

Detailed Case Study

VIVENTRA PROPERTY MANAGEMENT

Presented By : Pruthak Patel

Section No.	Heading	Page No.
1.0	Introduction	2
1.1	Industry Background	2
1.2	Business Problem	2
2.0	Mission and Vision	4
2.1	Mission Statement	4
2.2	Vision Statement	4
3.0	System Overview	6
3.1	Final Field List Table	6
3.1.1	Owners	6
3.1.2	Properties History	7
3.1.3	Tenants	7
3.1.4	Lease Agreements	8
3.1.5	Payments	8
3.1.6	Maintenance Requests	8
4.0	Entity Relationship Diagram	9
4.1	Explanation of ER Diagram with Each Table	9
4.1.1	Owners	9
4.1.2	Properties History	9
4.1.3	Tenants	10
4.1.4	Lease Agreements	10
4.1.5	Payments	10
4.1.6	Maintenance Requests	11
4.2	Relationship Table for Viventra Property Management System	11
5.0	SQL Database Queries	13
5.1	Query: Tenant Full Name	13
5.2	Query: Lease Status Active	14
5.3	Query: Payment Status for Each Lease	15
5.4	Query: Property Count per Owner	16
5.5	Query: In-Progress Maintenance Details	16
6.0	SQL Queries for Creating Views	17
6.0.1	View: Active Leases Details	17
6.0.2	View: Tenant Payment Summary	18
6.0.3	View: Maintenance Request Timelines	19

1. Introduction

1.1 Industry Overview

The real estate sector holds a vital position in the global economy, with residential rentals forming a substantial part of its activities. As cities grow and the need for housing expands, property management tasks have become more intricate. Key responsibilities—such as overseeing lease contracts, ensuring timely rent collection, responding to tenant concerns, and coordinating maintenance—demand accuracy, clear communication, and operational efficiency.

Historically, these duties were managed manually or through disconnected systems like spreadsheets, paper files, and rudimentary accounting software. This outdated approach frequently resulted in poor communication, missed or late payments, misplaced records, and delays in addressing maintenance—ultimately causing tenant dissatisfaction and hindering operational performance.

Recently, the industry has undergone a shift toward digitalization. There is an increasing push to adopt technology that can simplify and automate property management workflows. Landlords and property managers are now in search of intelligent, scalable platforms that not only reduce administrative load but also improve tenant satisfaction and ensure reliable, well-organized data management.

1.2 Business Challenge

Despite advancements in technology, many residential property management practices still rely on outdated, manual processes, scattered communication tools, and disjointed data storage. Managing multiple tenants and properties without an integrated system often creates organizational confusion. Essential records—such as lease documents, payment histories, and maintenance reports—are typically dispersed across spreadsheets, emails, paper archives, and physical files.

This disorganized method gives rise to several critical challenges:

- **Document Overload:** Maintaining physical and digital records for leases, rent receipts, and service requests consumes time, introduces errors, and becomes harder to manage over time.
- **Poor Communication:** Without a unified platform, tenants often experience delays when reporting issues or seeking information about their leases.
- **Rent Collection Difficulties:** The absence of automated payment tracking or reminders increases the likelihood of missed payments and financial discrepancies.
- **Lack of Tenant Visibility:** Limited access to important details—such as lease conditions, payment records, and maintenance request status—can erode tenant trust and lead to dissatisfaction.

- **Disconnected Data:** With information stored in multiple formats and locations, property managers struggle to maintain a comprehensive understanding of their operations or generate accurate reports.

These problems not only heighten the workload for property owners and managers but also damage tenant relations. Left unaddressed, they can result in financial setbacks, unresolved service issues, and even legal conflicts due to poor documentation and ineffective communication.

2.0 Mission and Vision

2.1 Mission Statement

“A smart digital platform that streamlines lease management, rent collection, and maintenance coordination for both tenants and landlords”.

Viventra is committed to transforming the property management process through the power of intelligent digital solutions. The platform is purpose-built to support two key user groups—landlords and tenants—who both face unique challenges in the traditional rental landscape.

For landlords, Viventra reduces the burden of administrative tasks such as drafting leases, monitoring payments, and addressing maintenance issues. For tenants, it delivers a simple and intuitive experience, allowing them to securely pay rent, access lease details, and submit maintenance requests—all within one unified platform.

By bringing these essential functions together, Viventra drives greater efficiency, improves response times, and builds stronger communication between users. The system is designed to replace outdated paperwork, minimize manual errors, and offer a more streamlined and transparent rental experience.

The focus on “smart” reflects Viventra’s advanced capabilities—it’s not just digital, but intelligent. Using structured data, dynamic dashboards, and automated processes, the platform provides meaningful insights and supports informed decision-making for both landlords and tenants.

2.2 Vision Statement

“To be a top-tier digital property management platform known for increasing transparency, fostering effective communication, and easing the administrative load in residential property rentals”.

Viventra looks toward a future where digital technology is fully integrated into all aspects of residential property management. The goal is not just to provide a software solution, but to become the trusted standard in the industry—recognized for dependability, openness, and empowering its users.

This vision is anchored in three key objectives:

- **Increasing Transparency:** By offering real-time access to records, payments, and updates, Viventra ensures that both tenants and landlords remain fully informed. This shared visibility helps prevent misunderstandings and builds long-term trust.
- **Enhancing Communication:** Acting as a centralized hub, the platform connects tenants and landlords with clear, consistent messaging—whether it’s for lease updates, rent reminders, or maintenance notices—ensuring no request or notification goes unnoticed.
- **Minimizing Administrative Burden:** Recognizing the heavy workload property managers face, Viventra automates routine processes like report generation, lease tracking, and

service scheduling. This allows managers to focus their time on strategy and growth, rather than paperwork.

Through this vision, Viventra aspires to reshape the property management experience into something smarter, faster, and more user-focused.

3.0 System Overview

Viventra is a comprehensive digital property management platform designed to facilitate smooth and efficient interactions between landlords and tenants regarding their rental arrangements.

Core Database Structure:

Table Name	Purpose
Owners	Stores detailed information about each property owner, including their name and contact details. This table helps track ownership of properties and supports communication with landlords.
Properties_History	Maintains records of all properties under management, including their features, status, and ownership. It connects properties to owners and acts as a central hub for lease, maintenance, and occupancy history.
Tenants	Contains personal and contact information of tenants. It is used to manage tenant profiles, track their lease agreements, and handle maintenance communications.
Lease_Agreements	Holds details of rental contracts between tenants and properties, including lease duration, rent amount, and status. It links tenants to properties and tracks rental activity over time.
Payments	Logs all rent payment transactions related to lease agreements. This includes payment dates, amounts, methods, and statuses, ensuring accurate financial tracking and reporting.
Maintenance_Requests	Records all service and repair requests submitted by tenants for specific properties. It includes issue details, status updates, and resolution dates to support timely property maintenance.

The system relies on a set of relational database tables that are interconnected using primary and foreign keys. This structure ensures consistent relationships between data points and preserves referential integrity across the platform.

3.1 Final field list table

3.1.1 Owners

Field Name	Data Type	Description
owner_id	INT	Unique ID for each property owner
first_name	VARCHAR(30)	First name of the owner
last_name	VARCHAR(30)	Last name of the owner
contact_number	BIGINT	Owner's phone number
email	VARCHAR(50)	Owner's email address

3.1.2 Properties_History

Field Name	Data Type	Description
property_id	INT (PK)	Unique ID for each property
address	VARCHAR(50)	Street address of the property
city	VARCHAR(20)	City where the property is located
state	VARCHAR(20)	State or province
zip_code	VARCHAR(10)	Postal or zip code
property_type	VARCHAR(20)	Type of property (e.g., Apartment, House)
num_bedrooms	INT	Number of bedrooms in the property
num_bathrooms	INT	Number of bathrooms in the property
availability_status	VARCHAR(15)	Current status (Available / Rented)
owner_id	INT (FK)	Links to the Owners table

3.1.3 Tenants

Field Name	Data Type	Description
tenant_id	INT	Unique ID for each tenant
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	Tenant's date of birth
emergency_contact	BIGINT	Emergency contact number

3.1.4 Lease Agreements

Field Name	Data Type	Description
lease_id	INT (PK)	Unique lease contract ID
property_id	INT (FK)	Links to Properties_History.property_id
tenant_id	INT (FK)	Links to Tenants.tenant_id
start_date	DATE	Lease start date
end_date	DATE	Lease end date
rent_amount	DECIMAL	Monthly rent agreed upon
lease_status	VARCHAR(15)	Status of lease (Active, Terminated, Pending)

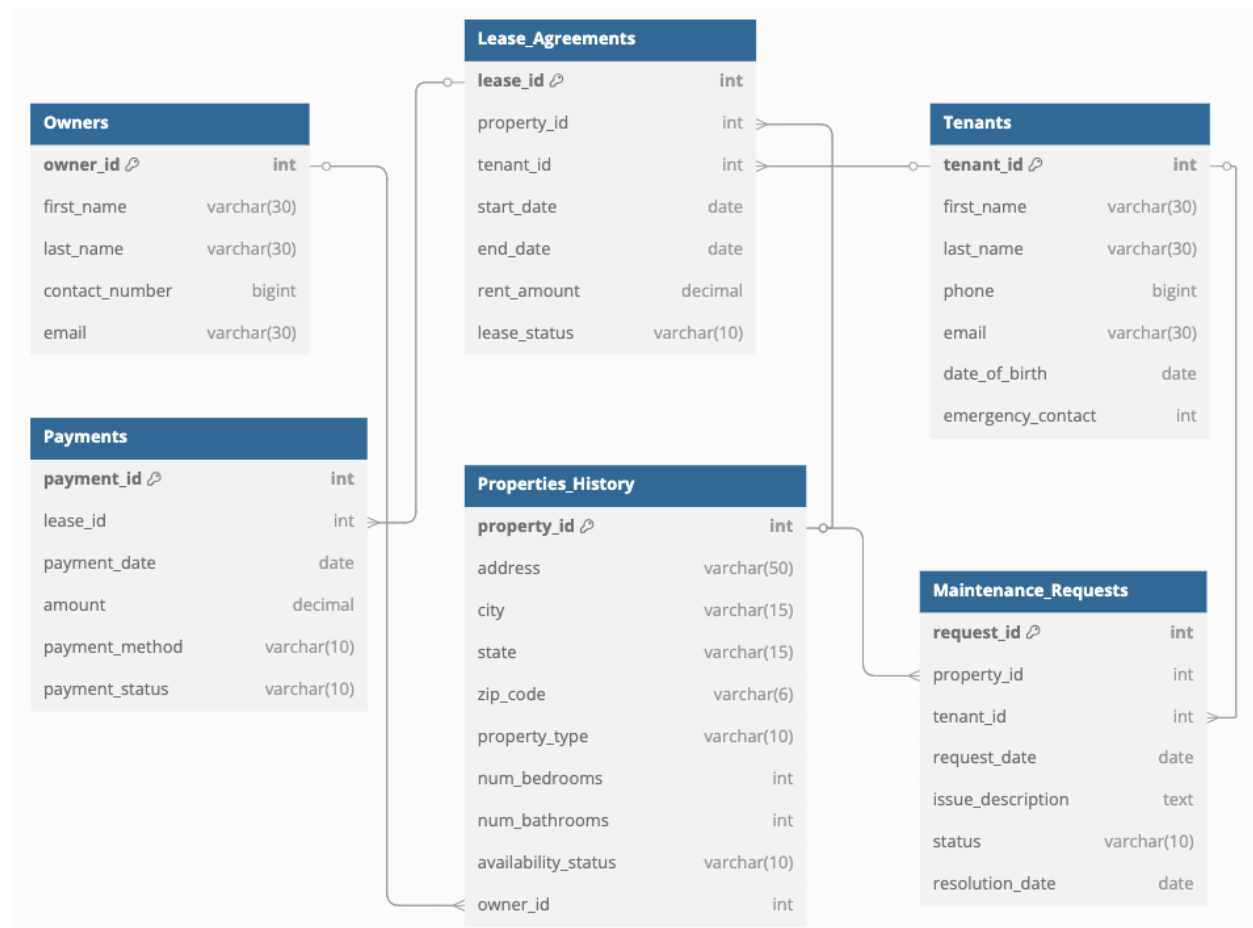
3.1.5 Payments

Field Name	Data Type	Description
payment_id	INT (PK)	Unique payment ID
lease_id	INT (FK)	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	INT (PK)	Unique ID for each maintenance request
property_id	INT (FK)	Property requiring maintenance
tenant_id	INT (FK)	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

4.0 Entity Relationship Diagram



4.1. Explanation of ER diagram with each table

4.1.1 Owners Table

The Owners table contains key details for each property owner registered in the system. Every owner is uniquely identified by the owner_id field, which functions as the table's primary key. Additional fields include the owner's first and last names, phone number, and email address. This data is essential for linking properties to their rightful owners and for enabling effective communication related to leases, property management, and maintenance tasks. There is a one-to-many relationship between the Owners table and the Properties_History table, where one owner can be associated with several properties, but each property is tied to only one owner.

4.1.2 Properties History Table

The Properties_History table maintains a record of all properties managed by the system. Each property entry is uniquely identified by a property_id, which acts as the primary key. This table captures detailed information including the property's full address, city, state, zip code, type (such as house or apartment), number of bedrooms and bathrooms, and its availability status (e.g., rented or available). Properties are associated with owners via the owner_id foreign key, creating a clear link between each property and its respective owner. This table plays a pivotal role in the system by connecting to lease records, maintenance tickets, and serving as the central repository for property-specific data.

4.1.3 Tenants

The Tenants table stores detailed personal and contact information for each tenant in the system. Every tenant is assigned a unique identifier, tenant_id, which serves as the primary key. The table includes fields for first and last names, phone number, email address, date of birth, and an emergency contact number. This information is essential for managing lease records, handling maintenance issues, and maintaining effective communication with tenants. The table is connected to both the Lease_Agreements and Maintenance_Requests tables, highlighting each tenant's role in lease contracts and maintenance activities.

4.1.4 Lease Agreements

The Lease_Agreements table outlines the formal rental agreements between tenants and properties. Each lease is uniquely identified by the lease_id primary key. It includes foreign keys for both property_id and tenant_id, thereby linking every lease to a particular property and tenant. The table also captures the lease start and end dates, the monthly rent amount, and the current status of the lease (e.g., active, terminated). This table is critical for monitoring which tenant occupies which property, the terms and duration of their lease, and for connecting to the Payments table, as multiple payments may be associated with a single lease.

4.1.5 Payments

The Payments table logs all rent-related financial transactions tied to lease agreements. Each payment entry is identified by a unique payment_id, which acts as the primary key. Payments are linked to specific leases through the lease_id foreign key. The table records vital payment details such as the date of payment, the amount paid, the payment method (e.g., cash, card, bank

transfer), and the current status (such as paid, late, or missed). This structure enables accurate tracking of payment histories, helps identify delinquencies, and ensures financial transparency for both landlords and tenants.

4.1.6 Maintenance Requests

The Maintenance_Requests table keeps a record of all service and repair requests submitted by tenants. Each request is uniquely identified using the request_id primary key. It is connected to both the property (property_id) and the tenant (tenant_id) through foreign keys. The table includes the date the request was submitted, a description of the issue, the status of the request (e.g., pending, in progress, or resolved), and the resolution date if applicable. This system allows for streamlined management and tracking of maintenance activities, ensuring that tenant issues are addressed promptly and that properties are well maintained.

4.2 Relationship Table in the Viventra Property Management System

Parent Table	Child Table	Relationship Type	Description
Owners	Properties_History	One-to-Many	One owner can have multiple properties, but each property is owned by one owner.
Properties_History	Lease_Agreements	One-to-Many	Each property may have several lease agreements over time.
Tenants	Lease_Agreements	One-to-Many	A tenant can sign multiple leases across properties or time periods.
Lease_Agreements	Payments	One-to-Many	Each lease has multiple payment records for ongoing rent transactions.
Tenants	Maintenance_Requests	One-to-Many	A tenant can submit multiple maintenance requests.
Properties_History	Maintenance_Requests	One-to-Many	A property can have numerous maintenance requests over time.

This section presents the core one-to-many relationships that define the structure of the Viventra property management database. At the highest level, the **Owners** table has a one-to-many connection with the **Properties_History** table. In this setup, a single owner can be associated with several properties, while each property belongs to only one owner. This linkage is critical for organizing and retrieving ownership data efficiently and for managing property portfolios accurately.

Next, the **Properties_History** table functions as a parent to the **Lease_Agreements** table. This relationship allows each property to be linked to multiple lease agreements over time—capturing changes in tenancy and lease duration. For instance, a property leased by different tenants across years will have separate records for each lease, enabling the system to maintain a comprehensive leasing history for every property.

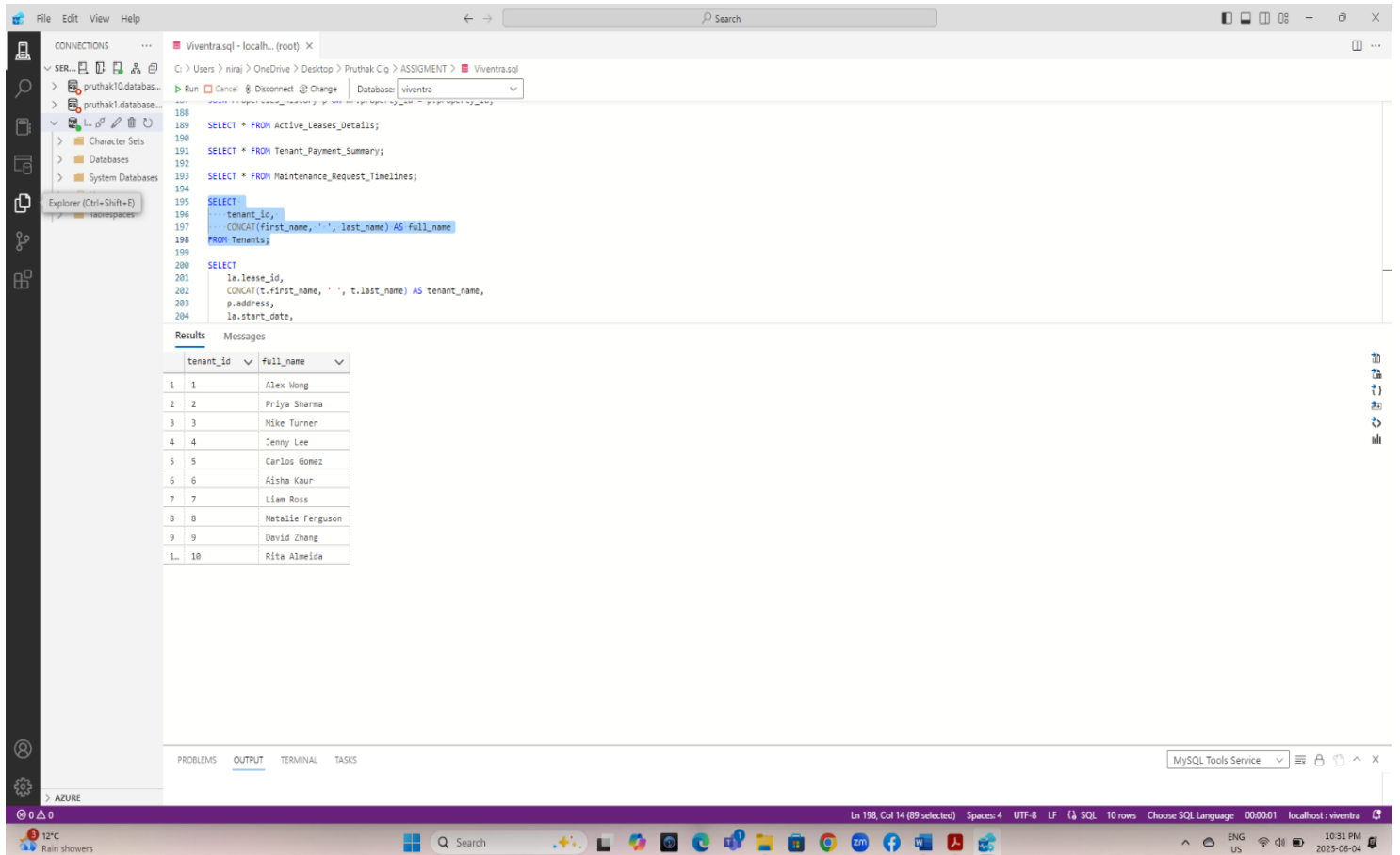
In parallel, the **Tenants** table is also related to the **Lease_Agreements** table in a one-to-many manner. A single tenant may have multiple lease entries—either for different units or for repeated leases over time. This makes it possible to track tenant movement across various properties or record multiple lease renewals within the system. The dual links between **Lease_Agreements**, **Properties_History**, and **Tenants** ensure that every lease is clearly tied to both the corresponding tenant and property.

Further, the **Lease_Agreements** table is related to the **Payments** table through a one-to-many relationship. Each lease may have several payment records tied to it, capturing monthly or periodic rental payments. This relationship is essential for tracking financial transactions, managing overdue payments, and maintaining accurate records of payment methods and amounts—ensuring financial transparency and accountability.

Moreover, both the **Tenants** and **Properties_History** tables are connected to the **Maintenance_Requests** table via one-to-many relationships. This structure means that a single tenant can make multiple maintenance requests and that any given property can accumulate several maintenance records over time. By linking each maintenance request to both the reporting tenant and the affected property, the system supports clear documentation, streamlined communication, and timely resolution of maintenance issues.

5.0. SQL Database Queries:

5.1. To know the tenants full name



The screenshot shows the SQL Server Enterprise Manager interface. The left pane displays the 'Servers' tree with 'pruthak10.databases' expanded. The right pane shows a query window with the following SQL code:

```
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,
```

The 'Results' pane shows the output of the query, displaying a table with two columns: 'tenant_id' and 'full_name'. The table contains 10 rows of data:

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

Query command:

```
SELECT

    tenant_id,

    CONCAT(first_name, ' ', last_name) AS full_name

FROM Tenants;
```

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

The screenshot shows the MySQL Workbench interface. The SQL editor contains the following query:

```
197 CONCAT(first_name, ' ', last_name) AS full_name
198 FROM Tenants;
199
200 SELECT
201     la.lease_id,
202     CONCAT(t.first_name, ' ', t.last_name) AS tenant_name,
203     p.address,
204     la.start_date,
205     la.end_date,
206     la.rent_amount
207 FROM Lease_Agreements la
208 JOIN Tenants t ON la.tenant_id = t.tenant_id
209 JOIN Properties_History p ON la.property_id = p.property_id
210 WHERE lease_status = 'Active';
211
212 SELECT
213     p.payment_id,
```

The Results tab shows the following data:

	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

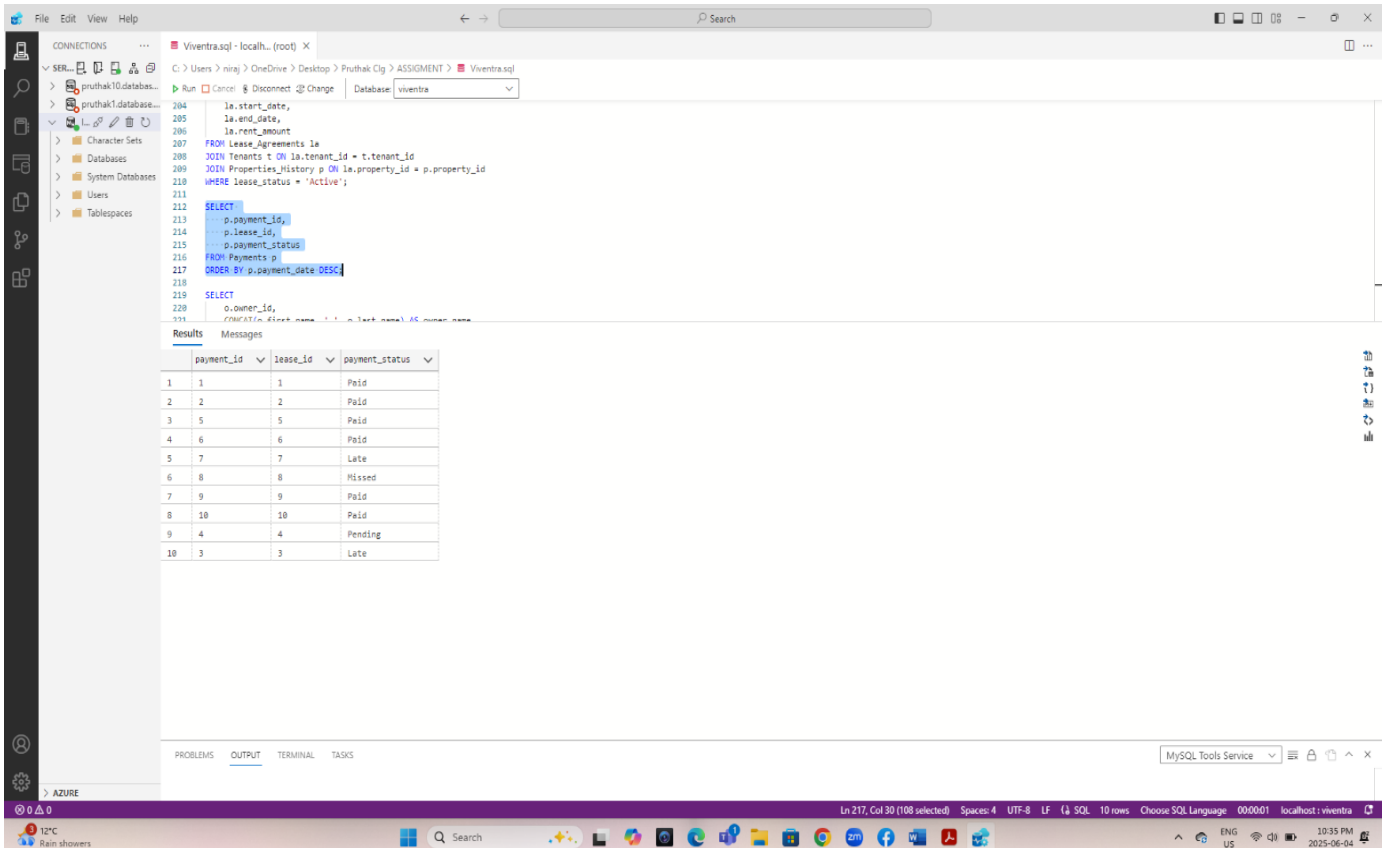
FROM Lease_Agreements la

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';
```

5.3. To know the payment status for each Lease



The screenshot shows the MySQL Workbench interface. The SQL editor contains the following query:

```
204  la.start_date,  
205  la.end_date,  
206  la.rent_amount  
207  FROM Lease_Agreements la  
208  JOIN Tenants t ON la.tenant_id = t.tenant_id  
209  JOIN Properties_History p ON la.property_id = p.property_id  
210  WHERE lease_status = 'Active';  
211  
212  SELECT  
213  p.payment_id,  
214  p.lease_id,  
215  p.payment_status  
216  FROM Payments p  
217  ORDER BY p.payment_date DESC;  
218  
219  SELECT  
220  CONCAT(t.first_name, ' ', t.last_name) AS tenant_name  
221
```

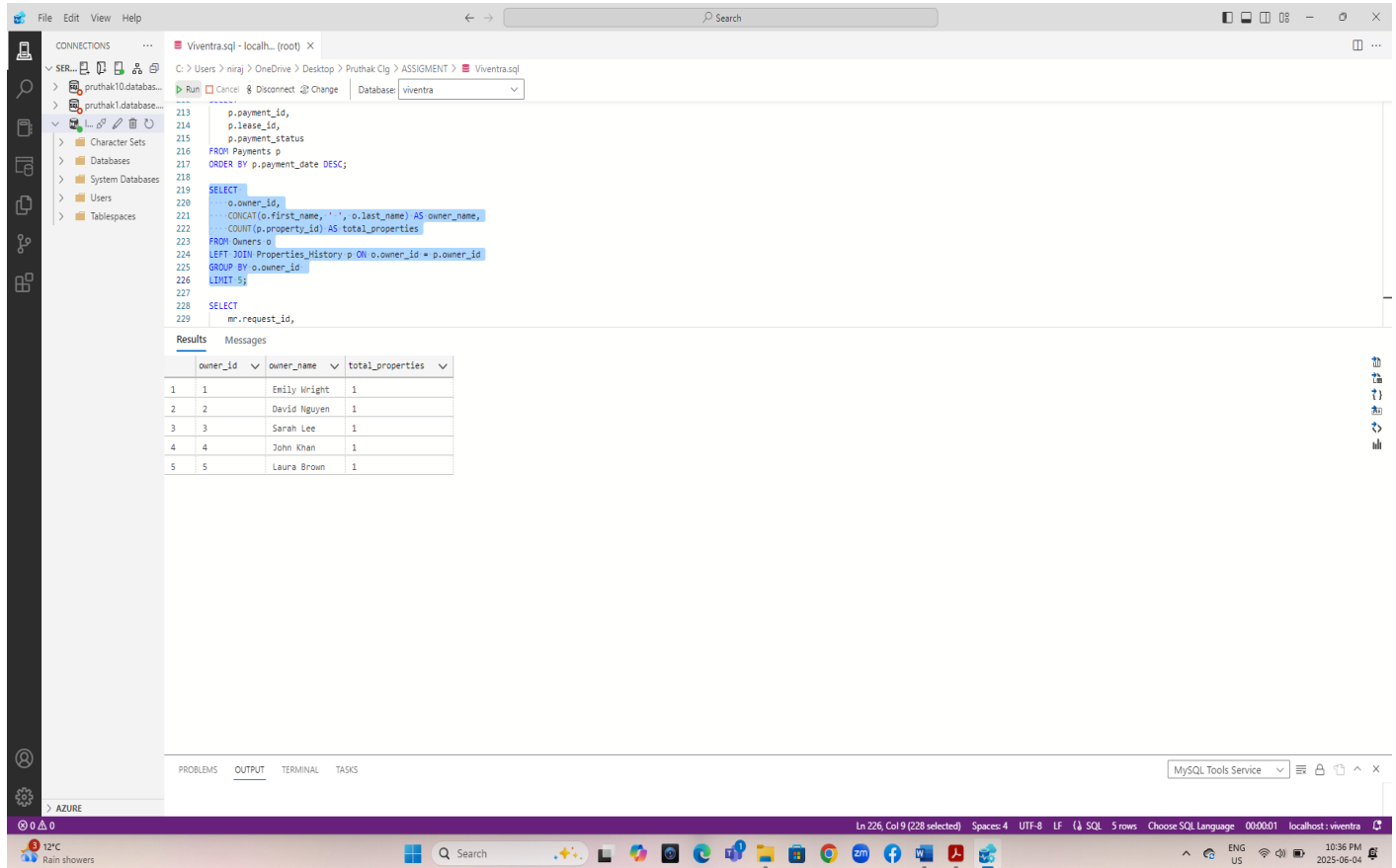
The Results tab shows the following data:

	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



The screenshot shows the MySQL Workbench interface. The SQL editor contains the following query:

```
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,
```

The Results tab shows the following data:

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

SQL query command

```
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

The screenshot displays the MySQL Tools Service application. The top toolbar includes buttons for Run, Cancel, Disconnect, and Change. The database 'viventra' is selected. The SQL editor contains the following query:

```
225 GROUP BY o.owner_id
226 LIMIT 5;
227
228 SELECT
229     mr.request_id,
230     mr.issue_description,
231     mr.status,
232     mr.request_date,
233     mr.resolution_date,
234     p.address AS property_address,
235     CONCAT(t.first_name, ' ', t.last_name) AS tenant_name
236 FROM Maintenance_Requests mr
237 JOIN Properties_History p ON mr.property_id = p.property_id
238 JOIN Tenants t ON mr.tenant_id = t.tenant_id
239 WHERE mr.status = 'InProgress';
240
241
```

The Results tab shows the following data:

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

The bottom status bar indicates: Ln 241, Col 1 (373 selected) Spaces: 4 UTF-8 LF 2 rows Choose SQL Language 00:00:01 localhost: viventra

Query Command:

```
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

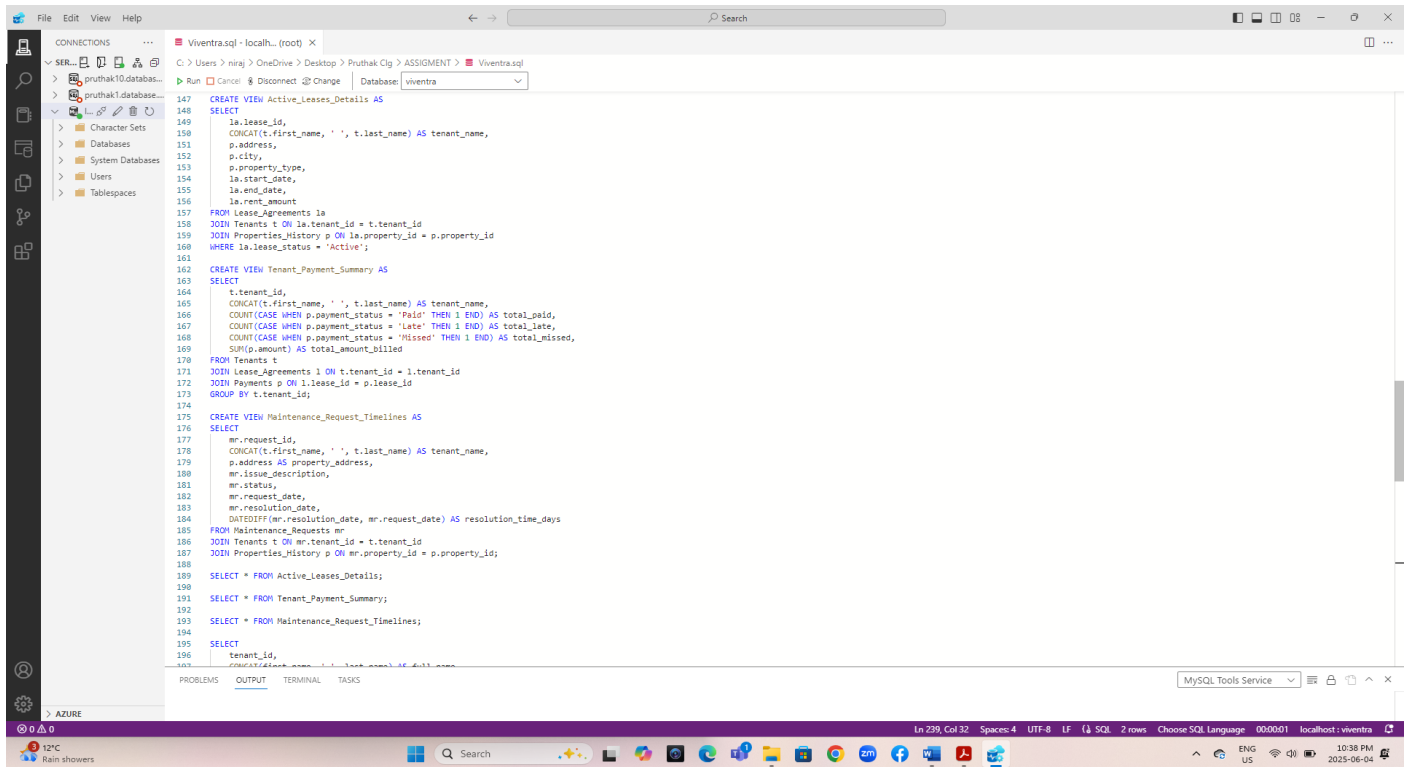
FROM Maintenance_Requests mr

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';
```

6.0. SQL queries for creating Views



The screenshot displays the MySQL Workbench interface with a SQL editor window titled 'Viventra.sql - localhost (root)'. The editor contains three SQL queries for creating views. The first query, 'CREATE VIEW Active_Leases_Details AS', selects tenant information and lease details from 'Lease_Agreements' and 'Properties_History' tables. The second query, 'CREATE VIEW Tenant_Payment_Summary AS', calculates payment statistics (total paid, total late, total missed) for each tenant. The third query, 'CREATE VIEW Maintenance_Request_Timelines AS', selects request details and calculates the resolution time in days from 'Maintenance_Requests' and 'Properties_History' tables. The interface includes a sidebar with a database tree, a top toolbar, and a bottom status bar showing the current file path and system information.

```
147 CREATE VIEW Active_Leases_Details AS
148 SELECT
149     la.lease_id,
150     CONCAT(t.first_name, ' ', t.last_name) AS tenant_name,
151     p.address,
152     p.city,
153     p.property_type,
154     la.start_date,
155     la.end_date,
156     la.rent_amount
157 FROM Lease_Agreements la
158 JOIN Tenants t ON la.tenant_id = t.tenant_id
159 JOIN Properties_History p ON la.property_id = p.property_id
160 WHERE la.lease_status = 'Active';
161
162 CREATE VIEW Tenant_Payment_Summary AS
163 SELECT
164     t.tenant_id,
165     CONCAT(t.first_name, ' ', t.last_name) AS tenant_name,
166     COUNT(CASE WHEN p.payment_status = 'Paid' THEN 1 END) AS total_paid,
167     COUNT(CASE WHEN p.payment_status = 'Late' THEN 1 END) AS total_late,
168     COUNT(CASE WHEN p.payment_status = 'Missed' THEN 1 END) AS total_missed,
169     SUM(p.amount) AS total_amount_billed
170 FROM Tenants t
171 JOIN Lease_Agreements l ON t.tenant_id = l.tenant_id
172 JOIN Payments p ON l.lease_id = p.lease_id
173 GROUP BY t.tenant_id;
174
175 CREATE VIEW Maintenance_Request_Timelines AS
176 SELECT
177     mr.request_id,
178     CONCAT(t.first_name, ' ', t.last_name) AS tenant_name,
179     p.address AS property_address,
180     mr.issue_description,
181     mr.status,
182     mr.request_date,
183     mr.resolution_date,
184     DATEDIFF(mr.resolution_date, mr.request_date) AS resolution_time_days
185 FROM Maintenance_Requests mr
186 JOIN Tenants t ON mr.tenant_id = t.tenant_id
187 JOIN Properties_History p ON mr.property_id = p.property_id;
188
189 SELECT * FROM Active_Leases_Details;
190
191 SELECT * FROM Tenant_Payment_Summary;
192
193 SELECT * FROM Maintenance_Request_Timelines;
194
195 SELECT
196     tenant_id,
```

6.0.1. View for active leases details

The screenshot shows a SQL Server Enterprise Manager interface. The left pane displays the 'Connections' tree with 'viventra.sql - localh... (root)' selected. The central query window shows a SQL script for creating a view named 'Maintenance_Request_Timelines'. The script includes a SELECT statement that joins 'Maintenance_Requests' with 'Tenants' and 'Properties_History'. Below the script, the 'Results' pane displays a table with 7 rows and 8 columns: lease_id, tenant_name, address, city, property_type, start_date, end_date, and rent_amount. The data includes tenants like Alex Wong, Priya Sharma, Carlos Gomez, Aisha Kaur, Llen Ross, David Zhang, and Rita Almeida.

lease_id	tenant_name	address	city	property_type	start_date	end_date	rent_amount
1	Alex Wong	323 Main St	Calgary	Apartment	2024-01-01	2024-12-31	1400
2	Priya Sharma	456 Lakeview	Edmonton	House	2023-07-01	2024-06-30	1800
3	Carlos Gomez	678 Prairie	Medicine Hat	Apartment	2024-05-01	2025-04-30	1600
4	Aisha Kaur	234 Willow	Airdrie	House	2023-10-01	2024-09-30	2000
5	Llen Ross	890 Sunset	Calgary	Apartment	2024-01-10	2025-01-09	1500
6	David Zhang	135 Riverb...	Okotoks	House	2023-11-01	2024-10-31	2100
7	Rita Almeida	246 Mounta...	Canmore	Apartment	2024-02-01	2025-01-31	1900

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

The screenshot shows the SQL Server Enterprise Manager interface. The left pane displays the 'Databases' folder expanded, showing 'pruthak10.databases' and 'pruthak11.databases'. The right pane shows a query window with the following SQL script:

```
174 CREATE VIEW Maintenance_Request_Timelines AS
175 SELECT
176     mr.request_id,
177     CONCAT(t.first_name, ' ', t.last_name) AS tenant_name,
178     p.address AS property_address,
179     mr.issue_description,
180     mr.status,
181     mr.request_date,
182     mr.resolution_date,
183     DATETIME(mr.resolution_date, mr.request_date) AS resolution_time_days
184 FROM Maintenance_Requests mr
185 JOIN Tenants t ON mr.tenant_id = t.tenant_id
186 JOIN Properties_History p ON mr.property_id = p.property_id;
187
188 SELECT * FROM Active_Leases_Details;
189
190 SELECT * FROM Tenant_Payment_Summary;
191
192 SELECT * FROM Maintenance_Request_Timelines;
193
194 SELECT
195     tenant_id,
196     CONCAT(first_name, ' ', last_name) AS full_name
197 FROM Tenants;
```

The 'Results' pane shows the output of the query, displaying a table with 10 rows and 7 columns:

tenant_id	tenant_name	total_paid	total_late	total_missed	total_amount_billed
1	Alex Wong	1	0	0	1400
2	Priya Sharma	1	0	0	1000
3	Mike Turner	0	1	0	1200
4	Jenny Lee	0	0	0	2200
5	Carlos Gomez	1	0	0	1600
6	Aisha Kaur	1	0	0	2000
7	Liam Ross	0	1	0	1500
8	Natalie Ferguson	0	0	1	2300
9	David Zhang	1	0	0	2100
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, 18
    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 Mintenance Request timelines

The screenshot displays the SQL Server Enterprise Manager interface. The left pane shows the database structure, including Databases, System Databases, Users, and Tablespace. The right pane shows the SQL script for creating the view `Maintenance_Request_Timelines`. The script includes a `SELECT` statement with columns: `request_id`, `tenant_name` (concatenated from `first_name` and `last_name`), `property_address`, `issue_description`, `status`, `request_date`, `resolution_date`, and `resolution_time_days` (calculated using `DATEDIFF`). The data is sourced from `Maintenance_Requests` joined with `Tenants` and `Properties_History`, and filtered by `Active_Leases_Details`. The results table shows 10 rows of data.

request_id	tenant_name	property_address	issue_description	status	request_date	resolution_date	resolution_time_days
1	Alex Wong	123 Main St	Leaky faucet in kitchen	Resolved	2024-05-15	2024-05-16	1
2	Priya Sharma	456 Lakeview Dr	Broken AC unit	InProgress	2024-06-01	NULL	NULL
3	Mike Turner	789 Oak St	Water heater malfunction	Resolved	2023-12-20	2023-12-21	1
4	Jenny Lee	321 Pine Ave	Pest issue in bedroom	Pending	2024-04-10	NULL	NULL
5	Carlos Gomez	678 Prairie Rd	Electrical outage	Resolved	2024-06-05	2024-06-06	1
6	Aisha Kaur	234 Willow Way	Clogged kitchen sink	InProgress	2024-05-25	NULL	NULL
7	Liam Ross	890 Sunset Blvd	Cracked window pane	Resolved	2024-04-30	2024-05-01	1
8	Natalie Ferguson	567 Forest Ln	Washing machine leak	Pending	2024-06-03	NULL	NULL
9	David Zhang	135 Riverbank Rd	Garage door issue	Resolved	2024-05-20	2024-05-21	1
10	Rita Almeida	246 Mountain View Dr	Smoke detector beeping	Resolved	2024-06-01	2024-06-02	1

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;
```