Detailed Case Study

Viventra Property Management

“Where smart living meets seamless management”

Presented By : Karan Acharya

Linkdin
GitHub

1. Introduction

1.1 Industry Background

The real estate industry plays a critical role in the global economy, with residential property rentals constituting a significant portion of its operations. As urban populations continue to rise and housing demands increase, property management has become increasingly complex. Managing lease agreements, collecting timely payments, addressing tenant concerns, and handling maintenance requests are essential functions that require precision, transparency, and efficiency.

Traditionally, these tasks were handled manually or through fragmented systems such as spreadsheets, physical documentation, and basic accounting tools. This approach often led to communication breakdowns, missed payments, lost records, and delayed maintenance—resulting in tenant dissatisfaction and operational inefficiencies.

In recent years, the sector has experienced a digital transformation, with a growing emphasis on integrating technology to automate and streamline these processes. Property managers and landlords are now looking for smart, scalable solutions that reduce administrative burdens, enhance tenant experience, and ensure accurate recordkeeping.

1.2 Business Problem

The traditional model of residential property management continues to rely heavily on manual processes, fragmented communication methods, and siloed data storage systems. Landlords often manage multiple properties and tenants without a unified system, resulting in organizational chaos. Critical documents such as lease agreements, payment records, and maintenance logs are scattered across spreadsheets, paper files, emails, and physical storage systems.

This fragmented approach leads to several operational issues:

- **Paperwork Overload:** Manually maintaining lease agreements, rent receipts, and service logs is time-consuming, error-prone, and difficult to track over time.
- **Ineffective Communication:** Tenants often face delays in getting responses to maintenance issues or lease inquiries due to lack of a centralized communication channel.
- **Payment Tracking Challenges:** Without automated reminders or tracking, rent payments are frequently missed or misrecorded, creating financial inconsistencies.
- **Lack of Transparency:** Tenants have limited visibility into their lease terms, payment history, or the status of their service requests, which can result in mistrust and dissatisfaction.

- **Data Disintegration:** With property, lease, and tenant data stored in different formats and locations, property managers struggle to gain a holistic view of operations or generate meaningful reports.

These inefficiencies not only increase the administrative burden on property owners and managers but also negatively impact tenant satisfaction and retention. Over time, they can lead to financial losses, unresolved maintenance issues, and legal disputes arising from mismanaged documentation or communication gaps.

2.0. Mission and Vision

2.1 Mission Statement

"A smart digital platform simplifying lease, payment, and maintenance management for tenants and landlords."

Viventra's mission revolves around streamlining the property management experience through the use of intelligent, digital technologies. The platform is designed to serve two primary user groups: landlords and tenants, each of whom faces distinct challenges in the traditional rental ecosystem.

For landlords, the platform simplifies time-consuming administrative tasks such as drafting lease agreements, tracking rent payments, and responding to maintenance issues. For tenants, it offers a transparent and user-friendly interface to view lease terms, make secure online payments, and raise maintenance requests—all from a single platform.

By centralizing these core functions, Viventra not only increases operational efficiency but also builds trust through improved communication, faster response times, and real-time access to important information. The platform aims to **eliminate paperwork, reduce human error, and enhance the overall experience** of managing or renting a property.

The mission emphasizes "smart" because Viventra goes beyond basic digitization—it uses structured data models, dynamic views, and automated query handling to deliver intelligent insights and decision-making capabilities for both users.

2.2 Vision Statement

"To become a leading digital property management solution that enhances transparency, improves communication, and reduces the administrative burden in residential rentals."

Viventra envisions a future where **technology is seamlessly integrated** into every aspect of property management, particularly in the residential rental market. The vision is not limited to being a software tool; rather, it aspires to become a **trusted industry standard**—a go-to platform known for **reliability, transparency, and user empowerment**.

Key goals embedded in this vision include:

- **Enhancing transparency:** By providing all parties access to real-time data and transaction history, the platform ensures that landlords and tenants are always on the same page. This fosters trust and minimizes conflicts.
- **Improving communication:** Viventra aims to serve as a communication hub, bridging the gap between landlords and tenants. Whether it's a lease renewal, rent reminder, or service update, the platform ensures timely and structured communication.
- **Reducing administrative burden:** One of the biggest pain points for property managers is juggling multiple responsibilities with limited tools. Viventra automates these workflows—such as generating reports, monitoring lease timelines, and managing maintenance pipelines—allowing property managers to focus on strategic decisions rather than manual tasks.

Linkdin
GitHub

3.0. System Overview:

Viventra is a property management system with key functionalities that support landlords and tenants by offering seamless digital interaction with their rental properties.

Core Database Tables:

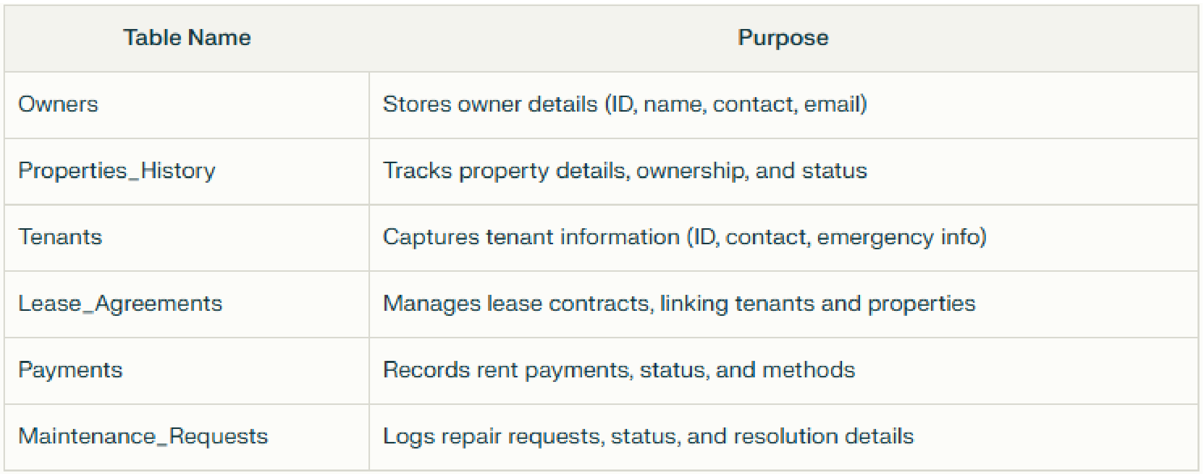
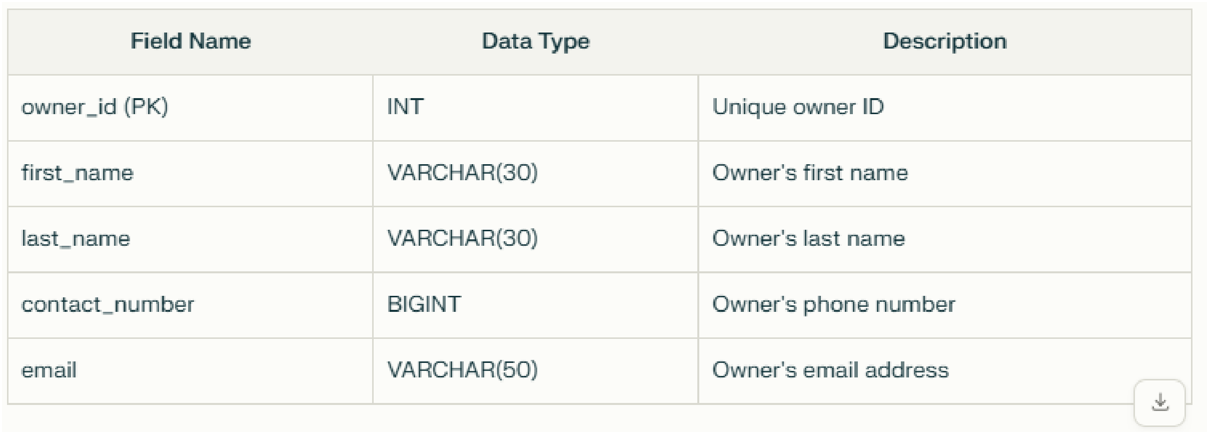


Table Name	Purpose
Owners	Stores owner details (ID, name, contact, email)
Properties_History	Tracks property details, ownership, and status
Tenants	Captures tenant information (ID, contact, emergency info)
Lease_Agreements	Manages lease contracts, linking tenants and properties
Payments	Records rent payments, status, and methods
Maintenance_Requests	Logs repair requests, status, and resolution details

These relational tables are connected through primary and foreign keys to maintain referential integrity.

3.1Final field list table

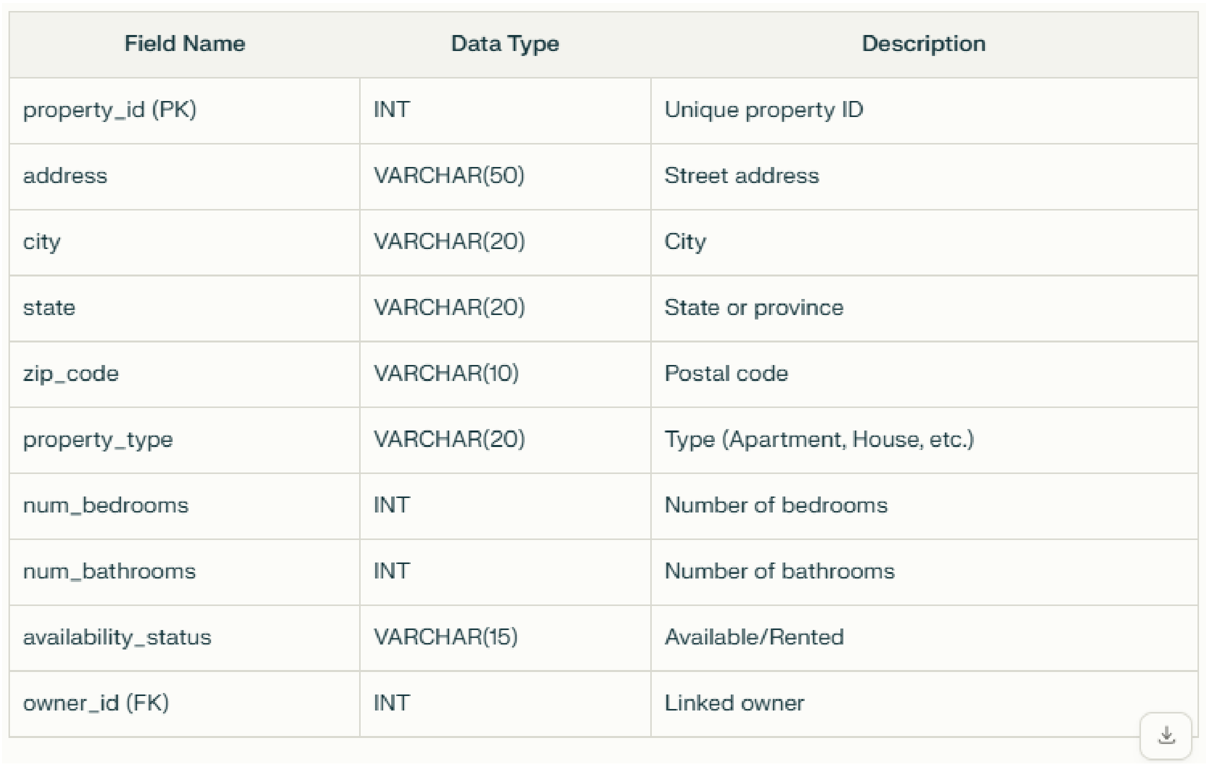
3.1.1 Owners



Field Name	Data Type	Description
owner_id (PK)	INT	Unique owner ID
first_name	VARCHAR(30)	Owner's first name
last_name	VARCHAR(30)	Owner's last name
contact_number	BIGINT	Owner's phone number
email	VARCHAR(50)	Owner's email address

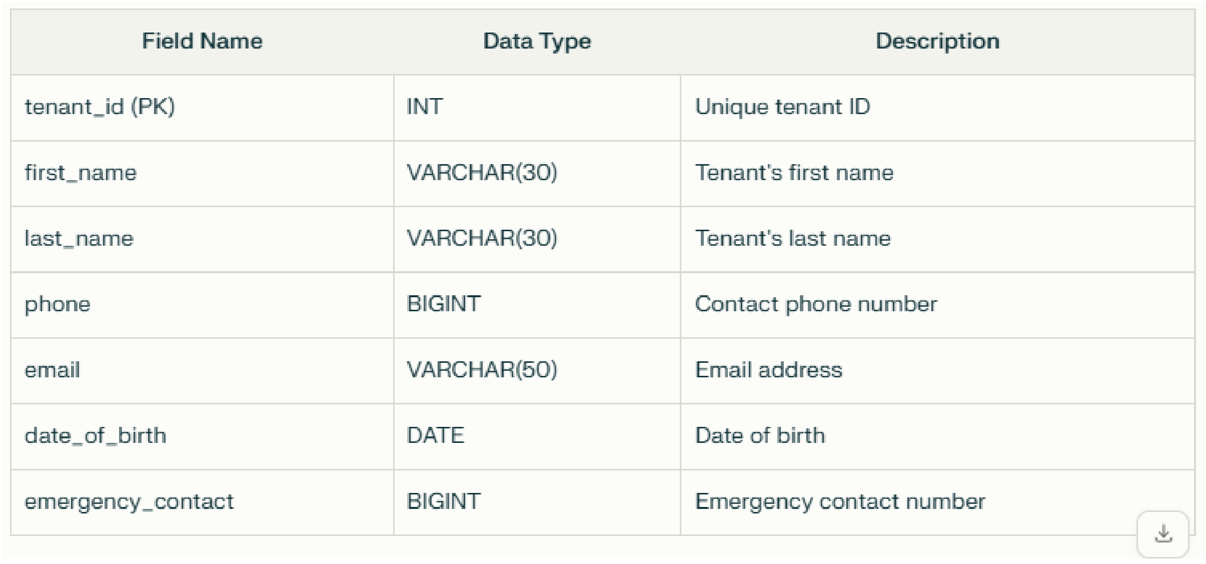
Linkdin
GitHub

3.1.2 Properties_History



Field Name	Data Type	Description
property_id (PK)	INT	Unique property ID
address	VARCHAR(50)	Street address
city	VARCHAR(20)	City
state	VARCHAR(20)	State or province
zip_code	VARCHAR(10)	Postal code
property_type	VARCHAR(20)	Type (Apartment, House, etc.)
num_bedrooms	INT	Number of bedrooms
num_bathrooms	INT	Number of bathrooms
availability_status	VARCHAR(15)	Available/Rented
owner_id (FK)	INT	Linked owner

3.1.3 Tenants



Field Name	Data Type	Description
tenant_id (PK)	INT	Unique tenant ID
first_name	VARCHAR(30)	Tenant's first name
last_name	VARCHAR(30)	Tenant's last name
phone	BIGINT	Contact phone number
email	VARCHAR(50)	Email address
date_of_birth	DATE	Date of birth
emergency_contact	BIGINT	Emergency contact number

3.1.4 Lease Agreements

Field Name	Data Type	Description
lease_id (PK)	INT	Unique lease contract ID
property_id (FK)	INT	Linked property
tenant_id (FK)	INT	Linked tenant
start_date	DATE	Lease start date
end_date	DATE	Lease end date
rent_amount	DECIMAL	Monthly rent
lease_status	VARCHAR(15)	Status (Active, Terminated, Pending)

3.1.5 Payments

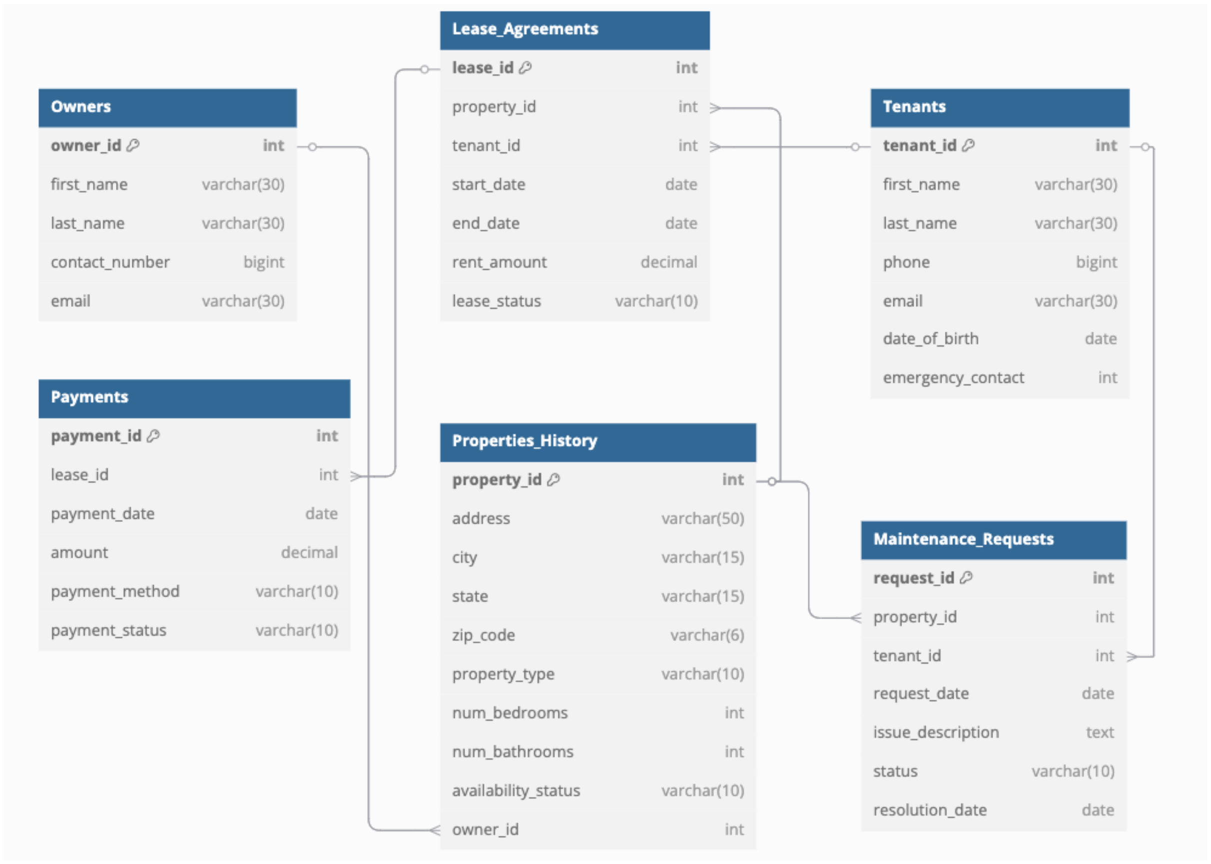
Field Name	Data Type	Description
payment_id (PK)	INT	Unique payment ID
lease_id (FK)	INT	Linked lease agreement
payment_date	DATE	Date payment was made
amount	DECIMAL	Payment amount
payment_method	VARCHAR(10)	Method (Cash, Card, EFT)
payment_status	VARCHAR(10)	Status (Paid, Late, Missed)

3.1.6. Maintenance_Requests

Field Name	Data Type	Description
request_id (PK)	INT	Unique request ID
property_id (FK)	INT	Property requiring maintenance
tenant_id (FK)	INT	Tenant who reported the issue
request_date	DATE	Date the request was made
issue_description	TEXT	Description of the issue
status	VARCHAR(15)	Request status (Pending, In Progress, Resolved)
resolution_date	DATE	Date issue was resolved

[Linkdin](#)
[GitHub](#)

4.0 Entity Relationship Diagram



4.1. Explanation of ER diagram with each table

4.1.1. Owners

The **Owners** table stores essential information about each property owner in the system. Each record is uniquely identified by the owner_id field, which serves as the primary key. The table also includes the owner's first and last names, contact number, and email address. This information is crucial for associating properties with their rightful owners and for facilitating communication regarding property management, lease agreements, and maintenance issues. The relationship between Owners and Properties_History is one-to-many, meaning a single owner can possess multiple properties, but each property is owned by only one owner.

4.1.2. Properties_History

The **Properties_History** table catalogs all properties managed within the system. Every property has a unique property_id as its primary key. The table records comprehensive details such as address, city, state, zip code, property type (like apartment or house), number of bedrooms and bathrooms, and the current availability status (e.g., available or rented). Each property is linked to an owner through the owner_id foreign key, establishing a direct relationship between properties and their owners. This table is central to the system, as it connects to lease agreements, maintenance requests, and thus acts as a hub for property-related data.

4.1.3. Tenants

The **Tenants** table contains personal and contact information for each tenant. Each tenant is uniquely identified by the tenant_id primary key. The table stores the tenant's first and last names, phone number, email address, date of birth, and an emergency contact number. This information is vital for managing lease agreements, processing maintenance requests, and ensuring clear communication with tenants. The Tenant's table is linked to both the Lease_Agreements and Maintenance_Requests tables, reflecting the tenant's involvement in property leases and maintenance activities.

4.1.4. Lease_Agreements

The **Lease_Agreements** table represents the contractual relationship between tenants and properties. Each lease agreement is uniquely identified by the lease_id primary key. The table includes foreign keys for both property_id and tenant_id, linking each lease to a specific property and tenant. It also records the lease's start and end dates, the agreed rent amount, and the current status of the lease (such as active or terminated). This table is crucial for tracking which tenant is leasing which property at any given time, the terms of their agreement, and the duration of occupancy. It also connects to the Payments table, as each lease can have multiple associated payments.

4.1.5. Payments

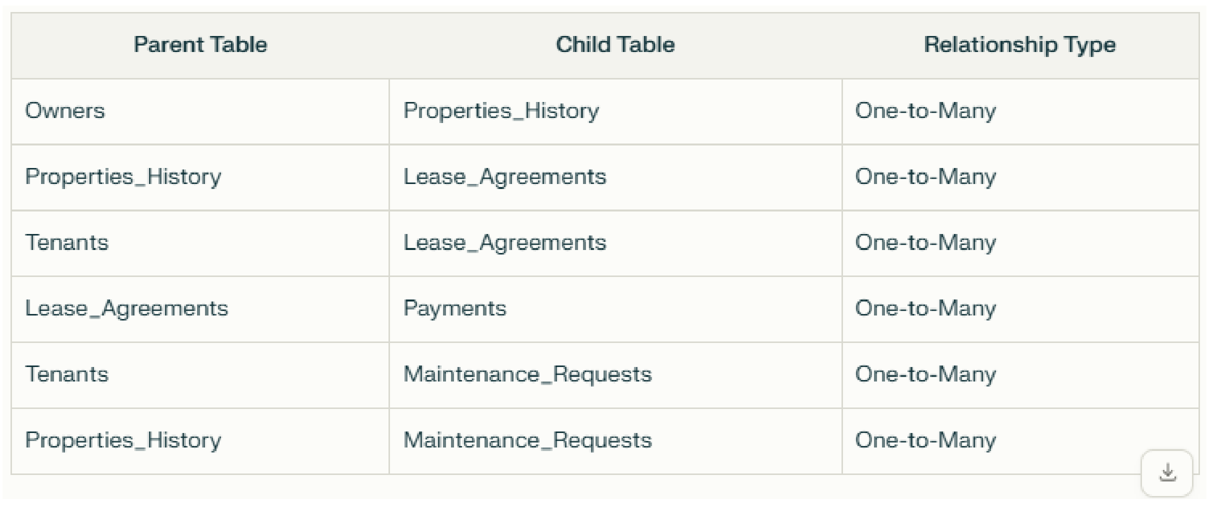
The **Payments** table tracks all financial transactions related to lease agreements. Every payment has a unique payment_id as its primary key and is associated with a specific lease agreement via the lease_id foreign key. The table records the payment date, the amount paid, the payment method (such as cash, card, or electronic transfer), and the payment status (e.g., paid, late, or missed). This structure allows the system to monitor rent payment histories, identify overdue payments, and ensure financial accountability for both tenants and property owners.

4.1.6. Maintenance_Requests

The **Maintenance_Requests** table logs all requests for property repairs and maintenance made by tenants. Each request is uniquely identified by the request_id primary key and is linked to both a property (property_id) and a tenant (tenant_id) through foreign keys. The table captures the date the request was made, a description of the issue, the current status of the request (such as pending, in progress, or resolved), and the date the issue was resolved. This table facilitates efficient management and tracking of maintenance work, ensuring timely responses to tenant concerns and proper upkeep of properties.

[Linkdin](#)
[GitHub](#)

4.2. Relationship Table for Viventra Property Management System



Parent Table	Child Table	Relationship Type
Owners	Properties_History	One-to-Many
Properties_History	Lease_Agreements	One-to-Many
Tenants	Lease_Agreements	One-to-Many
Lease_Agreements	Payments	One-to-Many
Tenants	Maintenance_Requests	One-to-Many
Properties_History	Maintenance_Requests	One-to-Many

The table outlines the key one-to-many relationships that form the backbone of the property management database. At the top level, the Owners table is related to the Properties_History table in a one-to-many fashion. This means that each owner can possess multiple properties, but each property is linked to only one owner. This relationship is essential for tracking property portfolios and ensuring that ownership information is accurately maintained and easily retrievable.

Moving forward, the Properties_History table serves as a parent to the Lease_Agreements table. This one-to-many relationship allows each property to have multiple lease agreements over time, reflecting different tenants or lease periods. For example, a property may be leased to various tenants in succession, and each of these leases is recorded as a separate entry in the Lease_Agreements table. This structure enables the system to maintain a historical record of all leasing activity associated with each property.

Similarly, the Tenants table is also a parent to the Lease_Agreements table, signifying that a single tenant can enter into multiple lease agreements, either for different properties or at different times. This is particularly useful for tracking tenants who move between properties managed by the same system or who renew leases over several years. The dual relationship of Lease_Agreements with both Properties_History and Tenants ensures that every lease is clearly linked to both the property and the tenant involved.

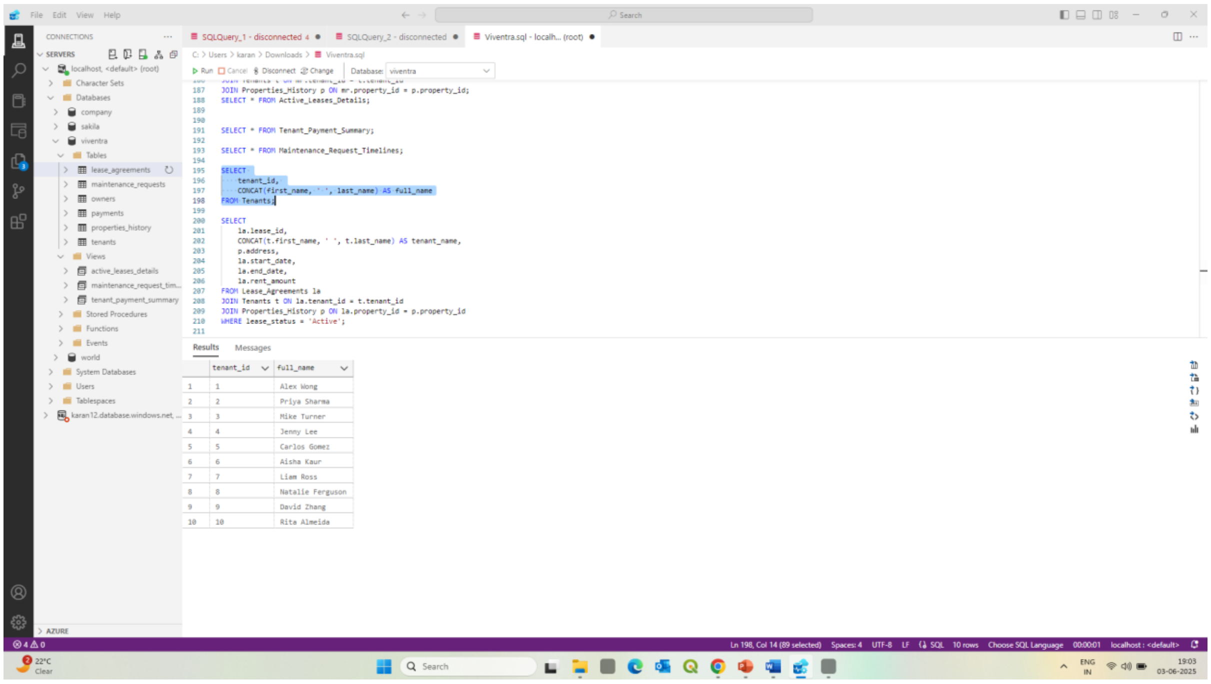
The Lease_Agreements table is connected to the Payments table in a one-to-many relationship, where each lease agreement can have multiple associated payment records. This reflects the recurring nature of rent payments, allowing the system to track monthly or periodic payments, missed payments, and payment methods for each lease. This setup is crucial for financial tracking, reporting, and ensuring accountability in the rental process.

Additionally, the Tenants and Properties_History tables both serve as parents to the Maintenance_Requests table, each in a one-to-many relationship. This means that each tenant can submit multiple maintenance requests, and each property can be the subject of multiple requests over time. By linking maintenance requests to both tenants and properties, the system enables precise tracking of who reported an issue and which property requires attention, streamlining communication and resolution of maintenance problems.

Linkdin
GitHub

5.0. SQL Database Queries:

5.1. To know the tenants full name



```
JOIN Properties_History p ON mr.property_id = p.property_id;
SELECT * FROM Active_Leases_Details;


SELECT * FROM Tenant_Payment_Summary;

SELECT * FROM Maintenance_Request_Timelines;

SELECT
    tenant_id,
    CONCAT(first_name, ' ', last_name) AS full_name
FROM Tenants;

SELECT
    la.lease_id,
    CONCAT(t.first_name, ' ', t.last_name) AS tenant_name,
    p.address,
    la.start_date,
    la.end_date,
    la.rent_amount
FROM Lease_Agreements la
JOIN Tenants t ON la.tenant_id = t.tenant_id
JOIN Properties_History p ON la.property_id = p.property_id
WHERE lease_status = 'Active';
```

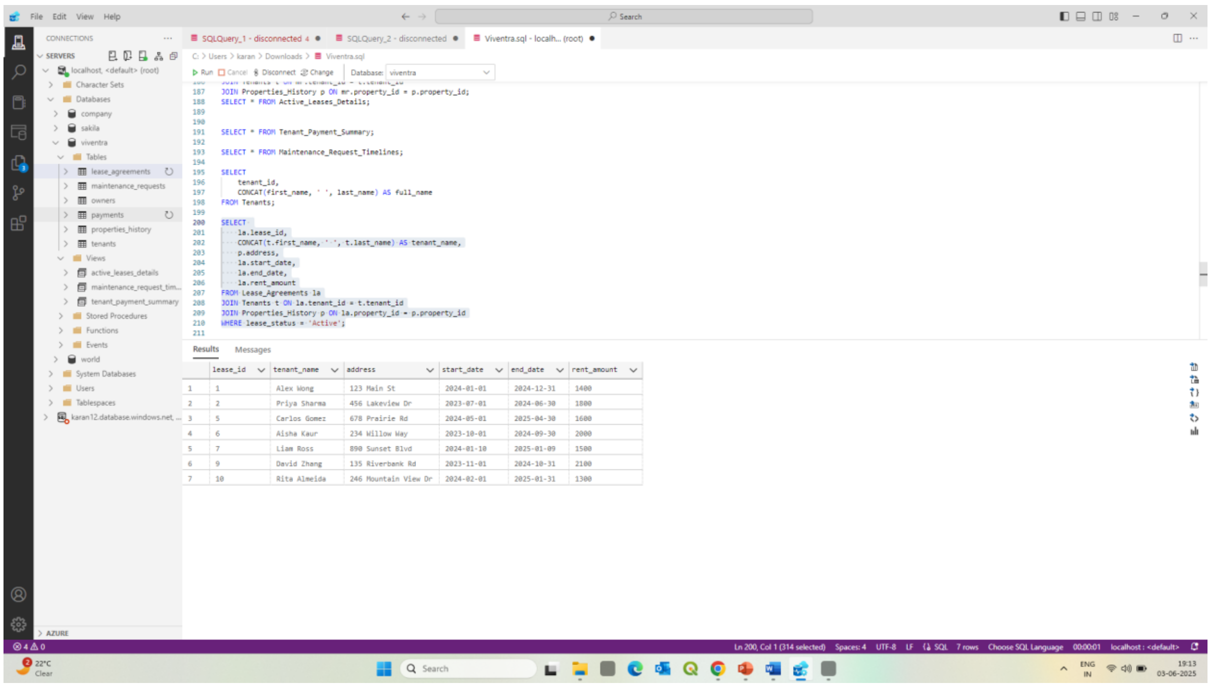
	tenant_id	full_name
1	1	Alex Wong
2	2	Priya Sharma
3	3	Mike Turner
4	4	Jenny Lee
5	5	Carlos Gomez
6	6	Aisha Kaur
7	7	Liam Ross
8	8	Natalie Ferguson
9	9	David Zhang
10	10	Rita Almeida

Query command:

```
SELECT

    tenant_id,
    CONCAT(first_name, ' ', last_name) AS full_name
FROM Tenants;
```

Linkdin
GitHub

5.2. To know the details where the lease status is still active or not



```
JOIN Properties_History p ON mr.property_id = p.property_id;
SELECT * FROM Active_Leases_Details;


SELECT * FROM Tenant_Payment_Summary;

SELECT * FROM Maintenance_Request_Timelines;

SELECT
    tenant_id,
    CONCAT(first_name, ' ', last_name) AS full_name
FROM Tenants;

SELECT
    la.lease_id,
    CONCAT(t.first_name, ' ', t.last_name) AS tenant_name,
    p.address,
    la.start_date,
    la.end_date,
    la.rent_amount
FROM Lease_Agreements la
JOIN Tenants t ON la.tenant_id = t.tenant_id
JOIN Properties_History p ON la.property_id = p.property_id
WHERE lease_status = 'Active';
```

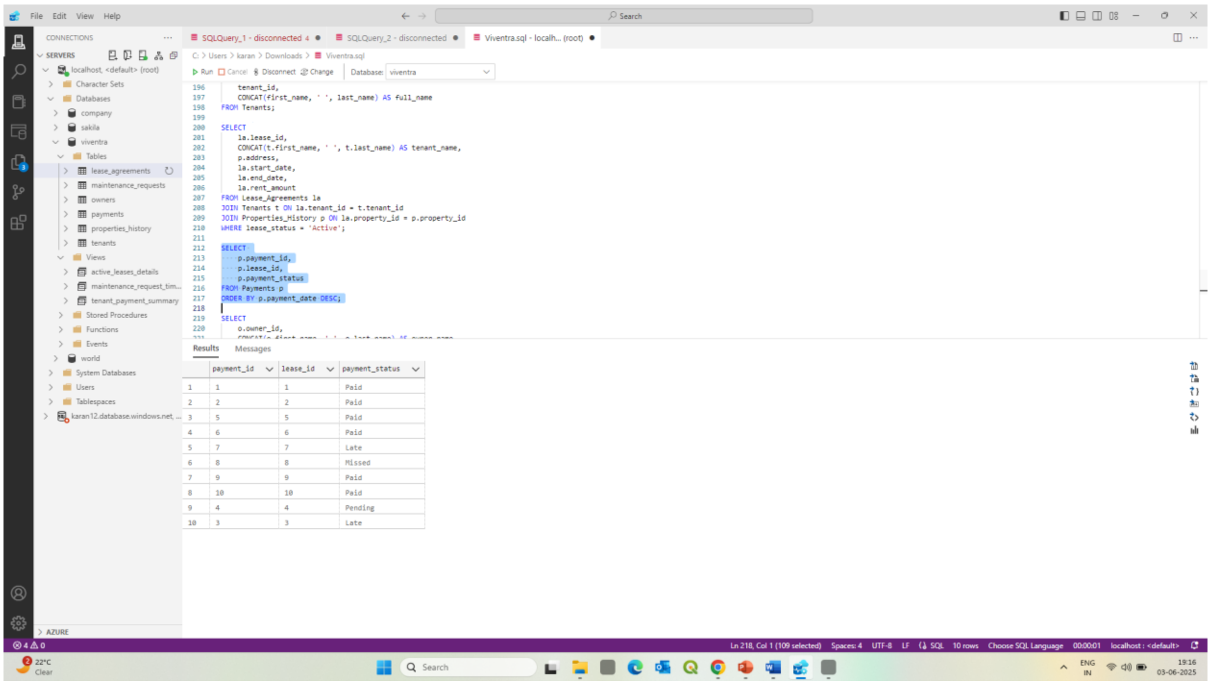
	lease_id	tenant_name	address	start_date	end_date	rent_amount
1	1	Alex Wong	123 Main St	2024-01-01	2024-12-31	1400
2	2	Priya Sharma	456 Lakeview Dr	2023-07-01	2024-06-30	1800
3	5	Carlos Gomez	678 Prairie Rd	2024-05-01	2025-04-30	1600
4	6	Aisha Kaur	234 Willow Way	2023-10-01	2024-09-30	2000
5	7	Liam Ross	890 Sunset Blvd	2024-01-10	2025-01-09	1500
6	9	David Zhang	135 Riverbank Rd	2023-11-01	2024-10-31	2100
7	10	Rita Almeida	246 Mountain View Dr	2024-02-01	2025-01-31	1300

Query Command:

```
SELECT
    la.lease_id,
    CONCAT(t.first_name, ' ', t.last_name) AS tenant_name,
    p.address,
    la.start_date,
    la.end_date,
    la.rent_amount
FROM Lease_Agreements la
JOIN Tenants t ON la.tenant_id = t.tenant_id
JOIN Properties_History p ON la.property_id = p.property_id
WHERE lease_status = 'Active';
```

Linkdin
GitHub

5.3. To know the payment status for each Lease



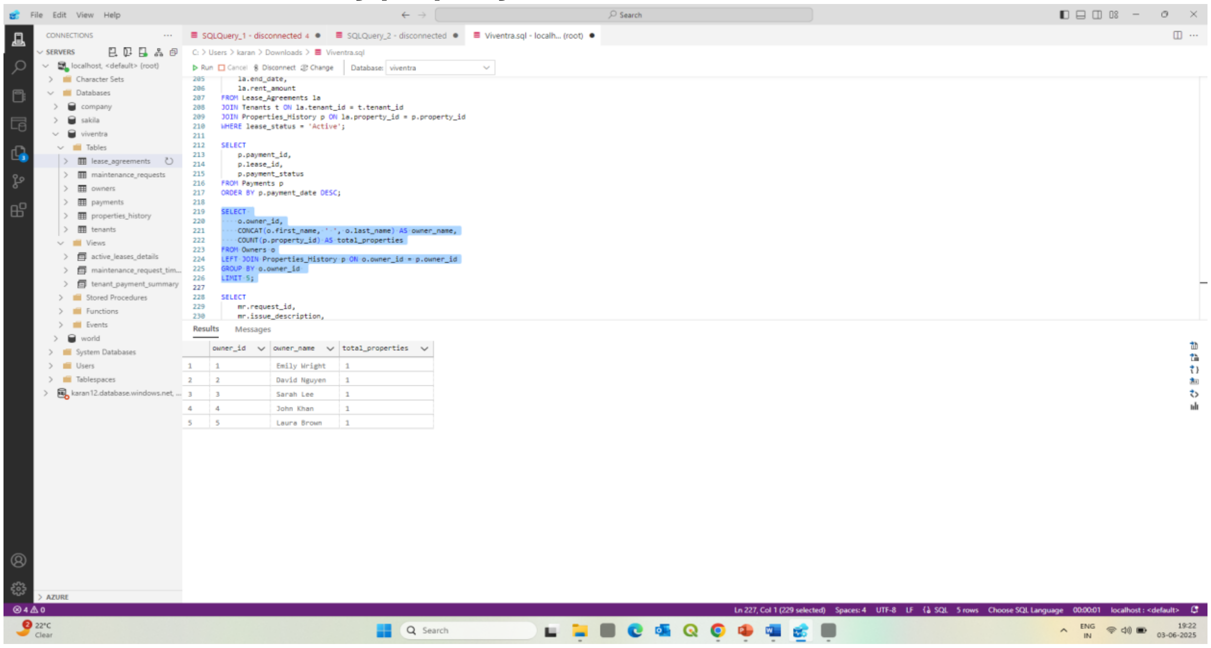
```
    tenant_id,
    CONCAT(first_name, ' ', last_name) AS full_name
FROM Tenants;

SELECT
    la.lease_id,
    CONCAT(t.first_name, ' ', t.last_name) AS tenant_name,
    p.address,
    la.start_date,
    la.end_date,
    la.rent_amount
FROM Lease_Agreements la
JOIN Tenants t ON la.tenant_id = t.tenant_id
JOIN Properties_History p ON la.property_id = p.property_id
WHERE lease_status = 'Active';

SELECT
    p.payment_id,
    p.lease_id,
    p.payment_status
FROM Payments p
ORDER BY p.payment_date DESC;

SELECT
    o.owner_id,
```

	payment_id	lease_id	payment_status
1	1	1	Paid
2	2	2	Paid
3	5	5	Paid
4	6	6	Paid
5	7	7	Late
6	8	8	Missed
7	9	9	Paid
8	10	10	Paid
9	4	4	Pending
10	3	3	Late

Query Command

```
SELECT
    p.payment_id,
    p.lease_id,
    p.payment_status
FROM Payments p
ORDER BY p.payment_date DESC;
```

5.4. To know how many property the owner owns



```
    la.end_date,
    la.rent_amount
FROM Lease_Agreements la
JOIN Tenants t ON la.tenant_id = t.tenant_id
JOIN Properties_History p ON la.property_id = p.property_id
WHERE lease_status = 'Active';

SELECT
    p.payment_id,
    p.lease_id,
    p.payment_status
FROM Payments p
ORDER BY p.payment_date DESC;

SELECT
    o.owner_id,
    CONCAT(o.first_name, ' ', o.last_name) AS owner_name,
    COUNT(p.property_id) AS total_properties
FROM Owners o
LEFT JOIN Properties_History p ON o.owner_id = p.owner_id
GROUP BY o.owner_id
LIMIT 5;

SELECT
    mr.request_id,
    mr.issue_description,
```

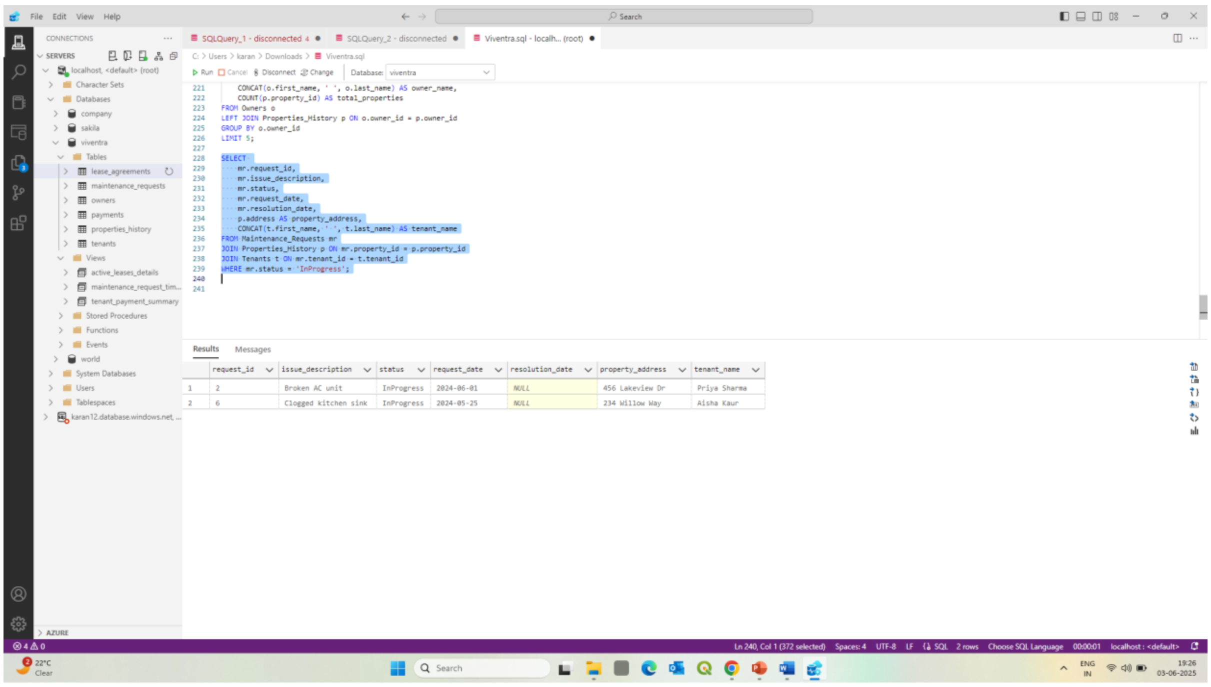
	owner_id	owner_name	total_properties
1	1	Emily Wright	1
2	2	David Nguyen	1
3	3	Sarah Lee	1
4	4	John Khan	1
5	5	Laura Brown	1

Linkdin
GitHub

SQL query command

```sql
SELECT
    o.owner_id,
    CONCAT(o.first_name, ' ', o.last_name) AS owner_name,
    COUNT(p.property_id) AS total_properties
FROM Owners o
LEFT JOIN Properties_History p ON o.owner_id = p.owner_id
GROUP BY o.owner_id
LIMIT 5;
```

5.5. To know the details where maintenance status is in progress or not



```sql
    CONCAT(o.first_name, ' ', o.last_name) AS owner_name,
    COUNT(p.property_id) AS total_properties
FROM Owners o
LEFT JOIN Properties_History p ON o.owner_id = p.owner_id
GROUP BY o.owner_id
LIMIT 5;

SELECT
    mr.request_id,
    mr.issue_description,
    mr.status,
    mr.request_date,
    mr.resolution_date,
    p.address AS property_address,
    CONCAT(t.first_name, ' ', t.last_name) AS tenant_name
FROM Maintenance_Requests mr
JOIN Properties_History p ON mr.property_id = p.property_id
JOIN Tenants t ON mr.tenant_id = t.tenant_id
WHERE mr.status = 'InProgress';
```

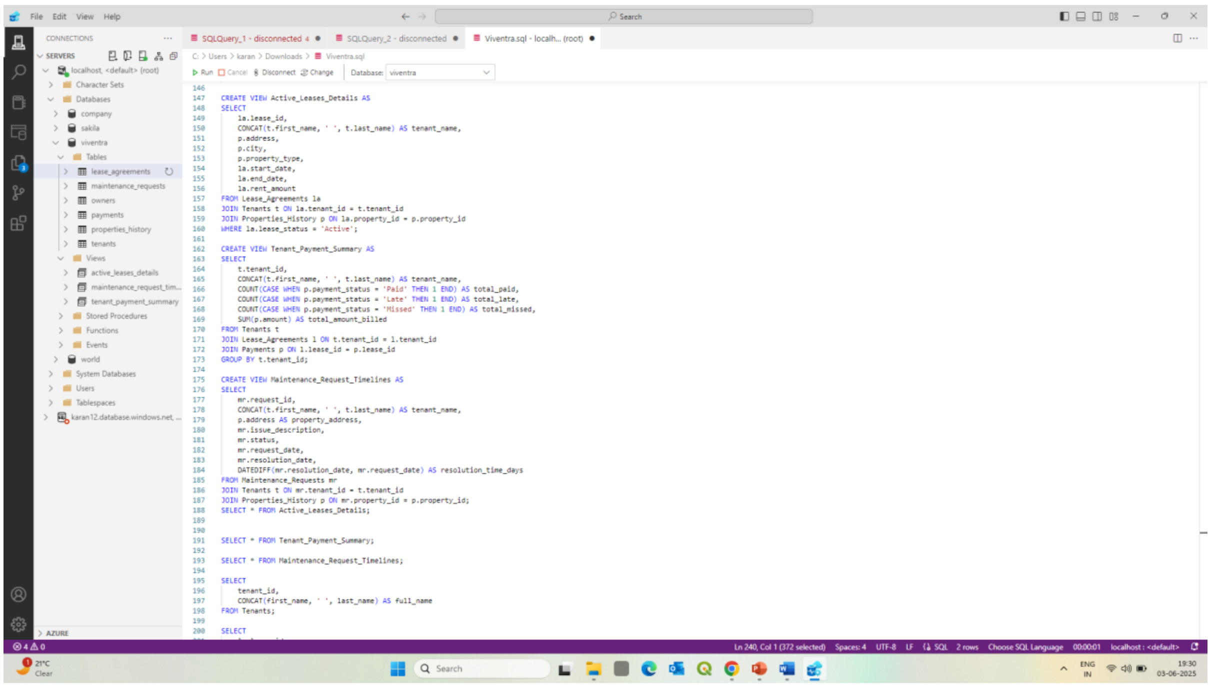
	request_id	issue_description	status	request_date	resolution_date	property_address	tenant_name
1	2	Broken AC unit	InProgress	2024-06-01	NULL	456 Lakeview Dr	Priya Sharma
2	6	Clogged kitchen sink	InProgress	2024-05-25	NULL	234 Willow Way	Aisha Kaur

Query Command:

```sql
SELECT
    mr.request_id,
    mr.issue_description,
    mr.status,
    mr.request_date,
    mr.resolution_date,
    p.address AS property_address,
    CONCAT(t.first_name, ' ', t.last_name) AS tenant_name
FROM Maintenance_Requests mr
JOIN Properties_History p ON mr.property_id = p.property_id
JOIN Tenants t ON mr.tenant_id = t.tenant_id
WHERE mr.status = 'InProgress';
```

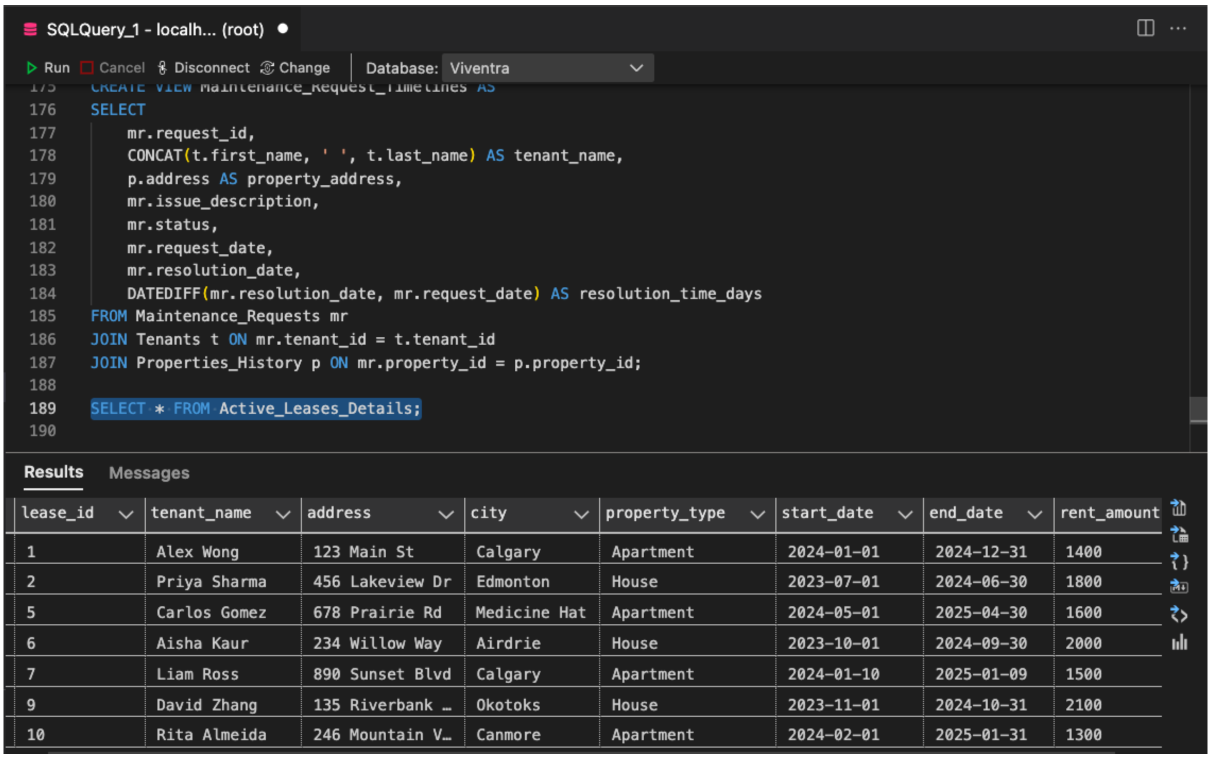
Linkdin
GitHub

6.0. SQL queries for creating Views



```sql
CREATE VIEW Active_Leases_Details AS
SELECT
    la.lease_id,
    CONCAT(t.first_name, ' ', t.last_name) AS tenant_name,
    p.address,
    p.city,
    p.property_type,
    la.start_date,
    la.end_date,
    la.rent_amount
FROM Lease_Agreements la
JOIN Tenants t ON la.tenant_id = t.tenant_id
JOIN Properties_History p ON la.property_id = p.property_id
WHERE la.lease_status = 'Active';

CREATE VIEW Tenant_Payment_Summary AS
SELECT
    t.tenant_id,
    CONCAT(t.first_name, ' ', t.last_name) AS tenant_name,
    COUNT(CASE WHEN p.payment_status = 'Paid' THEN 1 END) AS total_paid,
    COUNT(CASE WHEN p.payment_status = 'Late' THEN 1 END) AS total_late,
    COUNT(CASE WHEN p.payment_status = 'Missed' THEN 1 END) AS total_missed,
    SUM(p.amount) AS total_amount_billed
FROM Tenants t
JOIN Lease_Agreements l ON t.tenant_id = l.tenant_id
JOIN Payments p ON l.lease_id = p.lease_id
GROUP BY t.tenant_id;

CREATE VIEW Maintenance_Request_Timelines AS
SELECT
    mr.request_id,
    CONCAT(t.first_name, ' ', t.last_name) AS tenant_name,
    p.address AS property_address,
    mr.issue_description,
    mr.status,
    mr.request_date,
    mr.resolution_date,
    DATEDIFF(mr.resolution_date, mr.request_date) AS resolution_time_days
FROM Maintenance_Requests mr
JOIN Tenants t ON mr.tenant_id = t.tenant_id
JOIN Properties_History p ON mr.property_id = p.property_id;
SELECT * FROM Active_Leases_Details;


SELECT * FROM Tenant_Payment_Summary;

SELECT * FROM Maintenance_Request_Timelines;

SELECT
    tenant_id,
    CONCAT(first_name, ' ', last_name) AS full_name
FROM Tenants;

SELECT
```

6.0.1. View for active leases details



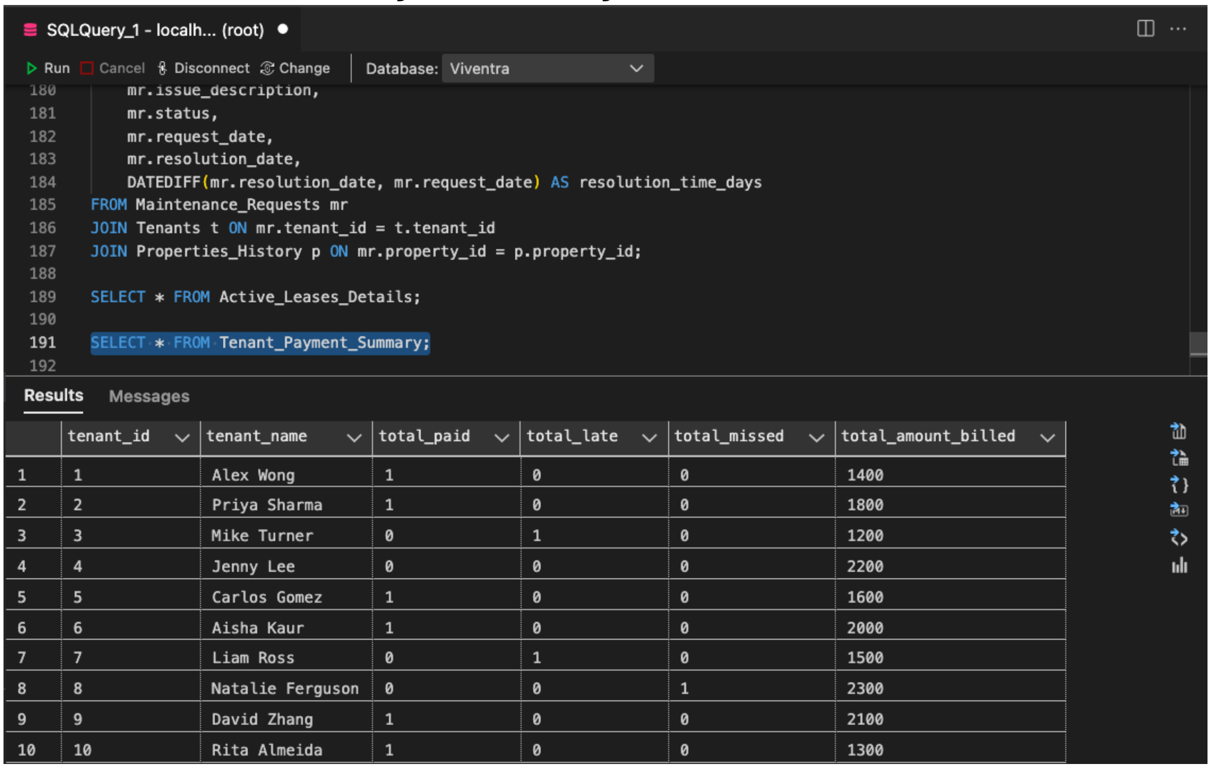
```sql
CREATE VIEW Maintenance_Request_Timelines AS
SELECT
    mr.request_id,
    CONCAT(t.first_name, ' ', t.last_name) AS tenant_name,
    p.address AS property_address,
    mr.issue_description,
    mr.status,
    mr.request_date,
    mr.resolution_date,
    DATEDIFF(mr.resolution_date, mr.request_date) AS resolution_time_days
FROM Maintenance_Requests mr
JOIN Tenants t ON mr.tenant_id = t.tenant_id
JOIN Properties_History p ON mr.property_id = p.property_id;

SELECT * FROM Active_Leases_Details;
```

lease_id	tenant_name	address	city	property_type	start_date	end_date	rent_amount
1	Alex Wong	123 Main St	Calgary	Apartment	2024-01-01	2024-12-31	1400
2	Priya Sharma	456 Lakeview Dr	Edmonton	House	2023-07-01	2024-06-30	1800
5	Carlos Gomez	678 Prairie Rd	Medicine Hat	Apartment	2024-05-01	2025-04-30	1600
6	Aisha Kaur	234 Willow Way	Airdrie	House	2023-10-01	2024-09-30	2000
7	Liam Ross	890 Sunset Blvd	Calgary	Apartment	2024-01-10	2025-01-09	1500
9	David Zhang	135 Riverbank …	Okotoks	House	2023-11-01	2024-10-31	2100
10	Rita Almeida	246 Mountain V…	Canmore	Apartment	2024-02-01	2025-01-31	1300

Query Command

```
CREATE VIEW Active_Leases_Details AS
SELECT
    la.lease_id,
    CONCAT(t.first_name, ' ', t.last_name) AS tenant_name,
    p.address,
    p.city,
    p.property_type,
    la.start_date,
    la.end_date,
    la.rent_amount
FROM Lease_Agreements la
JOIN Tenants t ON la.tenant_id = t.tenant_id
JOIN Properties_History p ON la.property_id = p.property_id
WHERE la.lease_status = 'Active';
```

6.0.2. View for Tenant Payment History



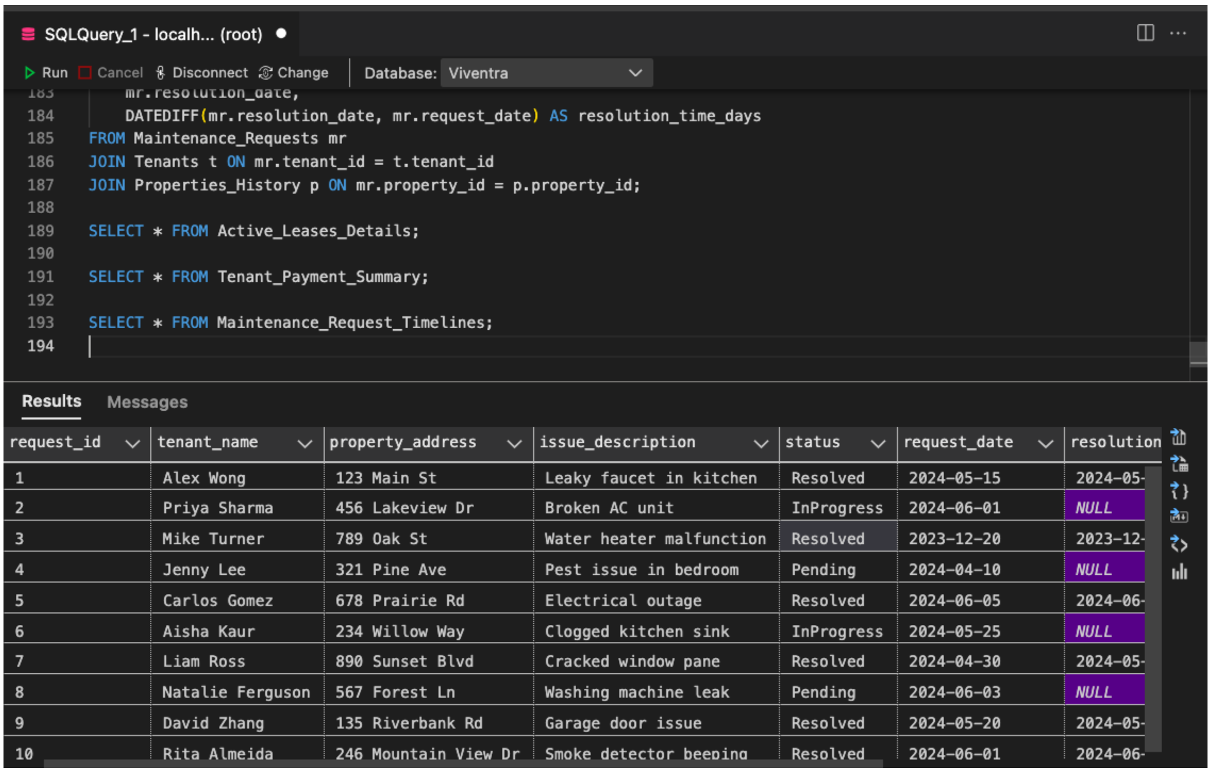
```
    mr.issue_description,
    mr.status,
    mr.request_date,
    mr.resolution_date,
    DATEDIFF(mr.resolution_date, mr.request_date) AS resolution_time_days
FROM Maintenance_Requests mr
JOIN Tenants t ON mr.tenant_id = t.tenant_id
JOIN Properties_History p ON mr.property_id = p.property_id;

SELECT * FROM Active_Leases_Details;

SELECT * FROM Tenant_Payment_Summary;
```

	tenant_id	tenant_name	total_paid	total_late	total_missed	total_amount_billed
1	1	Alex Wong	1	0	0	1400
2	2	Priya Sharma	1	0	0	1800
3	3	Mike Turner	0	1	0	1200
4	4	Jenny Lee	0	0	0	2200
5	5	Carlos Gomez	1	0	0	1600
6	6	Aisha Kaur	1	0	0	2000
7	7	Liam Ross	0	1	0	1500
8	8	Natalie Ferguson	0	0	1	2300
9	9	David Zhang	1	0	0	2100
10	10	Rita Almeida	1	0	0	1300

Query command

```
CREATE VIEW Tenant_Payment_Summary AS
SELECT
    t.tenant_id,
    CONCAT(t.first_name, ' ', t.last_name) AS tenant_name,
    COUNT(CASE WHEN p.payment_status = 'Paid' THEN 1 END) AS
total_paid,
    COUNT(CASE WHEN p.payment_status = 'Late' THEN 1 END) AS
total_late,
    COUNT(CASE WHEN p.payment_status = 'Missed' THEN 1 END) AS
total_missed,
    SUM(p.amount) AS total_amount_billed
FROM Tenants t
JOIN Lease_Agreements l ON t.tenant_id = l.tenant_id
JOIN Payments p ON l.lease_id = p.lease_id
GROUP BY t.tenant_id;
```

6.0.3. View for Maintenance Request timelines



SQLQuery_1 - localh... (root)

Database: Viventra

```
    mr.resolution_date,
    DATEDIFF(mr.resolution_date, mr.request_date) AS resolution_time_days
FROM Maintenance_Requests mr
JOIN Tenants t ON mr.tenant_id = t.tenant_id
JOIN Properties_History p ON mr.property_id = p.property_id;

SELECT * FROM Active_Leases_Details;

SELECT * FROM Tenant_Payment_Summary;

SELECT * FROM Maintenance_Request_Timelines;
```

request_id	tenant_name	property_address	issue_description	status	request_date	resolution
1	Alex Wong	123 Main St	Leaky faucet in kitchen	Resolved	2024-05-15	2024-05-
2	Priya Sharma	456 Lakeview Dr	Broken AC unit	InProgress	2024-06-01	NULL
3	Mike Turner	789 Oak St	Water heater malfunction	Resolved	2023-12-20	2023-12-
4	Jenny Lee	321 Pine Ave	Pest issue in bedroom	Pending	2024-04-10	NULL
5	Carlos Gomez	678 Prairie Rd	Electrical outage	Resolved	2024-06-05	2024-06-
6	Aisha Kaur	234 Willow Way	Clogged kitchen sink	InProgress	2024-05-25	NULL
7	Liam Ross	890 Sunset Blvd	Cracked window pane	Resolved	2024-04-30	2024-05-
8	Natalie Ferguson	567 Forest Ln	Washing machine leak	Pending	2024-06-03	NULL
9	David Zhang	135 Riverbank Rd	Garage door issue	Resolved	2024-05-20	2024-05-
10	Rita Almeida	246 Mountain View Dr	Smoke detector beeping	Resolved	2024-06-01	2024-06-

Query Command

```
CREATE VIEW Maintenance_Request_Timelines AS
SELECT
    mr.request_id,
    CONCAT(t.first_name, ' ', t.last_name) AS tenant_name,
    p.address AS property_address,
    mr.issue_description,
    mr.status,
    mr.request_date,
    mr.resolution_date,
    DATEDIFF(mr.resolution_date, mr.request_date) AS
resolution_time_days
FROM Maintenance_Requests mr
JOIN Tenants t ON mr.tenant_id = t.tenant_id
JOIN Properties_History p ON mr.property_id = p.property_id;
SELECT * FROM Active_Leases_Details;
```