Campus Trading Application: Database Design, UML Modelling, and ER Schema Development

CS 301: Database Management Systems — IIT Gandhinagar

Group 8 - Optimiser

Bhavik Patel
bhavik.patel@iitgn.ac.in
22110047
IIT Gandhinagar

Hitesh Kumar
hitesh.kumar@iitgn.ac.in
22110098
IIT Gandhinagar

Jinil Patel
jinilkumar.patel@iitgn.ac.in
22110184
IIT Gandhinagar

Pranav Patil
pranav.patil@iitgn.ac.in
22110199
IIT Gandhinagar

Sibtain
mdsibtain.raza@iitgn.ac.in
22110148
IIT Gandhinagar

Abstract

This report presents the complete conceptual and logical database design for the *Campus Trading Application* — a closed, trust-based digital marketplace for verified university students. We describe the full system through a set of UML and Entity-Relationship (ER) diagrams adhering to the Elmasri & Navathe notation standards. The design encompasses 14 relational tables supporting 8 core functionalities: User Management & Verification, Product Listing with Multi-Image Support, Offer-Based Negotiation System, Transaction Tracking & Confirmation, Mutual Rating & Reputation Building, Wish Request & Demand-Driven Supply, Real-Time Notification, and Administrative Moderation. We present seven UML Class Diagrams and six UML State Machine Diagrams, followed by a Chen-notation ER Diagram. Each diagram is accompanied by a detailed transition explanation from UML to ER, justification of all relationship cardinalities (1:1, 1:M, M:M), and a complete catalogue of database-level and application-level integrity constraints. All SQL artefacts, schema scripts, and synthetic data are available in the project repository [3].

Keywords

Database Design, UML, ER Diagram, Campus Marketplace, Relational Schema, MySQL

1 Introduction

University campuses generate a constant flow of second-hand goods: textbooks sold between semesters, electronics traded between batches, furniture exchanged as students move hostels. Yet current student trading relies on informal channels — WhatsApp groups, notice boards, and general-purpose platforms like OLX or Facebook Marketplace — that lack identity verification, trust mechanisms, or transaction tracking [4].

The **Campus Trading Application** addresses these shortcomings by providing a closed marketplace restricted to verified students holding a `@iitgn.ac.in` email address. The platform operates on three pillars:

1. **Marketplace**: Sellers list items with photos, pricing, condition grades, and meeting preferences; buyers browse, filter, and negotiate through a structured offer system.
2. **Wish Board**: A reverse marketplace where buyers post item requests with budget ranges, enabling demand-driven supply.
3. **Trust Framework**: Verified identities, transparent ratings, and immutable transaction history create accountability within the closed community.

2 System Overview

2.1 Core Functionalities

Table 1: Core Functionalities and Primary Database Tables

	Functionality	Primary Tables
1	User Management & Verification	Member, Administrator
2	Product Listing with Multi-Image Support	Listing, ListingImage, Category, Watchlist
3	Offer-Based Negotiation System	Offer, Listing, Member, Notification
4	Wish Request and Demand-Driven Supply	WishRequest, Listing, Member
5	Transaction Tracking and Confirmation	Transaction, Offer, Listing, Member
6	Mutual Rating	Rating, Transaction, Member
7	Real-Time Notification System	Notification, watchlist
8	Administrative Moderation	Report, Administrator

2.1.1 ***User Management and Verification.*** Manages student accounts using university email authentication, profile details (department, year, hostel), and account status tracking. Access is restricted to verified students through email domain validation with stored verification timestamps.

2.1.2 ***Product Listing with Multi-Image Support.*** Supports detailed item listings with pricing, condition ratings, and multiple images. Includes hierarchical categorization, course-specific tags, donation flags, and automatic 30-day listing expiry.

[0] GitHub Repository: https://github.com/Zeenu03/Campus_Trading_App/tree/main/Assignment_1

2.1.3 ***Offer-Based Negotiation System.*** Enables structured buyer–seller negotiations through offers on listings. Tracks offer states (*Submitted*, *Accepted*, *Declined*, *Withdrawn*, *Expired*), enforces 48-hour auto-expiry, and allows only one accepted offer per listing with messaging integration.

2.1.4 ***Transaction Tracking and Confirmation.*** Records completed trades with dual confirmation from both buyer and seller. Stores agreed prices, dates, meeting details, and references to originating offers for audit and status tracking.

2.1.5 ***Mutual Rating System.*** Allows both parties to submit 1–5 star ratings with optional feedback after a transaction. Limits ratings to two per transaction and aggregates scores to build user reputation.

2.1.6 ***Wish Request and Demand Matching.*** Enables users to post wish requests with budget ranges, preferred conditions, and deadlines. Sellers can create listings directly linked to specific requests for demand-driven matching.

2.1.7 ***Real-Time Notification System.*** Provides event-driven notifications for offers, watchlist updates, meetings, transactions, ratings, and wish request matches. Each notification includes category, read status, timestamps, and contextual links.

2.1.8 ***Administrative Moderation and Dispute Resolution.*** Implements a role-based admin system (*SuperAdmin*, *Moderator*, *Support*) for platform oversight. Supports user reports, tracks report lifecycle, logs admin actions, and manages dispute resolution workflows.

2.2 Core Database Entities

Table 2: Core Entities and Descriptions

Entity	Description
Member	Verified university students acting as buyers or sellers.
Listing	Items posted for sale with pricing and availability details.
Offer	Buyer-submitted price proposals for listings.
Transaction	Completed buyer–seller exchanges.
Category	Hierarchical structure for listing organization.
WishRequest	Buyer requests enabling demand-based matching.

3 UML Diagrams

This section presents the conceptual design of the Campus Trading Application using **Unified Modeling Language (UML)** diagrams [1]. UML provides a high-level, implementation-independent view of the system's structure and behaviour, serving as a foundation for deriving the Entity–Relationship (ER) model and, subsequently, the relational schema implemented in Module A.

The conceptual design is documented using the following two categories of UML diagrams:

- **UML Class Diagrams:** These diagrams capture the **static structure** of the system, including entities, attributes, operations, and structural relationships such as associations, composition, aggregation, and generalization.
- **UML State Machine Diagrams:** These diagrams model the **dynamic behaviour** of key entities by representing their lifecycle states and the transitions triggered by system events.

3.1 UML Notation and Conventions

Following Elmasri and Navathe (Chapter 10) [1], the UML class diagrams in this report adhere to the following notational conventions:

- **Classes** are represented as rectangles divided into three compartments: class name (top), attributes (middle), and operations (bottom).
- **Attributes** are shown with visibility markers (+ for public), data type, and attribute name.
- **Primary Keys** are the first attribute listed in each class (**suffixed with `ID`**).
- **Associations** are drawn as solid lines between classes, labelled with the role or association name.
- **Multiplicity** is denoted at each end of an association using standard UML notation: `1` (exactly one), `0..1` (zero or one), `*` (zero or more), `1..*` (one or more), `0..2` (zero to two).
- **Composition** (⧫) indicates a whole–part relationship where the part **cannot exist independently** of the whole and is cascade-deleted when the whole is removed.
- **Aggregation** (◊) indicates a whole–part relationship where the part can exist independently.
- **Generalization** (▷) indicates an inheritance (is-a) relationship between a superclass and its subclasses.
- **Association Classes** are shown with a dashed line connecting the class to the association it describes..
- **Stereotypes** (e.g., «`abstract`», «`association class`») are enclosed in guillemets above the class name.
- **Notes** attached to classes document key constraints and domain rules.

3.2 UML Class Diagrams

3.2.1 ***Complete System Class Diagram.*** The complete system class diagram [1] provides a holistic view of all fourteen entity classes and their interrelationships. The diagram identifies the *Member* class as the central entity, exhibiting direct associations with *Listing* (as seller), *Offer* (as buyer), *Transaction* (as both seller and buyer), *Rating* (as rater and rated), *WishRequest* (as requester), *Notification* (as recipient), *MessageThread* (as buyer participant), *Message* (as sender), and *Report* (as both reporter and reported target). This star-like topology reflects the member-centric architecture of the marketplace, where all system activities originate from or are directed toward student users.

Key structural patterns observable in the diagram include:

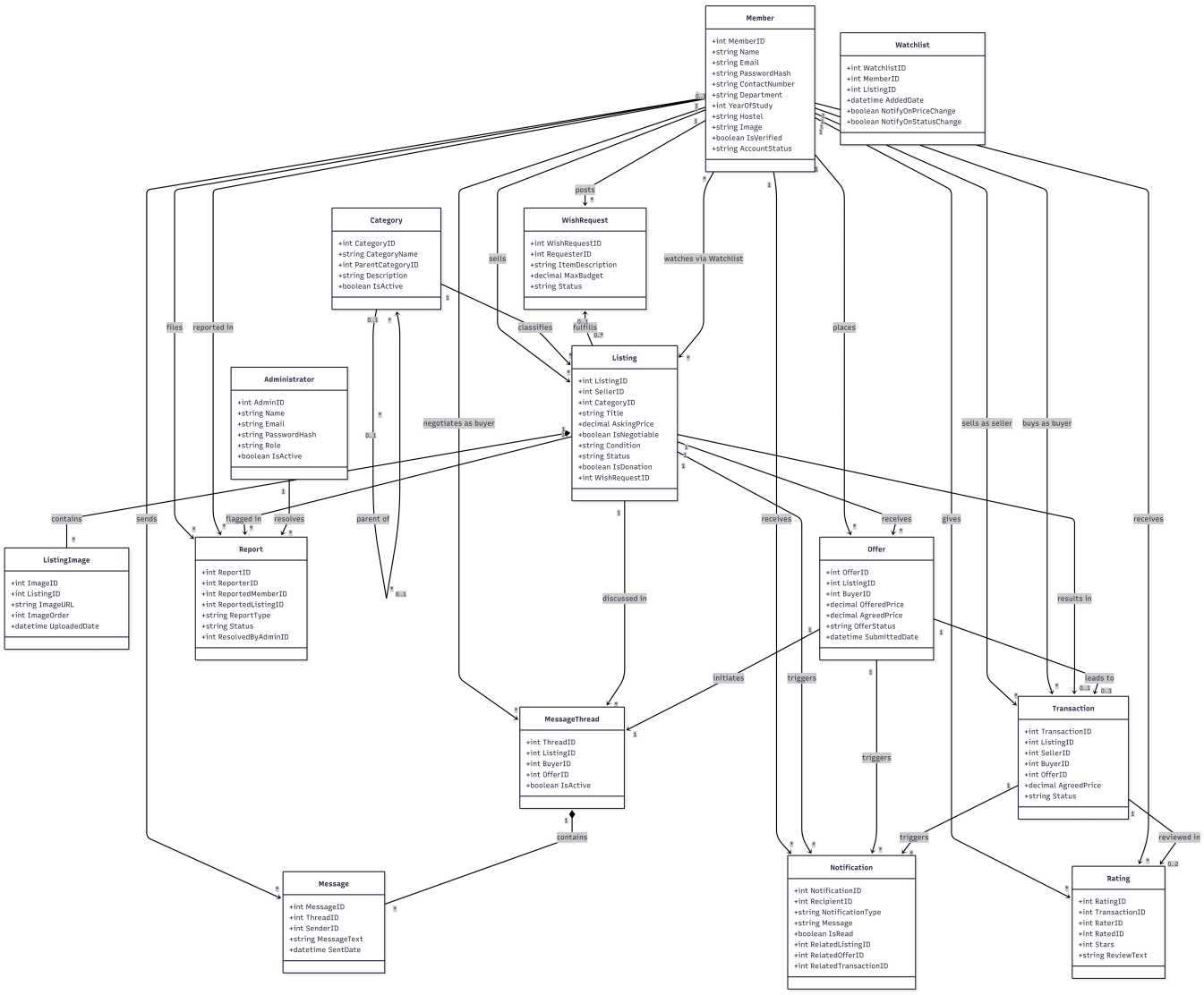

Figure 1: Complete System UML Class Diagram

- **Composition relationships:** *Listing–ListingImage* and *MessageThread–Message*, indicating that the composed entities have no independent existence outside their parent entities.
- **Many-to-many relationship:** *Member–Listing* implemented through the *Watchlist* junction entity.
- **Role-based associations:** *Member* participates in *Transaction* as both seller and buyer, and in *Rating* as both rater and rated, demonstrating role-differentiated multiplicities.
- **Self-referencing association:** *Category* maintains a recursive relationship through *ParentCategoryID* to model hierarchical product categorization.

3.2.2 ***User Management Module.*** This diagram [2] illustrates UML generalization (inheritance) as defined in Section 10.3 of Elmasri and Navathe. An abstract superclass, *User*, captures attributes common to all authenticated users, namely *Name*, *Email*, and *PasswordHash*.

Two concrete subclasses specialize this superclass:

- **Member:** Represents verified student accounts with campus-specific attributes and methods for registration, email verification, and profile management.
- **Administrator:** Represents platform staff with role-based access (*SuperAdmin*, *Moderator*, *Support*) and administrative methods for moderation and system management.

The generalization is *disjoint* and *total*, ensuring that every authenticated user belongs to exactly one subclass. In the relational schema (Module A), this hierarchy is implemented using the *Table-Per-Type* strategy with separate tables for *Member* and *Administrator*.

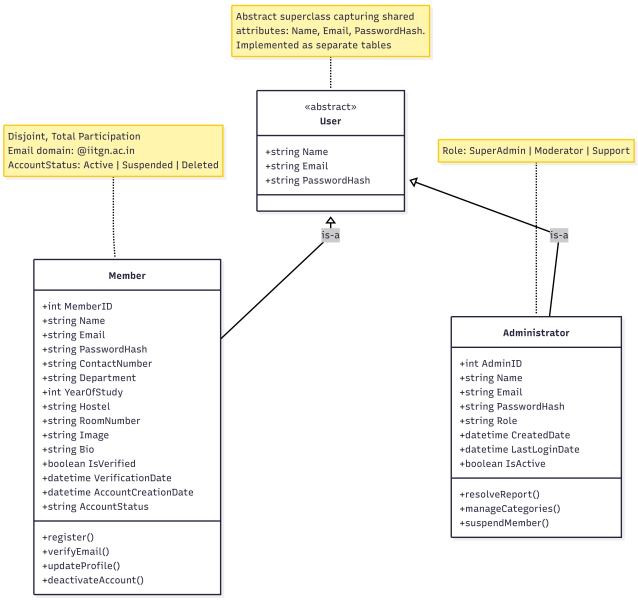

Figure 2: User Management UML Class Diagram

3.2.3 ***Marketplace and Catalog Module.*** This diagram [3] presents the UML class diagram for the Marketplace and Catalog module. The diagram captures the core marketplace structure and relationships among product-related entities.

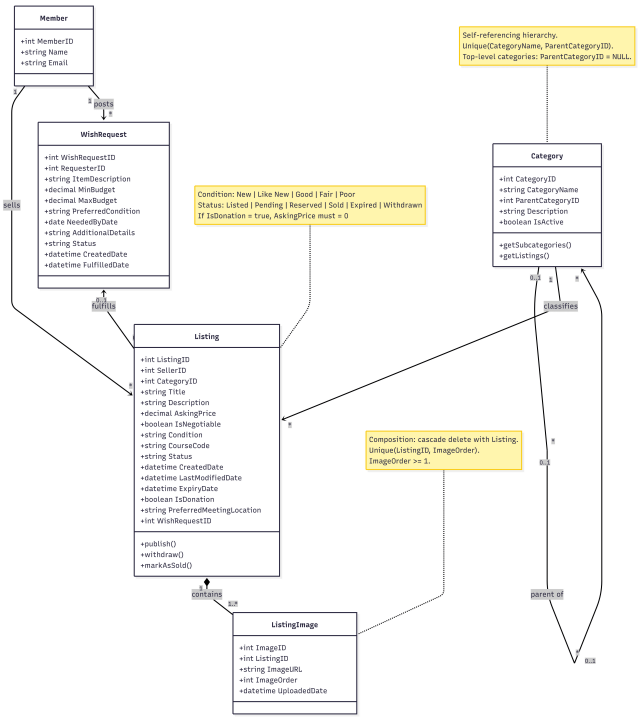

Figure 3: Marketplace & Catalog UML Class Diagram

Category uses a self-referencing association to represent hierarchical classification (e.g., Electronics → Computing → Laptops), with a unique constraint on (*CategoryName*, *ParentCategoryID*) to prevent duplicate names at the same level.

Listing serves as the central marketplace entity, linked to a seller (*Member*) and a *Category*. It includes attributes such as condition, course code, donation flag, and preferred meeting location.

ListingImage is compositionally dependent on *Listing*, indicating **lifecycle dependency** and **cascade deletion**. A uniqueness constraint on (*ListingID*, *ImageOrder*) ensures distinct image ordering.

WishRequest models demand-driven listings through an optional bidirectional association with *Listing*, supporting the "I Have This" feature.

3.2.4 ***Trading Pipeline Module.*** Figure [4] illustrates the UML class diagram for the Trading Pipeline module, modeling the end-to-end trading workflow.

Offer represents buyer-submitted price proposals on listings, with constraints enforcing a 48-hour auto-expiry and mandatory population of *AgreedPrice* upon acceptance. The one-to-many association from *Listing* to *Offer* allows multiple competing offers per listing.

Transaction records the outcome of accepted offers by linking buyers, sellers, and listings. It enforces the constraint *BuyerID* ≠ *SellerID* and implements dual confirmation, transitioning to *Completed* status only when both parties confirm.

Rating supports post-transaction reputation building. Each transaction may have up to two ratings, enforced through multiplicity constraints and uniqueness rules preventing self-rating.

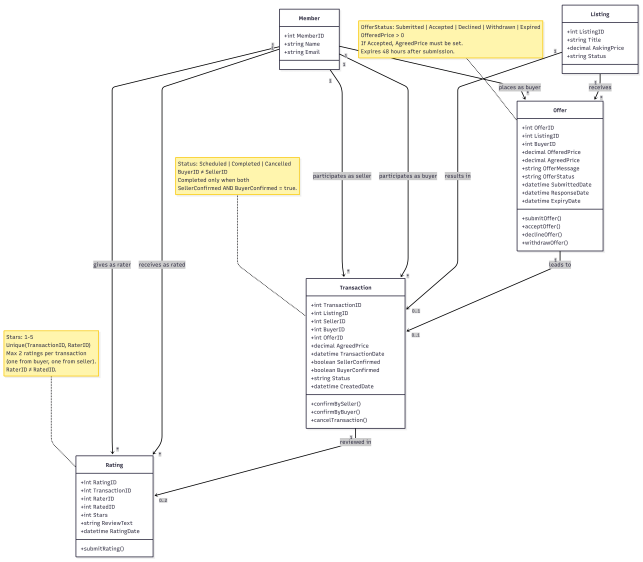

Figure 4: Trading Pipeline UML Class Diagram

The trading pipeline follows the lifecycle: *Listing* → *Offer* → *Transaction* → *Rating*.

3.2.5 ***Communication Module.*** Figure [5] presents the UML class diagram for the Communication module, which supports negotiation messaging between buyers and sellers.

MessageThread represents a dedicated conversation channel for a specific buyer–seller pair on a particular listing and is initiated by an *Offer*. A uniqueness constraint on (*ListingID*, *BuyerID*) ensures a single thread per buyer–listing combination.

Message is compositionally dependent on *MessageThread*, with cascade deletion applied when a thread is removed. The *SenderID* is restricted to either the buyer or the seller of the associated listing, preventing unauthorized participants.

The thread lifecycle is governed by the associated offer state: when an offer is accepted, declined, or expires, the thread is deactivated to prevent further messaging. .

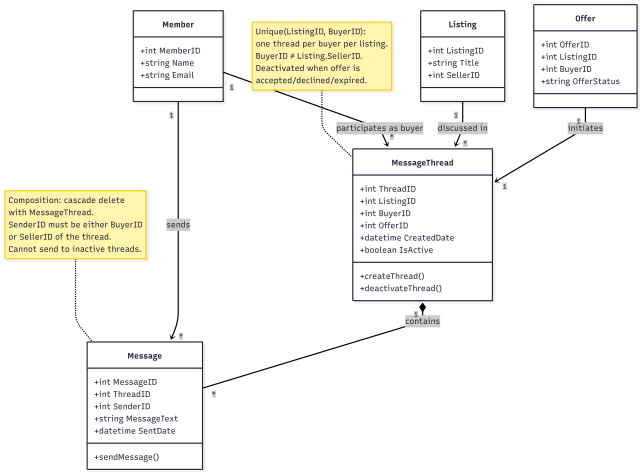

Figure 5: Communication UML Class Diagram

3.2.6 ***Discovery and Engagement Module***. Figure [6] illustrates the UML class diagram for the Discovery and Engagement module, which supports user-driven discovery and interaction.

WishRequest enables demand-driven discovery by allowing students to post requests for unavailable items. Sellers may respond by creating listings that reference the corresponding wish request.

Watchlist is modeled as an association class on the many-to-many relationship between *Member* and *Listing*. It extends this relationship with notification preferences, such as alerts for price changes or status updates.

Notification provides a polymorphic alert mechanism with optional references to *Listing*, *Offer*, and *Transaction*. Notifications are categorized by type and track read status to measure user engagement.

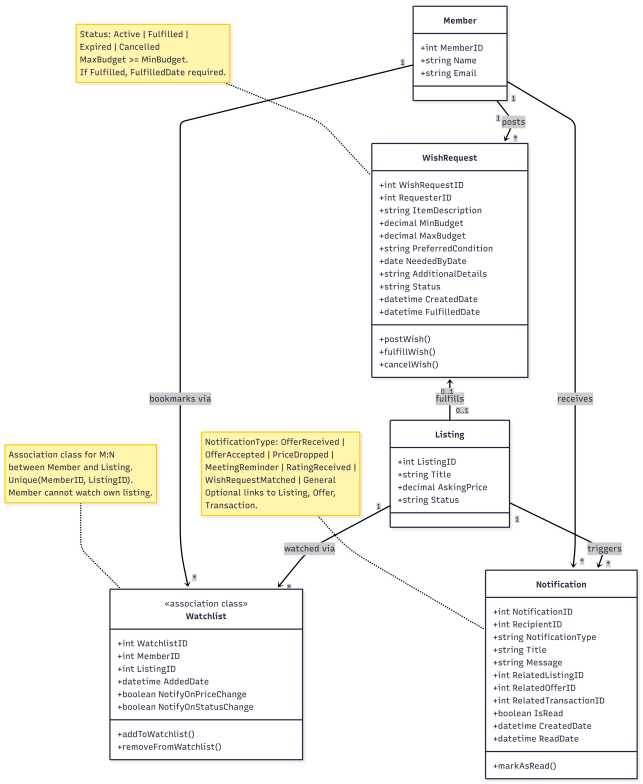

Figure 6: Discovery & Engagement UML Class Diagram

3.2.7 ***Moderation and Administration Module***. Figure [7] presents the UML class diagram for the Moderation and Administration module, which defines the platform's governance and control mechanisms.

Report enables users to submit complaints against members or listings, with the constraint that at least one reported target must be specified. Supported report types include misleading descriptions, scams, no-shows, inappropriate content, price manipulation, and fake offers.

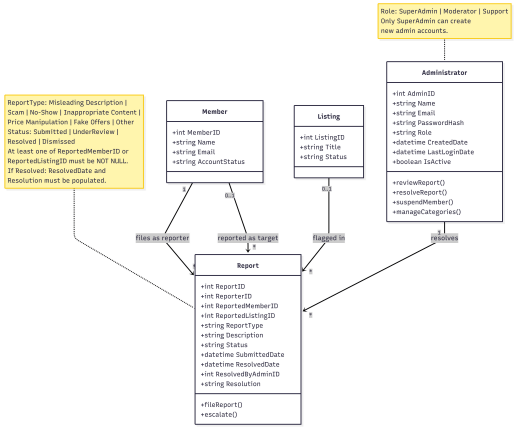

Figure 7: Moderation & Administration UML Class Diagram

Administrator manages report resolution through a defined workflow (*Submitted* → *UnderReview* → *Resolved* or *Dismissed*). When a report is resolved, administrator identity and resolution details are mandatory to ensure accountability.

Administrative responsibilities are isolated from regular users through the separation of *Administrator* and *Member* entities, enforced via UML generalization and role-based access control, where only *SuperAdmin* users may create new administrator accounts.

3.3 UML State Machine Diagrams

State machine diagrams (also called statechart diagrams) model the dynamic behaviour of objects by specifying the **valid states and transitions triggered by events**. Six stateful entities in the Campus Trading system warrant state machine modelling.

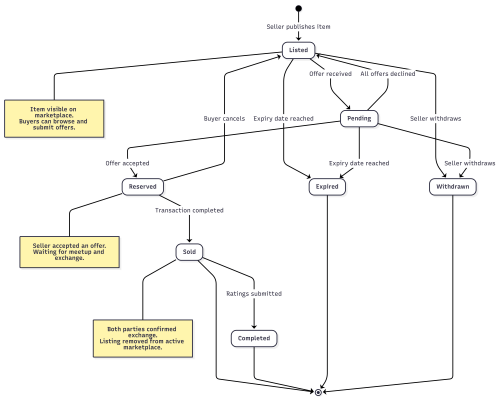

Figure 8: Listing State Machine Diagram

3.3.1 ***Listing Lifecycle.*** A listing [8] begins in the *Listed* state upon publication. Incoming offers transition it to *Pending*. If the seller accepts an offer, the listing moves to *Reserved*. Upon successful exchange with mutual confirmation, the listing reaches the terminal *Sold* state (optionally *Completed* after ratings). Listings may also reach terminal states *Expired* or *Withdrawn*. A declined offer can return the listing from *Pending* back to *Listed*, and a buyer cancellation can return it from *Reserved* back to *Listed*.

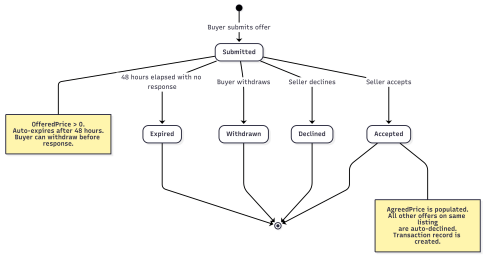

Figure 9: Offer State Machine Diagram

3.3.2 ***Offer Lifecycle.*** An offer [9] starts as *Submitted* and has four possible terminal transitions: *Accepted* (seller agrees, `AgreedPrice` populated, other offers auto-declined), *Declined* (seller rejects), *Withdrawn* (buyer cancels before response), or *Expired* (48-hour timeout with no seller response). All terminal states are irreversible, preserving the full negotiation audit trail.

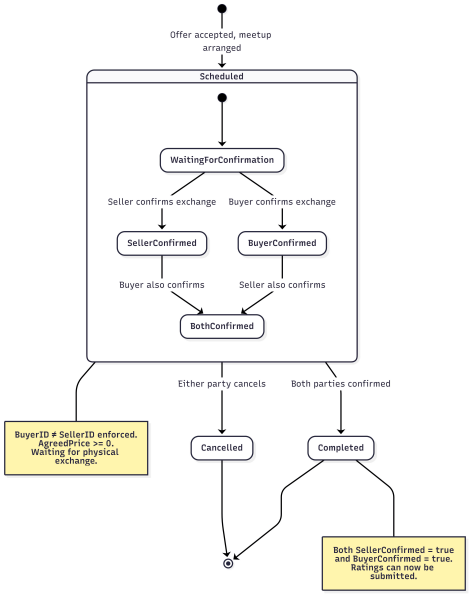

Figure 10: Transaction State Machine Diagram

3.3.3 ***Transaction Lifecycle.*** A transaction [10] begins as *Scheduled* after an offer is accepted. The *Scheduled* state has an internal sub-state machine modelling the dual-confirmation process: both `SellerConfirmed` and `BuyerConfirmed` must become true for the transaction to transition to *Completed*. Either party may initiate *Cancelled* at any point before mutual confirmation.

3.3.4 ***WishRequest Lifecycle.*** A wish request [11] starts as *Active* and transitions to *Fulfilled* when a matching listing is linked (`Listing.WishRequestID` is populated), *Expired* when the `NeededByDate` passes, or *Cancelled* by the requester. All three are terminal states.

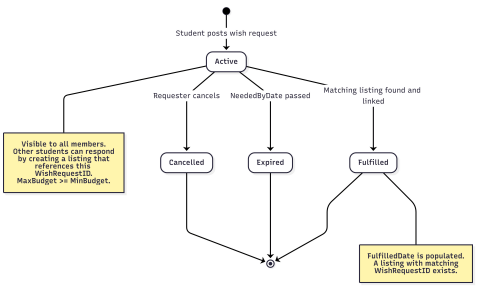

Figure 11: WishRequest State Machine Diagram

3.3.5 ***Report Lifecycle.*** A report [12] begins as *Submitted* when a member files a complaint. An administrator picks it up, transitioning it to *UnderReview*. The administrator then either *Resolves* it (taking action such as suspending the reported member or removing the flagged listing) or *Dismisses* it (no policy violation found). Resolved reports require mandatory `ResolvedDate`, `Resolution` text, and `ResolvedByAdminID` for accountability.

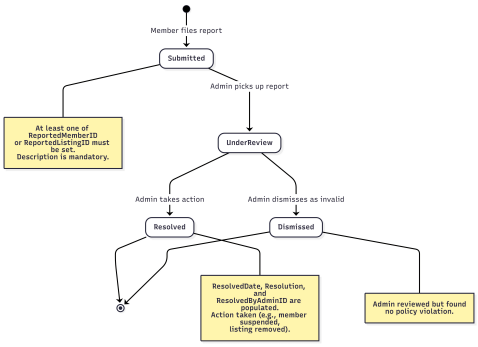

Figure 12: Report State Machine Diagram

3.3.6 ***Member Account Lifecycle.*** A member account [13] starts as *Active* after registration and email verification. Administrators can *Suspend* accounts for policy violations, and suspended accounts can be reinstated back to *Active*. Both *Active* and *Suspended* accounts can transition to the terminal *Deleted* state—either by the member's voluntary deactivation or by administrative permanent removal. Historical data for deleted accounts is retained to preserve referential integrity across transactions, ratings, and messages.

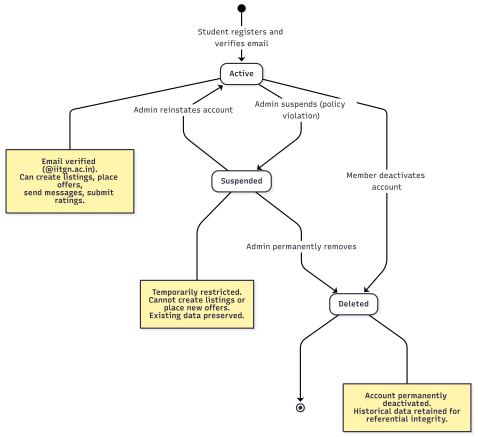

Figure 13: Member Account State Machine Diagram

4 Entity-Relationship Diagram

The ER Diagram uses **Chen notation** [2] as specified in the assignment. Entities are rectangles, relationships are diamonds, and attributes are shown inline within each entity box. Primary keys are underlined; foreign keys are noted explicitly.

4.1 ER Diagram Overview

Figure 14 shows the central entities and relationships. The complete A3-format diagram is available in the project repository [3].

4.2 Special ER Constructs

4.2.1 Self-Referencing Entity: Category. The `Category` entity includes a self-referencing relationship. The FK `ParentCategoryID` points back to `CategoryID` in the same table, modelling arbitrary-depth category hierarchies. In the ER diagram this is shown as a loop from `Category` back to itself, labelled "parent of (0..1 → 0..*)" with a dashed arc.

4.2.2 Multi-Role Foreign Keys: Member in Transaction. The `Transaction` entity references `Member` *twice*: once as `SellerID` and once as `BuyerID`. In Chen notation this is represented as two separate relationship diamonds: **SOLD BY** (Transaction → Member, M:1) and **BOUGHT BY** (Transaction → Member, M:1). The `CHECK (BuyerID ≠ SellerID)` constraint ensures these always reference different members.

4.2.3 Weak vs. Strong Entities. All 11 entities have their own `AUTO_INCREMENT` primary keys and are therefore **strong entities**. `ListingImage` and `Rating` are the closest to weak-entity semantics (they have no meaning outside their parent), but both have independent PKs (`ImageID`, `RatingID`) so they remain strong.

4.2.4 Associative Entity: Watchlist. `Watchlist` resolves the M:M relationship between `Member` and `Listing`. Because it carries its own attributes (`AddedDate`, `NotifyOnPriceChange`, `NotifyOnStatusChange`), it is modelled as an **associative entity** (rectangle with surrounding diamond in extended ER notation) rather than a plain bridge table. The composite unique key `(MemberID, ListingID)` is shown in the ER as a dashed underline.

5 UML-to-ER Transition

5.1 Conversion Methodology

The conversion from UML Class Diagram to ER Diagram follows four systematic rules derived from Elmasri & Navathe Chapter 9 [1]:

R1. Class → Entity: Every UML class with persistent state becomes an ER entity rectangle.

R2. Attribute → Attribute: UML class attributes map directly. Constraints expressed as stereotypes in UML («`PK`», «`unique`») become SQL constraints.

R3. Association → Relationship or FK: A 1:M UML association becomes a FK in the "many" side table. A M:M becomes a junction/associative entity. A 1:1 becomes a FK (on whichever side is optional or has lower cardinality).

R4. Generalisation → Ignored (Role-Based): No inheritance hierarchy exists in this design. The `Member` class serves multiple roles via contextual FK references.

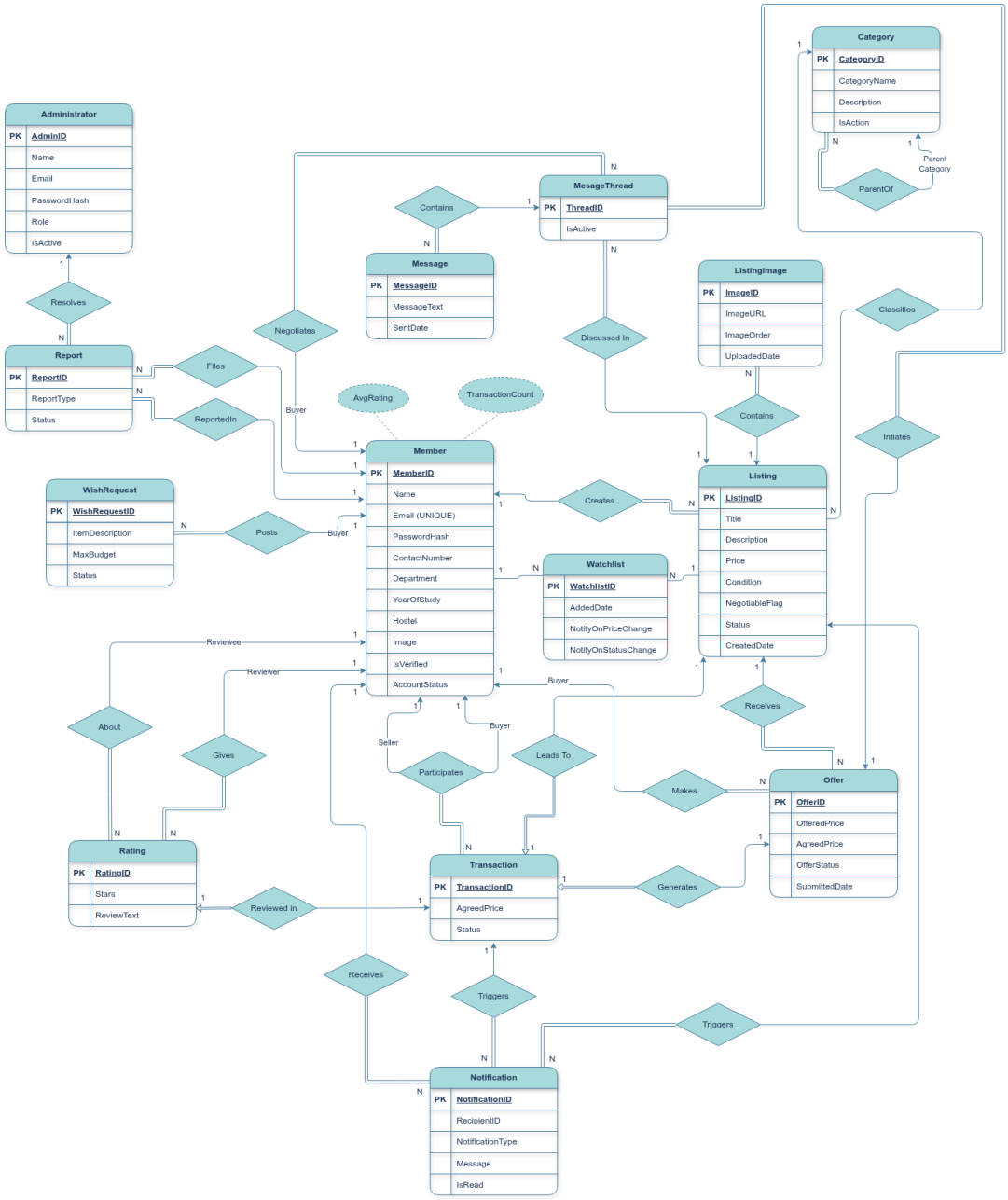

Figure 14: Complete System UML Class Diagram

5.2 Class-by-Class Mapping Table

Table 3 documents the complete conversion for all 11 classes.

5.3 Key Multiplicity Adjustments

When converting UML multiplicities to ER/relational form, several adjustments were necessary:

M:M → Junction Table. The UML M:M association between `Member` and `Listing` ("a member watches many listings; a listing is watched by many members") cannot be represented by a FK in either table. It becomes the `Watchlist` junction table with composite PK `(MemberID, ListingID)`.

1:1 as Optional FK. The UML 1:0..1 association between `Listing` and `Transaction` (a listing leads to at most one transaction) is represented as a FK `Transaction.ListingID` → `Listing`. No `UNIQUE` constraint is placed on this FK at the database level because the business rule (one accepted offer per listing) is enforced at the application layer.

Table 3: UML Class to ER Entity Mapping

UML Class	ER Entity	Conversion Notes
Member	Member	Direct mapping. No generalisation used; roles (buyer/seller) determined by FK context. `Email LIKE '%@iitgn.ac.in'` becomes a `CHECK` constraint.
Administrator	Administrator	Separate entity, not a subclass of Member. Referenced only via `Report.ResolvedByAdminID`.
Category	Category	Self-referencing FK: `ParentCategoryID` → `CategoryID`. `UNIQUE(CategoryName, ParentCategoryID)` enforces name uniqueness per level.
Listing	Listing	Three incoming FKs: `SellerID`, `CategoryID`, `WishRequestID`. UML composition to ListingImage → `ON DELETE CASCADE` FK.
ListingImage	ListingImage	UML composition (whole = Listing, part = Image). In ER: 1:M with `CASCADE DELETE`. `UNIQUE(ListingID, ImageOrder)` preserves display sequence.
Offer	Offer	Two 1:M associations from Listing (1) and Member (1) → two FKs. `ExpiryDate = SubmittedDate + 48h` enforced at application layer.
Transaction	Transaction	Three 1:M associations (from Listing, Member×2, Offer) → four FKs. `CHECK` on Completed status.
Rating	Rating	1:M from Transaction (max 2). `UNIQUE(TransactionID, RaterID)` enforces the two-ratings-per-transaction rule.
WishRequest	WishRequest	1:M from Member. Reverse link to Listing stored as `Listing.WishRequestID` (optional FK).
Watchlist	Watchlist	M:M association class between Member and Listing. `UNIQUE(MemberID, ListingID)` prevents duplicates. Own attributes justify an independent table over a plain bridge.
Report	Report	Multi-target entity: `ReportedMemberID` or `ReportedListingID` (one required). `ResolvedByAdminID` → `Administrator` (audit trail).

Self-referencing Association. The UML reflexive association on `Category` ("a category may have a parent category") becomes the nullable FK `ParentCategoryID` → `CategoryID` in the same table.

Multiple FK References to Same Entity. `Transaction` has two FKs to `Member` (`SellerID` and `BuyerID`) and two FKs to `Listing`/`Offer`. In UML this is two labelled associations; in the ER/relational model it is two separate FK columns. Naming disambiguation follows the convention `EntityID_Role` where ambiguous.

6 Relationship Justifications

6.1 One-to-Many (1:M) Relationships

Refer to Table 4.

6.2 One-to-One (1:1) Relationships

6.2.1 Listing ↔ Transaction (0..1 : 0..1). A listing leads to *at most* one completed transaction. This is a structural 1:1 at the business level: only one offer can be accepted, which spawns exactly one `Transaction` record.

Implementation: `Transaction.ListingID` is a FK to `Listing`. The uniqueness is enforced at the application layer ("accept one offer → auto-decline all others") rather than a DB-level `UNIQUE` constraint on `Transaction.ListingID`, because the relational schema allows the DB-level column to be non-unique while the application ensures only one transaction is ever inserted per listing.

Example: Listing #42 ("Samsung Galaxy A53") receives 3 offers. Seller accepts offer #7. Transaction #18 is created linking to Listing #42. No further transaction rows can reference Listing #42.

6.2.2 Offer ↔ Transaction (0..1 : 1). Each transaction references exactly one offer (the accepted one). An offer either leads to zero or one transaction.

Implementation: `Transaction.OfferID` FK to `Offer.OfferID`.

6.3 Many-to-Many (M:M) Relationships

6.3.1 Member ↔ Listing via Watchlist. A student can monitor many listings; a listing can be monitored by many students.

Why a junction table is necessary: Neither `Member` nor `Listing` can hold a FK to the other for this relationship without violating 1NF (a member watching 8 listings would require 8 FK columns or a repeating group).

Why an associative entity instead of a pure bridge table: `Watchlist` carries its own meaningful attributes:

- `AddedDate` — when the bookmark was created
- `NotifyOnPriceChange` — personalised alert preference
- `NotifyOnStatusChange` — personalised alert preference

These attributes depend on the *combination* of `(MemberID, ListingID)`, satisfying 2NF.

Example: Student A watches Listings #12, #17, #31 (3 rows in Watchlist). Listing #12 is watched by Students A, B, C (same 3 rows show `ListingID=12`).

Constraint: `UNIQUE(MemberID, ListingID)` prevents a student from adding the same listing to their watchlist twice.

7 Additional Integrity Constraints

Constraints are organised into three enforcement layers, following the principle of defence-in-depth for data integrity.

7.1 Database-Level Constraints (SQL)

CHECK Constraints — Member

- `CHK_Member_Email_Domain`: `Email LIKE '%@iitgn.ac.in'` — institutional email only.

Table 4: 1:M Relationships — Justification

Relationship	Card.	Justification & Example
Member → Listing	1:M	A student selling multiple items simultaneously (e.g., a textbook, a desk fan, and a bicycle) creates three separate listings under one account. FK: `Listing.SellerID` → `Member.MemberID`.
Category → Listing	1:M	Many listings share a category (e.g., all laptop listings belong to "Electronics → Computing → Laptops"). FK: `Listing.CategoryID` → `Category.CategoryID`.
Category → Category	1:M (self)	"Electronics" is the parent of "Computing" which is the parent of "Laptops". FK: `Category.ParentCategoryID` → `Category.CategoryID`.
Listing → ListingImage	1:M	A seller uploads 4 photos of their bicycle from different angles. Each is a separate `ListingImage` row with `ImageOrder` 1–4. FK with `ON DELETE CASCADE`.
Listing → Offer	1:M	One negotiable listing attracts 3 competing offers from different buyers. All share the same `ListingID`. This enables the seller to compare and choose the best bid.
Member → Offer (Buyer)	1:M	The same buyer submits offers on a desk lamp, a calculator, and a textbook simultaneously. FK: `Offer.BuyerID` → `Member.MemberID`.
Member → Transaction (Seller)	1:M	A seller who has completed 12 sales has 12 rows in `Transaction` where `SellerID` = their `MemberID`. Drives the public "12 sales completed" badge.
Member → Transaction (Buyer)	1:M	A buyer's purchase history: all transactions where `BuyerID` = their `MemberID`. Used for progressive trust limit calculation.
Transaction → Rating	1:(0..2)	After one transaction, both buyer and seller each leave a review — max 2 rows in `Rating` per `TransactionID`. `UNIQUE(TransactionID, RaterID)` enforces this cap.
Member → WishRequest	1:M	A student simultaneously seeks a TI-84 calculator, a desk lamp, and a data structures textbook — three wish requests. Limit of 5 active requests enforced at application level.
Member → Report	1:M	One vigilant member files 3 reports over the semester — against a misleading listing, a no-show seller, and scam attempt. FK: `Report.ReporterID`.
Listing → Report	1:M	A suspicious listing is reported by 5 different students before an admin reviews it. FK: `Report.ReportedListingID`.
Administrator → Report	1:M	One moderator resolves 14 reports in a month. FK: `Report.ResolvedByAdminID`; provides admin workload audit trail.

- `CHK_Member_AccountStatus`: `AccountStatus IN ('Active','Suspended','Deleted')`.
- `CHK_Member_YearOfStudy`: `YearOfStudy BETWEEN 1 AND 5`.
- `CHK_Member_Verification`: If `IsVerified=TRUE` then `VerificationDate IS NOT NULL`.

CHECK Constraints — Listing

- `CHK_Listing_Price`: `AskingPrice ≥ 0`.
- `CHK_Listing_Donation`: If `IsDonation=TRUE` then `AskingPrice=0`.
- `CHK_Listing_Status`: `Status IN ('Listed','Pending','Reserved','Completed','Sold','Expired','Withdrawn')`.
- `CHK_Listing_Condition`: `Condition IN ('New','Like New','Good','Fair','Poor') OR Condition IS NULL`.
- `CHK_Listing_Expiry`: `ExpiryDate IS NULL OR ExpiryDate > CreatedDate`.

CHECK Constraints: Offer / Transaction / Rating

- `CHK_Offer_Price`: `OfferedPrice > 0`.
- `CHK_Offer_Status`: `OfferStatus IN ('Submitted','Accepted','Declined','Withdrawn','Expired')`.
- `CHK_Offer_Agreed`: If `OfferStatus='Accepted'` then `AgreedPrice IS NOT NULL`.
- `CHK_Transaction_DifferentParties`: `BuyerID ≠ SellerID`.
- `CHK_Transaction_Completed`: If `Status='Completed'` then `SellerConfirmed=TRUE AND BuyerConfirmed=TRUE`.
- `CHK_Transaction_Price`: `AgreedPrice ≥ 0`.
- `CHK_Rating_Stars`: `Stars BETWEEN 1 AND 5`.
- `CHK_Rating_DifferentMembers`: `RaterID ≠ RatedID`.

CHECK Constraints: WishRequest and Report

- `CHK_WishRequest_Budget`: `MaxBudget IS NULL OR MinBudget IS NULL OR MaxBudget ≥ MinBudget`.
- `CHK_WishRequest_Fulfilled`: If `Status='Fulfilled'` then `FulfilledDate IS NOT NULL`.
- `CHK_Report_Target`: `ReportedMemberID IS NOT NULL OR ReportedListingID IS NOT NULL`.
- `CHK_Report_Resolved`: If `Status='Resolved'` then `ResolvedDate IS NOT NULL AND Resolution IS NOT NULL`.

UNIQUE Constraints

- `Member.Email` — no two accounts share an email address.
- `Administrator.Email` — same for admin accounts.
- `UNIQUE(CategoryName, ParentCategoryID)` — category names unique within the same parent level.

- UNIQUE(ListingID, ImageOrder) — no two images occupy the same display position.
- UNIQUE(MemberID, ListingID) on Watchlist — a member cannot watch the same listing twice.
- UNIQUE(TransactionID, RaterID) on Rating — each transaction participant rates once.

Foreign Key & Referential Integrity Rules

- Listing.SellerID → Member: ON UPDATE CASCADE, ON DELETE NO ACTION.
- ListingImage.ListingID → Listing: ON UPDATE CASCADE, ON DELETE CASCADE — photos removed with their listing.
- Watchlist.MemberID → Member: ON DELETE CASCADE — watchlist cleared when account deleted.
- Watchlist.ListingID → Listing: ON DELETE CASCADE — watchlist cleared when listing deleted.
- Report.ResolvedByAdminID → Administrator: ON DELETE SET NULL — report preserved if admin account removed.
- All other FK references: ON DELETE NO ACTION, ON UPDATE CASCADE.

7.2 Application-Level Constraints

These rules cannot be expressed in SQL and are enforced in application code:

Table 5: Application-Level Constraints

Constraint	Rule
Progressive listing limit	0 transactions: max 2 active listings; 1–4 transactions: max 5; 5+ transactions: unlimited.
Wish request limit	Max 5 active WishRequest rows per MemberID at any time.
No self-offer	Offer.BuyerID ≠ Listing.SellerID.
No self-watch	Watchlist.MemberID ≠ Listing.SellerID.
Single accepted offer	Only one Offer per ListingID may hold OfferStatus='Accepted'.
Contact reveal timing	ContactNumber and RoomNumber exposed only after offer accepted.
Rating gate	Rating rows can only be created when the referenced Transaction.Status='Completed'.

7.3 Trigger-Based Constraints

These are state-change side-effects implemented as database triggers or scheduled jobs:

Table 6: Trigger-Based Automated Constraints

Trigger Event	Automated Action
Offer submitted on listing	Listing.Status → 'Pending'.
Offer accepted	All other offers for same listing auto-Declined; Listing.Status → 'Reserved'; notifications sent.
Both parties confirm	Transaction.Status → 'Completed'; Listing.Status → 'Sold'; TotalTransactions++ for both.
New Rating inserted	Member.AverageRating recalculated via AVG(Stars) WHERE RatedID = MemberID.
Listing marked Sold/Withdrawn	All Watchlist entries for that listing removed.
AskingPrice decreased	Notification created for all watchers (type: PriceDropped).
Listing reaches ExpiryDate	Listing.Status → 'Expired' (scheduled job, runs nightly).
Offer reaches ExpiryDate	Offer.OfferStatus → 'Expired' (scheduled job).

8 Alignment with Module A Database Schema

The ER and UML diagrams presented in this report map directly and completely to the SQL schema submitted in Module A [3]. Table 7 confirms this alignment for all 11 tables.

Table 7: ER Entity to SQL Table Alignment

ER Entity	SQL Table	PK	FKs
Member	Member	MemberID	0
Administrator	Administrator	AdminID	0
Category	Category	CategoryID	1 (self)
Listing	Listing	ListingID	3
ListingImage	ListingImage	ImageID	1
Offer	Offer	OfferID	2
Transaction	Transaction	TransactionID	4
Rating	Rating	RatingID	3
WishRequest	WishRequest	WishRequestID	1
Watchlist	Watchlist	WatchlistID	2
Report	Report	ReportID	4
Total FK references			**21**

Every CHECK constraint, UNIQUE constraint, and NOT NULL column defined in the ER diagram and UML Class Diagram has a corresponding declaration in the Module A SQL CREATE TABLE statements. The complete SQL schema, INSERT statements (15–20 rows per table), and a database dump are available at [3].

9 Conclusion

This report has presented the complete conceptual and logical database design for the Campus Trading Application. Starting from the requirement specification [4], we derived four UML diagrams (Class, Use Case, Sequence, Activity) and an ER Diagram in Chen notation, covering all 11 entities and 21 foreign key relationships.

The UML-to-ER transition was performed systematically using the four conversion rules in Section 5: every class became an entity, every 1:M association became a FK, the one M:M association became the `Watchlist` associative entity, and no generalisation hierarchies were required. All relationship cardinalities were justified with concrete usage examples in Section 6.

The schema satisfies Third Normal Form across all tables, with two deliberate and documented denormalisations (`AverageRating` and `TotalTransactions`) justified on performance grounds. A total of 26 integrity constraints spanning database, application, and trigger layers protect data consistency throughout all seven core workflows.

All SQL artifacts, diagram source files, and this report are available in the project repository [3].

10 Team Member Contributions

Table 8: Team Member Contributions

Member	Roll No.	Contributions
Bhavik Patel	22110047	Ideation document and initial write-up [4]; UML Use Case Diagram; application-level constraint definition; report Section 6 (relationship justifications).
Hitesh Kumar	22110098	System requirements analysis (Phase 1); UML Class Diagram design; alignment verification between diagrams and SQL schema; report Section 2 and Section 3.
Jinil Patel	22110184	Module A SQL schema design (`CREATE TABLE` statements); CHECK and UNIQUE constraint specification; database dump and GitHub repository management [3]; report Section 7 (constraints).
Pranav Patil	22110199	ER Diagram design (Chen notation); UML Sequence Diagram; UML-to-ER transition analysis; normalisation verification (1NF/2NF/3NF); report Sections 4, 5, and 9.
Sibtain	22110148	UML Activity Diagram; synthetic data generation (15–20 rows per table); trigger-based constraint specification; report Section 8 and final compilation/review.

All team members participated in design review sessions and cross-checked each other's diagrams against the SQL schema for consistency.

References

[1] R. Elmasri and S. B. Navathe, *Fundamentals of Database Systems*, 7th ed. Pearson Education, 2015. Chapters 7 (ER Model), 9 (Relational DB Design), 10 (UML).

[2] A. Silberschatz, H. F. Korth, and S. Sudarshan, *Database System Concepts*, 7th ed. McGraw-Hill Education, 2019. Chapter 6 (E-R Model and Relational Design).

[3] Campus Trading Application Team, "Campus Trading Application — SQL Schema, Dump Files, and Diagram Scripts," GitHub Repository, 2025. [Online]. Available: https://github.com/Zeenu03/Campus_Trading_App

[4] Campus Trading Application Team, "Campus Trading Application — System Ideation and Requirement Specification (Phase 1 Write-up)," Project Documentation, IIT Gandhinagar, 2025. [Online]. Available: https://docs.google.com/document/d/1e_QuEpTuEb1YhiF_W-yfh9A6REIXPcwGeUspzoZSo4U/edit?usp=sharing