**Business Problems Being Addressed:**

The database underpins a **multi-tenant SaaS property management platform** and is designed to solve the following problems:

1. **Tenant (Company) Isolation with Shared Infrastructure**
   * Serve many property management companies in one deployment while guaranteeing strict logical isolation of each company’s data.
   * Prevent cross-tenant access, simplify upgrades, and keep costs predictable.
2. **End-to-End Tenant (Renter) Lifecycle**
   * Capture a renter’s journey: application → screening attributes → lease execution → renewals/transfer → move-out.
   * Preserve historical state for legal, financial, and operational analytics.
3. **Property & Unit Inventory Management**
   * Model properties and units with status (vacant/occupied/offline), amenities, physical attributes, and pricing to reduce vacancies and support listing workflows.
4. **Maintenance Workflow Orchestration**
   * Intake, triage, and prioritize service requests; assign to one or more vendors; track SLA milestones (created/assigned/in-progress/completed).
   * Provide durable records for cost, quality, and compliance.
5. **Financial Operations & Reconciliation**
   * Issue invoices for rent/fees, accept partial payments, reconcile transactions, and compute balances precisely per lease and tenant.
   * Support multiple payment methods and external gateway references.
6. **Centralized, Composable Document Management**
   * Store leases, IDs, inspection reports, photos, and receipts; attach documents to multiple business objects without duplication.
7. **Secure, Auditable Communication**
   * Enable messages between tenants, managers, vendors, and finance with delivery state (read/unread) and conversation threads for accountability.
8. **Auditability & Compliance**
   * Produce immutable activity logs across critical operations to satisfy legal and security requirements; enable forensic investigations.
9. **Analytics & Reporting Readiness**
   * Provide clean, consistent schemas for BI on occupancy, AR aging, collections rate, tenant churn, and maintenance SLA performance.
10. **Scalability & Operational Robustness**

* Sustain growth in companies, properties, tenants, and transactions; support batch billing, notifications, and archival without impacting hot paths.

**Entities and Relationships**

**2.1 Core Entities (with scope)**

* **ManagementCompany** *(tenant scope)* — Root tenant/company record.
* **User** *(tenant scope)* — People who use the system (managers, tenants, maintenance staff, accountants).
* **Role** — System roles/permissions (global catalog).
* **UserRole** *(junction)* — Assigns roles to users (supports many-to-many).
* **Property** *(tenant scope)* — A managed property owned/operated by a company.
* **Unit** *(tenant scope via Property)* — An individual rentable unit within a property.
* **Amenity** — Catalog of amenities/features.
* **UnitAmenity** *(junction)* — Links amenities to units (many-to-many).
* **Tenant** *(tenant scope via User)* — Renter profile (extension of a User when user\_type='tenant').
* **Lease** *(tenant scope)* — Agreement between a tenant and a unit across a date range.
* **Invoice** *(tenant scope)* — Billable record tied to a lease.
* **Payment** *(tenant scope)* — Money received, tied to a tenant and an invoice.
* **MaintenanceRequest** *(tenant scope)* — Service ticket for a unit (often initiated by a tenant).
* **Vendor** *(tenant scope)* — External service provider contracted by a company.
* **MaintenanceAssignment** *(junction)* — Assigns vendors to maintenance requests (many-to-many).
* **Document** *(tenant scope)* — Stored artifact (file) with type and metadata.
* **LeaseDocument** *(junction)* — Links documents to leases (many-to-many; extendable to other entities).
* **MessageThread** *(tenant scope)* — Conversation container for messages.
* **Message** *(tenant scope)* — Individual message (sender, recipient, body, read state).
* **AuditLog** *(tenant scope)* — Append-only record of critical actions.

**2.2 Canonical Relationships & Cardinalities**

* **ManagementCompany → Property / User / Vendor / MessageThread / Document / AuditLog**: **1:N**  
  Each tenant company owns its own resources; isolation enforced by company\_id.
* **Property → Unit**: **1:N**  
  Each property contains multiple units.
* **Unit ↔ Amenity (via UnitAmenity)**: **M:N**  
  Units can have many amenities; one amenity can appear on many units.
* **User ↔ Role (via UserRole)**: **M:N**  
  A user can hold multiple roles; roles can be assigned to many users.
* **User ↔ Tenant**: **1:1** *(conditional)*  
  When a user is a renter, a corresponding Tenant record stores renter-specific data.
* **Tenant → Lease**: **1:N**  
  A tenant can have multiple leases over time (e.g., renewals or transfers).
* **Lease → Invoice**: **1:N**  
  A lease produces periodic invoices (rent, fees).
* **Invoice → Payment**: **1:N**  
  An invoice can be paid partially or in multiple installments.
* **Tenant → Payment**: **1:N**  
  Payments are also tracked per tenant for statements and receipts.
* **Unit / Tenant → MaintenanceRequest**: **1:N** (each)  
  Requests are associated to the affected unit and optionally the requesting tenant.
* **MaintenanceRequest ↔ Vendor (via MaintenanceAssignment)**: **M:N**  
  A request can involve multiple vendors (diagnostic + remediation); a vendor can serve many requests.
* **Lease ↔ Document (via LeaseDocument)**: **M:N**  
  A document may be attached to multiple leases (e.g., standardized addenda), and a lease holds many documents.
* **MessageThread → Message**: **1:N**  
  A thread contains many messages; messages reference sender and recipient Users within the same company.
* **AuditLog → (Any Entity)**: **N:1 by (entity\_type, entity\_id)**  
  Each audit log entry references the target entity and the actor user.

**2.3 Entity Quick Reference (selected attributes)**

* **ManagementCompany**: company\_id, company\_name, subscription\_tier, created\_date, is\_active
* **User**: user\_id, company\_id, email, password\_hash, first\_name, last\_name, phone, user\_type, is\_active
* **Role**: role\_id, role\_name, scope; **UserRole**: (user\_id, role\_id)
* **Property**: property\_id, company\_id, property\_name, address, city, state, zip, property\_type, year\_built, total\_units
* **Unit**: unit\_id, property\_id, unit\_number, floor, bedrooms, bathrooms, square\_feet, base\_rent, status
* **Amenity**: amenity\_id, name; **UnitAmenity**: (unit\_id, amenity\_id)
* **Tenant**: tenant\_id, user\_id, emergency\_contact, employer, income, credit\_score, move\_in\_date
* **Lease**: lease\_id, tenant\_id, unit\_id, start\_date, end\_date, monthly\_rent, security\_deposit, lease\_status
* **Invoice**: invoice\_id, lease\_id, amount\_due, due\_date, invoice\_date, status
* **Payment**: payment\_id, tenant\_id, invoice\_id, amount, payment\_date, payment\_method, transaction\_id
* **MaintenanceRequest**: request\_id, unit\_id, tenant\_id, category, priority, description, status, created\_date, completed\_date
* **Vendor**: vendor\_id, company\_id, vendor\_name, contact\_email, phone, rating, is\_active
* **MaintenanceAssignment**: (request\_id, vendor\_id), assigned\_at, status
* **Document**: document\_id, company\_id, storage\_url, doc\_type, uploaded\_by, uploaded\_at, hash, size\_bytes
* **LeaseDocument**: (lease\_id, document\_id)
* **MessageThread**: thread\_id, company\_id, subject, created\_at
* **Message**: message\_id, company\_id, thread\_id, sender\_user\_id, recipient\_user\_id, body, sent\_at, is\_read
* **AuditLog**: audit\_id, company\_id, actor\_user\_id, entity\_type, entity\_id, action, payload\_json, created\_at

[**ER-Diagram**](https://drive.google.com/file/d/1AlyIalrJYk9FcC5G1FL5J7pqMvrzK__r/view?usp=sharingKey%20Database%20Design%20Decisions)

q**Key Database Design Decisions:**

Our database is designed as a multi-tenant property management platform. The goal is to create a system that is flexible, secure, and scalable, while also making sure the design supports the real-world needs of property managers, tenants, and service providers. Below are the major decisions we made and why:

1. **Multi-Tenancy with Company Isolation**  
   We included a ManagementCompany entity as the root for tenant isolation. Every other entity (Property, Unit, User, Vendor, etc.) connects back to the company through a foreign key. This ensures each property management company has its own data, with no overlap or leakage across companies, while still sharing one database system for efficiency.
2. **Users and Roles**  
   A single User table stores login and profile details. To manage permissions, we created a Role entity and a UserRole junction table. This supports many-to-many relationships so a user can have multiple roles, such as being both a Property Manager and an Accountant. This design provides flexibility without duplicating user data.
3. **Tenant Representation**  
   Rather than mixing tenant details directly into the User table, we designed a separate Tenant entity linked to a User when their role is renter. This keeps authentication and general user data clean while storing renter-specific information such as employment, credit score, and emergency contact separately.
4. **Property and Unit Structure**  
   We separated Property and Unit into distinct entities. Each property can contain many units, and each unit holds details such as rent, size, and number of bedrooms. Amenities are stored in a separate Amenity catalog and linked to units through UnitAmenity. This prevents duplication (e.g., “Pool” or “Gym” stored once instead of repeated across units) and supports flexible listings.
5. **Accurate Financial Tracking (Leases, Invoices, and Payments)**  
   A Lease links tenants to units, recording dates, rent, and deposits. We deliberately separated Invoice and Payment into their own entities. This allows tracking of partial payments, multiple payments for one invoice, or multiple invoices for one lease. This design mirrors real-world financial operations and provides accurate reporting.
6. **Flexible Maintenance Workflow**  
   Maintenance requests are stored in a MaintenanceRequest entity. Since one request can involve multiple vendors, we added a MaintenanceAssignment table to capture many-to-many relationships between requests and vendors. This supports real workflows where different vendors handle diagnostics, repairs, and inspections.
7. **Centralized Document Management**  
   We created a centralized Document entity to store files such as leases, IDs, receipts, and inspection reports. Documents are linked to business entities (like Leases or Units) through junction tables such as LeaseDocument. This design avoids duplication and ensures that documents remain accessible across related entities.
8. **Secure Communication System**  
   To support secure, auditable communication, we included MessageThread and Message entities. Threads organize conversations, and messages track sender, recipient, time, and read/unread status. This ensures accountability and provides a clear record of communication history.
9. **Audit and Compliance**  
   A dedicated AuditLog entity records important actions (who did what and when). Each entry references the affected entity and user. This design supports compliance, helps resolve disputes, and provides transparency for security and legal audits.
10. **Normalization and Relationships**  
    We deliberately separated entities to reduce redundancy (for example, splitting Invoice and Payment, or keeping Amenities in their own table). Relationships follow standard database design principles: one-to-many where appropriate (Property → Units), and many-to-many with junction tables (User ↔ Role, Unit ↔ Amenity, MaintenanceRequest ↔ Vendor). This ensures data consistency and scalability as the system grows.