Campus Trading Application - Database Design Document

Overview

This database design supports a campus trading marketplace where verified students can list items for sale, negotiate prices, post wish requests, complete transactions, and build reputation through ratings. The design ensures data integrity, supports complex workflows like offer management and transaction state tracking, and maintains referential relationships between all entities.

Complete List of Tables

1. **Member** - Core student user accounts
2. **Category** - Product categorization hierarchy
3. **Listing** - Items posted for sale
4. **ListingImage** - Multiple photos per listing
5. **Offer** - Price proposals from buyers to sellers
6. **Transaction** - Records of completed exchanges
7. **Rating** - Reviews and ratings after transactions
8. **WishRequest** - Buyer requests for items they're seeking
9. **Watchlist** - Items bookmarked by interested buyers
10. **Report** - Flagged content and user complaints
11. **Notification** - System alerts to users
12. **Administrator** - Platform moderators and admin users
13. **MessageThread** - Negotiation threads between buyers and sellers per listing
14. **Message** - Individual messages within a negotiation thread

---

Detailed Table Descriptions

1. Table: Member

Purpose:
Stores all verified student user accounts. This is the central entity representing students who can act as both buyers and sellers. The Member table is required by the project constraints and serves as the foundation for user identity and verification.

Attributes:

1. **MemberID** - Unique identifier for each student
2. **Name** - Full name of the student
3. **Email** - University email address (used for verification)
4. **PasswordHash** - Hashed password for authentication (never store plaintext)
5. **ContactNumber** - Mobile phone number
6. **Department** - Academic department (e.g., "Mechanical Engineering") [optional]
7. **YearOfStudy** - Current academic year (1, 2, 3, 4, or graduate) [optional]

8. **Hostel** - Residence hall name
9. **RoomNumber** - Specific room within hostel
10. **Image** - Profile photo URL or path
11. **Bio** - Optional self-introduction text
12. **IsVerified** - Whether email verification is complete [optional]
13. **VerificationDate** - When account was verified [optional]
14. **AccountCreationDate** - When the student registered
15. **AccountStatus** - Active, Suspended, or Deleted

Primary Key: `MemberID`

Important Constraints:

- Email must be unique (no duplicate accounts)
- Email must follow university domain format
- PasswordHash must be NOT NULL
- ContactNumber must be NOT NULL
- Name must be NOT NULL
- Email must be NOT NULL
- New members with fewer than 1 completed transaction can list maximum 2 items [application level constraint]

---

2. Table: Category

Purpose:
Organizes items into hierarchical categories to enable efficient browsing and filtering. Supports nested subcategories (e.g., Electronics → Computing → Laptops).

Attributes:

1. **CategoryID** - Unique identifier for each category
2. **CategoryName** - Display name (e.g., "Books & Textbooks", "Electronics")
3. **ParentCategoryID** - Reference to parent category (null for top-level categories)
4. **Description** - Brief explanation of what belongs in this category
5. **IsActive** - Whether category is currently available for listing [optional]

Primary Key: `CategoryID`

Important Constraints:

- CategoryName must be NOT NULL
- CategoryName must be unique within the same parent level
- ParentCategoryID references CategoryID (self-referencing foreign key)
- IsActive must be NOT NULL
- Top-level categories have ParentCategoryID = null

---

3. Table: Listing

Purpose:
Represents individual items posted for sale by students. This is the core marketplace entity that contains all information about products available for trade.

Attributes:

1. **ListingID** - Unique identifier for each listing
2. **SellerID** - Reference to the Member who created the listing
3. **CategoryID** - Product category classification
4. **Title** - Item name/headline
5. **Description** - Detailed condition and features explanation
6. **AskingPrice** - Initial price set by seller
7. **IsNegotiable** - Whether seller accepts price offers
8. **Condition** - Item state (New, Like New, Good, Fair, Poor)
9. **CourseCode** - For textbooks, the relevant course (e.g., "ME-201") [optional]
10. **Status** - Current state (Listed, Pending, Reserved, Completed, Sold, Expired, Withdrawn)
11. **CreatedDate** - When listing was published
12. **LastModifiedDate** - Most recent update timestamp
13. **ExpiryDate** - Automatic removal date if unsold
14. **IsDonation** - Whether item is free (donation)
15. **PreferredMeetingLocation** - Seller's preferred meetup spot (e.g., "Engineering lobby", "Library entrance")
16. **WishRequestID** - Reference to the wish request this listing fulfills (nullable, only populated when responding to a wish)

Primary Key: `ListingID`

Important Constraints:

- SellerID must be NOT NULL and reference Member(MemberID)
- CategoryID must be NOT NULL and reference Category(CategoryID)
- Title must be NOT NULL
- AskingPrice must be NOT NULL (can be 0 for donations)
- Status must be NOT NULL
- IsNegotiable must be NOT NULL
- Valid status transitions must be enforced (e.g., cannot go from Sold back to Listed)
- ExpiryDate automatically set to 30 days after CreatedDate
- If IsDonation = true, then AskingPrice must be 0
- If WishRequestID is populated, it must reference WishRequest(WishRequestID)

---

4. Table: ListingImage

Purpose:
Stores multiple photos for each listing since sellers can upload several images showing different angles, condition details, and proof of authenticity.

Attributes:

1. **ImageID** - Unique identifier for each image

2. **ListingID** - Reference to the parent listing
3. **ImageURL** - File path or storage location
4. **ImageOrder** - Display sequence (1st image is primary)
5. **UploadedDate** - When image was added

Primary Key: `ImageID`

Important Constraints:

- ListingID must be NOT NULL and reference Listing(ListingID)
- ImageURL must be NOT NULL
- ImageOrder must be NOT NULL
- When a listing is deleted, all associated images should be removed (cascade delete)
- First image (ImageOrder = 1) serves as thumbnail in search results

---

5. Table: Offer

Purpose:
Records price proposals submitted by buyers on negotiable listings. Serves as a log of all offers received for analytics and tracking. The actual negotiation (counter-offers, discussion) happens in the MessageThread. This table captures the initial bid and final outcome.

Attributes:

1. **OfferID** - Unique identifier for each offer
2. **ListingID** - Item being offered on
3. **BuyerID** - Member making the offer
4. **OfferedPrice** - Buyer's proposed price
5. **AgreedPrice** - Final negotiated price if offer is successful (nullable, populated when accepted)
6. **OfferMessage** - Optional note from buyer (e.g., "Can pick up today")
7. **OfferStatus** - Current state (Submitted, Accepted, Declined, Withdrawn, Expired)
8. **SubmittedDate** - When offer was created
9. **ResponseDate** - When seller acted on the offer
10. **ExpiryDate** - Offer automatically expires after 48 hours if no response

Primary Key: `OfferID`

Important Constraints:

- ListingID must be NOT NULL and reference Listing(ListingID)
- BuyerID must be NOT NULL and reference Member(MemberID)
- OfferedPrice must be NOT NULL
- OfferStatus must be NOT NULL
- BuyerID cannot equal the listing's SellerID (cannot offer on own items)
- ExpiryDate automatically set to SubmittedDate + 48 hours
- Only one offer per listing can have status = Accepted at a time
- When an offer is accepted, all other offers on that listing auto-decline
- If OfferStatus = Accepted, AgreedPrice must be populated

---

6. Table: Transaction

Purpose:
Records the complete history of successful trades between buyers and sellers, including meeting details, agreed prices, and confirmation status. This table links accepted offers to their final outcomes.

Attributes:

1. **TransactionID** - Unique identifier for each transaction
2. **ListingID** - Item that was traded
3. **SellerID** - Member who sold the item
4. **BuyerID** - Member who purchased the item
5. **OfferID** - The accepted offer that led to this transaction
6. **AgreedPrice** - Final negotiated price
7. **TransactionDate** - When exchange was completed
8. **SellerConfirmed** - Whether seller confirmed completion
9. **BuyerConfirmed** - Whether buyer confirmed completion
10. **Status** - Transaction state (Scheduled, Completed, Cancelled)
11. **CreatedDate** - When transaction was initiated

Primary Key: `TransactionID`

Important Constraints:

- ListingID must be NOT NULL and reference Listing(ListingID)
- SellerID must be NOT NULL and reference Member(MemberID)
- BuyerID must be NOT NULL and reference Member(MemberID)
- OfferID must reference Offer(OfferID)
- AgreedPrice must be NOT NULL
- SellerConfirmed must be NOT NULL (defaults to false)
- BuyerConfirmed must be NOT NULL (defaults to false)
- SellerID must match the listing's seller
- BuyerID cannot equal SellerID
- Transaction only moves to Completed when both SellerConfirmed and BuyerConfirmed are true
- If status = Completed, both confirmation flags must be true

---

7. Table: Rating

Purpose:
Store reviews and star ratings that buyers and sellers give each other after transactions complete. Builds reputation and accountability in the marketplace.

Attributes:

1. **RatingID** - Unique identifier for each rating
2. **TransactionID** - Which trade this rating is for
3. **RaterID** - Member giving the rating
4. **RatedID** - Member being rated
5. **Stars** - Numeric rating (1 to 5)

6. **ReviewText** - Optional written feedback
7. **RatingDate** - When review was submitted

Primary Key: `RatingID`

Important Constraints:

- TransactionID must be NOT NULL and reference Transaction(TransactionID)
- RaterID must be NOT NULL and reference Member(MemberID)
- RatedID must be NOT NULL and reference Member(MemberID)
- Stars must be NOT NULL and between 1 and 5
- RatingDate must be NOT NULL
- Each transaction can have maximum 2 ratings (one from buyer, one from seller)
- RaterID and RatedID must be different
- RaterID must be either the buyer or seller from the referenced transaction
- Ratings can only be created after transaction status = Completed

---

8. Table: WishRequest

Purpose:
Allows students to post requests for items they're seeking but can't find in active listings. Creates a reverse marketplace where demand drives supply.

Attributes:

1. **WishRequestID** - Unique identifier for each wish request
2. **RequesterID** - Member looking for the item
3. **ItemDescription** - What they're searching for
4. **MinBudget** - Minimum price willing to pay [optional]
5. **MaxBudget** - Maximum price willing to pay
6. **PreferredCondition** - Desired item state [optional]
7. **NeededByDate** - Deadline for when item is needed [optional]
8. **AdditionalDetails** - Extra requirements or preferences
9. **Status** - Request state (Active, Fulfilled, Expired, Cancelled)
10. **CreatedDate** - When wish was posted
11. **FulfilledDate** - When need was satisfied

Primary Key: `WishRequestID`

Important Constraints:

- RequesterID must be NOT NULL and reference Member(MemberID)
- ItemDescription must be NOT NULL
- Status must be NOT NULL by default active
- CreatedDate must be NOT NULL
- MaxBudget must be greater than or equal to MinBudget
- NeededByDate must be in the future when created
- If Status = Fulfilled, FulfilledDate must be populated

---

9. Table: Watchlist

Purpose:
Tracks which listings individual students are monitoring. Enables personalized notifications about price changes, new offers, or status updates on items of interest.

Attributes:

1. **WatchlistID** - Unique identifier for each watchlist entry
2. **MemberID** - Student watching the item
3. **ListingID** - Item being watched
4. **AddedDate** - When item was bookmarked
5. **NotifyOnPriceChange** - Whether to alert if price drops
6. **NotifyOnStatusChange** - Whether to alert if status changes

Primary Key: `WatchlistID` OR combination of `MemberID` and `ListingID`

Important Constraints:

- MemberID must be NOT NULL and reference Member(MemberID)
- ListingID must be NOT NULL and reference Listing(ListingID)
- AddedDate must be NOT NULL
- Each member can only watch a specific listing once (unique combination of MemberID + ListingID)
- Member cannot watch their own listings
- If listing status becomes Sold or Withdrawn, watchlist entries auto-remove

---

10. Table: Report

Purpose:
Captures user-submitted complaints about suspicious listings, inappropriate behavior, scams, or policy violations. Enables administrative moderation and dispute resolution.

Attributes:

1. **ReportID** - Unique identifier for each report
2. **ReporterID** - Member who filed the complaint
3. **ReportedMemberID** - Member being reported (nullable if reporting a listing)
4. **ReportedListingID** - Listing being reported (nullable if reporting a user)
5. **ReportType** - Category (Misleading Description, Scam, No-Show, Inappropriate Content, etc.)
6. **Description** - Detailed explanation of the issue
7. **Status** - Report state (Submitted, UnderReview, Resolved, Dismissed)
8. **SubmittedDate** - When report was filed
9. **ResolvedDate** - When admin took action
10. **ResolvedByAdminID** - Which administrator handled it
11. **Resolution** - Action taken and outcome [optional]

Primary Key: `ReportID`

Important Constraints:

- ReporterID must be NOT NULL and reference Member(MemberID)
- Description must be NOT NULL
- Status must be NOT NULL
- SubmittedDate must be NOT NULL
- At least one of ReportedMemberID or ReportedListingID must be populated / not null
- If ReportedMemberID is populated, it must reference Member(MemberID)
- If ReportedListingID is populated, it must reference Listing(ListingID)
- ResolvedByAdminID references Administrator(AdminID)
- If Status = Resolved, ResolvedDate and Resolution must be populated

---

11. Table: Notification

Purpose:
Manages all system-generated alerts sent to users about offers, watchlist updates, meeting reminders, ratings, and other important events. Ensures users stay informed about activity relevant to them.

Attributes:

1. **NotificationID** - Unique identifier for each notification
2. **RecipientID** - Member receiving the notification
3. **NotificationType** - Category (OfferReceived, OfferAccepted, PriceDropped, MeetingReminder, etc.)
4. **Title** - Brief notification headline
5. **Message** - Full notification text
6. **RelatedListingID** - Associated listing (if applicable)
7. **RelatedOfferID** - Associated offer (if applicable)
8. **RelatedTransactionID** - Associated transaction (if applicable)
9. **IsRead** - Whether user has viewed it
10. **CreatedDate** - When notification was generated
11. **ReadDate** - When user opened it

Primary Key: `NotificationID`

Important Constraints:

- RecipientID must be NOT NULL and reference Member(MemberID)
- NotificationType must be NOT NULL
- Message must be NOT NULL
- IsRead must be NOT NULL (defaults to false)
- CreatedDate must be NOT NULL
- If RelatedListingID is populated, it must reference Listing(ListingID)
- If RelatedOfferID is populated, it must reference Offer(OfferID)
- If RelatedTransactionID is populated, it must reference Transaction(TransactionID)

---

12. Table: Administrator

Purpose:
Stores admin and moderator accounts separate from regular student members. Admins verify accounts,

resolve disputes, manage categories, and monitor platform health.

Attributes:

1. **AdminID** - Unique identifier for each administrator
2. **Name** - Administrator's full name
3. **Email** - Contact email
4. **PasswordHash** - Hashed password for authentication (never store plaintext)
5. **Role** - Admin level (SuperAdmin, Moderator, Support)
6. **CreatedDate** - When admin account was created
7. **LastLoginDate** - Most recent access timestamp
8. **IsActive** - Whether account is currently enabled

Primary Key: `AdminID`

Important Constraints:

- Email must be NOT NULL and unique
- PasswordHash must be NOT NULL
- Name must be NOT NULL
- Role must be NOT NULL
- IsActive must be NOT NULL (defaults to true)
- Only SuperAdmin role can create new administrator accounts

---

13. Table: MessageThread

Purpose:
Represents a negotiation conversation between a specific buyer and seller for a particular listing. A thread is created when a buyer submits an offer, providing a channel for counter-offers, questions, and settlement discussion. Each buyer-listing pair has exactly one thread.

Attributes:

1. **ThreadID** - Unique identifier for each message thread
2. **ListingID** - The listing being negotiated on
3. **BuyerID** - The buyer participating in this thread (seller is derived from the listing)
4. **OfferID** - The offer that initiated this thread
5. **CreatedDate** - When the thread was created (same time as the offer)
6. **IsActive** - Whether the thread is still open for messages (closed when offer is accepted/declined/expired or listing is sold)

Primary Key: `ThreadID`

Important Constraints:

- ListingID must be NOT NULL and reference Listing(ListingID)
- BuyerID must be NOT NULL and reference Member(MemberID)
- OfferID must be NOT NULL and reference Offer(OfferID)
- CreatedDate must be NOT NULL
- IsActive must be NOT NULL (defaults to true)

- Unique constraint on (ListingID, BuyerID) - only one thread per buyer per listing
- BuyerID cannot equal the listing's SellerID (cannot negotiate with yourself)
- When the associated offer is accepted, declined, or expired, IsActive is set to false
- When the listing status becomes Sold or Withdrawn, all associated threads are deactivated

---

14. Table: Message

Purpose:
Stores individual messages exchanged within a negotiation thread. Messages can be sent by either the buyer or the seller. This captures the full negotiation history including counter-offers, questions, and agreement discussion.

Attributes:

1. **MessageID** - Unique identifier for each message
2. **ThreadID** - The thread this message belongs to
3. **SenderID** - The member who sent the message (either buyer or seller)
4. **MessageText** - Content of the message
5. **SentDate** - When the message was sent

Primary Key: `MessageID`

Important Constraints:

- ThreadID must be NOT NULL and reference MessageThread(ThreadID)
- SenderID must be NOT NULL and reference Member(MemberID)
- MessageText must be NOT NULL
- SentDate must be NOT NULL
- SenderID must be either the BuyerID or the SellerID (from the listing) of the associated thread - no third-party messages
- Messages cannot be sent to inactive threads (IsActive = false)
- When a thread is deleted, all associated messages should be removed (cascade delete)

Relationships Between Tables

One-to-Many Relationships

1. **Member** → **Listing** (One seller can have many listings)
    - `Member.MemberID` → `Listing.SellerID`
    - Supports: A student can sell multiple items simultaneously
2. **Category** → **Listing** (One category contains many listings)
    - `Category.CategoryID` → `Listing.CategoryID`
    - Supports: Product organization and browsing by category
3. **Category** → **Category** (Parent category contains subcategories)

    - `Category.CategoryID` → `Category.ParentCategoryID`
    - Supports: Hierarchical category structure (Electronics → Laptops)

4. **Listing** → **ListingImage** (One listing has multiple photos)
    - `Listing.ListingID` → `ListingImage.ListingID`
    - Supports: Multiple-angle views and condition documentation

5. **Listing** → **Offer** (One listing receives many offers)
    - `Listing.ListingID` → `Offer.ListingID`
    - Supports: Competitive bidding from multiple interested buyers

6. **Member** → **Offer** (One buyer can make many offers)
    - `Member.MemberID` → `Offer.BuyerID`
    - Supports: Students can bid on multiple items

7. **Listing** → **Transaction** (One listing eventually creates one transaction) [One-To-One Relationships]
    - `Listing.ListingID` → `Transaction.ListingID`
    - Supports: Tracking which listing led to which sale

8. **Member** → **Transaction** (as Seller) (One seller has many transactions)
    - `Member.MemberID` → `Transaction.SellerID`
    - Supports: Transaction history for sellers

9. **Member** → **Transaction** (as Buyer) (One buyer has many transactions)
    - `Member.MemberID` → `Transaction.BuyerID`
    - Supports: Purchase history for buyers

10. **Transaction** → **Rating** (One transaction can have up to two ratings) [One-to-Many, max 2]
    - `Transaction.TransactionID` → `Rating.TransactionID`
    - Supports: Mutual rating after each trade (buyer rates seller and seller rates buyer)

11. **Member** → **WishRequest** (One student can post many wish requests) [Limit to 5 for one user]
    - `Member.MemberID` → `WishRequest.RequesterID`
    - Supports: Students looking for multiple different items

12. **Member** → **Report** (as Reporter) (One member can file many reports)
    - `Member.MemberID` → `Report.ReporterID`
    - Supports: Users flagging multiple issues

13. **Member** → **Report** (as Reported Member) (One member can be reported multiple times)
    - `Member.MemberID` → `Report.ReportedMemberID`
    - Supports: Tracking complaints against problematic users

14. **Listing** → **Report** (One listing can be reported multiple times)

- `Listing.ListingID` → `Report.ReportedListingID`
- Supports: Multiple users flagging suspicious listings

15. **Administrator** → **Report** (One admin resolves many reports)
    - `Administrator.AdminID` → `Report.ResolvedByAdminID`
    - Supports: Admin workload tracking
16. **Member** → **Notification** (One member receives many notifications)
    - `Member.MemberID` → `Notification.RecipientID`
    - Supports: Personalized alert delivery
17. **Listing** → **Notification** (One listing generates many notifications)
    - `Listing.ListingID` → `Notification.RelatedListingID`
    - Supports: Alerts about listing activity
18. **Offer** → **Notification** (One offer generates notifications)
    - `Offer.OfferID` → `Notification.RelatedOfferID`
    - Supports: Alerts about offer status changes
19. **Transaction** → **Notification** (One transaction generates notifications)
    - `Transaction.TransactionID` → `Notification.RelatedTransactionID`
    - Supports: Meeting reminders and completion prompts
20. **WishRequest** → **Listing** (One wish request can be fulfilled by one listing)
    - `WishRequest.WishRequestID` → `Listing.WishRequestID`
    - Supports: Linking listings created in response to wish requests ("I Have This" feature)
21. **Listing** → **MessageThread** (One listing can have many negotiation threads, one per interested buyer)
    - `Listing.ListingID` → `MessageThread.ListingID`
    - Supports: Multiple buyers negotiating with the seller on the same listing
22. **Member** → **MessageThread** (One buyer can have many negotiation threads across different listings)
    - `Member.MemberID` → `MessageThread.BuyerID`
    - Supports: A buyer negotiating on multiple items simultaneously
23. **Offer** → **MessageThread** (One offer initiates one thread) [One-to-One]
    - `Offer.OfferID` → `MessageThread.OfferID`
    - Supports: Linking each negotiation thread to the offer that started it
24. **MessageThread** → **Message** (One thread contains many messages)
    - `MessageThread.ThreadID` → `Message.ThreadID`
    - Supports: Full negotiation conversation history
25. **Member** → **Message** (One member sends many messages across threads)

- `Member.MemberID` → `Message.SenderID`
- Supports: Tracking who sent each message

Many-to-Many Relationships (Using Junction Tables)

1. **Member ↔ Listing (Watchlist)**
   - Junction Table: `Watchlist`
   - Relationship: Many students can watch many listings
   - Supports: Students bookmarking items they're interested in
   - Keys: `Member.MemberID` ↔ `Watchlist.MemberID` and `Listing.ListingID` ↔ `Watchlist.ListingID`

End of Database Design Document