CS 6400 DATABASE SYSTEMS CONCEPTS AND DESIGN

TradePlaza
Phase 2 - Abstract Code + SQL

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*). **When in SQL code, to avoid ambiguity when copying, we also enclose in ` `:** (e.g. `*user.email*`)
- $UserInputInformation, a dollar sign goes in front of user-input information (e.g. $UserPassword in a registration form)
- *Stored_Variable*, sometimes it is clearer to split complex queries into separate queries. We pass a stored variable from query to query using this convention. This can also represent data about the page we are on (e.g. trade_id associated with a trade_url, item_id associated with a listing_url).

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
condition	string	Not Null
description	string	Null
listing_url	string	Not Null

Item_Collectable_Card_Game

Attribute	Data Type	Nullable
item_no	int	Not null
lister_email	string	Not null
number_of_cards	int	Not null

Item_Board_Game

Attribute	Data Type	Nullable
item_no	int	Not null
lister_email	string	Not null

Item_Playing_Card_Game

Attribute	Data Type	Nullable
item_no	int	Not null
lister_email	string	Not null

Item_Computer_Game

Attribute	Data Type	Nullable
item_no	int	Not null
lister_email	string	Not null
platform	string	Not Null

Item_Video_Game

Attribute	Data Type	Nullable
item_no	int	Not null
lister_email	string	Not null
platform	string	Not Null
media	string	Not Null

Platform

Attribute	Data Type	Nullable
name	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
auto_trade_id	int	Not Null

Location_Lookup (note: not in EER diagram, EER diagram just has a postal_code relation)

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

Distance_color_lookup (note: not in EER diagram)

Attribute	Data Type	Nullable
distance_lower_range	float	Not null
distance_upper_range	float	Not null
background_color	string	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 Abstract Code + SQL query

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).

```
Select password FROM User WHERE User.email = '$Email';
```

            - If user-entered *password* (‘$Password’) is not equal to user.password
                - Display an error message of “Password incorrect” and return to ***Login***
    - If user-entered *email/nickname* (‘$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**

```
SELECT COUNT(*) from User WHERE (User.email = '$Email' OR
User.nickname = '$Nickname');
```

        - Display an error message of “User email/nickname (‘$Email/Nickname’) not found”

User Registration

Abstract Code

- Click ***Register*** button to enter the *registration* form.
- Input the following info: (1) $Email (2) $Password (3) $Nickname (4) $First Name (5) $Last Name (6) $Postal Code.
- Perform data validation on the user input:
    - 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.

```
SELECT COUNT(*) FROM User WHERE
UPPER(User.email)=UPPER('$Email')
```

        - If count returned is greater than 0, display error
    - Is the $Nickname unique across the database? If not, show popup an error message.

```
SELECT COUNT(*) FROM USER where
UPPER(User.nickname)=Upper('$Nickname')
```

        - If count returned is greater than 0, display error
    - Does the $Postal Code exist in **Location_Lookup** table? If not, show a popup error message.

```
SELECT COUNT(*) FROM Location_Lookup WHERE
Location.Lookup.postal_code= '$Postal Code'
```

        - If count returned is less than 1, display error
- After passing required data validation, add the newly-registered info into **User** table.

```
INSERT INTO User(Email, password,first_name,last_night,nickname,postal_code)
VALUES ('$Email', '$Password', '$First Name', '$Last Name' , '$Nickname', '$Postal Code')
```

Main Menu

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

```
SELECT first_name, last_name, nickname FROM User WHERE
User.email=`user.email`
```

- Display the number of unaccepted trade
    - Lookup status = unaccepted in **Trade** table to get count

```
SELECT COUNT(*) FROM Trade WHERE (Trade.status= 'unaccepted' AND
Trade.proposer_email=`user.email`)
```

        - 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

```
SELECT
coalesce(avg(julianday(accept_reject_date)-julianday(proposed_date)),0)  as
avg_response_time
FROM Trade WHERE
(Trade.status = "accepted" OR Trade.status = "rejected")) and
(Trade.proposer_email = `user.email` or
Trade.counterparty_email=`user.email`)
```

    - Store the result of the query as a variable called *avg_response_time*
    - 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.

```
SELECT text_color FROM Response_color_lookup WHERE
response_lower_range<=round(*avg_response_time* ,1) AND
response_upper_range=>round(*avg_response_time* ,1)
```

- Display the current user's current trader rank
    - Count total instances where status = 'accepted'

```
SELECT COUNT(*) FROM Trade WHERE Trade.status= 'accepted'
AND (Trade.proposer_email= `user.email` OR
Trade.counterparty_email=`user.email`)
```

        - Store this as *completed_trade_count*
    - 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.

```
SELECT rank_label from Rank_lookup WHERE
trade_lower_range<=*completed_trade_count* and
trade_upper_range>=*completed_trade_count*
```

- 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

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. Pull these from the database using the query below

```
SELECT name from Platform
```

    - 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 Trade.counterparty_email = *user.email*

```
SELECT COUNT(*) FROM Trade WHERE Trade.status= 'unaccepted' AND Trade.counterparty_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

```
INSERT INTO Item (lister_email, title, item_no, condition, description, listing_url) VALUES (`user.email`, '$Title', #item_no, '$Condition', '$Description', '$Listing URL');
```

- Depending on which game_type was selected, the relevant insert statement will insert the game_type information into the relevant table

```
INSERT INTO Item_Collectable_Card_Game (item_no, number_of_cards) VALUES (#item_no, $number_of_cards);

INSERT INTO Item_Board_Game (item_no) VALUES (#item_no);

INSERT INTO Item_Playing_Card_Game (item_no) VALUES (#item_no);

INSERT INTO Item_Computer_Game (item_no, media) VALUES (#item_no, '$Platform')

INSERT INTO Item_Video_Game (item_no, media) VALUES (#item_no, '$Media','$Platform')
```

My Items

Abstract Code

- The user clicks the ***my items*** button in the main menu and enters the **My Items** task
- Show *item counts* and *my items* tables
    - Under *item counts*
        - Using the lister_email attribute, search the **item** table where lister_emai=*user.email*, count and display number of listed:
            - Board games
            - Playing card games
            - Computer games
            - Collectible card games
            - Video Games

```
SELECT CASE WHEN cg.item_no IS NOT NULL then
‘Collectable Card Game’ WHEN bg.item_no IS NOT NULL then
‘Board Game’ WHEN pcg.item_no IS NOT NULL then ‘Playing
Card Game’ WHEN comg.item_no is NOT NULL then
‘Computer Game’ WHEN vg.item_no is NOT NULL then ‘Video
Game’ END as game_type, count(*)
FROM Item i
LEFT JOIN Item_Collectable_Card_Game cg on
i.item_no=cg.item_no
LEFT JOIN Item_Board_Game bg on
i.item_no=bg.item_no
LEFT JOIN Item_Playing_Card_Game pcg on
i.item_no=pcg.item_no
LEFT JOIN Item_Computer_Game comg on
i.item_no=comg.item_no
LEFT JOIN Item_Video_Game vg on i.item_no=vg.item_no
WHERE lister_email=`user.email`
and i.item_no not in (select proposed_item_no from Trade
where status= ‘accepted’)
and i.item_no not in (select counterparty_item_no from Trade
WHERE status= ‘accepted’)
GROUP BY CASE WHEN cg.item_no IS NOT NULL then
‘Collectable Card Game’ WHEN bg.item_no IS NOT NULL then
‘Board Game’ WHEN pcg.item_no IS NOT NULL then ‘Playing
Card Game’ WHEN comg.item_no is NOT NULL then
‘Computer Game’ WHEN vg.item_no is NOT NULL then ‘Video
Game’ END
```

- Add up total number of listed games and display in table

```
SELECT COUNT(*) FROM Item
WHERE lister_email=`user.email`
and item_no not in (select proposed_item_no from Trade
where status= 'accepted')
and item_no not in (select counterparty_item_no from Trade where
status= 'accepted')
```

- 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

```
SELECT i.item_no
, CASE WHEN cg.item_no IS NOT NULL then 'Collectable Card Game'
WHEN bg.item_no IS NOT NULL then 'Board Game' WHEN
pcg.item_no IS NOT NULL then 'Playing Card Game' WHEN
comg.item_no is NOT NULL then 'Computer Game' WHEN vg.item_no
is NOT NULL then 'Video Game' END as game_type
, title, condition, description, listing_url
FROM Item i
LEFT JOIN Item_Collectable_Card_Game cg on i.item_no=cg.item_no
LEFT JOIN Item_Board_Game bg on
i.item_no=bg.item_no
LEFT JOIN Item_Playing_Card_Game pcg on i.item_no=pcg.item_no
LEFT JOIN Item_Computer_Game comg on i.item_no=comg.item_no
LEFT JOIN Item_Video_Game vg on i.item_no=vg.item_no
WHERE lister_email=`user.email`
and i.item_no not in (select proposed_item_no from Trade where
status= 'accepted')
and i.item_no not in (select counterparty_item_no from Trade where
status= 'accepted')
```

- If a user has no listed items, this table should still be visible, albeit empty.

Searching for Items

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%'. Cross-reference with **Trade** to ensure that these item numbers are not associated with trades that have already been accepted.

```
SELECT item_no
FROM Item
WHERE (description like '%$KeyWordSearchTerm%' OR title
LIKE '%$KeyWordSearchTerm%')
AND (item_no not in (select distinct proposer_item_no from
Trade where status= 'accepted')
AND item_no not in (select distinct counterparty_item_no from
Trade where status= 'accepted'))
```

            - Store the returned results as *items_returned_by_search*
        - 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** to ensure that these items numbers are not associated with trades that have already been accepted

```
SELECT item_no
FROM Item
WHERE lister_email in (SELECT email from User where
postal_code= `user.postalcode`)
AND (item_no not in (select distinct proposer_item_no from
Trade where status= 'accepted')
AND item_no not in (select distinct counterparty_item_no from
Trade where status= 'accepted'))
```

            - Store the returned results as *items_returned_by_search*

- 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**. Restrict to a list of postal codes that are within $MilesWithin

```
SELECT postal_code
FROM
(
SELECT loc2.postal_code, (((acos(sin((loc2.lat*pi()/180)) *
sin((loc1.lat*pi()/180)) + cos((loc2.lat*pi()/180)) *
cos((loc1.lat*pi()/180)) * cos(((loc2.lng- loc1.lng) * pi()/180)))) *
180/pi()) * 60 * 1.1515 * 1.609344) as distance
FROM
(select latitude as lat, longitude as lng, postal_code from
Location_lookup where postal_code in (select postal_code from
users where email=`user.email`)) loc1
cross join
(SELECT latitude as lat, longitude as lng, postal_code from
Location_lookup) loc2
) dist
--the above formula calculates in KM, convert to miles
WHERE distance*0.621371 <= $MilesWithin
```

    - Store the results above as an array map called *postal_codes_within_search*.
    - 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. Ensure that any items associated with completed trades are not included.

```
SELECT item_no from Item
where lister_email in
(SELECT email from User where postal_code in
(*postal_codes_within_search*)) and
 (item_no not in (select distinct proposer_item_no from Trade
where status= 'accepted')
AND item_no not in (select distinct counterparty_item_no from
Trade where status= 'accepted'))
```

    - Store the returned results as *items_returned_by_search*

- 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”. The below query should return a value >1

```
SELECT count(*)
FROM Location_lookup
WHERE postal_code= ‘$PostalCodeSearch’
```

    - 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. Ensure that any items associated with completed trades are not included.

```
SELECT item_no from Item
WHERE lister_email in
(SELECT email from User where postal_code=
‘$PostalCodeSearch’)
AND
 (item_no not in (select distinct proposer_item_no from
Trade where status= ‘accepted’)
AND item_no not in (select distinct counterparty_item_no
from Trade where status= ‘accepted’))
```

    - Store the returned results as *items_returned_by_search*
- 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

```
SELECT title
, CASE WHEN cg.item_no IS NOT NULL then ‘Collectable Card
Game’ WHEN bg.item_no IS NOT NULL then ‘Board Game’
WHEN pcg.item_no IS NOT NULL then ‘Playing Card Game’
WHEN comg.item_no is NOT NULL then ‘Computer Game’
```

```
WHEN vg.item_no is NOT NULL then 'Video Game' END as
game_type
, condition, listing_url, description
FROM Item i
LEFT JOIN Item_Collectable_Card_Game cg on
i.item_no=cg.item_no
LEFT JOIN Item_Board_Game bg on
i.item_no=bg.item_no
LEFT JOIN Item_Playing_Card_Game pcg on
i.item_no=pcg.item_no
LEFT JOIN Item_Computer_Game comg on
i.item_no=comg.item_no
LEFT JOIN Item_Video_Game vg on i.item_no=vg.item_no
WHERE i.item_no in (*items_returned_by_search*)
```

- 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 . Store the result of this as *lister_avg_response_time*

```
SELECT item_no,
COALESCE(lister_avg_response_time,0.0) as
lister_avg_response_time
FROM
(SELECT item_no, lister_email from Item where
item_no in (*items_returned_by_search*))
rev_items
JOIN
(SELECT item_sellers.lister_email
,avg(julianday(accept_reject_date)-julianday(prop
osed_date)) as lister_avg_response_time
FROM
(SELECT lister_email, item_no from Item where
item_no in (*items_returned_by_search*))
item_sellers
JOIN
Trade t
on item_sellers.lister_email=t.counterparty_email
WHERE (t.status= 'accepted' or
t.status='rejected')
GROUP BY lister_email) response_times
```

```
on
rev_items.lister_email=response_times.lister_em
ail
```

- 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.
    - Count total instances where status = 'Accepted'

```
SELECT item_no,
COALESCE(completed_trades,0) as
lister_completed_trade_count
 FROM
(SELECT item_no, lister_email from Item
where item_no in
(*items_returned_by_search*)) rev_items
JOIN
(SELECT item_sellers.lister_email
, count(*) as completed_trades
FROM
(SELECT lister_email, item_no from Item
where item_no in
(*items_returned_by_search*))
item_sellers
JOIN
Trade t1
on
item_sellers.lister_email=t1.counterparty_
email
JOIN
Trade t2
on
item_sellers.lister_email=t2.proposer_em
ail
WHERE (t1.status= 'accepted' OR
t2.status= 'accepted')
GROUP BY lister_email)
completed_trades
on
rev_items.lister_email=completed_trades.
lister_email
```

```

```

- Store these values as *lister_completed_trade_count*

- 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.

```
SELECT rank_label from Rank_lookup
WHERE
trade_lower_range<=*lister_completed_trade_count* and
trade_upper_range>=*lister_completed_trade_count*
```

- Find the lister distance from searchee
  - If user searched by “In my postal code”, this value should be 0.0
  - Else if user searched by keyword, by postal code, or ‘Within X miles of me’, perform Haversine formula between user and the lister’s location. Unfortunately we cannot just take this from the earlier calculation (since it would not have been performed for keyword search or postal code search).

```
SELECT item_no, distance*0.621371 as
lister_distance
FROM
(
SELECT loc2.item_no,
(((acos(sin((loc2.lat*pi()/180)) *
sin((loc1.lat*pi()/180)) + cos((loc2.lat*pi()/180)) *
cos((loc1.lat*pi()/180)) * cos(((loc2.lng- loc1.lng) *
pi()/180)))) * 180/pi()) * 60 * 1.1515 * 1.609344)
as distance
FROM
(select latitude as lat, longitude as lng,
postal_code from Location_lookup where
postal_code in (select postal_code from users
```

```
where email=`user.email`)) loc1
cross join
(SELECT item_no, latitude as lat, longitude as
lng, Location_lookup.postal_code
        from Item join User
        on item.lister_email=user.email
        join Location_lookup
        on
user.postal_code=Location_lookup.postal_code
        where item.item_no in
(*items_returned_by_search*)) loc2
) dist
```

- 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

```
SELECT text_color FROM Response_color_lookup WHERE
response_lower_range<=round(*lister_avg_response_time*,1)
AND
response_upper_range=>round(*lister_avg_response_time*,1)
```

  - 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

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)

```
SELECT i.item_no, title
, CASE WHEN cg.item_no IS NOT NULL then ‘Collectable Card Game’
WHEN bg.item_no IS NOT NULL then ‘Board Game’ WHEN
pcg.item_no IS NOT NULL then ‘Playing Card Game’ WHEN
comg.item_no is NOT NULL then ‘Computer Game’ WHEN vg.item_no
is NOT NULL then ‘Video Game’ END as game_type
, media
, COALESCE(vg.platform,comg.platform) as platform
, no_cards, condition, description
FROM Item i
LEFT JOIN Item_Collectable_Card_Game cg on i.item_no=cg.item_no
LEFT JOIN Item_Board_Game bg on
i.item_no=bg.item_no
LEFT JOIN Item_Playing_Card_Game pcg on i.item_no=pcg.item_no
LEFT JOIN Item_Computer_Game comg on i.item_no=comg.item_no
LEFT JOIN Item_Video_Game vg on i.item_no=vg.item_no
WHERE i.item_no=*item_no*
```

    - 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)

```
SELECT i.item_no, title
```

```
, CASE WHEN cg.item_no IS NOT NULL then 'Collectable Card
Game' WHEN bg.item_no IS NOT NULL then 'Board Game'
WHEN pcg.item_no IS NOT NULL then 'Playing Card Game'
WHEN comg.item_no is NOT NULL then 'Computer Game'
WHEN vg.item_no is NOT NULL then 'Video Game' END as
game_type
, media
,  COALESCE(vg.platform,comg.platform) as platform
, no_cards, condition, description
FROM Item i
LEFT JOIN Item_Collectable_Card_Game cg on
i.item_no=cg.item_no
LEFT JOIN Item_Board_Game bg on
i.item_no=bg.item_no
LEFT JOIN Item_Playing_Card_Game pcg on
i.item_no=pcg.item_no
LEFT JOIN Item_Computer_Game comg on
i.item_no=comg.item_no
LEFT JOIN Item_Video_Game vg on i.item_no=vg.item_no
WHERE i.item_no=*item_no*
```

        - 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)

```
SELECT i.item_no, title
, CASE WHEN cg.item_no IS NOT NULL then 'Collectable Card
Game' WHEN bg.item_no IS NOT NULL then 'Board Game'
WHEN pcg.item_no IS NOT NULL then 'Playing Card Game'
WHEN comg.item_no is NOT NULL then 'Computer Game'
WHEN vg.item_no is NOT NULL then 'Video Game' END as
game_type
, media
,  COALESCE(vg.platform,comg.platform) as platform
, no_cards
, condition, description
FROM Item i
LEFT JOIN Item_Collectable_Card_Game cg on
i.item_no=cg.item_no
LEFT JOIN Item_Board_Game bg on
i.item_no=bg.item_no
```

```
LEFT JOIN Item_Playing_Card_Game pcg on
i.item_no=pcg.item_no
LEFT JOIN Item_Computer_Game comg on
i.item_no=comg.item_no
LEFT JOIN Item_Video_Game vg on i.item_no=vg.item_no
WHERE i.item_no=*item_no*
```

- 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.

```
SELECT u.first_name, u.last_name, loc.city, loc.state,
loc.postal_code
FROM
User u join Location_lookup loc
ON u.postal_code=loc.postal_code
WHERE u.email in (select lister_email from Item where
item_no=*item_no*)
```

    - 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

```
SELECT
coalesce(avg(julianday(accept_reject_dat
e)-julianday(proposed_date)),0.0) as
avg_response_time
FROM Trade
WHERE (Trade.status = 'accepted' OR
Trade.status = 'rejected')
AND counterparty_email in (select
lister_email from Item where
item_no=*item_no*)
```

            - Store this as *avg_response_time*
        - Map the number to a color by using **Response_color_lookup**

```
SELECT text_color FROM
Response_color_lookup WHERE
```

```
response_lower_range<=round(
*avg_response_time*,1) AND
response_upper_range=>round(
*avg_response_time*,1)
```

- To show rank:
  - Count total numbers of completed trades of the User either as a proposer or as a counterparty.
  - 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)

```
SELECT count(*) as
completed_trade_count
FROM Trade
WHERE (Trade.status =
'accepted' OR Trade.status =
'rejected')
AND (counterparty_email in
(select lister_email from Item
where item_no=*item_no*)
        OR proposer_email in
(select lister_email from Item
where item_no=*item_no*))
```

    - Store this as *completed_trade_count*
  - Map this count to a rank in the **Rank_lookup** table.

```
SELECT rank_label from Rank_lookup
WHERE
trade_lower_range<=*completed_trade_c
ount* and trade_upper_range>=
*completed_trade_count*
```

- 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. Store this as *distance_between_users*

```
SELECT item_no, distance*0.621371 as
distance_between_users
FROM
(
SELECT loc2.item_no,
(((acos(sin((loc2.lat*pi()/180)) *
sin((loc1.lat*pi()/180)) +
cos((loc2.lat*pi()/180)) *
cos((loc1.lat*pi()/180)) * cos(((loc2.lng-
loc1.lng) * pi()/180)))) * 180/pi()) * 60 *
1.1515 * 1.609344) as distance
FROM
(select latitude as lat, longitude as lng,
postal_code from Location_lookup where
postal_code in (select postal_code from
users where email=`user.email`)) loc1
cross join
(SELECT item_no, latitude as lat,
longitude as lng,
Location_lookup.postal_code
        from Item join User
        on item.lister_email=user.email
        join Location_lookup
        on
user.postal_code=Location_lookup.postal
_code
        where item.item_no=*item_no*)
loc2
) dist
```

- Use **Distance_color_lookup** to map the distance to the appropriate background color

```
SELECT background_color FROM
Distance_color_lookup
WHERE
distance_lower_range<=round(*distance_
between_users*,1) AND
distance_upper_range>=round(*distance_
```

```
between_users*,1)
```

Proposing a Trade

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.

```
SELECT postal_code, round(distance*0.621371,2) as distance_miles
FROM
(
SELECT loc2.postal_code, (((acos(sin((loc2.lat*pi()/180)) *
sin((loc1.lat*pi()/180)) + cos((loc2.lat*pi()/180)) * cos((loc1.lat*pi()/180))
* cos(((loc2.lng- loc1.lng) * pi()/180)))) * 180/pi()) * 60 * 1.1515 *
1.609344) as distance
FROM
(select latitude as lat, longitude as lng, postal_code from
Location_lookup where postal_code in (select postal_code from users
where email=`user.email`)) loc1
cross join
(SELECT latitude as lat, longitude as lng, postal_code
        from Location_lookup l
        join User u
        on l.postal_code=u.postal_code
        join Item i
        on u.email=i.lister_email
        where i.item_no=*item_no*) loc2
) dist
```

        - 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.

```
SELECT i.item_no
, CASE WHEN cg.item_no IS NOT NULL then ‘Collectable Card Game’
WHEN bg.item_no IS NOT NULL then ‘Board Game’ WHEN
pcg.item_no IS NOT NULL then ‘Playing Card Game’ WHEN
comg.item_no is NOT NULL then ‘Computer Game’ WHEN vg.item_no
is NOT NULL then ‘Video Game’ END as game_type
, title, condition
FROM Item i
LEFT JOIN Item_Collectable_Card_Game cg on i.item_no=cg.item_no
LEFT JOIN Item_Board_Game bg on
i.item_no=bg.item_no
LEFT JOIN Item_Playing_Card_Game pcg on i.item_no=pcg.item_no
LEFT JOIN Item_Computer_Game comg on i.item_no=comg.item_no
LEFT JOIN Item_Video_Game vg on i.item_no=vg.item_no
WHERE lister_email=`user.email`
ORDER BY i.item_no
```

    Display item selector
        - Display a radio button to select items from item list
        - Proposed_item_no is defined by the item_no of the selected item and stored as *proposed_item_no*
- Display “Confirm” link and store trade information
    - 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

```
SELECT lister_email as counterparty_email
, item_no as counterparty_item_no
FROM Item
WHERE item_no=*item_no*
```

        - Store these as *counterparty_email* and *counterparty_item_no*
        - #autogenerated_trade_id represents an autoincremented trade_id:

```
SELECT coalesce(max(auto_trade_id)+1,1) from Trade
```

- Store the information

```
INSERT INTO Trade (proposer_email, counterparty_email,
proposer_item_no, counterparty_item_no, proposed_date,
status,auto_trade_id) values (`user.email`, *counterparty_email*,
*proposed_item_no*, *counterparty_item_no*, `current_date`,
“unaccepted”,#autogenerated_trade_id)
```

- Display confirmation message
  - User will have the option to return to the main menu

Accept and Reject Trades

Abstract Code

- Find non-calculated information to display about the proposed trades
  - 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 the Proposed Item
    - Display the proposed item associated with a trade
  - The above bullets would be accomplished with this SQL code:

```
SELECT t.auto_trade_id, t.proposed_date as date_sk, i1.name as desired_item,
i1.listing_url as desired_item_url, u.nickname as proposer, i2.name as proposed_item,
i2.list_url as proposed_item_url, u.postal_code as proposer_postal_code
FROM
Trade t
```

```
join Item i1
on t.counterparty_item_no=i1.item_no
join Item i2
on t.proposed_item_no=i2.item_no
join User u
on t.proposer_email=u.email
where t.counterparty_email=`user.email`
```

    - Store proposer postal_code as *proposer_postal_code*

- Calculate and 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.

```
SELECT t.auto_trade_id,
count(t2.auto_trade_id) + count(t3.auto_trade_id) as completed_trades
FROM
Trade t1
join Trade t2
on t1.proposer_email=t2.proposer_email
join Trade t3
on t1.proposer_email=t3.counterparty_email
where t1.counterparty_email=`UserEmail`
and (t2.status='accepted' or t3.status='accepted')
```

        - Store the *proposer_completed_trade_count* associated with Trade.auto_trade_id
    - 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.

```
SELECT rank_label from Rank_lookup WHERE
trade_lower_range<=*proposer_completed_trade_count* and
trade_upper_range>=*proposer_completed_trade_count*
```

- 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*

        - This was stored above with each Trade.auto_trade_id as *proposer_postal_code*
    - 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.

```
SELECT postal_code, round(distance*0.621371,1) as distance_miles
FROM
(
SELECT loc2.postal_code, (((acos(sin((loc2.lat*pi()/180)) *
sin((loc1.lat*pi()/180)) + cos((loc2.lat*pi()/180)) * cos((loc1.lat*pi()/180))
* cos(((loc2.lng- loc1.lng) * pi()/180)))) * 180/pi()) * 60 * 1.1515 *
1.609344) as distance
FROM
(select latitude as lat, longitude as lng, postal_code from
Location_lookup where postal_code in (select postal_code from users
where email=`user.emaill`)) loc1
cross join
(SELECT latitude as lat, longitude as lng, postal_code
        from Location_lookup
        where postal_code=*proposer_postal_code*) loc2
) dist
```

- 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

```
SELECT u.first_name, u.email
FROM Trade t join User u
on t.proposer_email=u.email
 WHERE auto_trade_id=*trade_id*
```

        - In the **Trade** table update the status of the trade to 'Accepted'

```
UPDATE Trade SET status= ‘accepted’ where
auto_trade_id=*trade_id*
```

- After this, re-query **Trade** for trades where proposer’s email address is either counterparty_email and status= ‘unaccepted’.
  - 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’

```
UPDATE Trade SET status= ‘rejected’ where
auto_trade_id=*trade_id*
```

  - After this, re-query **Trade** for trades where proposer’s email address is counterparty_email and status= ‘unaccepted’.
    - 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

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

```
SELECT CASE WHEN `user.email`=proposer_email then ‘Proposer’
WHEN `user.email`=counterparty_email then ‘Counterparty’ END as
role
, count(*) as Total
, sum(CASE WHEN status=’accepted’ then 1 end) as Accepted,
```

```
sum(CASE WHEN status= 'rejected' then 1 end) as Rejected
,sum(CASE WHEN status= 'rejected' then 1 end)/count(*) as
Rejected_Perc
FROM Trade
WHERE status in ('accepted, 'rejected') AND
(counterparty_email=`user.email` OR proposer_email=`user.email`)
GROUP BY CASE WHEN `user.email`=proposer_email then 'Proposer'
WHEN `user.email`=counterparty_email then 'Counterparty' END
```

- 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

```
SELECT t.proposal_date
, t.accepted_reject_date
, t.status
, julianday(t.accept_reject_date)-julianday(t.proposal_date) as
response_time
, CASE WHEN `user.email`=t.proposer_email then 'Proposer' WHEN
`user.email`=t.counterparty_email then 'Counterparty' END as role
, i1.title as proposed_item
, i2.title as desired_item
, CASE WHEN `user.email`=t.proposer_email then u2.nickname WHEN
`user.email`=t.counterparty_email then u1.nickname END as
other_user
, t.trade_history_link as detail_link
FROM Trade t
join Item i1 on t.proposer_item_id=i1.item_no
 join Item i2 on t.counterparty_item_id=i2.item_no
```

```
 join User u1 on t.proposer_email=u1.email
 join User u2 on t.counterparty_email=u2.email
WHERE status in ('accepted, 'rejected') AND
(counterparty_email=`user.email` OR proposer_email=`user.email`)
```

Trade Details

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)

```
SELECT proposed_date
, accept_reject_date
, status
, CASE WHEN `user.email`=t.proposer_email then 'Proposer'
WHEN `user.email`=t.counterparty_email then 'Counterparty'
END as role
,julianday(t.accept_reject_date)-julianday(t.proposal_date) as
response_time
 FROM trade
WHERE auto_trade_id=*trade_id*
```

    - 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 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.

```
SELECT round(distance*0.621371,2) as distance_miles
FROM
(
SELECT (((acos(sin((loc2.lat*pi()/180)) *
sin((loc1.lat*pi()/180)) + cos((loc2.lat*pi()/180)) *
cos((loc1.lat*pi()/180)) * cos(((loc2.lng- loc1.lng) *
pi()/180)))) * 180/pi()) * 60 * 1.1515 * 1.609344) as
distance
FROM
(select latitude as lat, longitude as lng, postal_code from
Location_lookup where postal_code in (select
postal_code from users where email=`user.email`)) loc1
cross join
(SELECT CASE WHEN `user.email`=t.proposer_email
then l2.latitude WHEN `user.email`=t.counterparty_email
then l1.latitude as lat
        , CASE WHEN `user.email`=t.proposer_email
then l2.longitude WHEN
`user.email`=t.counterparty_email then l1.longitude as
lng
        from
        Trade t
        join
        User u1
        on t.proposer_email_id=u1.email
        join
        Location_lookup l1
        on u1.postal_code=l1.postal_code
        join
        User u2
        on t.counterparty_email_id=u2.email
        join
        Location_lookup l2
        on u2.postal_code=l2.postal_code
```

```
        where t.auto_trade_id=*trade_id*) loc2
) dist
```

- Other user's nickname. (If trade is accepted), find and display
  - Other user's name
  - Other user's email address

```
CASE WHEN `user.email`=t.proposer_email THEN
u2.nickname WHEN `user.email`=t.counterparty_email
then u1.nickname END as nickname
, CASE WHEN status='accepted' AND
`user.email`=t.proposer_email THEN u2.first_name
WHEN status= 'accepted' and
`user.email`=t.counterparty_email then u1.first_name
END as name
, CASE WHEN status='accepted' AND
`user.email`=t.proposer_email THEN
t.counterparty_email WHEN status= 'accepted' AND
`user.email`=t.counterparty_email THEN
t.proposer_email END as email
FROM Trade t join User u1 on
t.proposer_email=u1.email
 join User u2 on t.counterparty_email=u2.email
WHERE t.auto_trade_id=*trade_id*
```

- 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)

```
SELECT i.item_no, i.name
, CASE WHEN cg.item_no IS NOT NULL then 'Collectable Card Game'
WHEN bg.item_no IS NOT NULL then 'Board Game' WHEN
pcg.item_no IS NOT NULL then 'Playing Card Game' WHEN
comg.item_no is NOT NULL then 'Computer Game' WHEN vg.item_no
is NOT NULL then 'Video Game' END as game_type
, i.condition, i.description FROM Trade t
join Item i on t.proposed_item_id=i.item_id
LEFT JOIN Item_Collectable_Card_Game cg on i.item_no=cg.item_no
LEFT JOIN Item_Board_Game bg on
```

```
i.item_no=bg.item_no
LEFT JOIN Item_Playing_Card_Game pcg on i.item_no=pcg.item_no
LEFT JOIN Item_Computer_Game comg on i.item_no=comg.item_no
LEFT JOIN Item_Video_Game vg on i.item_no=vg.item_no
```

- 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

```
SELECT i.item_no, i.name
, CASE WHEN cg.item_no IS NOT NULL then 'Collectable Card Game'
WHEN bg.item_no IS NOT NULL then 'Board Game' WHEN
pcg.item_no IS NOT NULL then 'Playing Card Game' WHEN
comg.item_no is NOT NULL then 'Computer Game' WHEN vg.item_no
is NOT NULL then 'Video Game' END as game_type
, i.condition, i.description FROM Trade t
join Item i on t.counterparty_item_id=i.item_id
LEFT JOIN Item_Collectable_Card_Game cg on i.item_no=cg.item_no
LEFT JOIN Item_Board_Game bg on
i.item_no=bg.item_no
LEFT JOIN Item_Playing_Card_Game pcg on i.item_no=pcg.item_no
LEFT JOIN Item_Computer_Game comg on i.item_no=comg.item_no
LEFT JOIN Item_Video_Game vg on i.item_no=vg.item_no
```