistics

Abstract Code

- User clicked on ***Trade History*** from **Main Menu**
- Run **Trade History** Task
    - Based on the *user.email*, use the **Trade** table to find the following grouped by the current user's role:
        - Total trades
        - Count of accepted trades
        - Count of rejected trades
        - Count of rejected trades divided by the total number of trades
            - If the percentage is greater than or equal to 50%, highlight the background in red
    - Based on the *user.email*, use the **Trade** table to find the following by sorted by acception/rejection date and then proposed date (descending):
        - Trade.Proposed Date
        - Trade.Accepted/Rejected Date
        - Trade.Trade Status
        - Trade.Response Time (days) (calculated as trade.accept_reject_date-trade.proposal_date)
        - Trade.User Role (If *user.email*=trade.counterparty_email, display "Counterparty". If *user.email*=trade.lister_email, display "Proposer")
        - Trade.Proposed Item
        - Trade.Desired Item
        - Trade.Other User
        - Trade.Trade ID masked as "Detail"
            - Clicking on "Detail" for a row will run the *Trade Detail* form associated with the row

Trade Details



Task Decomp

Locktypes: 1 lookup on **Trade** table, 1 lookup on the **User** table, 2 lookups on the **Item** table, all read-only
Number of Locks: Single
Enabling Conditions: User must be logged in, current user must have engaged in at least one trade
Frequency: Low
Consistency (ACID): not critical, order is not critical
Subtasks: Mother task, “trade details” split into subtasks, “trade details”, “user details”, “proposed item”, and “desired item”

Abstract Code

- The user clicks on the ***detail*** button in the **Trade History** task and enters the **Trade Details** task
- If the user is logged in and has participated in at least one trade,
    - Find the current user in the **user** table using *user.email*
    - Find the current trade (trade that prompted the jump to this task) in the **trade** table, then
- Show *trade details*, *user details*, *proposed item*, and *desired item* forms
    - Under *trade details*
        - Using the proposer_item_no, counterparty_item_no attributes, find in the **trade** table and display:
            - Proposed date of trade
            - Date of acceptance or rejection of trade
            - Status of trade

- Current user's role in trade (If *user.email*=trade.counterparty_email, display "Counterparty". If *user.email*=trade.lister_email, display "Proposer")
- The response for this trade in days (calculated from trade.accept_reject_date-trade.proposal_date)

- Under *user details*
  - If *user.email* is the prosposer_email, use the counterparty_email attribute. If *user.email* is the counterparty_email, use the proposer_email attribute. Find this email in the **user** table and display:
    - Other user's nickname
    - Other user's distance away from current user (in miles, hundredths)
      - Use the haversine formula to calculate the distance between the two users. Round the figure to the nearest hundredth.
    - (If trade is accepted), find and display
      - Other user's name
      - Other user's email address
- Under *proposed item*
  - Using the proposer_item_no attribute, find in the **item** table, and display for the item proposed to be traded away:
    - Item number
    - Title of the game
    - Type of the game
    - Condition of the game
    - Description of the game (full)
- Under *desired item*
  - Using the counterparty_item_no attribute, find in the **item** table, and display for the item desired:
    - Item number
    - Title of the game
    - Game type
    - Condition of the game