Apartment Maintenance & Complaints Tracker Database Design Document

1. Business Problems Being Addressed

Apartment complexes face frequent maintenance issues (plumbing, electrical, HVAC, pest control, etc.) and tenant complaints. Without a structured system, issues are often lost, delayed, or untracked.

Business Issues:

1. **Broken reporting** — Residents' complaints are not centrally recorded or managed.
2. **No assignment/audit trail** — Requests are not consistently assigned or tracked to completion.
3. **Lack of transparency** — Tenants cannot monitor complaint status; managers cannot track staff performance.
4. **Escalation & accountability gaps** — No clear process for escalating unresolved issues.
5. **Missing cost reconciliation** — Labor and material costs are not accurately linked to specific maintenance work.

Objective:
To unify logging, assignment, notification, escalation, and billing so that property management can monitor accountability, timeliness, and ensure transparency for residents.

---

2. Entities and Relationships

The design is based on the **Enhanced ER Model (EER)**, with specialization, weak entities, and associative relationships.

Entities:

1. **Resident (Strong Entity)**
   PK: ResidentID
   Attributes: Name, PrimaryPhone, Email, MoveInDate, LeaseStatus,

PreferredContactChannel
Purpose: Identifies residents who file maintenance or complaint requests.

2. **ApartmentUnit (Strong Entity)**
   PK: UnitID
   Attributes: Building, Floor, ApartmentNumber, UnitType, SquareFeet
   Purpose: Represents apartment units associated with residents and complaints.

3. **Lease (Associative Entity)**
   PK: LeaseID
   Attributes: StartDate, EndDate, LeaseStatus, IsPrimaryResident
   Purpose: Resolves the many-to-many relationship between residents and apartment units.

4. **Request (Strong Entity)**
   PK: RequestID
   Attributes: Title, Description, DateSubmitted, UrgencyLevel, Status, Deadline
   Purpose: Core entity capturing complaints and maintenance requests.

5. **RequestCategory (Strong Entity)**
   PK: CategoryID
   Attributes: CategoryName, Description
   Purpose: Normalizes complaint types for better tracking and reporting.

6. **Worker (Supertype)**
   PK: WorkerID
   Attributes: Name, Phone, Email, Specialty, WorkerType (Staff/Contractor)
   Purpose: Represents any individual performing maintenance work.

7. **MaintenanceStaff (Subtype)**
   PK: WorkerID → Worker
   Attributes: EmployeeNumber, Role, AvailabilityStatus, HireDate
   Purpose: Represents internal employees.

8. **Contractor (Subtype)**
   PK: WorkerID → Worker
   Attributes: ContractorLicenseNo, RatePerHour, ContractStartDate, ContractEndDate, InsuranceProvider, ContractStatus
   Purpose: Represents external service providers.

9. **WorkOrder (Associative Entity)**
   PK: WorkOrderID

Attributes: StartDate, EndDate, Status, TimeSpentHours, MaterialsUsed, ResolutionNotes
Purpose: Tracks the execution of maintenance tasks linked to requests.

10. **Invoice (Weak Entity)**
PK: InvoiceID
Attributes: LaborCost, MaterialCost, TotalCost, TaxAmount, PaymentStatus, PaymentDate
Purpose: Captures financial details of completed work.

11. **PropertyManager (Strong Entity)**
PK: ManagerID
Attributes: Name, Phone, Email, Office
Purpose: Oversees property operations and escalations.

12. **Escalation (Weak Entity)**
PK: EscalationID
Attributes: EscalationDate, EscalationReason, ResolutionStatus, Notes
Purpose: Records complaints escalated due to delays or issues.

13. **Notification (Weak Entity)**
PK: NotificationID
Attributes: RecipientType, RecipientID, Channel, Message, DateSent, DeliveryStatus
Purpose: Tracks communication to residents, staff, and managers.

Relationships Overview:

- Resident — Lease
  a. One Resident ↔ Many Leases
  b. One Lease ↔ One Resident

- ApartmentUnit — Lease
  a. One ApartmentUnit ↔ Many Leases
  b. One Lease ↔ One ApartmentUnit

- Resident — Complaint/Request
  a. One Resident ↔ Many Requests
  b. One Request ↔ One Resident

- ApartmentUnit — Complaint/Request
  a. One ApartmentUnit ↔ Many Requests
  b. One Request ↔ One ApartmentUnit

- Complaint/Request — WorkOrder
  a. One Request ↔ Many WorkOrders
  b. One WorkOrder ↔ One Request

- WorkOrder — Worker
  a. One Worker ↔ Many WorkOrders
  b. One WorkOrder ↔ One Worker

- Worker — MaintenanceStaff (Subtype)
  a. One Worker ↔ Zero or One MaintenanceStaff
  b. One MaintenanceStaff ↔ One Worker

- Worker — Contractor (Subtype)
  a. One Worker ↔ Zero or One Contractor
  b. One Contractor ↔ One Worker

- Complaint/Request — Escalation
  a. One Request ↔ Zero or One Escalation
  b. One Escalation ↔ One Request

- Escalation — PropertyManager
  a. One PropertyManager ↔ Many Escalations
  b. One Escalation ↔ One PropertyManager

- Complaint/Request — Notification
  a. One Request ↔ Many Notifications
  b. One Notification ↔ One Request

- RequestCategory — Complaint/Request
  a. One Category ↔ Many Requests
  b. One Request ↔ One Category

- WorkOrder — Invoice
  a. One WorkOrder ↔ One Invoice
  b. One Invoice ↔ One WorkOrder

3. Key Database Design Decisions

- **Worker Supertype with Subtypes:** Allows shared attributes and avoids redundancy.
- **Lease as Associative Entity:** Captures the history of resident occupancy and supports many-to-many mapping.
- **Request as Central Entity:** Serves as the hub for all related entities (WorkOrder, Escalation, Invoice, Notification).
- **WorkOrder Linking Requests to Workers:** Enables detailed tracking of each job assignment.
- **Weak Entities (Invoice, Escalation, Notification):** Depend on parent entities to ensure referential consistency.
- **Scalability:** Structured to support integration with a RESTful backend or web dashboard.

---

4. Diagram Description

The conceptual ERD depicts both structural and operational relationships:

- **Specialization:** Worker supertype branching into MaintenanceStaff and Contractor (disjoint, total participation).
- **Associative Entities:** Lease and WorkOrder resolve many-to-many relationships.
- **Weak Entities:** Invoice, Escalation, and Notification depend on Request or WorkOrder.
- **One-to-many and one-to-one mappings:** Simplify reporting and allow flexible extensions.

---

5. Design Rules and Integrity Constraints

- **Primary Keys:** Guarantee unique identification (e.g., ResidentID, RequestID).
- **Foreign Keys:** Enforce valid cross-entity links (e.g., RequestID in WorkOrder).
- **Domain Constraints:** Enforce valid attribute ranges (e.g., UrgencyLevel = {Low, Medium, High}).
- **Entity Integrity:** Prevents nulls in primary keys.
- **Referential Integrity:** Ensures every Request must have a valid Resident and ApartmentUnit.
- **Update and Delete Rules:** Cascade updates to maintain consistency when parent data changes.

---

6. Scalability and Future Enhancements

The design can be extended to include:

- **MaintenanceVendor** entity for third-party service providers.
- **BuildingInspection** entity for periodic safety checks.
- Integration with **IoT-based sensors** to auto-log maintenance requests.
- Addition of **AuditLogs** for detailed user activity tracking.

Conclusion:

This database model establishes a centralized, normalized, and transparent structure for managing maintenance and complaint workflows in apartment communities. It balances clarity, accountability, and scalability, ensuring operational efficiency for both residents and property managers.