Name: Pranav Raj Sowrirajan Balaji

Date: 19 October, 2024

Home Rental Management System

This system is designed to help property owners manage rental properties and tenants efficiently. It includes features for tracking property listings, tenant applications, lease agreements, payments, maintenance requests and tenant referrals. The Tenant Referral Program adds a layer of community engagement by allowing current tenants to refer friends or colleagues. If the referral leads to a lease, the referring tenant earns rewards, such as rent discounts or cash incentives. This feature encourages tenants to spread the word, improving property occupancy rates for property owners.

Rules

- A **property owner** can own multiple properties and list them on the website.
- A **property** can only be rented by one tenant at a time, but over time, it can have multiple tenants.
- A **tenant** can apply to rent multiple properties but can have only one active lease.
- Each **application** is associated with a specific tenant and property.
- A **lease agreement** connects a tenant and a property for a specific duration.
- Tenants must make **payments** for their rent, which are tracked for each lease.
- Tenants can submit **maintenance requests** for properties they rent.
- Tenants can refer other potential tenants and earn rewards if the referral leads to a lease.

Nouns & Verbs:

Entities

1. **Property Owner**: Represents a person or company that owns rental properties.

   Attributes:

   - propertyowner_ID
   - Name
   - Email
   - PhoneNumber
   - Address
   - TaxID

   Verbs/actions: List

2. **Property**: Represents a rental property owned by a property owner.
    a. Attributes:
        - property_ID,
        - Address
        - City
        - State
        - ZipCode
        - PropertyType (e.g., Apartment, House)
        - NumberOfRooms
        - RentAmount
        - propertyowner_ID

Verbs/actions: List

3. **Tenant**: Represents a person renting a property.

Attributes:

- tenant_ID,
- Name
- Email
- PhoneNumber
- LeaseStartDate
- LeaseEndDate
- property_id

Verbs: Apply, Rent, submit

4. Application: Represents an application submitted by a tenant to rent a property. (Class serving as medium to split many to many relation from property - tenant UML diagram to the ERD diagram)

Attributes:

- application_ID,
- tenant_ID
- property_ID
- ApplicationDate,
- ApplicationStatus (e.g., Pending, Approved, Rejected).

Verbs: Apply

5. Lease Agreement: Represents the rental agreement between a property owner and a tenant.

Attributes:

- lease_ID
- property_ID
- tenant_ID
- LeaseStartDate
- LeaseEndDate
- RentAmount
- SecurityDepositAmount.

Verbs: connects, active

6. Payment: Represents a payment made by a tenant for rent.

Attributes:

- payment_ID
- tenant_ID
- lease_ID
- PaymentDate
- Amount
- PaymentStatus (e.g., Paid, Late)

7. Maintenance Request: Represents a maintenance request made by a tenant.

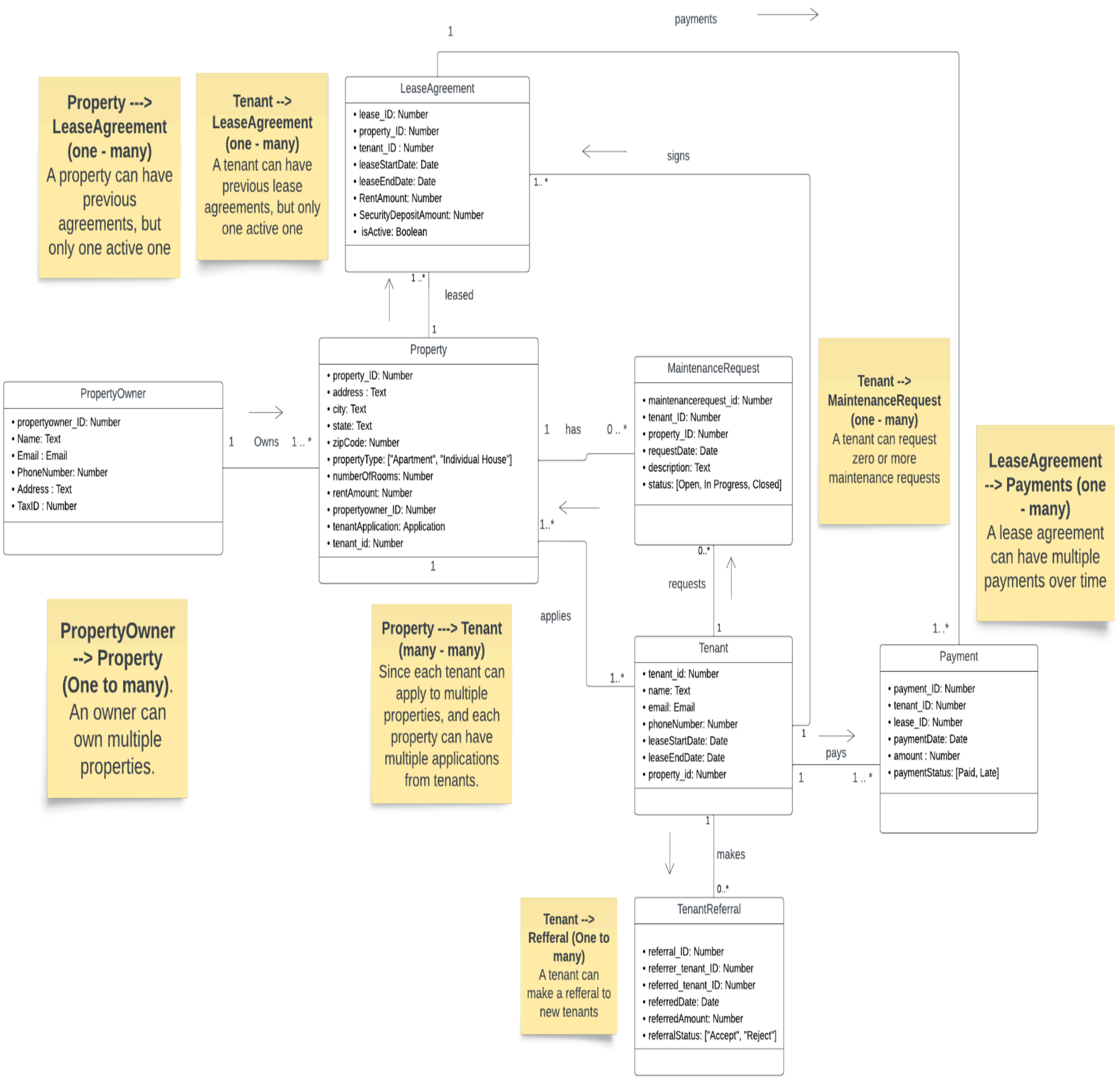
Attributes:

- request_ID
- tenant_ID
- property_ID
- RequestDate
- Description
- Status (e.g., Open, In Progress, Completed)

8. **TenantReferral**: Tenant can give referrals to potential clients / tenants.

- referral_ID
- referrer_tenant_ID
- referred_tenant_ID
- ReferralDate
- RewardAmount
- ReferralStatus (Accept, Reject)

Conceptual Model (UML):



Property ---> LeaseAgreement (one - many)
A property can have previous agreements, but only one active one

Tenant --> LeaseAgreement (one - many)
A tenant can have previous lease agreements, but only one active one

Tenant --> MaintenanceRequest (one - many)
A tenant can request zero or more maintenance requests

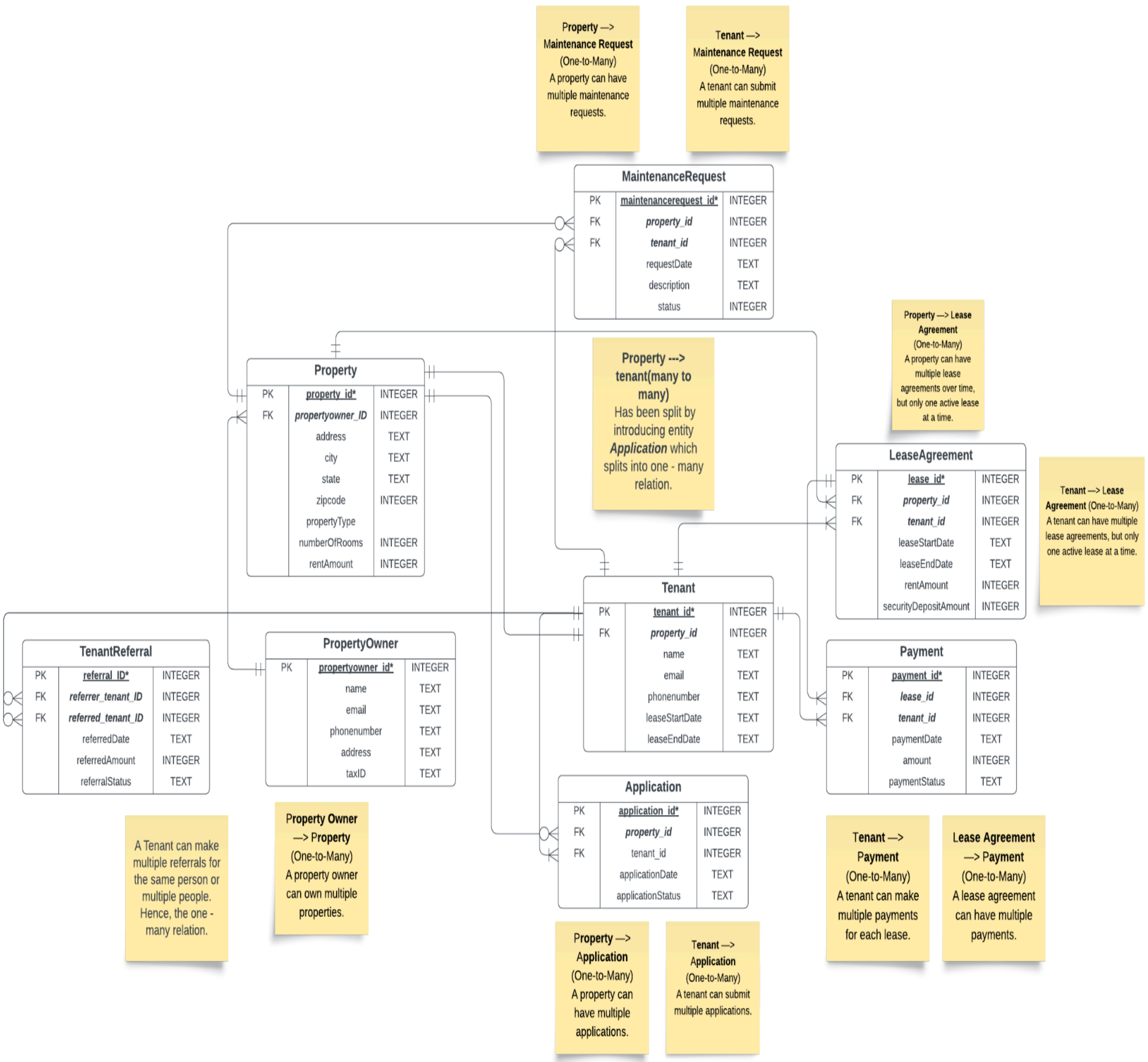
LeaseAgreement --> Payments (one - many)
A lease agreement can have multiple payments over time

PropertyOwner --> Property (One to many).
An owner can own multiple properties.

Property ---> Tenant (many - many)
Since each tenant can apply to multiple properties, and each property can have multiple applications from tenants.

Tenant --> Refferal (One to many)
A tenant can make a refferal to new tenants

ERD Crow's Foot Diagram:



Property —> Maintenance Request (One-to-Many) A property can have multiple maintenance requests.

Tenant —> Maintenance Request (One-to-Many) A tenant can submit multiple maintenance requests.

MaintenanceRequest		
PK	maintenancerequest_id*	INTEGER
FK	*property_id*	INTEGER
FK	*tenant_id*	INTEGER
	requestDate	TEXT
	description	TEXT
	status	INTEGER

Property —> Lease Agreement (One-to-Many) A property can have multiple lease agreements over time, but only one active lease at a time.

Property		
PK	property_id*	INTEGER
FK	*propertyowner_ID*	INTEGER
	address	TEXT
	city	TEXT
	state	TEXT
	zipcode	INTEGER
	propertyType	
	numberOfRooms	INTEGER
	rentAmount	INTEGER

Property ---> tenant(many to many) Has been split by introducing entity *Application* which splits into one - many relation.

LeaseAgreement		
PK	lease_id*	INTEGER
FK	*property_id*	INTEGER
FK	*tenant_id*	INTEGER
	leaseStartDate	TEXT
	leaseEndDate	TEXT
	rentAmount	INTEGER
	securityDepositAmount	INTEGER

Tenant —> Lease Agreement (One-to-Many) A tenant can have multiple lease agreements, but only one active lease at a time.

Tenant		
PK	tenant_id*	INTEGER
FK	*property_id*	INTEGER
	name	TEXT
	email	TEXT
	phonenumber	TEXT
	leaseStartDate	TEXT
	leaseEndDate	TEXT

TenantReferral		
PK	referral_ID*	INTEGER
FK	*referrer_tenant_ID*	INTEGER
FK	*referred_tenant_ID*	INTEGER
	referredDate	TEXT
	referredAmount	INTEGER
	referralStatus	TEXT

PropertyOwner		
PK	propertyowner_id*	INTEGER
	name	TEXT
	email	TEXT
	phonenumber	TEXT
	address	TEXT
	taxID	TEXT

Payment		
PK	payment_id*	INTEGER
FK	*lease_id*	INTEGER
FK	*tenant_id*	INTEGER
	paymentDate	TEXT
	amount	INTEGER
	paymentStatus	TEXT

Application		
PK	application_id*	INTEGER
FK	*property_id*	INTEGER
FK	tenant_id	INTEGER
	applicationDate	TEXT
	applicationStatus	TEXT

A Tenant can make multiple referrals for the same person or multiple people. Hence, the one - many relation.

Property Owner —> Property (One-to-Many) A property owner can own multiple properties.

Tenant —> Payment (One-to-Many) A tenant can make multiple payments for each lease.

Lease Agreement —> Payment (One-to-Many) A lease agreement can have multiple payments.

Property —> Application (One-to-Many) A property can have multiple applications.

Tenant —> Application (One-to-Many) A tenant can submit multiple applications.

Relational Schema BCNF:

- **PropertyOwner**(**propertyowner_ID*** PK, Name, Email, PhoneNumber, Address, TaxID)
- **Property**(**property_ID*** PK, Address, City, State, ZipCode, PropertyType, NumberOfRooms, RentAmount, ***propertyowner_ID*** FK)
- **Tenant**(**tenant_ID*** PK, Name, Email, PhoneNumber)
- **Application**(**application_ID*** PK, ***tenant_ID*** FK, ***property_ID*** FK, ApplicationDate, ApplicationStatus)
- **LeaseAgreement**(**lease_ID*** PK, ***tenant_ID*** FK, ***property_ID*** FK, LeaseStartDate, LeaseEndDate, RentAmount, SecurityDepositAmount)
- **Payment**(**payment_ID*** PK, ***tenant_ID*** FK, ***lease_ID*** FK, PaymentDate, Amount, PaymentStatus)
- **MaintenanceRequest**(**maintenancerequest_ID*** PK, ***tenant_ID*** FK, ***property_ID*** FK, RequestDate, Description, Status)
- **TenantReferral**(**referral_ID*** PK, ***referrer_tenant_ID*** FK, ***referred_tenant_ID*** FK, ReferralDate, RewardAmount, ReferralStatus)

Proof for BCNF:

PropertyOwner:

FD: propertyowner_ID → Name, Email, PhoneNumber, Address, TaxID

propertyowner_ID is a superkey, so this is in BCNF.

Property:

FD: property_ID → Address, City, State, ZipCode, PropertyType, NumberOfRooms, RentAmount, propertyowner_ID

property_ID is a superkey, so this is in BCNF.

Tenant:

FD: tenant_ID → Name, Email, PhoneNumber

tenant_ID is a superkey, so this is in BCNF.

Application:

FD: application_ID → tenant_ID, property_ID, ApplicationDate, ApplicationStatus

application_ID is a superkey, so this is in BCNF.

Lease Agreement:

FD: lease_ID → tenant_ID, property_ID, LeaseStartDate, LeaseEndDate, RentAmount, SecurityDepositAmount

lease_ID is a superkey, so this is in BCNF.

Payment:

FD: payment_ID → tenant_ID, lease_ID, PaymentDate, Amount, PaymentStatus

payment_ID is a superkey, so this is in BCNF.

Maintenance Request:

FD: maintenancerequest_ID → tenant_ID, property_ID, RequestDate, Description, Status

maintenancerequest_ID is a superkey, so this is in BCNF.

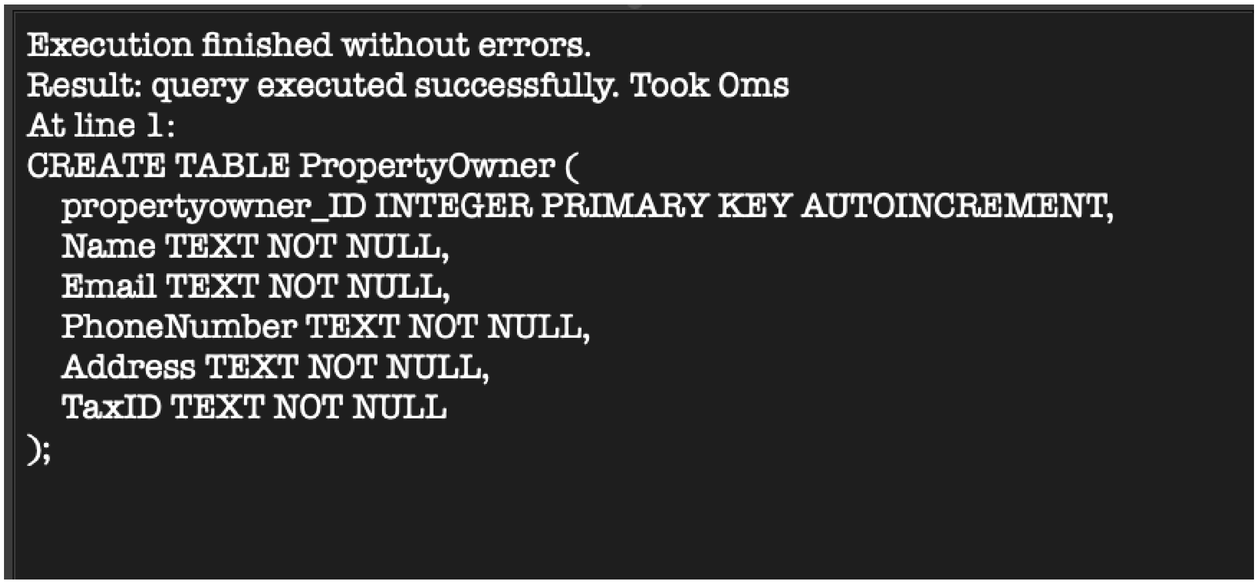
TenantReferral:

FD: referral_ID → referrer_tenant_ID, referred_tenant_ID, ReferralDate, RewardAmount, ReferralStatus

referral_ID is a superkey, so this is in BCNF.

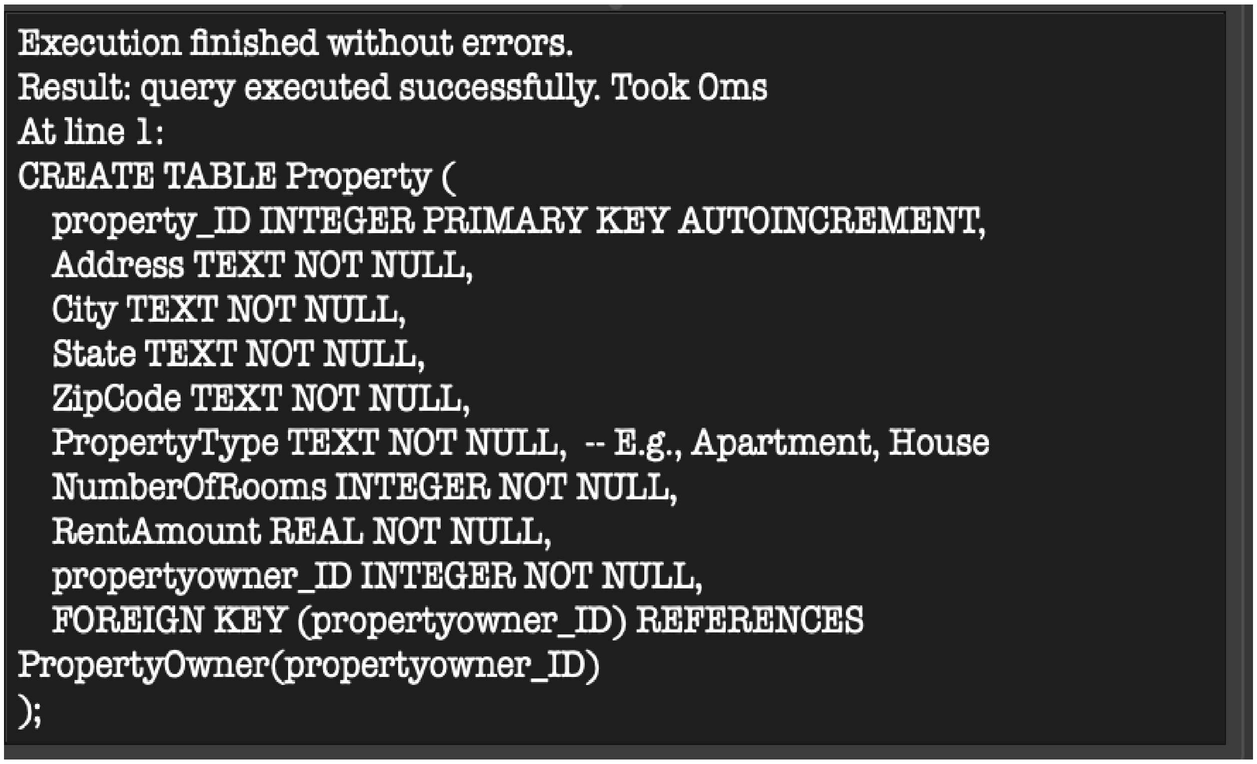
Creating Tables using SQLite3 & DB Browser:

Table 1: PropertyOwner



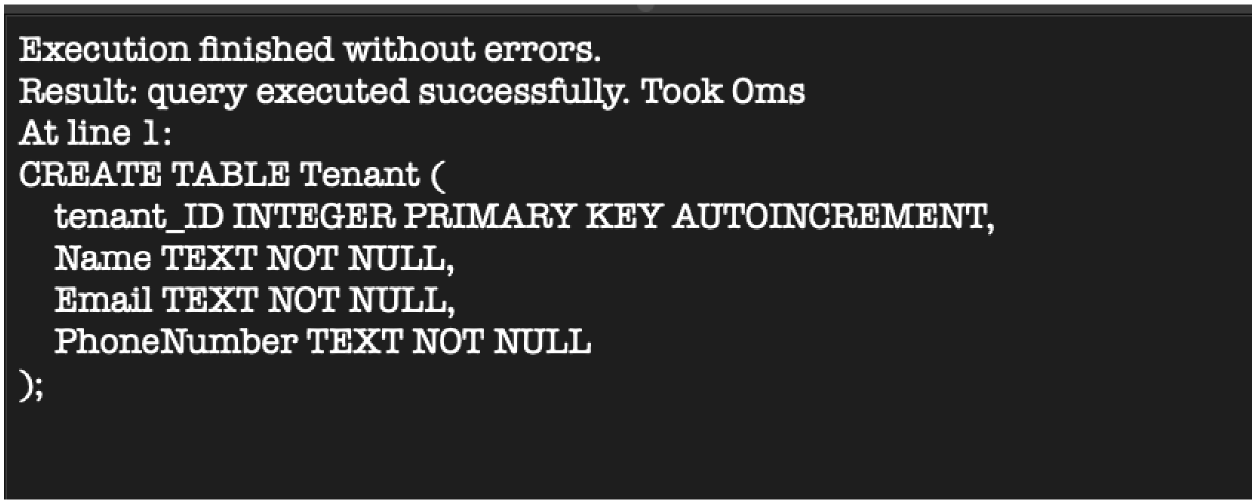
```
Execution finished without errors.
Result: query executed successfully. Took 0ms
At line 1:
CREATE TABLE PropertyOwner (
   propertyowner_ID INTEGER PRIMARY KEY AUTOINCREMENT,
   Name TEXT NOT NULL,
   Email TEXT NOT NULL,
   PhoneNumber TEXT NOT NULL,
   Address TEXT NOT NULL,
   TaxID TEXT NOT NULL
);
```

Table 2: Property



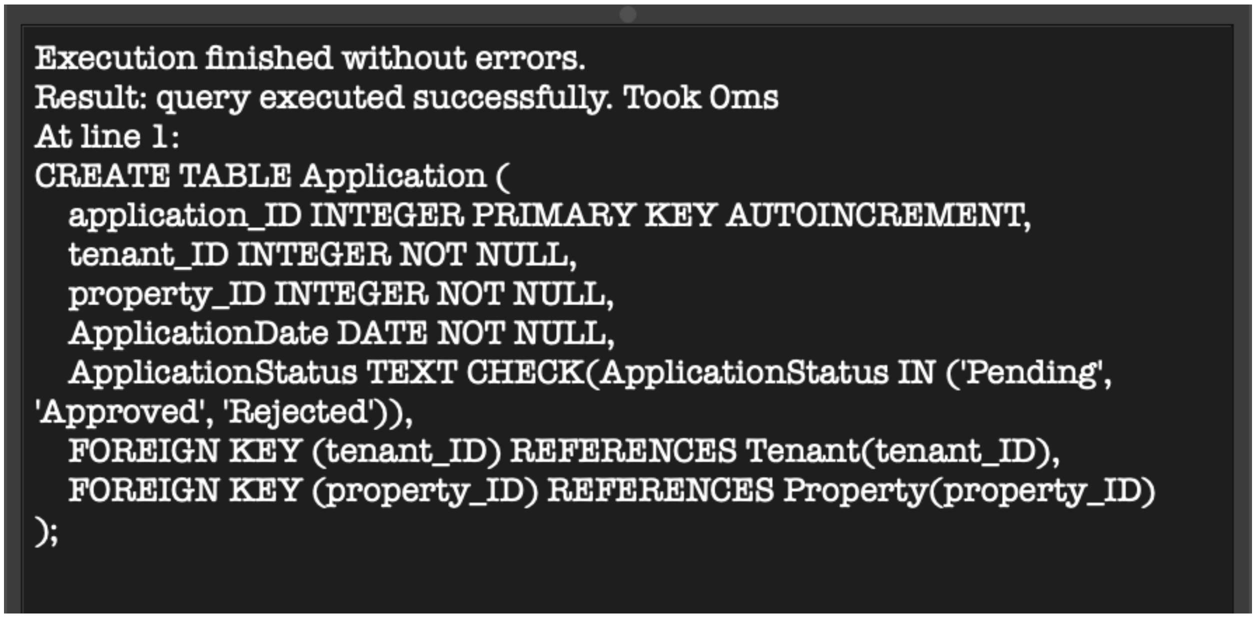
```
Execution finished without errors.
Result: query executed successfully. Took 0ms
At line 1:
CREATE TABLE Property (
   property_ID INTEGER PRIMARY KEY AUTOINCREMENT,
   Address TEXT NOT NULL,
   City TEXT NOT NULL,
   State TEXT NOT NULL,
   ZipCode TEXT NOT NULL,
   PropertyType TEXT NOT NULL,  -- E.g., Apartment, House
   NumberOfRooms INTEGER NOT NULL,
   RentAmount REAL NOT NULL,
   propertyowner_ID INTEGER NOT NULL,
   FOREIGN KEY (propertyowner_ID) REFERENCES
PropertyOwner(propertyowner_ID)
);
```

Table 3: Tenant



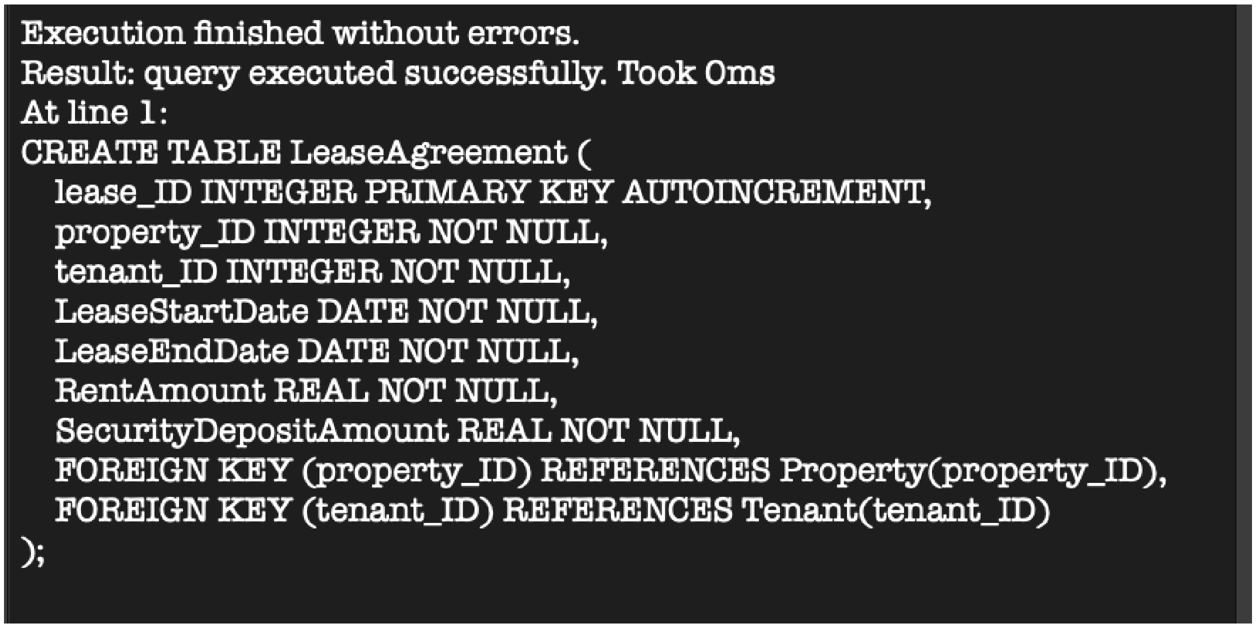
```
Execution finished without errors.
Result: query executed successfully. Took 0ms
At line 1:
CREATE TABLE Tenant (
   tenant_ID INTEGER PRIMARY KEY AUTOINCREMENT,
   Name TEXT NOT NULL,
   Email TEXT NOT NULL,
   PhoneNumber TEXT NOT NULL
);
```

Table 4: Application



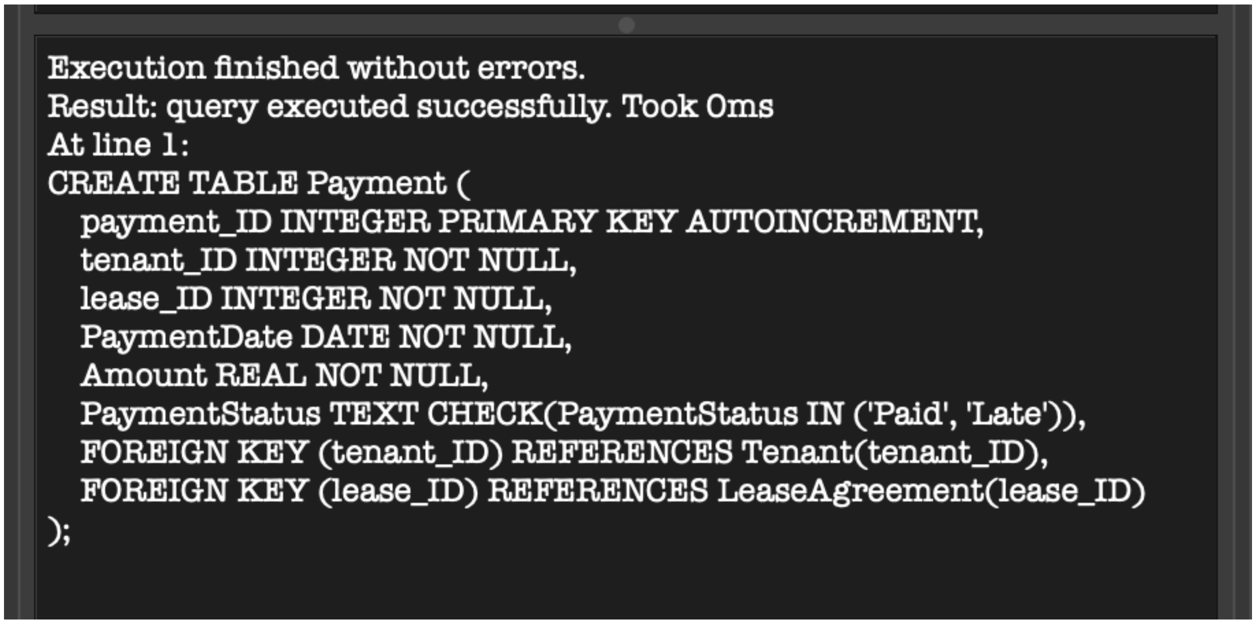
```
Execution finished without errors.
Result: query executed successfully. Took 0ms
At line 1:
CREATE TABLE Application (
   application_ID INTEGER PRIMARY KEY AUTOINCREMENT,
   tenant_ID INTEGER NOT NULL,
   property_ID INTEGER NOT NULL,
   ApplicationDate DATE NOT NULL,
   ApplicationStatus TEXT CHECK(ApplicationStatus IN ('Pending',
'Approved', 'Rejected')),
   FOREIGN KEY (tenant_ID) REFERENCES Tenant(tenant_ID),
   FOREIGN KEY (property_ID) REFERENCES Property(property_ID)
);
```

Table 5: Lease Agreement



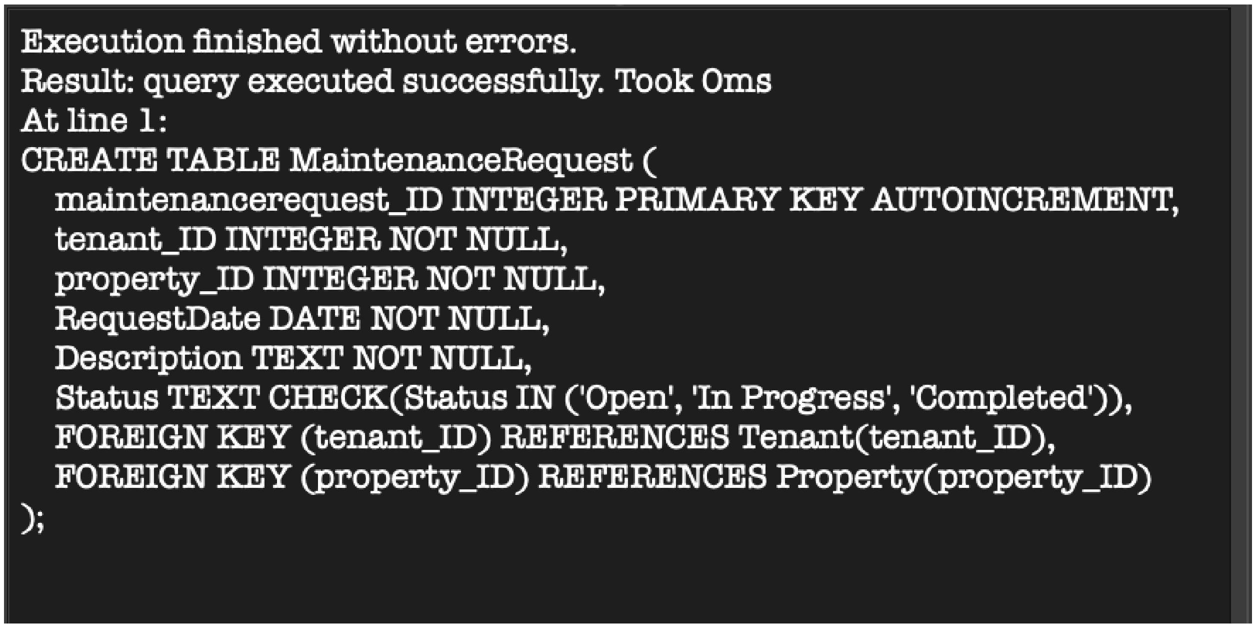
```
Execution finished without errors.
Result: query executed successfully. Took 0ms
At line 1:
CREATE TABLE LeaseAgreement (
   lease_ID INTEGER PRIMARY KEY AUTOINCREMENT,
   property_ID INTEGER NOT NULL,
   tenant_ID INTEGER NOT NULL,
   LeaseStartDate DATE NOT NULL,
   LeaseEndDate DATE NOT NULL,
   RentAmount REAL NOT NULL,
   SecurityDepositAmount REAL NOT NULL,
   FOREIGN KEY (property_ID) REFERENCES Property(property_ID),
   FOREIGN KEY (tenant_ID) REFERENCES Tenant(tenant_ID)
);
```

Table 6: Payment



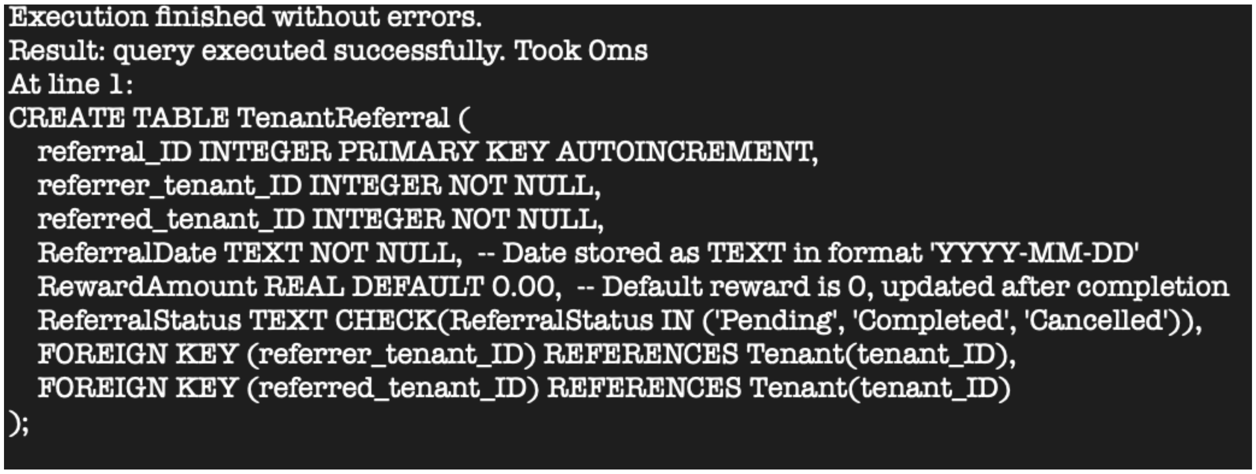
```
Execution finished without errors.
Result: query executed successfully. Took 0ms
At line 1:
CREATE TABLE Payment (
   payment_ID INTEGER PRIMARY KEY AUTOINCREMENT,
   tenant_ID INTEGER NOT NULL,
   lease_ID INTEGER NOT NULL,
   PaymentDate DATE NOT NULL,
   Amount REAL NOT NULL,
   PaymentStatus TEXT CHECK(PaymentStatus IN ('Paid', 'Late')),
   FOREIGN KEY (tenant_ID) REFERENCES Tenant(tenant_ID),
   FOREIGN KEY (lease_ID) REFERENCES LeaseAgreement(lease_ID)
);
```

Table 7: MaintenanceRequest:



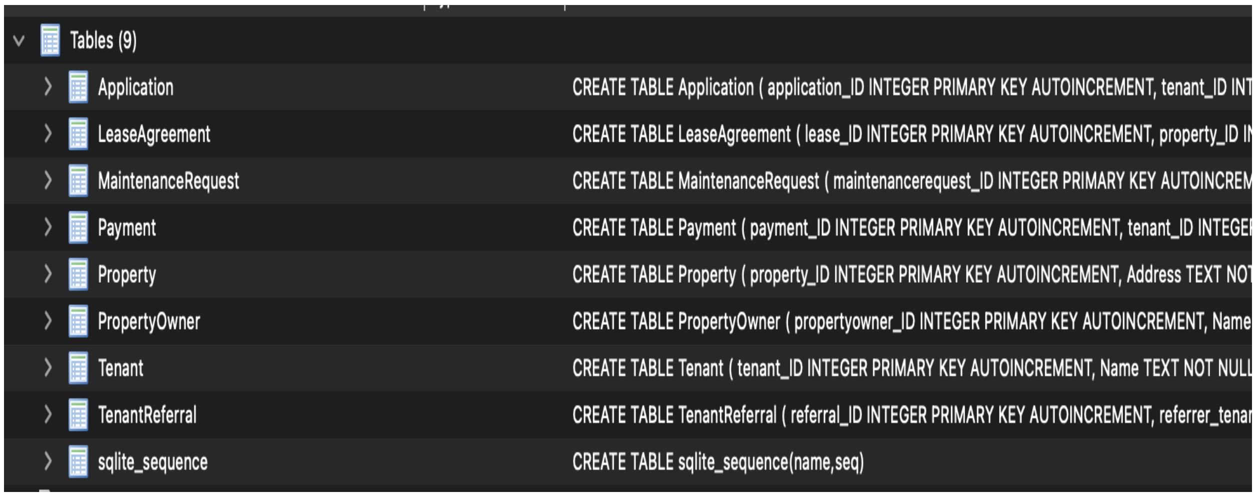
```
Execution finished without errors.
Result: query executed successfully. Took 0ms
At line 1:
CREATE TABLE MaintenanceRequest (
   maintenancerequest_ID INTEGER PRIMARY KEY AUTOINCREMENT,
   tenant_ID INTEGER NOT NULL,
   property_ID INTEGER NOT NULL,
   RequestDate DATE NOT NULL,
   Description TEXT NOT NULL,
   Status TEXT CHECK(Status IN ('Open', 'In Progress', 'Completed')),
   FOREIGN KEY (tenant_ID) REFERENCES Tenant(tenant_ID),
   FOREIGN KEY (property_ID) REFERENCES Property(property_ID)
);
```

Table 8: TenantReferral:



```
Execution finished without errors.
Result: query executed successfully. Took 0ms
At line 1:
CREATE TABLE TenantReferral (
   referral_ID INTEGER PRIMARY KEY AUTOINCREMENT,
   referrer_tenant_ID INTEGER NOT NULL,
   referred_tenant_ID INTEGER NOT NULL,
   ReferralDate TEXT NOT NULL,  -- Date stored as TEXT in format 'YYYY-MM-DD'
   RewardAmount REAL DEFAULT 0.00,  -- Default reward is 0, updated after completion
   ReferralStatus TEXT CHECK(ReferralStatus IN ('Pending', 'Completed', 'Cancelled')),
   FOREIGN KEY (referrer_tenant_ID) REFERENCES Tenant(tenant_ID),
   FOREIGN KEY (referred_tenant_ID) REFERENCES Tenant(tenant_ID)
);
```

Final DB Structure:



```
Tables (9)
  Application          CREATE TABLE Application ( application_ID INTEGER PRIMARY KEY AUTOINCREMENT, tenant_ID INT
  LeaseAgreement       CREATE TABLE LeaseAgreement ( lease_ID INTEGER PRIMARY KEY AUTOINCREMENT, property_ID IN
  MaintenanceRequest   CREATE TABLE MaintenanceRequest ( maintenancerequest_ID INTEGER PRIMARY KEY AUTOINCREM
  Payment              CREATE TABLE Payment ( payment_ID INTEGER PRIMARY KEY AUTOINCREMENT, tenant_ID INTEGE
  Property             CREATE TABLE Property ( property_ID INTEGER PRIMARY KEY AUTOINCREMENT, Address TEXT NOT
  PropertyOwner        CREATE TABLE PropertyOwner ( propertyowner_ID INTEGER PRIMARY KEY AUTOINCREMENT, Name
  Tenant               CREATE TABLE Tenant ( tenant_ID INTEGER PRIMARY KEY AUTOINCREMENT, Name TEXT NOT NULL
  TenantReferral       CREATE TABLE TenantReferral ( referral_ID INTEGER PRIMARY KEY AUTOINCREMENT, referrer_tena
  sqlite_sequence      CREATE TABLE sqlite_sequence(name,seq)
```

The database **“RentalDB”** has also been populated with data, you can find and download the database by using this Github Link: https://github.com/Pranav2501/databases-RentalDB

1. **Join of Three Tables: Query 1** demonstrates joining the Tenant, LeaseAgreement, and Property tables to get a complete view of tenant leases.
2. **Subquery: Query 2** uses a subquery to find tenants with pending applications.
3. **Group By with HAVING Clause: Query 3** groups property owners and filters based on ownership of 3 or more properties.
4. **Complex Search Criteria: Query 4** uses multiple logical conditions to find tenants in specific cities, renting houses and paying high rent.
5. **Advanced Query with PARTITION BY and CASE/WHEN: Query 5** uses advanced SQL features to rank tenants based on their payments and categorize them into High Payers or Low Payers.

More details on the query can be found in the **README** file present in the Github Link provided above.