



SACRAMENTO STATE  
COLLEGE OF CONTINUING EDUCATION

---

# MSBA202

## Data Management for Business Analytics

Section - 70

Prepared by – Group 1

Md Tarbiar Swat - 304454332

Bindu Nithyananda – 304429021

Hashim Gilani - 304439369

Suleman Ahmad - 304467046

Group Project - Part 1 & 2

Submission Date

May 8, 2025

Prepared for

Beom-Jin Choi



# PART 1

# 1.0 Business Objective

To upgrade the existing operational database for a property management company by enhancing data integrity, reducing redundancy, and providing improved support for complex queries. The upgraded system will effectively manage detailed information on buildings, apartments, managers, staff, inspections, leases, maintenance requests, and corporate clients. In addition, the solution will support an analytical database for historical analysis and decision-making, enabling the company to monitor leasing trends, maintenance costs, and overall property performance.

## 1.1 Database & Business Requirement

- **Operational Database Requirements:**
  - **Manage Buildings & Managers:**  
Record details of buildings and the managers who oversee them, including tracking which building a manager resides in.
  - **Apartment & Leasing Management:**  
Capture apartment details (number of bedrooms, occupancy status) and manage leasing information for corporate clients. This includes tracking lease start/end dates, monthly rent, and security deposits.
  - **Maintenance Requests:**  
Record maintenance requests from tenants with details such as request date, description, and status.
  - **Inspections:**  
Manage and track building inspections by inspectors, including inspection dates and next scheduled inspection dates.
  - **Cleaning Operations:**  
Track cleaning assignments by linking staff members to the apartments they clean.

## 1.2 List of Entities and Attributes

- **Building** (Building\_ID,Street,City,ZipCode,Manager\_ID)
- **Manager** (Manager\_ID,MFull\_Name,Salary, MPhone, MEmail, Building\_ID)
- **Apartment** (Apartment\_ID, No\_of\_bedrooms, Rental\_Status, Building\_ID)
- **MaintenanceRequest** (Request\_ID, Building\_ID, AptNo, Request\_Date, Description, Status)
- **Inspector** (Inspector\_ID, InspectorName, IPhone, IEmail)
- **Inspects** (Inspector\_ID, Insp\_date, next\_insp, Building\_ID)
- **Staff** (Staff\_ID, S\_Name, SPhone, SEmail)
- **Cleans** (Apartment\_ID, Staff\_ID, Building\_ID)

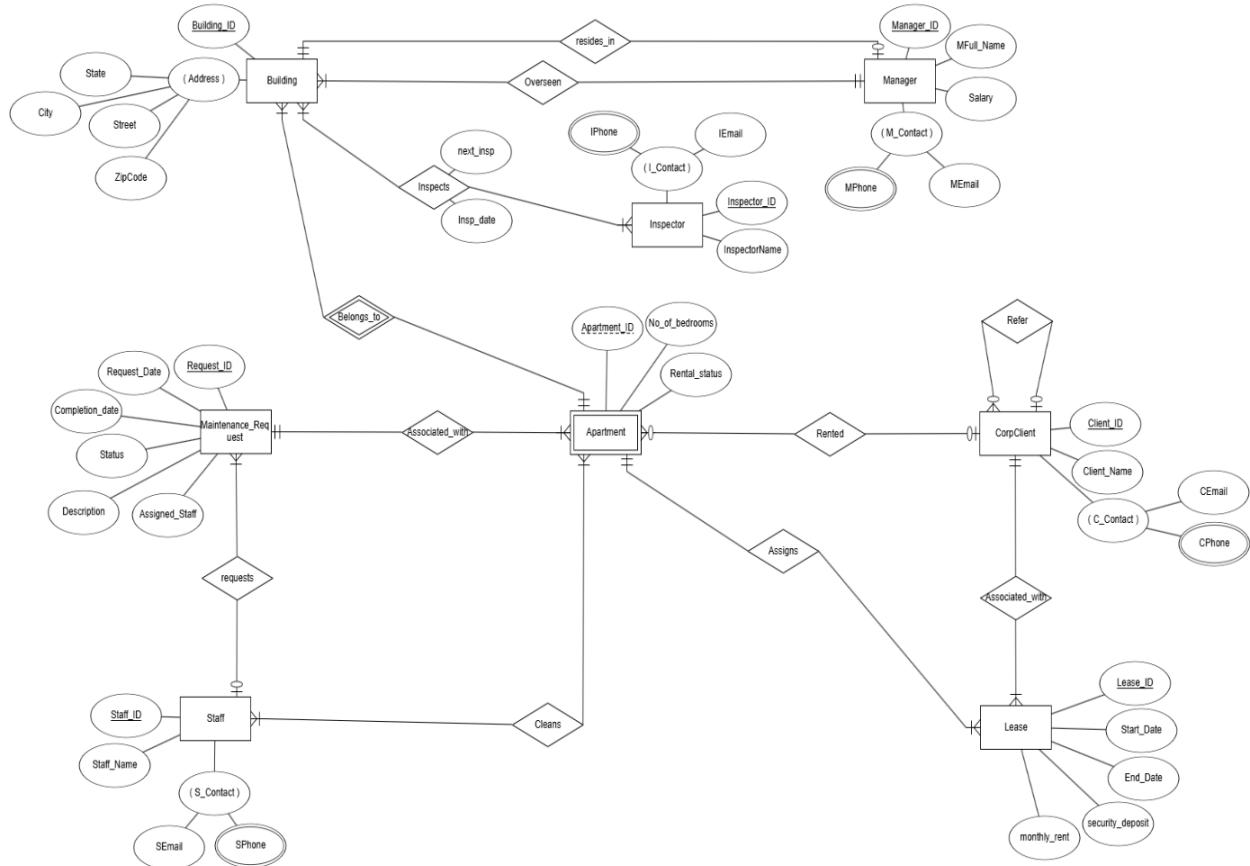
- **Resident\_Feedback** (Feedback\_ID, Feedback\_Date, Category, Description, Status, Apartment\_ID)
- **CorpClient** (Client\_ID, Client\_Name, CEmail, CPhone)
- **Lease** (Lease\_ID, Building\_ID, AptNo, Client\_ID, LeaseStartDate, LeaseEndDate, MonthlyRent, SecurityDeposit)

## 2.0 ER Diagram

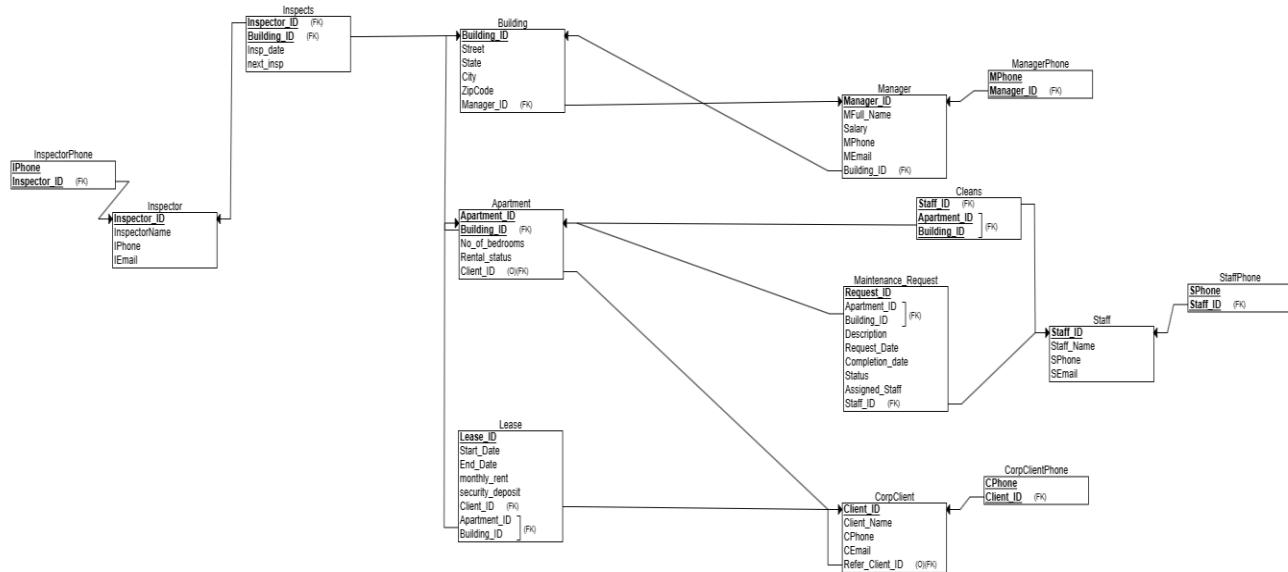
The ER Diagram visually represents the conceptual design of the property management system. It illustrates all major entities—such as Building, Manager, Apartment, MaintenanceRequest, Lease, CorpClient, Inspector, Staff, and related associative entities like Overseen, Inspection, and Cleans—and the relationships between them.

### 2.1 Create an ER diagram and the relational schema for the database (20 Points)

#### 2.1.1 Entity-Relationship Diagram: PROPERTY MANAGEMENT SYSTEM



## 2.1.2 Relational-Schema: PROPERTY MANAGEMENT SYSTEM



**2.2 Create the data dictionary (an example is shown below) that documents metadata about tables and related constraints. (15 points)**

### 2.2.1 Data-Dictionary: PROPERTY MANAGEMENT SYSTEM

| Entity:             | MANAGER TABLE                      | Contains all data about property managers.           |                             |         |          |        |                |
|---------------------|------------------------------------|--|-----------------------------|---------|----------|--------|----------------|
| Field Name          | Description                        | Type   | Specifications              | Default | Required | Unique | Key(s)         |
| Manager_ID          | Unique manager ID                  | INT  | 10 digits, unique ID        | -       | YES      | YES    | PK             |
| MFull_Name          | Manager's full name                | CHAR   | 25 characters               | -       | YES      | NO     | -              |
| Salary              | Manager's annual salary            | INT  | 20 digits                   | -       | YES      | NO     | -              |
| M_Contact           | Manager's contact number           | VARCHAR  | 15 alpha numeric characters | -       | YES      | NO     | -              |
| MEmail              | Manager's email address            | VARCHAR  | 50 alpha numeric characters | -       | YES      | YES    | -              |
| Building_ID         | Building where the manager resides | INT  | 10 digits                   | -       | YES      | NO     | FK             |
| MANAGER PHONE TABLE |                                    |  |                             |         |          |        |                |
| Mphone              | Phone number of Manager            | INT  | 10 digits                   | -       | YES      | NO     | Null Allowed   |
| Manager_ID          | Unique ID of Manager               | INT  | 10 digits, unique ID        | -       | YES      | YES    | FK             |
| Entity:             | BUILDING TABLE                     | Stores building details and assigned manager.        |                             |         |          |        |                |
| Field Name          | Description                        | Type   | Specifications              | Default | Required | Unique | Key(s)         |
| Building_ID         | Building ID                        | INT  | 10 digits, unique ID        | -       | YES      | YES    | PK             |
| Street              | Building street address            | VARCHAR  | 30 alpha numeric characters | -       | -        | -      | -              |
| City                | Building in which city             | VARCHAR  | 20 alpha numeric characters | -       | -        | -      | -              |
| State               | Building in which state            | VARCHAR  | 20 alpha numeric characters | -       | -        | -      | -              |
| ZipCode             | Building postal code               | INT  | 10 digits                   | -       | -        | -      | -              |
| Manager_ID          | Assigned Manager ID                | INT  | 10 digits, unique ID        | -       | YES      | YES    | FK             |
| Entity:             | APARTMENT TABLE                    | Tracks apartments within buildings and their status. |                             |         |          |        |                |
| Field Name          | Description                        | Type   | Specifications              | Default | Required | Unique | Key(s)         |
| Apartment_ID        | Apartment Number                   | INT  | 10 digits, unique ID        | -       | YES      | YES    | PK (Composite) |
| Building_ID         | Building ID                        | INT  | 10 digits, unique ID        | -       | YES      | NO     | PK(Composite)  |
| No_of_Bedrooms      | Number of Bedrooms                 | INT  | 2 digits, (1 - 5 Bedrooms)  | -       | YES      | NO     | -              |
| Rental_Status       | Occupancy Status                   | VARCHAR  | Occupied or Vacant          | Vacant  | YES      | NO     | -              |
| Client_ID           | Corporate Client ID                | INT  | 10 digits (if occupied)     | NULL    | NO       | NO     | FK             |
| Entity:             | CORPCLIENT TABLE                   | Tracks apartments within buildings and their status. |                             |         |          |        |                |
| Field Name          | Description                        | Type   | Specifications              | Default | Required | Unique | Key(s)         |
| Client_ID           | Corporate Client ID                | INT  | 10 digits, unique ID        | -       | YES      | YES    | PK             |
| Client_Name         | Client Name                        | VARCHAR  | 30 alpha numeric characters | -       | YES      | NO     | -              |
| CEmail              | Client Phone Number                | VARCHAR  | 20 alpha numeric characters | -       | YES      | YES    | -              |
| Refer_Client_ID     | Referrer Client ID                 | INT  | 10 digits (if referred)     | NULL    | NO       | NO     | FK             |

| CORPCLIENT PHONE TABLE |                                  |   |                             |         |          |        |                |
|------------------------|----------------------------------|---|-----------------------------|---------|----------|--------|----------------|
| CPhone                 | Phone number of corporate client | INT   | 10 digits                   | -       | YES      | NO     | Null Allowed   |
| Client_ID              | Unique ID of corporate client    | INT   | 10 digits, unique ID        | -       | YES      | YES    | FK             |
| Entity:                | LEASE TABLE                      | Tracks apartment lease agreements.              |                             |         |          |        |                |
| Field Name             | Description                      | Type  | Specifications              | Default | Required | Unique | Key(s)         |
| Lease_ID               | Lease ID                         | INT   | 10 digits, unique ID        | -       | YES      | YES    | PK             |
| Apartment_ID           | Apartment Number                 | VARCHAR   | 10 digits                   | -       | YES      | NO     | FK (Composite) |
| BuildingID             | Building ID                      | INT   | 10 digits                   | -       | YES      | NO     | FK (Composite) |
| Start_Date             | Lease Start Date                 | DATE  | Format: YYYY-MM-DD          | -       | YES      | NO     | -              |
| End_Date               | Lease End Date                   | DATE  | Format: YYYY-MM-DD          | -       | YES      | NO     | -              |
| monthly_rent           | Monthly Rent                     | DECIMAL(10,2)                                   | Currency                    | -       | YES      | NO     | -              |
| security_deposit       | Security Deposit                 | DECIMAL(10,2)                                   | Currency                    | -       | YES      | NO     | -              |
| Client_ID              | Corporate Client ID              | INT   | 10 digits                   | -       | YES      | NO     | FK             |
| Entity:                | STAFF TABLE                      | Records staff members responsible for cleaning. |                             |         |          |        |                |
| Field Name             | Description                      | Type  | Specifications              | Default | Required | Unique | Key(s)         |
| Staff_ID               | Staff ID                         | INT   | 10 digits, unique ID        | -       | YES      | YES    | PK             |
| Staff_Name             | Staff Name                       | VARCHAR   | 30 alpha numeric characters | -       | YES      | NO     | -              |
| SEmail                 | Staff email address              | VARCHAR   | 20 alpha numeric characters | -       | YES      | YES    | -              |
| STAFF PHONE TABLE      |                                  |   |                             |         |          |        |                |
| SPhone                 | Phone number of staff            | INT   | 10 digits                   | -       | YES      | NO     | Null Allowed   |
| Staff_ID               | Unique ID of staff               | INT   | 10 digits, unique ID        | -       | YES      | YES    | FK             |
| Entity:                | INSPECTOR TABLE                  | Tracks building inspectors.                     |                             |         |          |        |                |
| Field Name             | Description                      | Type  | Specifications              | Default | Required | Unique | Key(s)         |
| Inspector_ID           | Inspector ID                     | INT   | 10 digits, unique ID        | -       | YES      | YES    | PK             |
| InspectorName          | Inspector Name                   | VARCHAR   | 30 alpha numeric characters | -       | YES      | NO     | -              |
| iEmail                 | Inspector email address          | VARCHAR   | 20 alpha numeric characters | -       | YES      | YES    | -              |
| INSPECTOR PHONE TABLE  |                                  |   |                             |         |          |        |                |
| iPhone                 | Phone number of inspector        | INT   | 10 digits                   | -       | YES      | NO     | Null Allowed   |
| Inspector_ID           | Unique ID of inspector           | INT   | 10 digits, unique ID        | -       | YES      | YES    | FK             |
| Entity:                | INSPECTS TABLE                   | Logs building inspections and schedules.        |                             |         |          |        |                |
| Field Name             | Description                      | Type  | Specifications              | Default | Required | Unique | Key(s)         |
| Inspector_ID           | Inspector ID                     | INT   | 10 digits, unique ID        | -       | YES      | YES    | PK             |
| Building_ID            | Building ID                      | INT   | 10 digits                   | -       | YES      | NO     | FK             |
| Insp_date              | Inspection date                  | DATE  | Format: YYYY-MM-DD          | -       | YES      | NO     | -              |

| Insp_date       | Inspection date                | DATE  | Format: YYYY-MM-DD            | -       | YES      | NO     | -              |
|-----------------|--------------------------------|---|-------------------------------|---------|----------|--------|----------------|
| next_insp       | Next inspection date           | DATE  | Format: YYYY-MM-DD            | -       | YES      | NO     | -              |
| Entity:         | CLEANS TABLE                   | Assigns staff to clean specific apartments. |                               |         |          |        |                |
| Field Name      | Description                    | Type  | Specifications                | Default | Required | Unique | Key(s)         |
| Staff_ID        | Identifier of the staff member | INT   | 10 digits                     | -       | YES      | NO     | FK             |
| Apartment_ID    | Identifier of the apartment    | INT   | 10 digits                     | -       | YES      | NO     | FK (Composite) |
| Building_ID     | Building ID                    | INT   | 10 digits                     | -       | YES      | NO     | FK (Composite) |
| Entity:         | MAINTENANCE REQUEST TABLE      | Records tenant maintenance requests.        |                               |         |          |        |                |
| Field Name      | Description                    | Type  | Specifications                | Default | Required | Unique | Key(s)         |
| Request_ID      | Unique id for each request     | INT   | 10 digits, unique ID          | -       | YES      | YES    | PK             |
| Building_ID     | Identifier of the building     | INT   | 10 digits                     | -       | YES      | NO     | FK (Composite) |
| Apartment_ID    | Apartment Number               | INT   | 10 digits                     | -       | YES      | NO     | FK (Composite) |
| Request_Date    | Date of Request                | DATE  | Format: YYYY-MM-DD            | -       | YES      | NO     | -              |
| Completion_date | Date of completion             | DATE  | Format: YYYY-MM-DD            | -       | NO       | NO     | -              |
| Status          | Maintenance status             | VARCHAR                                     | 20 alpha numeric characters   | -       | YES      | NO     | -              |
| Assigned_Staff  | Staff Member Assigned          | VARCHAR                                     | 30 alpha numeric characters   | -       | YES      | NO     | FK             |
| Staff_ID        | Request Status                 | INT   | 10 digits                     | -       | NO       | NO     | -              |
| Entity:         | RESIDENT FEEDBACK TABLE        | Track utility billing of apartments         |                               |         |          |        |                |
| Field Name      | Description                    | Type  | Specifications                | Default | Required | Unique | Key(s)         |
| FeedbackID      | Unique feedback ID             | INT   | Auto-increment                | -       | YES      | YES    | PK             |
| BuildingID      | Building ID                    | INT   | Links to Apartment            | -       | YES      | NO     | -              |
| AptNo           | Apartment number               | INT   | Links to Apartment            | -       | YES      | NO     | -              |
| FeedbackDate    | Date of feedback submission    | DATE  | Format: YYYY-MM-DD            | -       | YES      | NO     | -              |
| Category        | Feedback category              | ENUM  | MaINTenance, Noise, Facility  | -       | YES      | NO     | -              |
| Description     | Feedback details               | TEXT  | -                             | -       | YES      | NO     | -              |
| Status          | Feedback resolution status     | VARCHAR(10)                                 | CHECK: Open/Resolved          | -       | YES      | NO     | -              |
| Entity:         | UTILITY BILLING                | Track feedbacks from residents              |                               |         |          |        |                |
| Field Name      | Description                    | Type  | Specifications                | Default | Required | Unique | Key(s)         |
| UtilityID       | Unique utility bill ID         | INT   | Auto-increment                | -       | YES      | YES    | PK             |
| BuildingID      | Building ID                    | INT   | Links to Apartment            | -       | YES      | NO     | -              |
| AptNo           | Apartment number               | INT   | Links to Apartment            | -       | YES      | NO     | -              |
| UtilityType     | Type of utility                | ENUM  | Electricity, Water, Gas       | -       | YES      | NO     | -              |
| MonthlyCost     | Monthly utility cost           | DECIMAL(10,2)                               | Positive value (e.g., 150.00) | -       | YES      | NO     | -              |
| DueDate         | Payment due date               | DATE  | Format: YYYY-MM-DD            | -       | YES      | NO     | -              |

## 2.3 Populate the database with sample data using SQL DDL codes (at least 10 rows). (10 points)

### 2.3.1 SQL Database creation and table population

The screenshot shows a database management interface with the following details:

- Navigator:** On the left, under "SCHEMAS", the "pms" schema is selected, showing its structure with Tables, Views, Stored Procedures, and Functions.
- Main Area:** The title bar says "CreateDB\_Table\_Query\_Group...". The query editor contains the following SQL DDL code:

```
1  /* Enable foreign key checks */
2 * create database PMS;
3 * use PMS;
4 * SET FOREIGN_KEY_CHECKS = 0;
5
6  /* Manager and Building Tables */
7 * CREATE TABLE MANAGER (
8     Manager_ID INT PRIMARY KEY,
9     MFull_Name CHAR(25) NOT NULL,
10    Salary INT NOT NULL,
11    M_Contact VARCHAR(15) NOT NULL,
12    MEmail VARCHAR(50) NOT NULL UNIQUE,
13    Building_ID INT NOT NULL,
14    FOREIGN KEY (Building_ID) REFERENCES BUILDING(Building_ID)
15 );
16
17 * CREATE TABLE BUILDING (
18     Building_ID INT PRIMARY KEY,
19     Street VARCHAR(30),
20     City VARCHAR(20),
21     State VARCHAR(20),
22     ZipCode INT,
23     Manager_ID INT NOT NULL UNIQUE,
24     FOREIGN KEY (Manager_ID) REFERENCES MANAGER(Manager_ID)
25 );
26
27  /* Manager Phone Table */
28 * CREATE TABLE MANAGER_PHONE (
29     Mphone VARCHAR(15) NOT NULL,
30     Manager_ID INT NOT NULL,
31     FOREIGN KEY (Manager_ID) REFERENCES MANAGER(Manager_ID)
32 );
33
34  /* Apartment Table */
35 * CREATE TABLE APARTMENT (
36     Apartment_ID INT,
37     Building_ID INT,
38     No_of_Bedrooms INT NOT NULL CHECK (No_of_Bedrooms BETWEEN 1 AND 5),
39     Rental_Status VARCHAR(10) DEFAULT 'Vacant' NOT NULL,
40     Client_ID INT,
41     PRIMARY KEY (Apartment_ID, Building_ID),
42     FOREIGN KEY (Building_ID) REFERENCES BUILDING(Building_ID),
43     FOREIGN KEY (Client_ID) REFERENCES CORPCLIENT(Client_ID)
44 );
45
```

**Navigator**

**SCHEMAS**

Filter objects

- pms
  - Tables
  - Views
  - Stored Procedures
  - Functions
- sakila
- sys
- world

Administration Schemas

Information

No object selected

CreateDB\_Table\_Query\_Group...

```

45  /* Corporate Client Tables */
46  CREATE TABLE CORPCLIENT (
47    Client_ID INT PRIMARY KEY,
48    Client_Name VARCHAR(30) NOT NULL,
49    CEmail VARCHAR(20) NOT NULL UNIQUE,
50    Refer_Client_ID INT,
51    FOREIGN KEY (Refer_Client_ID) REFERENCES CORPCLIENT(Client_ID)
52  );
53
54
55  CREATE TABLE CORPCLIENT_PHONE (
56    CPhone VARCHAR(15) NOT NULL,
57    Client_ID INT NOT NULL,
58    FOREIGN KEY (Client_ID) REFERENCES CORPCLIENT(Client_ID)
59  );
60
61  /* Lease Table */
62  CREATE TABLE LEASE (
63    Lease_ID INT(10) PRIMARY KEY,
64    Apartment_ID INT(10) NOT NULL,
65    Building_ID INT(10) NOT NULL,
66    Start_Date DATE NOT NULL,
67    End_Date DATE NOT NULL,
68    monthly_rent DECIMAL(10,2) NOT NULL,
69    security_deposit DECIMAL(10,2) NOT NULL,
70    Client_ID INT(10) NOT NULL,
71    FOREIGN KEY (Apartment_ID, Building_ID) REFERENCES APARTMENT(Apartment_ID, Building_ID),
72    FOREIGN KEY (Client_ID) REFERENCES CORPCLIENT(Client_ID)
73  );
74
75  -- Staff Tables
76  CREATE TABLE STAFF (
77    Staff_ID INT PRIMARY KEY,
78    Staff_Name VARCHAR(30) NOT NULL,
79    SEmail VARCHAR(20) NOT NULL UNIQUE
80  );
81
82  CREATE TABLE STAFF_PHONE (
83    SPhone VARCHAR(15) NOT NULL,
84    Staff_ID INT NOT NULL,
85    FOREIGN KEY (Staff_ID) REFERENCES STAFF(Staff_ID)
86  );

```

**Navigator**

**SCHEMAS**

Filter objects

- pms
  - Tables
  - Views
  - Stored Procedures
  - Functions
- sakila
- sys
- world

Administration Schemas

Information

No object selected

CreateDB\_Table\_Query\_Group...

```

133
134  /* Resident Feedback Table */
135  CREATE TABLE RESIDENT_FEEDBACK (
136    FeedbackID INT AUTO_INCREMENT PRIMARY KEY,
137    Building_ID INT NOT NULL,
138    Apartment_ID INT NOT NULL,
139    FeedbackDate DATE NOT NULL,
140    Category ENUM('Maintenance', 'Noise', 'Facility') NOT NULL,
141    Description TEXT NOT NULL,
142    Status VARCHAR(10) NOT NULL CHECK (Status IN ('Open', 'Resolved')),
143    FOREIGN KEY (Apartment_ID, Building_ID) REFERENCES APARTMENT(Apartment_ID, Building_ID)
144  );
145
146  /* Utility Billing Table */
147  CREATE TABLE UTILITY_BILLING (
148    UtilityID INT AUTO_INCREMENT PRIMARY KEY,
149    Building_ID INT NOT NULL,
150    Apartment_ID INT NOT NULL,
151    UtilityType ENUM('Electricity', 'Water', 'Gas') NOT NULL,
152    MonthlyCost DECIMAL(10,2) NOT NULL CHECK (MonthlyCost > 0),
153    DueDate DATE NOT NULL,
154    FOREIGN KEY (Apartment_ID, Building_ID) REFERENCES APARTMENT(Apartment_ID, Building_ID)
155  );

```

**Schemas**

Filter objects

- pms
  - Tables
  - Views
  - Stored Procedures
  - Functions
- sakila
- sys
- world

No object selected

```

87  /* Inspector Tables */
88  * CREATE TABLE INSPECTOR (
89  *   Inspector_ID INT PRIMARY KEY,
90  *   InspectorName VARCHAR(30) NOT NULL,
91  *   IEmail VARCHAR(20) NOT NULL UNIQUE
92  * );
93
94
95  * CREATE TABLE INSPECTOR_PHONE (
96  *   IPHONE VARCHAR(15) NOT NULL,
97  *   Inspector_ID INT NOT NULL,
98  *   FOREIGN KEY (Inspector_ID) REFERENCES INSPECTOR(Inspector_ID)
99  * );
100
101 /* Inspection Table */
102 * CREATE TABLE INSPECTS (
103 *   Inspector_ID INT,
104 *   Building_ID INT,
105 *   Insp_date DATE NOT NULL,
106 *   next_insp DATE NOT NULL,
107 *   PRIMARY KEY (Inspector_ID, Building_ID, Insp_date),
108 *   FOREIGN KEY (Inspector_ID) REFERENCES INSPECTOR(Inspector_ID),
109 *   FOREIGN KEY (Building_ID) REFERENCES BUILDING(Building_ID)
110 * );
111
112 /* Cleans Table */
113 * CREATE TABLE CLEANS (
114 *   Staff_ID INT NOT NULL,
115 *   Apartment_ID INT NOT NULL,
116 *   Building_ID INT NOT NULL,
117 *   FOREIGN KEY (Staff_ID) REFERENCES STAFF(Staff_ID),
118 *   FOREIGN KEY (Apartment_ID, Building_ID) REFERENCES APARTMENT(Apartment_ID, Building_ID)
119 * );
120
121 /* Maintenance Request Table */
122 * CREATE TABLE MAINTENANCE_REQUEST (
123 *   Request_ID INT PRIMARY KEY,
124 *   Building_ID INT NOT NULL,
125 *   Apartment_ID INT NOT NULL,
126 *   Request_Date DATE NOT NULL,
127 *   Completion_date DATE,
128 *   Status VARCHAR(20) NOT NULL,
129 *   Assigned_Staff INT NOT NULL,
130 *   FOREIGN KEY (Apartment_ID, Building_ID) REFERENCES APARTMENT(Apartment_ID, Building_ID),
131 *   FOREIGN KEY (Assigned_Staff) REFERENCES STAFF(Staff_ID)
132 * );

```

**Navigator**

**Schemas**

Filter objects

- pms
  - Tables
  - Views
  - Stored Procedures
  - Functions
- sakila
- sys
- world

Administration Schemas Information

No object selected

**InsertValues\_Query\_GroupProj\_**

```

1 SET FOREIGN_KEY_CHECKS = 0;
2 /* Insert Managers with temporary Building_IDs */
3
4 * INSERT INTO MANAGER (Manager_ID, MFull_Name, Salary, M_Contact, MEmail, Building_ID) VALUES
5 (103, 'Saanvi Desai', 68000, '555-0103', 'sdesai@gmail.com', 3),
6 (104, 'Arjun Kumar', 72000, '555-0203', 'akumar@gmail.com', 4),
7 (105, 'Ishaan Rao', 70000, '555-0301', 'irao@gmail.com', 5),
8 (106, 'Meera Nair', 75000, '555-0401', 'mnair@gmail.com', 6),
9 (107, 'Reyansh Singh', 69000, '555-0501', 'rsingh@gmail.com', 7),
10 (108, 'Aarohi Patel', 71000, '555-0601', 'apatel@gmail.com', 8),
11 (109, 'Krishna Verma', 73000, '555-0701', 'kverma@gmail.com', 9),
12 (110, 'Navya Reddy', 72000, '555-0801', 'nreddy@gmail.com', 10),
13 (111, 'Sai Prakash', 74000, '555-0901', 'sprakash@gmail.com', 11),
14 (112, 'Diya Sharma', 76000, '555-1001', 'dsharma@gmail.com', 12);
15
16 /* Insert Buildings with actual Manager_IDs */
17 * INSERT INTO BUILDING (Building_ID, Street, City, State, ZipCode, Manager_ID) VALUES
18 (3, '789 Pine Rd', 'Greenwich', 'CT', 06830, 103),
19 (4, '101 Birch Blvd', 'Fairfield', 'CT', 06824, 104),
20 (5, '202 Cedar Ln', 'Stamford', 'CT', 06901, 105),
21 (6, '303 Elm St', 'Norwalk', 'CT', 06851, 106),
22 (7, '404 Maple Dr', 'Danbury', 'CT', 06810, 107),
23 (8, '505 Oak Ct', 'Bridgeport', 'CT', 06604, 108),
24 (9, '606 Pine Ave', 'New Haven', 'CT', 06510, 109),
25 (10, '707 Birch Rd', 'Hartford', 'CT', 06103, 110),
26 (11, '808 Cedar Blvd', 'Waterbury', 'CT', 06702, 111),
27 (12, '909 Elm Ave', 'New London', 'CT', 06320, 112);
28
29

```

**Action Output**

| #  | Time     | Action  |
|----|----------|---|
| 24 | 08:48:54 | INSERT INTO BUILDING (Building_ID, Street, City, State, ZipCode, Manager_ID) VALUES (3, '789 Pine Rd', 'Greenwich', 'CT', 06830, 103)   |
| 25 | 08:48:54 | UPDATE MANAGER SET Building_ID = 3 WHERE Manager_ID = 103   |
| 26 | 08:48:54 | UPDATE MANAGER SET Building_ID = 4 WHERE Manager_ID = 104   |
| 27 | 08:48:54 | UPDATE MANAGER SET Building_ID = 5 WHERE Manager_ID = 105   |
| 28 | 08:48:54 | UPDATE MANAGER SET Building_ID = 6 WHERE Manager_ID = 106   |
| 29 | 08:48:54 | UPDATE MANAGER SET Building_ID = 7 WHERE Manager_ID = 107   |
| 30 | 08:48:54 | UPDATE MANAGER SET Building_ID = 8 WHERE Manager_ID = 108   |
| 31 | 08:48:54 | UPDATE MANAGER SET Building_ID = 9 WHERE Manager_ID = 109   |
| 32 | 08:48:54 | UPDATE MANAGER SET Building_ID = 10 WHERE Manager_ID = 110  |
| 33 | 08:48:54 | UPDATE MANAGER SET Building_ID = 11 WHERE Manager_ID = 111  |
| 34 | 08:48:54 | UPDATE MANAGER SET Building_ID = 12 WHERE Manager_ID = 112  |
| 35 | 08:48:54 | INSERT INTO MANAGER_PHONE (Mphone, Manager_ID) VALUES ('555-0103', 103), ('555-0104', 104), ('555-0301', 105), ('555-0401', 106)  |
| 36 | 08:48:54 | INSERT INTO CORPCLIENT (Client_ID, Client_Name, CEmail, Refer_Client_ID) VALUES (504, 'Innovative Solutions Ltd', 'innovate@gmail.com', null)   |
| 37 | 08:48:54 | INSERT INTO CORPCLIENT_PHONE (CPhone, Client_ID) VALUES ('555-1101', 504), ('555-1102', 504), ('555-2001', 505), ('555-2002', 505)  |
| 38 | 08:48:54 | INSERT INTO APARTMENT (Apartment_ID, Building_ID, No_of_Bedrooms, Rental_Status, Client_ID) VALUES (203, 3, 2, 'Vacant', null)  |
| 39 | 08:48:54 | INSERT INTO LEASE (Lease_ID, Apartment_ID, Building_ID, Start_Date, End_Date, monthly_rent, security_deposit, Client_ID) VALUES (1, 203, 3, '2024-05-15', '2024-05-15', 1000, 100, null)            |
| 40 | 08:48:54 | INSERT INTO STAFF (Staff_ID, Staff_Name, SEmail) VALUES (203, 'Ravi Mehta', 'rmehta@gmail.com'), (204, 'Priya Sharma', 'psharma@gmail.com')   |
| 41 | 08:48:54 | INSERT INTO STAFF_PHONE (SPhone, Staff_ID) VALUES ('555-0602', 203), ('555-0603', 204), ('555-0604', 205), ('555-0605', 206), ('555-0606', 207)   |
| 42 | 08:48:54 | INSERT INTO INSPECTOR (Inspector_ID, InspectorName, IEmail) VALUES (303, 'Ritika Das', 'rdas@exam.com'), (304, 'Kabir Malhotra', 'kmalhotra@exam.com')  |
| 43 | 08:48:54 | INSERT INTO INSPECTOR_PHONE (IPhone, Inspector_ID) VALUES ('555-0702', 303), ('555-0703', 304), ('555-0704', 305), ('555-0705', 306)  |
| 44 | 08:48:54 | INSERT INTO INSPECTS (Inspector_ID, Building_ID, Insp_date, next_insp) VALUES (303, 3, '2023-05-15', '2024-05-15'), (304, 4, '2023-06-15', '2024-06-15')  |
| 45 | 08:48:54 | INSERT INTO CLEANS (Staff_ID, Apartment_ID, Building_ID) VALUES (203, 203, 3), (204, 204, 3), (205, 205, 4), (206, 206, 4), (207, 207, 4)   |
| 46 | 08:48:54 | INSERT INTO MAINTENANCE_REQUEST (Request_ID, Building_ID, Apartment_ID, Request_Date, Completion_date, Status, Assigned_Supervisor_ID) VALUES (1, 203, 3, '2024-05-15', '2024-05-15', 'Open', null) |

**ACTION OUTPUT: SUCCESSFUL QUERY EXECUTION**

Navigator

SCHEMAS

Filter objects

▼ pms

► Tables

Views

Stored Procedures

Functions

► sakila

► sys

► world

Administration Schemas

Information

No object selected

InsertValues\_Query\_GroupProj\_ x

Limit to 1000 rows

```
29
30    /* Update Managers to reference the correct Building_IDs */
31    UPDATE MANAGER SET Building_ID = 3 WHERE Manager_ID = 103;
32    UPDATE MANAGER SET Building_ID = 4 WHERE Manager_ID = 104;
33    UPDATE MANAGER SET Building_ID = 5 WHERE Manager_ID = 105;
34    UPDATE MANAGER SET Building_ID = 6 WHERE Manager_ID = 106;
35    UPDATE MANAGER SET Building_ID = 7 WHERE Manager_ID = 107;
36    UPDATE MANAGER SET Building_ID = 8 WHERE Manager_ID = 108;
37    UPDATE MANAGER SET Building_ID = 9 WHERE Manager_ID = 109;
38    UPDATE MANAGER SET Building_ID = 10 WHERE Manager_ID = 110;
39    UPDATE MANAGER SET Building_ID = 11 WHERE Manager_ID = 111;
40    UPDATE MANAGER SET Building_ID = 12 WHERE Manager_ID = 112;
41
42    /* Insert Manager Phone Numbers */
43    INSERT INTO MANAGER_PHONE (Mphone, Manager_ID) VALUES
44        ('555-0103', 103),
45        ('555-0104', 104),
46        ('555-0301', 105),
47        ('555-0401', 106),
48        ('555-0501', 107),
49        ('555-0601', 108),
50        ('555-0701', 109),
51        ('555-0801', 110),
52        ('555-0901', 111),
53        ('555-1001', 112);
54
55    /* Insert Corporate Clients with referrals */
56    INSERT INTO CORPCLIENT (Client_ID, Client_Name, CEmail, Refer_Client_ID) VALUES
57        (504, 'Innovative Solutions Ltd', 'innovate@gmail.com', NULL),
58        (505, 'Eco Friendly Enterprises', 'eco@gmail.com', 504),
59        (506, 'Smart Tech Corp', 'smart@gmail.com', 505),
60        (507, 'Global Logistics Inc', 'log@gmail.com', 506),
61        (508, 'Creative Designs Studio', 'desi@gmail.com', 507),
62        (509, 'Health Plus Clinic', 'health@gmail.com', 508),
63        (510, 'EduTech Solutions', 'edutech@gmail.com', 509),
64        (511, 'Green Energy Co', 'green@gmail.com', 510),
65        (512, 'Fast Delivery Services', 'del@gmail.com', 511),
66        (513, 'Bright Future School', 'sc@gmail.com', 512);
67
68    /* Insert Corporate Client Phone Numbers */
69    INSERT INTO CORPCLIENT_PHONE (CPhone, Client_ID) VALUES
70        ('555-1101', 504),
71        ('555-1102', 504),
72        ('555-2001', 505),
73        ('555-2002', 505),
74        ('555-3001', 506),
75        ('555-3002', 506),
76        ('555-4001', 507),
77        ('555-4002', 507),
78        ('555-5001', 508),
79        ('555-5002', 508);
```

**Navigator**

**SCHEMAS**

Filter objects

- pms
  - Tables
  - Views
  - Stored Procedures
  - Functions
- sakila
- sys
- world

Administration Schemas Information

No object selected

InsertValues\_Query\_GroupProj... X Limit to 1000 rows

```

81  /* Insert Apartments */
82 * INSERT INTO APARTMENT (Apartment_ID, Building_ID, No_of_Bedrooms, Rental_Status, Client_ID) VALUES
83  (203, 3, 2, 'Vacant', NULL),
84  (204, 3, 3, 'Occupied', 504),
85  (205, 4, 1, 'Vacant', NULL),
86  (206, 4, 2, 'Occupied', 505),
87  (207, 5, 2, 'Vacant', NULL),
88  (208, 5, 3, 'Occupied', 506),
89  (209, 6, 1, 'Vacant', NULL),
90  (210, 6, 2, 'Occupied', 507),
91  (211, 7, 2, 'Vacant', NULL),
92  (212, 7, 3, 'Occupied', 508);

93
94  /* Insert Leases */
95 * INSERT INTO LEASE (Lease_ID, Apartment_ID, Building_ID, Start_Date, End_Date, monthly_rent, security_deposit, Client_ID) VALUES
96  (1003, 203, 3, '2023-05-01', '2024-04-30', 2200.00, 3300.00, 504),
97  (1004, 204, 3, '2023-06-15', '2024-06-14', 2400.00, 3600.00, 505),
98  -- Continue Lease Inserts
99  (1005, 205, 4, '2023-07-01', '2024-06-30', 1900.00, 2850.00, 506),
100  (1006, 206, 4, '2023-08-01', '2024-07-31', 2100.00, 3150.00, 507),
101  (1007, 207, 5, '2023-09-01', '2024-08-31', 1700.00, 2550.00, 508),
102  (1008, 208, 5, '2023-10-01', '2024-09-30', 1800.00, 2700.00, 509),
103  (1009, 209, 6, '2023-11-01', '2024-10-31', 1600.00, 2400.00, 510),
104  (1010, 210, 6, '2023-12-01', '2024-11-30', 2000.00, 3000.00, 511),
105  (1011, 211, 7, '2024-01-01', '2024-12-31', 2300.00, 3450.00, 512),
106  (1012, 212, 7, '2024-02-01', '2025-01-31', 2500.00, 3750.00, 513);

107
108  /* Insert Staff Members */
109 * INSERT INTO STAFF (Staff_ID, Staff_Name, SEmail) VALUES
110  (203, 'Ravi Mehta', 'rmehta@gmail.com'),
111  (204, 'Priya Sharma', 'psharma@gmail.com'),
112  (205, 'Kunal Joshi', 'kjoshi@gmail.com'),
113  (206, 'Nikita Rao', 'nrao@gmail.com'),
114  (207, 'Amit Kapoor', 'akapoor@gmail.com'),
115  (208, 'Sneha Iyer', 'siyer@gmail.com'),
116  (209, 'Rahul Das', 'rdas@gmail.com'),
117  (210, 'Pooja Sen', 'psen@gmail.com'),
118  (211, 'Manoj Varma', 'mvarma@gmail.com'),
119  (212, 'Tanya Roy', 'troy@gmail.com');

120
121  /* Insert Staff Phone Numbers */
122 * INSERT INTO STAFF_PHONE (SPhone, Staff_ID) VALUES
123  ('555-0602', 203),
124  ('555-0603', 204),
125  ('555-0604', 205),
126  ('555-0605', 206),
127  ('555-0606', 207),
128  ('555-0607', 208),
129  ('555-0608', 209),
130  ('555-0609', 210),
131  ('555-0610', 211),
132  ('555-0611', 212);

133

```

**Navigator**

**SCHEMAS**

Filter objects

- ▼ **pms**
  - Tables
  - Views
  - Stored Procedures
  - Functions
- **sakila**
- **sys**
- **world**

Administration Schemas Information

No object selected

**InsertValues\_Query\_GroupProj... x**

133  
134   /\* Insert Inspectors \*/  
135 •   INSERT INTO INSPECTOR (Inspector\_ID, InspectorName, IEmail) VALUES  
136   (303, 'Ritika Das', 'rdas@exam.com'),  
137   (304, 'Kabir Malhotra', 'kmalh@exam.com'),  
138   (305, 'Leena Bhatt', 'lbhatt@exam.com'),  
139   (306, 'Aditya Singh', 'asingh@exam.com'),  
140   (307, 'Tanvi Saxena', 'tsax@exam.com'),  
141   (308, 'Mohit Jain', 'mjain@gmail.com'),  
142   (309, 'Nidhi Kapoor', 'nkapoor@gmail.com'),  
143   (310, 'Ajay Deshmukh', 'adesh@gmail.com'),  
144   (311, 'Simran Kohli', 'skohli@gmail.com'),  
145   (312, 'Dev Arora', 'darora@gmail.com');

146  
147   /\* Insert Inspector Phone Numbers \*/  
148 •   INSERT INTO INSPECTOR\_PHONE (IPhone, Inspector\_ID) VALUES  
149   ('555-0702', 303),  
150   ('555-0703', 304),  
151   ('555-0704', 305),  
152   ('555-0705', 306),  
153   ('555-0706', 307),  
154   ('555-0707', 308),  
155   ('555-0708', 309),  
156   ('555-0709', 310),  
157   ('555-0710', 311),  
158   ('555-0711', 312);

159  
160   /\* Insert Inspections \*/  
161 •   INSERT INTO INSPECTS (Inspector\_ID, Building\_ID, Insp\_date, next\_insp) VALUES  
162   (303, 3, '2023-05-15', '2024-05-15'),  
163   (304, 4, '2023-06-01', '2024-06-01'),  
164   (305, 5, '2023-06-20', '2024-06-20'),  
165   (306, 6, '2023-07-10', '2024-07-10'),  
166   (307, 7, '2023-08-01', '2024-08-01'),  
167   (308, 8, '2023-09-01', '2024-09-01'),  
168   (309, 9, '2023-10-01', '2024-10-01'),  
169   (310, 10, '2023-11-01', '2024-11-01'),  
170   (311, 11, '2023-12-01', '2024-12-01'),  
171   (312, 12, '2024-01-01', '2025-01-01');

172  
173   /\* Insert Cleaning Assignments \*/  
174 •   INSERT INTO CLEANS (Staff\_ID, Apartment\_ID, Building\_ID) VALUES  
175   (203, 203, 3),  
176   (204, 204, 3),  
177   (205, 205, 4),  
178   (206, 206, 4),  
179   (207, 207, 5),  
180   (208, 208, 5),  
181   (209, 209, 6),  
182   (210, 210, 6),  
183   (211, 211, 7),  
184   (212, 212, 7);

185

Navigator > Schemas > pms

Filter objects

Administration Schemas Information

No object selected

```

185      /* Insert Maintenance Requests */
186      • INSERT INTO MAINTENANCE_REQUEST (Request_ID, Building_ID, Apartment_ID, Request_Date, Completion_date, Status, Assigned_Staff) VALUES
187          (403, 3, 203, '2023-05-20', '2023-05-22', 'Completed', 203),
188          (404, 3, 204, '2023-06-05', NULL, 'Pending', 204),
189          (405, 4, 205, '2023-06-15', '2023-06-17', 'Completed', 205),
190          (406, 4, 206, '2023-07-10', NULL, 'Pending', 206),
191          (407, 5, 207, '2023-08-01', NULL, 'Pending', 207),
192          (408, 5, 208, '2023-08-15', '2023-08-16', 'Completed', 208),
193          (409, 6, 209, '2023-09-01', NULL, 'Pending', 209),
194          (410, 6, 210, '2023-10-01', '2023-10-02', 'Completed', 210),
195          (411, 7, 211, '2023-11-01', NULL, 'Pending', 211),
196          (412, 7, 212, '2023-12-01', NULL, 'Pending', 212);

198      /* Insert Resident Feedback */
199      • INSERT INTO RESIDENT_FEEDBACK (Building_ID, Apartment_ID, FeedbackDate, Category, Description, Status) VALUES
200          (3, 203, '2023-06-10', 'Facility', 'Gym equipment needs maintenance.', 'Open'),
201          (3, 204, '2023-06-12', 'Noise', 'Parties late at night.', 'Open'),
202          (4, 205, '2023-07-01', 'Maintenance', 'Air conditioning not working.', 'Resolved'),
203          (4, 206, '2023-07-10', 'Noise', 'Construction noise in the morning.', 'Open'),
204          (5, 207, '2023-08-01', 'Facility', 'Pool water is not clean.', 'Resolved'),
205          (5, 208, '2023-08-05', 'Noise', 'Elevator noise.', 'Open'),
206          (6, 209, '2023-09-01', 'Maintenance', 'Plumbing issue in bathroom.', 'Resolved'),
207          (6, 210, '2023-09-10', 'Facility', 'Lights in hallway flickering.', 'Resolved'),
208          (7, 211, '2023-10-01', 'Noise', 'Neighbors playing loud music.', 'Open'),
209          (7, 212, '2023-10-05', 'Maintenance', 'Broken window in living room.', 'Resolved');

211      /* Insert Utility Bills */
212      • INSERT INTO UTILITY_BILLING (Building_ID, Apartment_ID, UtilityType, MonthlyCost, DueDate) VALUES
213          (3, 203, 'Electricity', 160.00, '2023-06-01'),
214          (3, 204, 'Water', 80.00, '2023-06-01'),
215          (4, 205, 'Gas', 100.00, '2023-07-01'),
216          (4, 206, 'Electricity', 170.00, '2023-07-01'),
217          (5, 207, 'Water', 85.00, '2023-08-01'),
218          (5, 208, 'Gas', 95.00, '2023-08-01'),
219          (6, 209, 'Electricity', 150.00, '2023-09-01'),
220          (6, 210, 'Water', 90.00, '2023-09-01'),
221          (7, 211, 'Gas', 110.00, '2023-10-01'),
222          (7, 212, 'Electricity', 175.00, '2023-10-01');

224      /* Re-enable foreign key checks */
225      • SET FOREIGN_KEY_CHECKS = 1;

```

### 2.3.2 The SELECT \* FROM TABLE statement in SQL would return all columns and all rows from the specified table.

The following snippets show the details of all the tables from the query:

The screenshot shows a database interface with a left sidebar for 'Navigator' and 'Administration' tabs, and a main area for the 'apartment' table.

**Table: apartment**

**Columns:**

- Apartment\_ID int PK
- Building\_ID int PK
- No\_of\_Bedrooms int
- Rental\_Status varchar(10)
- Client\_ID int

**Result Grid:**

| Apartment_ID | Building_ID | No_of_Bedrooms | Rental_Status | Client_ID |
|--------------|-------------|----------------|---------------|-----------|
| 203          | 3           | 2              | Vacant        | NULL      |
| 204          | 3           | 3              | Occupied      | 504       |
| 205          | 4           | 1              | Vacant        | NULL      |
| 206          | 4           | 2              | Occupied      | 505       |
| 207          | 5           | 2              | Vacant        | NULL      |
| 208          | 5           | 3              | Occupied      | 506       |
| 209          | 6           | 1              | Vacant        | NULL      |
| 210          | 6           | 2              | Occupied      | 507       |
| 211          | 7           | 2              | Vacant        | NULL      |
| 212          | 7           | 3              | Occupied      | 508       |
| NULL         | NULL        | NULL           | NULL          | NULL      |

The screenshot shows a database interface with a left sidebar for 'Navigator' and 'Administration' tabs, and a main area for the 'cleans' table.

**Table: cleans**

**Columns:**

- Staff\_ID int
- Apartment\_ID int
- Building\_ID int

**Result Grid:**

| Staff_ID | Apartment_ID | Building_ID |
|----------|--------------|-------------|
| 203      | 203          | 3           |
| 204      | 204          | 3           |
| 205      | 205          | 4           |
| 206      | 206          | 4           |
| 207      | 207          | 5           |
| 208      | 208          | 5           |
| 209      | 209          | 6           |
| 210      | 210          | 6           |
| 211      | 211          | 7           |
| 212      | 212          | 7           |

**Navigator**

**SCHEMAS**

Filter objects

**pms**

Tables

- apartment
- building
- cleans
- corpclient
- corpclient\_phone
- inspector
- inspector\_phone
- inspects
- lease
- maintenance\_reques
- manager
- manager\_phone
- resident\_feedback
- staff
- staff\_phone
- utility\_billing

Views

Administration Schemas

Information

**Table: corpclient**

**Columns:**

| Client_ID | Client_Name              | CEmail             | Refer_Client_ID |
|-----------|--------------------------|--------------------|-----------------|
| 504       | Innovative Solutions Ltd | innovate@gmail.com | NULL            |
| 505       | Eco Friendly Enterprises | eco@gmail.com      | 504             |
| 506       | Smart Tech Corp          | smart@gmail.com    | 505             |
| 507       | Global Logistics Inc     | log@gmail.com      | 506             |
| 508       | Creative Designs Studio  | desi@gmail.com     | 507             |
| 509       | Health Plus Clinic       | health@gmail.com   | 508             |
| 510       | EduTech Solutions        | edutech@gmail.com  | 509             |
| 511       | Green Energy Co          | green@gmail.com    | 510             |
| 512       | Fast Delivery Services   | del@gmail.com      | 511             |
| 513       | Bright Future School     | sc@gmail.com       | 512             |
| NULL      | NULL                     | NULL               | NULL            |

apartment x

1. SELECT \* FROM corpclient;

Result Grid Filter Rows: Edit: Export/Import: Wrap Cell Content:

**Navigator**

**SCHEMAS**

Filter objects

**pms**

Tables

- apartment
- building
- cleans
- corpclient
- corpclient\_phone
- inspector
- inspector\_phone
- inspects
- lease
- maintenance\_reques
- manager
- manager\_phone
- resident\_feedback
- staff
- staff\_phone
- utility\_billing

Views

Administration Schemas

Information

**Table: corpclient**

**Columns:**

| Client_ID | Client_Name              | CEmail             | Refer_Client_ID |
|-----------|--------------------------|--------------------|-----------------|
| 504       | Innovative Solutions Ltd | innovate@gmail.com | NULL            |
| 505       | Eco Friendly Enterprises | eco@gmail.com      | 504             |
| 506       | Smart Tech Corp          | smart@gmail.com    | 505             |
| 507       | Global Logistics Inc     | log@gmail.com      | 506             |
| 508       | Creative Designs Studio  | desi@gmail.com     | 507             |
| 509       | Health Plus Clinic       | health@gmail.com   | 508             |
| 510       | EduTech Solutions        | edutech@gmail.com  | 509             |
| 511       | Green Energy Co          | green@gmail.com    | 510             |
| 512       | Fast Delivery Services   | del@gmail.com      | 511             |
| 513       | Bright Future School     | sc@gmail.com       | 512             |
| NULL      | NULL                     | NULL               | NULL            |

apartment x

1. SELECT \* FROM corpclient\_phone;

Result Grid Filter Rows: Export: Wrap Cell Content:

| CPhone   | Client_ID |
|----------|-----------|
| 555-1101 | 504       |
| 555-1102 | 504       |
| 555-2001 | 505       |
| 555-2002 | 505       |
| 555-3001 | 506       |
| 555-3002 | 506       |
| 555-4001 | 507       |
| 555-4002 | 507       |
| 555-5001 | 508       |
| 555-5002 | 508       |

**Navigator**

**SCHEMAS**

Filter objects

**pms**

Tables

- apartment
- building
- cleans
- corpclient
- corpclient\_phone
- inspector
- inspector\_phone
- inspects
- lease
- maintenance\_requests
- manager
- manager\_phone
- resident\_feedback
- staff
- staff\_phone
- utility\_billing

Views

Administration Schemas

Information

**Table: inspector**

**Columns:**

| Inspector_ID  | int PK         |                   |
|---------------|----------------|-------------------|
| InspectorName | varchar(30)    |                   |
| IEmail        | varchar(20)    |                   |
| 303           | Ritika Das     | rdas@exam.com     |
| 304           | Kabir Malhotra | kmalh@exam.com    |
| 305           | Leena Bhatt    | lbhatt@exam.com   |
| 306           | Aditya Singh   | asingh@exam.com   |
| 307           | Tanvi Saxena   | tsax@exam.com     |
| 308           | Mohit Jain     | mjain@gmail.com   |
| 309           | Nidhi Kapoor   | nkapoor@gmail.com |
| 310           | Ajay Deshmukh  | adesh@gmail.com   |
| 311           | Simran Kohli   | skohli@gmail.com  |
| 312           | Dev Arora      | darora@gmail.com  |
| *             | NULL           | NULL              |

Result Grid Filter Rows: Edit: Export/Import: Wrap Cell Content:

**Navigator**

**SCHEMAS**

Filter objects

**pms**

Tables

- apartment
- building
- cleans
- corpclient
- corpclient\_phone
- inspector
- inspector\_phone
- inspects
- lease
- maintenance\_requests
- manager
- manager\_phone
- resident\_feedback
- staff
- staff\_phone
- utility\_billing

Views

Administration Schemas

Information

**Table: inspector\_phone**

**Columns:**

| IPhone   | Inspector_ID |
|----------|--------------|
| 555-0702 | 303          |
| 555-0703 | 304          |
| 555-0704 | 305          |
| 555-0705 | 306          |
| 555-0706 | 307          |
| 555-0707 | 308          |
| 555-0708 | 309          |
| 555-0709 | 310          |
| 555-0710 | 311          |
| 555-0711 | 312          |

Result Grid Filter Rows: Export: Wrap Cell Content:

**Navigator**

**SCHEMAS**

Filter objects

**pms**

Tables

- apartment
- building
- dleans
- corpclient
- corpclient\_phone
- inspector
- inspector\_phone
- inspects
- lease
- maintenance\_reques
- manager
- manager\_phone
- resident\_feedback
- staff
- staff\_phone
- utility\_billing

Views

Administration Schemas

Information

**Table: inspects**

Columns:

| Inspector_ID | int PK  |            |            |
|--------------|---------|------------|------------|
| Building_ID  | int PK  |            |            |
| Insp_date    | date PK |            |            |
| next_insp    | date    |            |            |
| 303          | 3       | 2023-05-15 | 2024-05-15 |
| 304          | 4       | 2023-06-01 | 2024-06-01 |
| 305          | 5       | 2023-06-20 | 2024-06-20 |
| 306          | 6       | 2023-07-10 | 2024-07-10 |
| 307          | 7       | 2023-08-01 | 2024-08-01 |
| 308          | 8       | 2023-09-01 | 2024-09-01 |
| 309          | 9       | 2023-10-01 | 2024-10-01 |
| 310          | 10      | 2023-11-01 | 2024-11-01 |
| 311          | 11      | 2023-12-01 | 2024-12-01 |
| 312          | 12      | 2024-01-01 | 2025-01-01 |
| *            | HULL    | HULL       | HULL       |

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content: IA

apartment x

1 • SELECT \* FROM inspects;

Limit to 1000 rows

**Navigator**

**SCHEMAS**

Filter objects

**pms**

Tables

- apartment
- building
- dleans
- corpclient
- corpclient\_phone
- inspector
- inspector\_phone
- inspects
- lease
- maintenance\_reques
- manager
- manager\_phone
- resident\_feedback
- staff
- staff\_phone
- utility\_billing

Views

Administration Schemas

Information

**Table: inspects**

Columns:

| Inspector_ID | int PK  |      |            |            |         |         |      |
|--------------|---------|------|------------|------------|---------|---------|------|
| Building_ID  | int PK  |      |            |            |         |         |      |
| Insp_date    | date PK |      |            |            |         |         |      |
| next_insp    | date    |      |            |            |         |         |      |
| 1003         | 203     | 3    | 2023-05-01 | 2024-04-30 | 2200.00 | 3300.00 | 504  |
| 1004         | 204     | 3    | 2023-06-15 | 2024-06-14 | 2400.00 | 3600.00 | 505  |
| 1005         | 205     | 4    | 2023-07-01 | 2024-06-30 | 1900.00 | 2850.00 | 506  |
| 1006         | 206     | 4    | 2023-08-01 | 2024-07-31 | 2100.00 | 3150.00 | 507  |
| 1007         | 207     | 5    | 2023-09-01 | 2024-08-31 | 1700.00 | 2550.00 | 508  |
| 1008         | 208     | 5    | 2023-10-01 | 2024-09-30 | 1800.00 | 2700.00 | 509  |
| 1009         | 209     | 6    | 2023-11-01 | 2024-10-31 | 1600.00 | 2400.00 | 510  |
| 1010         | 210     | 6    | 2023-12-01 | 2024-11-30 | 2000.00 | 3000.00 | 511  |
| 1011         | 211     | 7    | 2024-01-01 | 2024-12-31 | 2300.00 | 3450.00 | 512  |
| 1012         | 212     | 7    | 2024-02-01 | 2025-01-31 | 2500.00 | 3750.00 | 513  |
| *            | HULL    | HULL | HULL       | HULL       | HULL    | HULL    | HULL |

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content: IA

apartment x

1 • SELECT \* FROM lease;

Limit to 1000 rows

**Navigator**

**SCHEMAS**

Filter objects

**pms**

Tables

- apartment
- building
- cleans
- corpclient
- corpclient\_phone
- inspector
- inspector\_phone
- inspects
- lease
- maintenance\_requests
- manager
- manager\_phone
- resident\_feedback
- staff
- staff\_phone
- utility\_billing

Views

Administration Schemas

Information

**Table:** maintenance\_request

**Columns:**

| Request_ID | Building_ID | Apartment_ID | Request_Date | Completion_date | Status    | Assigned_Staff |
|------------|-------------|--------------|--------------|-----------------|-----------|----------------|
| 403        | 3           | 203          | 2023-05-20   | 2023-05-22      | Completed | 203            |
| 404        | 3           | 204          | 2023-06-05   | NULL            | Pending   | 204            |
| 405        | 4           | 205          | 2023-06-15   | 2023-06-17      | Completed | 205            |
| 406        | 4           | 206          | 2023-07-10   | NULL            | Pending   | 206            |
| 407        | 5           | 207          | 2023-08-01   | NULL            | Pending   | 207            |
| 408        | 5           | 208          | 2023-08-15   | 2023-08-16      | Completed | 208            |
| 409        | 6           | 209          | 2023-09-01   | NULL            | Pending   | 209            |
| 410        | 6           | 210          | 2023-10-01   | 2023-10-02      | Completed | 210            |
| 411        | 7           | 211          | 2023-11-01   | NULL            | Pending   | 211            |
| 412        | 7           | 212          | 2023-12-01   | NULL            | Pending   | 212            |
| NULL       | NULL        | NULL         | NULL         | NULL            | NULL      | NULL           |

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content: |

apartment x

1 SELECT \* FROM maintenance\_request;

Limit to 1000 rows

**Navigator**

**SCHEMAS**

Filter objects

**pms**

Tables

- apartment
- building
- cleans
- corpclient
- corpclient\_phone
- inspector
- inspector\_phone
- inspects
- lease
- maintenance\_requests
- manager
- manager\_phone
- resident\_feedback
- staff
- staff\_phone
- utility\_billing

Views

Administration Schemas

Information

**Table:** manager

**Columns:**

| Manager_ID | MFull_Name    | Salary | M_Contact | MEmail             | Building_ID |
|------------|---------------|--------|-----------|--------------------|-------------|
| 103        | Saanvi Desai  | 68000  | 555-0103  | sdesai@gmail.com   | 3           |
| 104        | Arjun Kumar   | 72000  | 555-0203  | akumar@gmail.com   | 4           |
| 105        | Ishaan Rao    | 70000  | 555-0301  | irao@gmail.com     | 5           |
| 106        | Meera Nair    | 75000  | 555-0401  | mnair@gmail.com    | 6           |
| 107        | Reyansh Singh | 69000  | 555-0501  | rsingh@gmail.com   | 7           |
| 108        | Aarohi Patel  | 71000  | 555-0601  | apatel@gmail.com   | 8           |
| 109        | Krishna Verma | 73000  | 555-0701  | kverma@gmail.com   | 9           |
| 110        | Navya Reddy   | 72000  | 555-0801  | nreddy@gmail.com   | 10          |
| 111        | Sai Prakash   | 74000  | 555-0901  | sprakash@gmail.com | 11          |
| 112        | Diya Sharma   | 76000  | 555-1001  | dsharma@gmail.com  | 12          |
| NULL       | NULL          | NULL   | NULL      | NULL               | NULL        |

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content: |

apartment x

1 SELECT \* FROM managers;

Limit to 1000 rows

**Navigator**

**SCHEMAS**

Filter objects

**pms**

Tables

- apartment
- building
- cleans
- corpclient
- corpclient\_phone
- inspector
- inspector\_phone
- inspects
- lease
- maintenance\_reqes
- manager
- manager\_phone
- resident\_feedback
- staff
- staff\_phone
- utility\_billing

Views

Administration Schemas

Information

**Table: manager\_phone**

**Columns:**

|            |             |
|------------|-------------|
| Mphone     | varchar(15) |
| Manager_ID | int         |

Result Grid

| Mphone   | Manager_ID |
|----------|------------|
| 555-0103 | 103        |
| 555-0104 | 104        |
| 555-0301 | 105        |
| 555-0401 | 106        |
| 555-0501 | 107        |
| 555-0601 | 108        |
| 555-0701 | 109        |
| 555-0801 | 110        |
| 555-0901 | 111        |
| 555-1001 | 112        |

**Navigator**

**SCHEMAS**

Filter objects

**pms**

Tables

- apartment
- building
- cleans
- corpclient
- corpclient\_phone
- inspector
- inspector\_phone
- inspects
- lease
- maintenance\_reqes
- manager
- manager\_phone
- resident\_feedback
- staff
- staff\_phone
- utility\_billing

Views

Administration Schemas

Information

**Table: staff**

**Columns:**

|            |             |
|------------|-------------|
| Staff_ID   | int PK      |
| Staff_Name | varchar(30) |
| SEmail     | varchar(20) |

Result Grid

| Staff_ID | Staff_Name   | SEmail            |
|----------|--------------|-------------------|
| 203      | Ravi Mehta   | rmehta@gmail.com  |
| 204      | Priya Sharma | psharma@gmail.com |
| 205      | Kunal Joshi  | kjoshi@gmail.com  |
| 206      | Nikita Rao   | nrao@gmail.com    |
| 207      | Amit Kapoor  | akapoor@gmail.com |
| 208      | Sneha Iyer   | siyer@gmail.com   |
| 209      | Rahul Das    | rdas@gmail.com    |
| 210      | Pooja Sen    | psen@gmail.com    |
| 211      | Manoj Varma  | mvarma@gmail.com  |
| 212      | Tanya Roy    | troy@gmail.com    |
| HULL     | HULL         | HULL              |

MySQL Workbench Screenshot:

**Navigator**

- SCHEMAS
  - pms
    - Tables: apartment, building, cleans, corpclient, corpclient\_phone, inspector, inspector\_phone, inspects, lease, maintenance\_requests, manager, manager\_phone, resident\_feedback, staff, staff\_phone, utility\_billing

**Editor**

Query: SELECT \* FROM staff\_phone;

Result Grid:

| SPhone   | Staff_ID |
|----------|----------|
| 555-0602 | 203      |
| 555-0603 | 204      |
| 555-0604 | 205      |
| 555-0605 | 206      |
| 555-0606 | 207      |
| 555-0607 | 208      |
| 555-0608 | 209      |
| 555-0609 | 210      |
| 555-0610 | 211      |
| 555-0611 | 212      |

**2.4 Create 10 business questions and provide the SQL codes and results. Submit the questions, SQL codes and the screenshots of the results. Use different types of joins, aggregation function, sub query, CHECK, stored procedure, trigger at least once. (30 points)**

#### 2.4.1 Show the occupancy rate (occupied/total apartments) for each building.

MySQL Workbench Screenshot:

**Navigator**

- SCHEMAS
  - pms
    - Tables: apartment, building, cleans, corpclient, corpclient\_phone, inspector, inspector\_phone, inspects, lease, maintenance\_requests, manager, manager\_phone, resident\_feedback, staff, staff\_phone, utility\_billing

**Editor**

```

1 /* Show the occupancy rate (occupied/total apartments) for each building. */
2 •  SELECT b.Building_ID,
3     COUNT(a.Apartment_ID) AS Total_Apartments,
4     SUM(CASE WHEN a.Rental_Status = 'Occupied' THEN 1 ELSE 0 END) AS Occupied,
5     ROUND((SUM(CASE WHEN a.Rental_Status = 'Occupied' THEN 1 ELSE 0 END) / COUNT(a.Apartment_ID)) * 100, 2) AS Occupancy_Rate
6   FROM BUILDING b
7   INNER JOIN APARTMENT a ON b.Building_ID = a.Building_ID
8 GROUP BY b.Building_ID;

```

**Result Grid**

| Building_ID | Total_Apartments | Occupied | Occupancy_Rate |
|-------------|------------------|----------|----------------|
| 3           | 2                | 1        | 50.00          |
| 4           | 2                | 1        | 50.00          |
| 5           | 2                | 1        | 50.00          |
| 6           | 2                | 1        | 50.00          |
| 7           | 2                | 1        | 50.00          |

## 2.4.2 List managers earning above average salary with their building addresses.

The screenshot shows the MySQL Workbench interface. On the left, the Navigator pane displays the schema structure under the 'pms' database, including tables like apartment, building, cleans, corpclient, corpclient\_phone, inspector, and inspector\_phone. Below this, the Information pane shows details for the 'utility\_billing' table, including its columns: UtilityID (int AI PK), Building\_ID (int), Apartment\_ID (int), UtilityType (enum('Elec...')), MonthlyCost (decimal(10,0)), and DueDate (date).

The main query editor window contains the following SQL code:

```

9
10  /* List managers earning above average salary with their building addresses. */
11 *  SELECT m.MFull_Name AS Manager_Name, m.Salary, CONCAT(b.Street, ' ', b.City, ' ', b.State, ' ', b.ZipCode) AS Building_Address
12   FROM MANAGER m
13  LEFT JOIN BUILDING b ON m.Building_ID = b.Building_ID
14  WHERE m.Salary > (SELECT AVG(Salary) FROM MANAGER);
15
16  /* Find leases with durations longer than 1 year. */

```

The Result Grid shows the output of the query:

| Manager_Name  | Salary | Building_Address                   |
|---------------|--------|------------------------------------|
| Mera Nair     | 75000  | 303 Elm St, Norwalk, CT 6851       |
| Krishna Verma | 73000  | 606 Pine Ave, New Haven, CT 6510   |
| Sai Prakash   | 74000  | 808 Cedar Blvd, Waterbury, CT 6702 |
| Diya Sharma   | 76000  | 909 Elm Ave, New London, CT 6320   |

## 2.4.3 Count maintenance requests by status per building.

The screenshot shows the MySQL Workbench interface. The Navigator pane displays the schema structure under the 'pms' database, including tables like apartment, building, cleans, corpclient, corpclient\_phone, inspector, and inspector\_phone. Below this, the Information pane shows details for the 'utility\_billing' table, including its columns: UtilityID (int AI PK), Building\_ID (int), Apartment\_ID (int), UtilityType (enum('Elec...')), MonthlyCost (decimal(10,0)), and DueDate (date).

The main query editor window contains the following SQL code:

```

23  /* Count maintenance requests by status per building. */
24 *  SELECT b.Building_ID,
25        mr.Status,
26        COUNT(mr.Request_ID) AS Total_Requests
27   FROM BUILDING b
28  INNER JOIN MAINTENANCE_REQUEST mr ON b.Building_ID = mr.Building_ID
29  GROUP BY b.Building_ID, mr.Status;
30

```

The Result Grid shows the output of the query:

| Building_ID | Status    | Total_Requests |
|-------------|-----------|----------------|
| 3           | Completed | 1              |
| 3           | Pending   | 1              |
| 4           | Completed | 1              |
| 4           | Pending   | 1              |
| 5           | Pending   | 1              |
| 5           | Completed | 1              |
| 6           | Pending   | 1              |
| 6           | Completed | 1              |
| 7           | Pending   | 2              |

#### 2.4.4 Show unresolved feedback with apartment details.

The screenshot shows the MySQL Workbench interface with the following details:

- Navigator:** Shows the schema **pms** with tables: apartment, building, cleans, corpclient, corpclient\_phone, inspector, and inspector\_phone.
- Query Editor:** A query titled "infoQuery\_GroupProjectPart1" is displayed:
 

```

31  /* Show unresolved feedback with apartment details. */
32  •  SELECT rf.FeedbackID, rf.Description, a.Building_ID, a.Apartment_ID
33  FROM RESIDENT_FEEDBACK rf
34  LEFT JOIN APARTMENT a ON rf.Building_ID = a.Building_ID AND rf.Apartment_ID = a.Apartment_ID
35  WHERE rf.Status = 'Open';
36
      
```
- Result Grid:** The results show 9 rows of feedback with their descriptions and associated apartment details:
 

| FeedbackID | Description                        | Building_ID | Apartment_ID |
|------------|------------------------------------|-------------|--------------|
| 1          | Gym equipment needs maintenance.   | 3           | 203          |
| 2          | Parties late at night.             | 3           | 204          |
| 4          | Construction noise in the morning. | 4           | 206          |
| 6          | Elevator noise.                    | 5           | 208          |
| 9          | Neighbors playing loud music.      | 7           | 211          |
- Table Definition:** The **utility\_billing** table is defined with columns:
 

| UtilityID | Building_ID | Apartment_ID | UtilityType | MonthlyCost | DueDate |
|-----------|-------------|--------------|-------------|-------------|---------|
|-----------|-------------|--------------|-------------|-------------|---------|

#### 2.4.5 Calculate total utility costs per building.

The screenshot shows the MySQL Workbench interface with the following details:

- Navigator:** Shows the schema **pms** with tables: apartment, building, cleans, corpclient, corpclient\_phone, inspector, and inspector\_phone.
- Query Editor:** A query titled "infoQuery\_GroupProjectPart1" is displayed:
 

```

37  /* Calculate total utility costs per building. */
38  •  SELECT b.Building_ID,
39            SUM(ub.MonthlyCost) AS Total_Utility_Cost
40  FROM BUILDING b
41  INNER JOIN UTILITY_BILLING ub ON b.Building_ID = ub.Building_ID
42  GROUP BY b.Building_ID;
43
44  /* List corporate clients who didn't refer anyone. */
      
```
- Result Grid:** The results show 7 buildings and their total utility costs:
 

| Building_ID | Total_Utility_Cost |
|-------------|--------------------|
| 3           | 240.00             |
| 4           | 270.00             |
| 5           | 180.00             |
| 6           | 240.00             |
| 7           | 285.00             |
- Table Definition:** The **utility\_billing** table is defined with columns:
 

| UtilityID | Building_ID | Apartment_ID | UtilityType | MonthlyCost | DueDate |
|-----------|-------------|--------------|-------------|-------------|---------|
|-----------|-------------|--------------|-------------|-------------|---------|

#### 2.4.6 List corporate clients who didn't refer anyone.

The screenshot shows the MySQL Workbench interface. On the left, the Navigator pane displays the schema structure under the 'pms' database, including tables like apartment, building, cleans, corpclient, corpclient\_phone, inspector, and inspector\_shm. The central area shows a query editor window titled 'infoQuery\_GroupProjectPart1' with the following SQL code:

```
43
44  /* List corporate clients who didn't refer anyone. */
45  SELECT Client_ID AS CCID, Client_Name AS CCName
46  FROM CORPCLIENT
47  WHERE Client_ID NOT IN (SELECT Refer_Client_ID FROM CORPCLIENT WHERE Refer_Client_ID IS NOT NULL);
48
```

The result grid shows one row of data:

| CCID | CCName               |
|------|----------------------|
| 513  | Bright Future School |

#### 2.4.7 Count inspections per inspector.

The screenshot shows the MySQL Workbench interface. On the left, the Navigator pane displays the schema structure under the 'pms' database, including tables like apartment, building, cleans, corpclient, corpclient\_phone, inspector, and inspector\_shm. The central area shows a query editor window titled 'infoQuery\_GroupProjectPart1' with the following SQL code:

```
49  /* Count inspections per inspector. */
50  SELECT i.InspectorName AS Inspector_Name, COUNT(ins.Inspector_ID) AS Total_Inspections
51  FROM INSPECTOR i
52  LEFT JOIN INSPECTS ins ON i.Inspector_ID = ins.Inspector_ID
53  GROUP BY i.InspectorName;
54
```

The result grid shows the count of inspections for each inspector:

| Inspector_Name | Total_Inspections |
|----------------|-------------------|
| Ritika Das     | 1                 |
| Kabir Malhotra | 1                 |
| Leena Bhatt    | 1                 |
| Aditya Singh   | 1                 |
| Tanvi Saxena   | 1                 |
| Mohit Jain     | 1                 |
| Nidhi Kapoor   | 1                 |
| Ajay Deshmukh  | 1                 |
| Simran Kohli   | 1                 |
| Dev Arora      | 1                 |

## 2.4.8 Create a trigger to mark apartments as occupied when leased.

The screenshot shows the MySQL Workbench interface with the Navigator pane on the left displaying the 'Schemas' tree under the 'pms' schema, which contains tables like apartment, building, cleans, corpclient, corpclient\_phone, and inspector. The main query editor window titled 'infoQuery\_GroupProjectPart1' contains the following SQL code:

```
65  AFTER INSERT ON LEASE
66  FOR EACH ROW
67  BEGIN
68      UPDATE APARTMENT
69      SET Rental_Status = 'Occupied'
70      WHERE Building_ID = NEW.Building_ID AND Apartment_ID = NEW.Apartment_ID;
71  END;
72  //
```

Below the code, the history pane shows several previous queries and the creation of the trigger:

```
77 09:06:00 SELECT Client_ID AS CCID, Client_Name AS CCName FROM CORPCLIENT WHERE Client_ID NOT IN (SELECT Refer_Client_ID FROM CORPCLIENT WHERE Refer_Client_ID IS NOT NULL) LIMIT 0, 1000
78 09:06:18 SELECT i.InspectorName AS Inspector_Name, COUNT(i.Inspector_ID) AS Total_Inspections FROM INSPECTOR i LEFT JOIN INSPECTS ins ON i.Inspector_ID = ins.Inspector_ID GROUP BY i.InspectorName LIMIT 0, 1000
79 09:06:39 SELECT s.Staff_Name AS Staff_Name, COUNT(s.Apartment_ID) AS Total_Assignments FROM STAFF s INNER JOIN CLEANS ca ON s.Staff_ID = ca.Staff_ID GROUP BY s.Staff_Name HAVING Total_Assignments > 1 LIMIT 0, 1000
80 [09:06:51] CREATE TRIGGER UpdateApartmentStatus AFTER INSERT ON LEASE FOR EACH ROW BEGIN UPDATE APARTMENT SET Rental_Status = 'Occupied' WHERE Building_ID = NEW.Building_ID AND Apartment_ID = NEW.Apartment_ID; END;
```

## 2.4.9 Automatically log completed maintenance requests

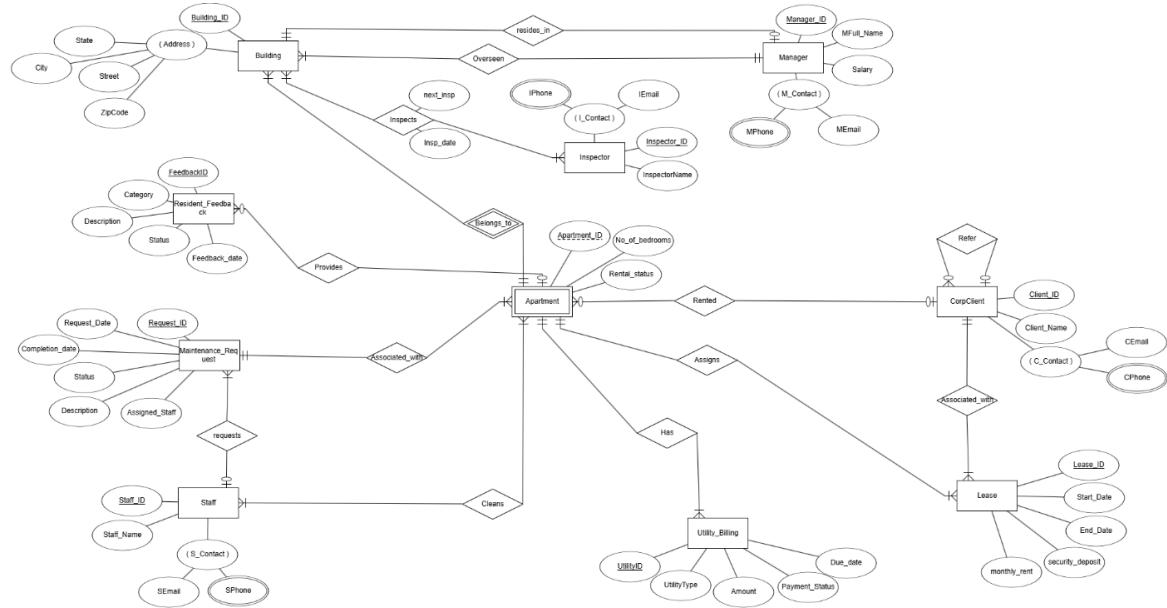
The screenshot shows the MySQL Workbench interface with the Navigator pane on the left displaying the 'Schemas' tree under the 'pms' schema, which contains tables like apartment, building, cleans, corpclient, corpclient\_phone, and inspector. The main query editor window titled 'infoQuery\_GroupProjectPart1' contains the following SQL code:

```
104  DELIMITER //
105  * CREATE TRIGGER LogCompletedMaintenance
106  AFTER UPDATE ON MAINTENANCE_REQUEST
107  FOR EACH ROW
108  BEGIN
109      IF NEW.Status = 'Completed' AND OLD.Status != 'Completed' THEN
110          INSERT INTO MAINTENANCE_LOG (Request_ID, CompletionDate)
111          VALUES (NEW.Request_ID, NOW());
112      END IF;
113  END;
```

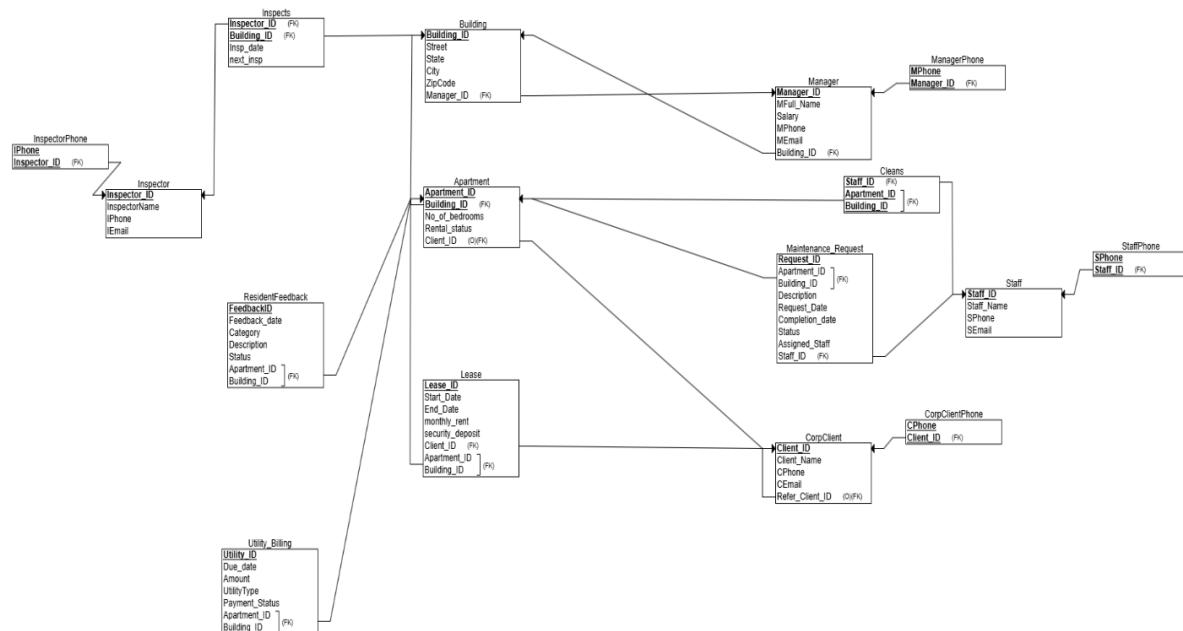
Below the code, the history pane shows several previous queries and the creation of the trigger:

```
79 09:06:39 SELECT s.Staff_Name AS Staff_Name, COUNT(s.Apartment_ID) AS Total_Assignments FROM STAFF s INNER JOIN CLEANS ca ON s.Staff_ID = ca.Staff_ID GROUP BY s.Staff_Name HAVING Total_Assignments > 1 LIMIT 0, 1000
80 09:06:51 CREATE TRIGGER UpdateApartmentStatus AFTER INSERT ON LEASE FOR EACH ROW BEGIN UPDATE APARTMENT SET Rental_Status = 'Occupied' WHERE Building_ID = NEW.Building_ID AND Apartment_ID = NEW.Apartment_ID; END;
81 09:09:47 SELECT s.Staff_Name AS Staff_Name, COUNT(DISTINCT ca.Building_ID) AS Buildings_Covered FROM STAFF s JOIN CLEANS ca ON s.Staff_ID = ca.Staff_ID GROUP BY s.Staff_Name HAVING Buildings_Covered > 1 LIMIT 0, 1000
82 [09:10:56] CREATE TRIGGER LogCompletedMaintenance AFTER UPDATE ON MAINTENANCE_REQUEST FOR EACH ROW BEGIN IF NEW.Status = 'Completed' AND OLD.Status != 'Completed' THEN INSERT INTO MAINTENANCE_LOG (Request_ID, CompletionDate) VALUES (NEW.Request_ID, NOW()); END IF;
```

## 3.0 Entity-Relationship: Property Management System: 2 New Entities



### 3.1 Relational-Schema: PROPERTY MANAGEMENT SYSTEM: 2 NEW ENTITIES



## 3.2 DATA-DICTIONARY: PROPERTY MANAGEMENT SYSTEM: 2 NEW ENTITIES

| Entity:      | RESIDENT FEEDBACK TABLE     | Track utility billing of apartments |                               |         |          |        |        |  |
|--------------|-----------------------------|-------------------------------------|-------------------------------|---------|----------|--------|--------|--|
| Field Name   | Description                 | Type                                | Specifications                | Default | Required | Unique | Key(s) |  |
| FeedbackID   | Unique feedback ID          | INT                                 | Auto-increment                | -       | YES      | YES    | PK     |  |
| BuildingID   | Building ID                 | INT                                 | Links to Apartment            | -       | YES      | NO     | -      |  |
| AptNo        | Apartment number            | INT                                 | Links to Apartment            | -       | YES      | NO     | -      |  |
| FeedbackDate | Date of feedback submission | DATE                                | Format: YYYY-MM-DD            | -       | YES      | NO     | -      |  |
| Category     | Feedback category           | ENUM                                | MalINTenance, Noise, Facility | -       | YES      | NO     | -      |  |
| Description  | Feedback details            | TEXT                                |                               | -       | YES      | NO     | -      |  |
| Status       | Feedback resolution status  | VARCHAR(10)                         | CHECK: Open/Resolved          | -       | YES      | NO     | -      |  |
| Entity:      | UTILITY BILLING             | Track feedbacks from residents      |                               |         |          |        |        |  |
| Field Name   | Description                 | Type                                | Specifications                | Default | Required | Unique | Key(s) |  |
| UtilityID    | Unique utility bill ID      | INT                                 | Auto-increment                | -       | YES      | YES    | PK     |  |
| BuildingID   | Building ID                 | INT                                 | Links to Apartment            | -       | YES      | NO     | -      |  |
| AptNo        | Apartment number            | INT                                 | Links to Apartment            | -       | YES      | NO     | -      |  |
| UtilityType  | Type of utility             | ENUM                                | Electricity, Water, Gas       | -       | YES      | NO     | -      |  |
| MonthlyCost  | Monthly utility cost        | DECIMAL(10,2)                       | Positive value (e.g., 150.00) | -       | YES      | NO     | -      |  |
| DueDate      | Payment due date            | DATE                                | Format: YYYY-MM-DD            | -       | YES      | NO     | -      |  |

## 3.3 BUSINESS QUERY: 2 NEW ENTITIES – RESIDENT FEEDBACK AND UTILITY BILLING

### 3.3.1 High Utility Cost Apartments with Pending Feedback

The screenshot shows a database interface with a query editor and a results grid.

**Query Editor:**

```

SELECT
    ub.Apartment_ID,
    ub.Building_ID,
    FORMAT(ub.MonthlyCost, 2) AS Current_UtilityCost,
    (
        SELECT FORMAT(AVG(MonthlyCost), 2)
        FROM UTILITY_BILLING
        WHERE UtilityType = ub.UtilityType
    ) AS Type_Avg,
    (
        SELECT GROUP_CONCAT(Description SEPARATOR ' | ')
        FROM RESIDENT_FEEDBACK
        WHERE Apartment_ID = ub.Apartment_ID
        AND Building_ID = ub.Building_ID
        AND Status = 'Open'
    ) AS Open_Issues
    FROM UTILITY_BILLING ub
    WHERE ub.MonthlyCost > (
        SELECT AVG(MonthlyCost)
        FROM UTILITY_BILLING
        WHERE UtilityType = ub.UtilityType
    )
    AND EXISTS (
        SELECT 1
        FROM RESIDENT_FEEDBACK
        WHERE Apartment_ID = ub.Apartment_ID
        AND Building_ID = ub.Building_ID
        AND Status = 'Open'
    )
    ORDER BY ub.MonthlyCost DESC;

```

**Results Grid:**

| Apartment_ID | Building_ID | Current_UtilityCost | Type_Avg | Open_Issues                        |
|--------------|-------------|---------------------|----------|------------------------------------|
| 206          | 4           | 170.00              | 163.75   | Construction noise in the morning. |
| 211          | 7           | 110.00              | 101.67   | Neighbors playing loud music.      |

### 3.3.2 Utility payment compliance check

Navigator      BusinessQuery\_2\_GroupProject... x

**SCHEMAS**

Filter objects

**pms**

- Tables
  - apartment
  - building
  - cleans
  - corpclient
  - corpclient\_phone
  - inspector
  - inspector\_phone
  - inspects
  - lease
  - maintenance\_request
  - manager
  - manager\_phone
  - resident\_feedback
  - staff
  - staff\_phone
  - utility\_billing
- Views
- Stored Procedures
- Functions

sakila

sys

world

**BusinessQuery\_2\_GroupProject... x**

```

1 •   SELECT
2     ub.Building_ID,
3     ub.Apartment_ID,
4     ub.UtilityType,
5     FORMAT(ub.MonthlyCost, 2) AS Due_Amount,
6     ub.DueDate,
7     CASE
8       WHEN DATEDIFF(CURDATE(), ub.DueDate) > 30 THEN 'Delinquent'
9       WHEN DATEDIFF(CURDATE(), ub.DueDate) > 15 THEN 'Overdue'
10      ELSE 'Current'
11    END AS Payment_Status,
12    (
13      SELECT COUNT(*)
14      FROM RESIDENT_FEEDBACK
15      WHERE Apartment_ID = ub.Apartment_ID
16          AND Building_ID = ub.Building_ID
17          AND Category = 'Facility'
18    ) AS Facility_Complaints
19  FROM UTILITY_BILLING ub
20  ORDER BY Payment_Status, DueDate DESC;
21

```

Administration Schemas

Information

Result Grid | Filter Rows: | Export: | Wrap Cell Content: |

Schema: pms

|   | Building_ID | Apartment_ID | UtilityType | Due_Amount | DueDate    | Payment_Status | Facility_Complaints |
|---|-------------|--------------|-------------|------------|------------|----------------|---------------------|
| ▶ | 7           | 211          | Gas         | 110.00     | 2023-10-01 | Delinquent     | 0                   |
|   | 7           | 212          | Electricity | 175.00     | 2023-10-01 | Delinquent     | 0                   |
|   | 6           | 209          | Electricity | 150.00     | 2023-09-01 | Delinquent     | 0                   |
|   | 6           | 210          | Water       | 90.00      | 2023-09-01 | Delinquent     | 1                   |
|   | 5           | 207          | Water       | 85.00      | 2023-08-01 | Delinquent     | 1                   |
|   | 5           | 208          | Gas         | 95.00      | 2023-08-01 | Delinquent     | 0                   |
|   | 4           | 205          | Gas         | 100.00     | 2023-07-01 | Delinquent     | 0                   |
|   | 4           | 206          | Electricity | 170.00     | 2023-07-01 | Delinquent     | 0                   |
|   | 3           | 203          | Electricity | 160.00     | 2023-06-01 | Delinquent     | 1                   |
|   | 3           | 204          | Water       | 80.00      | 2023-06-01 | Delinquent     | 0                   |

### 3.3.3 Apartments with Unresolved Feedback & High Utility Costs

The screenshot shows a database interface with a query editor and a results grid.

**Navigator:** Shows the schema **pms** with tables: apartment, building, cleans, corpclient, corpclient\_phone, inspector, inspector\_phone, inspects, lease, maintenance\_requests, manager, manager\_phone, resident\_feedback, staff, staff\_phone.

**BusinessQuery\_3\_GroupProject...:** A query window titled "BusinessQuery\_3\_GroupProject..." containing the following SQL code:

```
1 *   SELECT
2       rf.Apartment_ID,
3       rf.Building_ID,
4       rf.Description AS Open_Issue,
5       FORMAT(ub.MonthlyCost, 2) AS Utility_Cost
6   FROM RESIDENT_FEEDBACK rf
7   JOIN UTILITY_BILLING ub
8       ON rf.Apartment_ID = ub.Apartment_ID
9       AND rf.Building_ID = ub.Building_ID
10      WHERE rf.Status = 'Open'
11      AND ub.MonthlyCost > (
12          SELECT AVG(MonthlyCost)
13          FROM UTILITY_BILLING
14      );
15
```

**Information:** Shows the schema **pms**.

**Result Grid:** A table displaying the results of the query:

|   | Apartment_ID | Building_ID | Open_Issue                         | Utility_Cost |
|---|--------------|-------------|------------------------------------|--------------|
| ▶ | 203          | 3           | Gym equipment needs maintenance.   | 160.00       |
| ▶ | 206          | 4           | Construction noise in the morning. | 170.00       |

### 3.3.4 Utility Costs vs Building Average

The screenshot shows the Business Studio interface with the Navigator and BusinessQuery\_4\_GroupProject... tabs active. The Navigator pane displays the 'pms' schema with its tables: apartment, building, cleans, corpclient, corpclient\_phone, inspector, inspector\_phone, inspects, lease, maintenance\_requests, manager, manager\_phone, resident\_feedback, staff, and staff\_phone. The BusinessQuery\_4\_GroupProject... tab contains the following SQL code:

```

1 •   SELECT
2     | ub.Apartment_ID,
3     | ub.Building_ID,
4     | FORMAT(ub.MonthlyCost, 2) AS Current_Cost,
5     | FORMAT(
6     |   (
7         |     SELECT AVG(MonthlyCost)
8         |     FROM UTILITY_BILLING
9         |     WHERE Building_ID = ub.Building_ID
10    |   ), 2
11  | ) AS Building_Avg_Cost
12  | FROM UTILITY_BILLING ub
13  | WHERE ub.MonthlyCost > (
14    |   SELECT AVG(MonthlyCost)
15    |   FROM UTILITY_BILLING
16    |   WHERE Building_ID = ub.Building_ID
17  );

```

The Result Grid shows the following data:

|   | Apartment_ID | Building_ID | Current_Cost | Building_Avg_Cost |
|---|--------------|-------------|--------------|-------------------|
| ▶ | 203          | 3           | 160.00       | 120.00            |
|   | 206          | 4           | 170.00       | 135.00            |
|   | 208          | 5           | 95.00        | 90.00             |
|   | 209          | 6           | 150.00       | 120.00            |
|   | 212          | 7           | 175.00       | 142.50            |

### 3.3.5 Open Feedback Count with Total Utility Cost

The screenshot shows the Business Studio interface with the Navigator and BusinessQuery\_5\_GroupProject... tabs active. The Navigator pane displays the 'pms' schema with its tables: apartment, building, cleans, corpclient, corpclient\_phone, inspector, inspector\_phone, inspects, lease, maintenance\_requests, manager, manager\_phone, resident\_feedback, staff, and staff\_phone. The BusinessQuery\_5\_GroupProject... tab contains the following SQL code:

```

1 •   SELECT
2     | ub.Apartment_ID,
3     | ub.Building_ID,
4     | FORMAT(SUM(ub.MonthlyCost), 2) AS Total_Utility,
5     | (
6       |   SELECT COUNT(*)
7       |   FROM RESIDENT_FEEDBACK
8       |   WHERE Apartment_ID = ub.Apartment_ID
9       |   AND Building_ID = ub.Building_ID
10      |   AND Status = 'Open'
11    | ) AS Open_Feedback_Count
12  | FROM UTILITY_BILLING ub
13  | GROUP BY ub.Apartment_ID, ub.Building_ID
14  | HAVING Open_Feedback_Count > 0;

```

The Result Grid shows the following data:

|   | Apartment_ID | Building_ID | Total_Utility | Open_Feedback_Count |
|---|--------------|-------------|---------------|---------------------|
| ▶ | 203          | 3           | 160.00        | 1                   |
|   | 204          | 3           | 80.00         | 1                   |
|   | 206          | 4           | 170.00        | 1                   |
|   | 208          | 5           | 95.00         | 1                   |
|   | 211          | 7           | 110.00        | 1                   |

This project successfully addressed the property management company's operational and analytical needs by modernizing its database infrastructure.

## **Part 1: Operational Database Upgrade**

### **1. Enhanced Functionality:**

- The upgraded database now efficiently tracks leases and maintenance requests, streamlining rental management and repair workflows.
- New entities like Resident Feedback and Utility Billing were added to improve tenant satisfaction monitoring and cost tracking.

### **2. Data Integrity & Efficiency:**

- Triggers (e.g., auto-updating apartment vacancy status) and stored procedures (e.g., rent revenue calculation) reduced manual effort.
- Referential integrity was enforced through foreign keys and constraints (e.g., CHECK clauses for valid statuses).

### **3. Business Insight**

- Queries using joins, aggregations, and sub queries answered critical questions, such as identifying unoccupied apartments or analyzing referral trends.

### **1. Scalability:**

- The relational schema and ER diagram ensure flexibility to accommodate future expansions, such as integrating IoT devices for smart buildings.

### **Impact - The upgraded system will:**

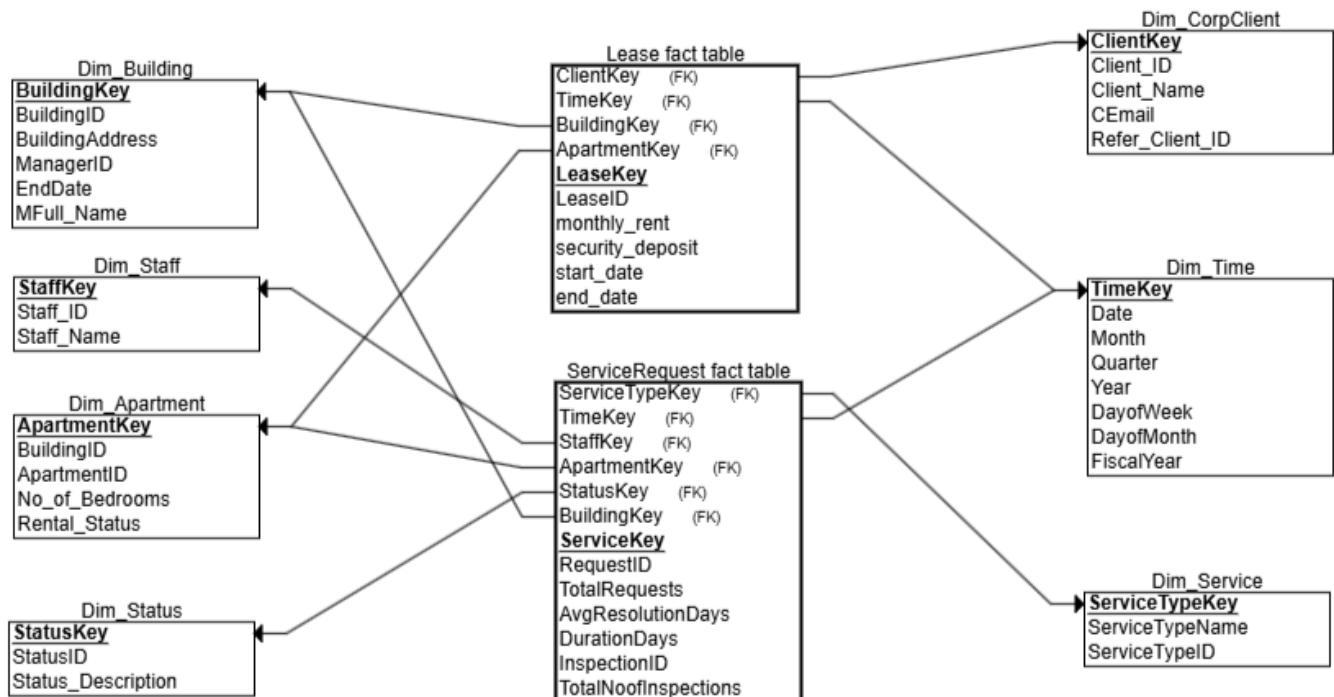
- Reduce vacancies by identifying leasing trends and high-demand apartment features.
- Lower operational costs through predictive maintenance alerts and utility usage optimization.
- Improve tenant retention by resolving feedback faster and ensuring compliance with safety inspections.
- Support strategic growth by analyzing referral patterns and corporate client preferences.

By bridging operational efficiency with analytical depth, this database redesign positions the company to thrive in a competitive real estate market while maintaining scalability for future innovations.

# PART 2

## 4.0 Fact and Dimension Schema Design

This schema is designed using a star structure to support efficient analytical processing within a property management data warehouse. At the center are two fact tables that track key business processes—rental activity and service requests—each surrounded by multiple descriptive dimensions. The rental-related table records lease transactions across time, units, and tenants, capturing metrics such as rent, deposits, and lease durations. The service-related table tracks maintenance activities and operational performance indicators including request volume, resolution times, and inspection counts. Dimensions enrich the analysis by providing attributes for time periods, property structures, units, personnel, client relationships, and service categories. This model enables flexible slicing and historical trend analysis across various perspectives such as time, location, client, or operational category, supporting informed decision-making in real estate operations.



### 4.1 Executive Summary

To support strategic decision-making and historical analysis, a data warehouse was designed and implemented for a property management company. This warehouse enables executives, managers, and analysts to monitor performance indicators such as rental income, occupancy trends, maintenance efficiency, and inspection patterns over time. The solution leverages

dimensional modeling and ETL processes to transform operational data into a structured, analytics-ready format that supports business intelligence and reporting needs.

## 5.0 Fact Tables:

The data warehouse includes two primary fact tables that capture the core transactional processes of the property management system:

### 5.1 Lease Transaction Fact Table

The **Lease Transaction Fact** table serves as the central hub for tracking all leasing activities within the property management system. Each record in this table represents a unique lease agreement between a corporate tenant and a property unit (apartment).

- **Structure:**
  - The primary key, LeaseKey, uniquely identifies each lease transaction.
  - Foreign keys such as ClientKey, TimeKey, BuildingKey, and ApartmentKey link this fact table to dimension tables like Corporate Client, Time, Building, and Apartment, respectively. These connections allow analysts to contextualize leases by tenant, location, and time.
  - Key attributes include LeaseID (the original lease identifier from the source system), monthly\_rent (the recurring payment amount), security\_deposit (the upfront deposit held for the lease), and start\_date/end\_date (defining the lease term).
- **Business Purpose:**

This table enables stakeholders to analyze rental income trends, occupancy rates, and tenant retention. For example, by joining with the Time Dimension, managers can compare quarterly revenue across buildings or identify seasonal fluctuations in lease signings. It also supports compliance reporting by tracking deposit amounts and lease durations.

### 5.2 Service Request Management Fact Table

The Service Request Management Fact table captures all maintenance and service requests raised by tenants or property managers. It provides insights into operational efficiency and resource allocation.

- **Structure:**
  - The primary key, ServiceKey, uniquely identifies each service request.
  - Foreign keys like ServiceTypeKey, TimeKey, StaffKey, ApartmentKey, StatusKey, and BuildingKey connect to dimensions such as Service Type, Time, Staff, Apartment, Status, and Building. These relationships allow for granular analysis of request types, resolution times, and staff performance.
  - Metrics such as TotalRequests (aggregating requests by category), AvgResolutionDays (average time to resolve issues), and DurationDays (time taken for a specific request) quantify operational efficiency.
- **Business Purpose:**

This table helps identify recurring maintenance issues (e.g., plumbing failures in older buildings) and evaluate staff productivity. For instance, by linking to the Staff Dimension, managers can determine which employees resolve requests fastest or handle the most complex tasks. It also supports root-cause analysis by correlating service types (e.g., electrical repairs) with apartment attributes (e.g., building age).

## 6.0 Database Schema Recap: Property Management System

This schema supports a comprehensive property management system, covering building administration, leasing, maintenance, staff, inspections, and tenant interactions.

**Manager & Building:** The `MANAGER` table stores manager details, each linked to a building via `BUILDING`, which captures location and structural attributes. Managers can also be contacted through multiple numbers stored in `MANAGER_PHONE`.

**Corporate Clients:** `CORPCLIENT` holds tenant company information, including referral links to other clients. Their contact numbers are maintained in `CORPCLIENT_PHONE`.

**Apartments & Leasing:** The `APARTMENT` table defines each unit by its building, bedroom count, and rental status. Leasing agreements are tracked in the `LEASE` table with terms, rent, deposits, and associated clients.

**Staff & Cleaning:** Staff records are maintained in `STAFF` with contact data in `STAFF_PHONE`. Cleaning responsibilities are mapped through the `CLEANS` table, linking staff to specific apartments.

**Inspection Management:** The `INSPECTOR` and `INSPECTS` tables manage inspection personnel and their assignments to buildings, including inspection and follow-up dates. Inspector contacts are stored in `INSPECTOR_PHONE`.

**Maintenance Requests:** `MAINTENANCE_REQUEST` logs service requests, resolution durations, and staff assignments, supporting operational tracking and performance analysis.

**Resident Feedback & Utilities:** Tenant-submitted issues are captured in `RESIDENT_FEEDBACK`, categorized and tracked by status. The `UTILITY_BILLING` table logs apartment-level utility costs, types, and due dates for billing purposes.

This relational design ensures robust data integrity, traceability across operations, and strong analytical potential for building management and service delivery.

## 7.0 Data Warehouse Schema

The schema is designed using a dimensional star model to support analytical reporting for lease agreements and service request management within the housing system. It includes clearly defined dimension and fact tables to enable time-based, client-centric, and operational analysis.

**Dim\_CorpClient:** Captures corporate tenant details including IDs, names, contact emails, and referral links. Acts as a key entity for lease and client-based analysis.

**Dim\_Time:** Stores temporal breakdowns such as date, month, quarter, year, day of the week, and fiscal year. Supports time-based slicing of metrics in both fact tables.

**Dim\_Building:** Represents building entities with address, manager details, and end-of-use dates. Enables performance and usage tracking by property location and assigned managers.

**Dim\_Staff:** Stores maintenance or service staff identities for productivity and assignment analysis across requests.

**Dim\_Apartment:** Defines each apartment by building ID, apartment number, bedroom count, and rental status. Serves as the property unit reference in both lease and service transactions.

**Dim\_Status:** Classifies service request states (e.g., open, closed, pending), useful for monitoring request pipelines and backlogs.

**Dim\_Service:** Describes types of services or maintenance issues (e.g., electrical, plumbing), enabling breakdowns of operational demand.

**LeaseFact**: Central fact table for lease agreements, linking corporate clients, properties, and lease periods. Tracks lease IDs, monthly rent, deposits, and lease duration, enabling analysis of revenue and occupancy trends.

**ServiceRequestFact**: Logs all tenant or manager-raised service requests with metrics such as total requests, resolution times, and inspections. Links to staff, apartments, buildings, time, status, and service types for end-to-end performance insights.

## 7.1 Creating Dimension & Fact Tables

The screenshot shows a database management interface with a 'Navigator' sidebar on the left and a main 'create\_dim\_projectpart2' query editor on the right. The 'Navigator' sidebar displays the 'SCHEMAS' section with several schemas listed: pms, pms\_backup, pms\_p2, pms\_part2, sakila, sys, and world. The 'pms\_part2' schema is expanded, showing its sub-objects: Tables, Views, Stored Procedures, and Functions. The main query editor contains the following SQL code:

```
1 -- Dim_Building: Stores flattened building + manager info for analytics
2 * CREATE TABLE Dim_Building (
3     BuildingKey INT AUTO_INCREMENT PRIMARY KEY,          -- Surrogate key for DW
4     BuildingID INT NOT NULL UNIQUE,                     -- Natural/business key
5     Street VARCHAR(30),
6     City VARCHAR(20),
7     State VARCHAR(20),
8     ZipCode INT,
9     ManagerID INT,                                     -- From operational schema
10    ManagerName VARCHAR(30),                           -- Concatenated full name
11    ManagerEmail VARCHAR(50)
12 );
13
14 -- Dim_Staff: Contains staff identity and contact info
15 * DROP TABLE IF EXISTS Dim_Staff;
16 * CREATE TABLE Dim_Staff (
17     StaffKey INT AUTO_INCREMENT PRIMARY KEY,           -- Natural/business key
18     StaffID INT NOT NULL UNIQUE,
19     StaffName VARCHAR(30),
20     StaffEmail VARCHAR(100)
21 );
22
23 -- Dim_Apartment: Apartment units and status
24 * CREATE TABLE Dim_Apartment (
25     ApartmentKey INT AUTO_INCREMENT PRIMARY KEY,      -- Apartment number
26     ApartmentID INT,                                 -- FK to building
27     BuildingID INT,
28     NoOfBedrooms INT,
29     RentalStatus VARCHAR(10)                         -- e.g., 'Vacant', 'Occupied'
30 );
31
32 -- Dim_Status: Lookup for service request statuses
33 * CREATE TABLE Dim_Status (
34     StatusKey INT AUTO_INCREMENT PRIMARY KEY,         -- e.g., 1=Open
35     StatusID INT NOT NULL UNIQUE,                    -- e.g., 'Completed'
36     StatusDescription VARCHAR(20)
37 );
```

**Navigator**

**SCHEMAS**

- pms
- pms\_backup
- pms\_p2
- pms\_part2**
  - Tables
  - Views
  - Stored Procedures
  - Functions
- sakila
- sys
- world

**Administration Schemas**

**Information**

**Schema:** pms\_part2

**create\_dim\_projectpart2\***

```

40 • CREATE TABLE Dim_CorpClient (
41     ClientKey INT AUTO_INCREMENT PRIMARY KEY,
42     ClientID INT NOT NULL UNIQUE,
43     ClientName VARCHAR(30),
44     ClientEmail VARCHAR(20),                                -- May want to increase length
45     ReferredClientID INT                                -- FK to self for referral tracking
46 );
47     -- Dim_Time: Date-related breakdown for analysis
48 • CREATE TABLE Dim_Time (
49     TimeKey INT AUTO_INCREMENT PRIMARY KEY,
50     Date DATE NOT NULL UNIQUE,                            -- Calendar date
51     Year INT,
52     Month INT,
53     Quarter INT,
54     DayOfWeek INT,                                     -- 1=Sunday, 7=Saturday
55     DayOfMonth INT,
56     FiscalYear INT                                    -- Varies by org, e.g., April-March
57 );
58     -- Dim_Service: Types of service events
59 • CREATE TABLE Dim_Service (
60     ServiceTypeKey INT AUTO_INCREMENT PRIMARY KEY,
61     ServiceTypeID INT NOT NULL UNIQUE,                  -- Natural key, e.g., 1=Plumbing
62     ServiceTypeName VARCHAR(30)
63 );
64     -- FACT TABLES
65     -- Lease_Fact: Captures leasing transactions
66 • CREATE TABLE Lease_Fact (
67     LeaseKey INT AUTO_INCREMENT PRIMARY KEY,
68     ClientKey INT,                                      -- FK to Dim_CorpClient
69     TimeKey INT,                                       -- FK to Dim_Time
70     BuildingKey INT,                                  -- FK to Dim_Building
71     ApartmentKey INT,                                 -- FK to Dim_Apartment
72     LeaseID INT,                                     -- Operational Lease ID
73     MonthlyRent DECIMAL(10,2),
74     SecurityDeposit DECIMAL(10,2),
75     StartDate DATE,
76     EndDate DATE,
77     -- Foreign Keys
78     FOREIGN KEY (ClientKey) REFERENCES Dim_CorpClient(ClientKey),
79     FOREIGN KEY (TimeKey) REFERENCES Dim_Time(TimeKey),
80     FOREIGN KEY (BuildingKey) REFERENCES Dim_Building(BuildingKey),
81     FOREIGN KEY (ApartmentKey) REFERENCES Dim_Apartment(ApartmentKey)
82 );

```

Navigator: create\_dim\_projectpart2\*

**SCHEMAS**

Filter objects

- pms
- pms\_backup
- pms\_p2
- pms\_part2**
  - Tables
  - Views
  - Stored Procedures
  - Functions
- sakila
- sys
- world

```

84      -- ServiceRequest_Fact: Tracks maintenance, cleaning, etc.
85  CREATE TABLE ServiceRequest_Fact (
86      ServiceKey INT AUTO_INCREMENT PRIMARY KEY,
87      ServiceTypeKey INT,                                -- e.g., Plumbing
88      TimeKey INT,                                     -- Request date
89      StaffKey INT,                                    -- Who handled it
90      ApartmentKey INT,
91      StatusKey INT,
92      BuildingKey INT,
93      RequestID INT,
94      TotalRequests INT,                             -- Typically = 1
95      AvgResolutionDays DECIMAL(5,2),
96      DurationDays INT,
97      -- Foreign Keys
98      FOREIGN KEY (ServiceTypeKey) REFERENCES Dim_Service(ServiceTypeKey),
99      FOREIGN KEY (TimeKey) REFERENCES Dim_Time(TimeKey),
100     FOREIGN KEY (StaffKey) REFERENCES Dim_Staff(StaffKey),
101     FOREIGN KEY (ApartmentKey) REFERENCES Dim_Apartment(ApartmentKey),
102     FOREIGN KEY (StatusKey) REFERENCES Dim_Status(StatusKey),
103     FOREIGN KEY (BuildingKey) REFERENCES Dim_Building(BuildingKey)
104 );

```

### 7.1.1 Success Log

Output: Action Output

| #    | Time     | Action  |
|------|----------|---|
| ✓ 1  | 15:15:01 | CREATE TABLE Dim_Building ( BuildingKey INT AUTO_INCREMENT PRIMARY KEY, -- Surrogate key for DW BuildingID INT NC |
| ⚠ 2  | 15:15:05 | DROP TABLE IF EXISTS Dim_Staff  |
| ✓ 3  | 15:15:08 | CREATE TABLE Dim_Staff ( StaffKey INT AUTO_INCREMENT PRIMARY KEY, StaffID INT NOT NULL UNIQUE, -- Nat             |
| ✓ 4  | 15:15:13 | CREATE TABLE Dim_Apartment ( ApartmentKey INT AUTO_INCREMENT PRIMARY KEY, ApartmentID INT, -- I                   |
| ✓ 5  | 15:15:17 | CREATE TABLE Dim_Status ( StatusKey INT AUTO_INCREMENT PRIMARY KEY, StatusID INT NOT NULL UNIQUE, -               |
| ✓ 6  | 15:15:20 | CREATE TABLE Dim_CorpClient ( ClientKey INT AUTO_INCREMENT PRIMARY KEY, ClientID INT NOT NULL UNIQUE, ClientN     |
| ✓ 7  | 15:15:23 | CREATE TABLE Dim_Time ( TimeKey INT AUTO_INCREMENT PRIMARY KEY, Date DATE NOT NULL UNIQUE, -- C                   |
| ✓ 8  | 15:15:29 | CREATE TABLE Dim_Service ( ServiceTypeKey INT AUTO_INCREMENT PRIMARY KEY, ServiceTypeID INT NOT NULL UNIQU        |
| ✓ 9  | 15:15:33 | CREATE TABLE Lease_Fact ( LeaseKey INT AUTO_INCREMENT PRIMARY KEY, ClientKey INT, -- FK to Dim,                   |
| ✓ 10 | 15:15:38 | CREATE TABLE ServiceRequest_Fact ( ServiceKey INT AUTO_INCREMENT PRIMARY KEY, ServiceTypeKey INT,                 |

## 7.2 Populating the warehouse

The screenshot shows the SQL Server Management Studio interface. The left pane, titled 'Navigator', displays the database schema for 'pms\_part2'. It includes sections for 'SCHEMAS' (containing 'pms', 'pms\_backup', 'pms\_p2', and 'pms\_part2'), 'Tables' (containing 'apartment', 'building', 'corporateclient', 'dim\_apartment', 'dim\_building', 'dim\_corpclient', 'dim\_service', 'dim\_staff', 'dim\_status', 'dim\_time', 'lease', 'lease\_fact', 'maintenancerequest', 'manager', 'servicerequest\_fact', and 'staff'), and 'Views', 'Stored Procedures', and 'Functions'. Below these are 'sakila', 'sys', and 'world'. The 'Information' tab is also visible. The right pane, titled 'create\_dim\_projectpart2' and 'Insert\_dim\_projectpart2', contains a SQL script for populating the 'Dim\_Building', 'Dim\_Staff', 'Dim\_Apartment', 'Dim\_Status', and 'Dim\_CorpClient' tables. The script uses INSERT INTO statements with SELECT clauses to copy data from various source tables like 'Building', 'Manager', 'Apartment', 'Status', and 'CorporateClient' into their respective dimensions. The code includes several 'LIMIT 10' statements to insert only the first 10 records of each dimension.

```
89
90 *   INSERT INTO Dim_Building (BuildingID, Street, City, State, ZipCode, ManagerID, ManagerName, ManagerEmail)
91     SELECT
92         b.Building_ID,
93         b.Street,
94         b.City,
95         b.State,
96         b.ZipCode,
97         m.Manager_ID,
98         m.MFull_Name,
99         m.MEmail
100    FROM Building b
101   JOIN Manager m ON b.Manager_ID = m.Manager_ID
102   LIMIT 10; -- Insert first 10 buildings
103
104 *   select * from Dim_Building;
105
106 *   INSERT INTO Dim_Staff (StaffID, StaffName, StaffEmail)
107     SELECT Staff_ID, Staff_Name, SEmail
108    FROM Staff
109   LIMIT 10; -- Insert all 10 staff members
110
111 *   INSERT INTO Dim_Apartment (ApartmentID, BuildingID, NoOfBedrooms, RentalStatus)
112     SELECT Apartment_ID, Building_ID, No_of_Bedrooms, Rental_Status
113    FROM Apartment
114   LIMIT 10; -- Insert first 10 apartments
115
116 *   INSERT INTO Dim_Status (StatusID, StatusDescription)
117     VALUES
118     (1, 'Completed'),
119     (2, 'Open'),
120     (3, 'In Progress');
121
122
123 -- Increase the column length (e.g., VARCHAR(100))
124 *   ALTER TABLE Dim_CorpClient
125     MODIFY COLUMN ClientEmail VARCHAR(100);
126 *   INSERT INTO Dim_CorpClient (ClientID, ClientName, ClientEmail, ReferredClientID)
127     SELECT
128         Client_ID,
129         Client_Name,
130         CEmail,
131         Refer_Client_ID
132    FROM CorporateClient
133   LIMIT 10; -- Insert all 10 clients
```

Navigator: create\_dim\_projectpart2 Insert\_dim\_projectpart2 SQL File 27\*

Schemas

- pms
- pms\_backup
- pms\_p2
- pms\_part2**
  - Tables
    - apartment
    - building
    - corporatedclient
    - dim\_apartment
    - dim\_building
    - dim\_corplient
    - dim\_service
    - dim\_staff
    - dim\_status
    - dim\_time
    - lease
    - lease\_fact
    - maintenancerequest
    - manager
    - servicerequest\_fact
    - staff
  - Views
  - Stored Procedures
  - Functions
- sakila
- sys
- world

134 • INSERT INTO Dim\_Time (Date, Year, Month, Quarter, DayOfWeek, DayOfMonth, FiscalYear)  
 135     SELECT DISTINCT  
 136         Start\_Date,  
 137         YEAR(Start\_Date),  
 138         MONTH(Start\_Date),  
 139         QUARTER(Start\_Date),  
 140         DAYOFWEEK(Start\_Date),  
 141         DAYOFMONTH(Start\_Date),  
 142         YEAR(Start\_Date) + IF(MONTH(Start\_Date) >= 4, 1, 0) -- Indian fiscal year (April-March)  
 143     FROM Lease  
 144     LIMIT 10; -- Insert 10 unique dates  
 145 • INSERT INTO Dim\_Service (ServiceTypeID, ServiceTypeName)  
 146     VALUES  
 147         (1, 'Plumbing'),  
 148         (2, 'Electrical'),  
 149         (3, 'HVAC');  
 150  
 151 • INSERT INTO Lease\_Fact (ClientKey, TimeKey, BuildingKey, ApartmentKey, LeaseID, MonthlyRent, SecurityDeposit, StartDate, EndDate)  
 152     SELECT  
 153         dc.ClientKey,  
 154         dt.TimeKey,  
 155         db.BuildingKey,  
 156         da.ApartmentKey,  
 157         l.Lease\_ID,  
 158         l.monthly\_rent,  
 159         l.security\_deposit,  
 160         l.Start\_Date,  
 161         l.End\_Date  
 162     FROM Lease l  
 163     JOIN Dim\_CorpClient dc ON l.Client\_ID = dc.ClientID  
 164     JOIN Dim\_Time dt ON l.Start\_Date = dt.Date  
 165     JOIN Dim\_Building db ON l.Building\_ID = db.BuildingID  
 166     JOIN Dim\_Apartment da ON l.Apartment\_ID = da.ApartmentID AND l.Building\_ID = da.BuildingID  
 167     LIMIT 10; -- Insert all 10 leases  
 168  
 169

Administration Schemas Output

Navigator: create\_dim\_projectpart2 Insert\_dim\_projectpart2 SQL File 27\*

Schemas

- pms
- pms\_backup
- pms\_p2
- pms\_part2**
  - Tables
    - apartment
    - building
    - corporatedclient
    - dim\_apartment
    - dim\_building
    - dim\_corplient
    - dim\_service
    - dim\_staff
    - dim\_status
    - dim\_time
    - lease
    - lease\_fact
    - maintenancerequest
    - manager
    - servicerequest\_fact
    - staff
  - Views
  - Stored Procedures
  - Functions
- sakila
- sys
- world

170 • INSERT INTO ServiceRequest\_Fact (ServiceTypeKey, TimeKey, StaffKey, ApartmentKey, StatusKey, BuildingKey, RequestID, TotalRequests, AvgResolutionDays)  
 171     SELECT  
 172         ds.ServiceTypeKey,  
 173         dt.TimeKey,  
 174         dst.StaffKey,  
 175         da.ApartmentKey,  
 176         ds2.StatusKey,  
 177         db.BuildingKey,  
 178         mr.Request\_ID,  
 179         1 AS TotalRequests, -- Assume 1 request per row  
 180         DATEDIFF(mr.Completion\_date, mr.Request\_Date) AS AvgResolutionDays  
 181     FROM MaintenanceRequest mr  
 182     JOIN Dim\_Service ds ON mr.Request\_ID % 3 + 1 = ds.ServiceTypeID -- Assign service types arbitrarily  
 183     JOIN Dim\_Time dt ON mr.Request\_Date = dt.Date  
 184     JOIN Dim\_Staff dst ON mr.Assigned\_Staff = dst.StaffID  
 185     JOIN Dim\_Apartment da ON mr.Apartment\_ID = da.ApartmentID AND mr.Building\_ID = da.BuildingID  
 186     JOIN Dim\_Status ds2 ON mr.Status = ds2.StatusDescription  
 187     JOIN Dim\_Building db ON mr.Building\_ID = db.BuildingID  
 188     LIMIT 10; -- Insert first 10 requests  
 189  
 190  
 191

## 7.2.1 Success Log

| Action Output |          |  |
|---------------|----------|--|
| #             | Time     | Action   |
| 1             | 15:31:11 | INSERT INTO Dim_Building (BuildingID, Street, City, State, ZipCode, ManagerID, ManagerName, ManagerEmail) SELECT b.Building_ID,  |
| 2             | 15:31:20 | select * from Dim_Building LIMIT 0, 1000   |
| 3             | 15:31:23 | INSERT INTO Dim_Staff (StaffID, StaffName, StaffEmail) SELECT Staff_ID, Staff_Name, SEmail FROM Staff LIMIT 10   |
| 4             | 15:31:26 | INSERT INTO Dim_Apartment (ApartmentID, BuildingID, NoOfBedrooms, RentalStatus) SELECT Apartment_ID, Building_ID, No_of_Bedrooms, Rental_Status FROM Apartment LIMIT 10                        |
| 5             | 15:31:29 | INSERT INTO Dim_Status (StatusID, StatusDescription) VALUES (1, 'Completed'), (2, 'Open'), (3, 'In Progress')  |
| 6             | 15:31:34 | ALTER TABLE Dim_CorpClient MODIFY COLUMN ClientEmail VARCHAR(100)  |
| 7             | 15:31:37 | ALTER TABLE Dim_CorpClient MODIFY COLUMN ClientEmail VARCHAR(100)  |
| 8             | 15:31:40 | INSERT INTO Dim_CorpClient (ClientID, ClientName, ClientEmail, ReferredClientID) SELECT Client_ID, Client_Name, CEmail, ReferredClientID FROM CorpClient LIMIT 10                              |
| 9             | 15:31:45 | INSERT INTO Dim_Time (Date, Year, Month, Quarter, DayOfWeek, DayOfMonth, FiscalYear) SELECT DISTINCT Start_Date, YEAR(Start_Date), Month, Quarter, DayOfWeek, DayOfMonth, FiscalYear FROM Time |
| 10            | 15:31:49 | INSERT INTO Dim_Service (ServiceTypeID, ServiceTypeName) VALUES (1, 'Plumbing'), (2, 'Electrical'), (3, 'HVAC')  |
| 11            | 15:31:53 | INSERT INTO ServiceRequest_Fact (ServiceTypeKey, TimeKey, StaffKey, ApartmentKey, StatusKey, BuildingKey, RequestID, TotalRequest) VALUES (1, 1, 1, 1, 1, 1, 1, 1)                             |

## 7.3 Retrieving the Data

The screenshot shows a database management interface with the following details:

- Schemas:** pms, pms\_backup, pms\_p2, pms\_part2.
- Tables:** apartment, building, corporateclient, dim\_apartment, dim\_building, dim\_corpclient, dim\_service, dim\_staff, dim\_status, dim\_time, inspects, inspector, lease, lease\_fact, maintenancerequest, manager, servicerequest\_fact.
- Query:** SELECT \* FROM pms\_part2.dim\_staff;
- Result Grid:** A table displaying 10 rows of staff data.

|    | StaffKey | StaffID         | StaffName                   | StaffEmail |
|----|----------|-----------------|-----------------------------|------------|
| 1  | 1        | Anjali Mehta    | anjali.mehta@example.com    |            |
| 2  | 2        | Ravi Shastri    | ravi.shastri@example.com    |            |
| 3  | 3        | Sunita Rao      | sunita.rao@example.com      |            |
| 4  | 4        | Kiran Bedi      | kiran.bedi@example.com      |            |
| 5  | 5        | Sanjay Kapoor   | sanjay.kapoor@example.com   |            |
| 6  | 6        | Pooja Choudhary | pooja.choudhary@example.com |            |
| 7  | 7        | Amitabh Roy     | amitabh.roy@example.com     |            |
| 8  | 8        | Nandini Das     | nandini.das@example.com     |            |
| 9  | 9        | Harish Pandey   | harish.pandey@example.com   |            |
| 10 | 10       | Swati Menon     | swati.menon@example.com     |            |
| *  | HULL     | HULL            | HULL                        | HULL       |

Navigator: create\_dim\_projectpart2 Insert\_dim\_projectpart2\* dim\_building

SCHEMAS

Filter objects

- pms
- pms\_backup
- pms\_p2
- pms\_part2**
  - Tables
    - apartment
    - building
    - corporatedclient
    - dim\_apartment
    - dim\_building
    - dim\_corpclient
    - dim\_service
    - dim\_staff
    - dim\_status
    - dim\_time
    - inspects
    - inspector
    - lease
    - lease\_fact
    - maintenancerequest
    - manager
    - servicerequest\_fact
    - staff

```
1 • SELECT * FROM pms_part2.dim_building;
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

| BuildingKey | BuildingID | Street          | City      | State       | ZipCode | ManagerID | ManagerName   | ManagerEmail              |
|-------------|------------|-----------------|-----------|-------------|---------|-----------|---------------|---------------------------|
| 1           | 101        | Marine Drive    | Mumbai    | Maharashtra | 400001  | 1         | Arjun Patel   | arjun.patel@example.com   |
| 2           | 102        | Connaught Place | Delhi     | Delhi       | 110001  | 2         | Priya Sharma  | priya.sharma@example.com  |
| 3           | 103        | MG Road         | Bangalore | Karnataka   | 560001  | 3         | Rajesh Kumar  | rajesh.kumar@example.com  |
| 4           | 104        | Park Street     | Kolkata   | West Bengal | 700016  | 4         | Ananya Reddy  | ananya.reddy@example.com  |
| 5           | 105        | Juhu Beach      | Mumbai    | Maharashtra | 400049  | 5         | Vikram Singh  | vikram.singh@example.com  |
| 6           | 106        | Hauz Khas       | Delhi     | Delhi       | 110016  | 6         | Meera Desai   | meera.desai@example.com   |
| 7           | 107        | Indiranagar     | Bangalore | Karnataka   | 560038  | 7         | Suresh Iyer   | suresh.iyer@example.com   |
| 8           | 108        | Salt Lake City  | Kolkata   | West Bengal | 700091  | 8         | Deepika Joshi | deepika.joshi@example.com |
| 9           | 109        | Bandra West     | Mumbai    | Maharashtra | 400050  | 9         | Rahul Gupta   | rahul.gupta@example.com   |
| 10          | 110        | Lajpat Nagar    | Delhi     | Delhi       | 110024  | 10        | Neha Verma    | neha.verma@example.com    |
| *           | NULL       | NULL            | NULL      | NULL        | NULL    | NULL      | NULL          | NULL                      |

Navigator: create\_dim\_projectpart2 Insert\_dim\_projectpart2\* dim\_building dim\_staff dim\_apartment

SCHEMAS

Filter objects

- pms
- pms\_backup
- pms\_p2
- pms\_part2**
  - Tables
    - apartment
    - building
    - corporatedclient
    - dim\_apartment
    - dim\_building
    - dim\_corpclient
    - dim\_service
    - dim\_staff
    - dim\_status
    - dim\_time
    - inspects
    - inspector
    - lease
    - lease\_fact
    - maintenancerequest
    - manager
    - servicerequest\_fact

```
1 • SELECT * FROM pms_part2.dim_apartment;
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

| ApartmentKey | ApartmentID | BuildingID | NoOfBedrooms | RentalStatus |
|--------------|-------------|------------|--------------|--------------|
| 1            | 201         | 101        | 2            | Occupied     |
| 2            | 202         | 101        | 1            | Vacant       |
| 3            | 301         | 102        | 3            | Occupied     |
| 4            | 302         | 102        | 2            | Occupied     |
| 5            | 401         | 103        | 1            | Vacant       |
| 6            | 402         | 103        | 2            | Occupied     |
| 7            | 501         | 104        | 3            | Occupied     |
| 8            | 502         | 104        | 2            | Vacant       |
| 9            | 601         | 105        | 1            | Occupied     |
| 10           | 602         | 105        | 3            | Occupied     |
| *            | NULL        | NULL       | NULL         | NULL         |

**Navigator**

**SCHEMAS**

Filter objects

- pms
- pms\_backup
- pms\_p2
- pms\_part2**
  - Tables
    - apartment
    - building
    - corporatedclient
    - dim\_apartment
    - dim\_building
    - dim\_corpclient
    - dim\_service
    - dim\_staff
    - dim\_status
    - dim\_time

**dim\_status x**

1 \* SELECT \* FROM pms\_part2.dim\_status;

**Result Grid**

|   | StatusKey | StatusID | StatusDescription |
|---|-----------|----------|-------------------|
| ▶ | 1         | 1        | Completed         |
|   | 2         | 2        | Open              |
|   | 3         | 3        | In Progress       |
| * | NULL      | NULL     | NULL              |

**Navigator**

**SCHEMAS**

Filter objects

- pms
- pms\_backup
- pms\_p2
- pms\_part2**
  - Tables
    - apartment
    - building
    - corporatedclient
    - dim\_apartment
    - dim\_building
    - dim\_corpclient
    - dim\_service
    - dim\_staff
    - dim\_status
    - dim\_time
    - inspects
    - inspector
    - lease
    - lease\_fact
    - maintenancerequest
    - manager
    - servicerequest\_fat
    - staff
  - Views

**dim\_status dim\_corpclient x**

1 \* SELECT \* FROM pms\_part2.dim\_corpclient;

**Result Grid**

|   | ClientKey | ClientID | ClientName                | ClientEmail           | ReferredClientID |
|---|-----------|----------|---------------------------|-----------------------|------------------|
| ▶ | 1         | 1001     | TechMahindra Ltd          | info@techmahindra.com | NULL             |
|   | 2         | 1002     | Infosys Technologies      | contact@infosys.com   | 1001             |
|   | 3         | 1003     | Wipro Enterprises         | support@wipro.com     | 1001             |
|   | 4         | 1004     | Tata Consultancy Services | help@tcs.com          | 1002             |
|   | 5         | 1005     | Reliance Industries       | contact@reliance.com  | NULL             |
|   | 6         | 1006     | Aditya Birla Group        | info@adityabirla.com  | 1005             |
|   | 7         | 1007     | HDFC Bank                 | support@hdfcbank.com  | NULL             |
|   | 8         | 1008     | ICICI Bank                | contact@icici.com     | 1007             |
|   | 9         | 1009     | Bajaj Auto                | info@bajajauto.com    | 1005             |
|   | 10        | 1010     | Larsen & Toubro           | help@larsentoubro.com | 1006             |
| * | NULL      | NULL     | NULL                      | NULL                  | NULL             |

The screenshot shows a database interface with the following details:

- Schemas:** pms, pms\_backup, pms\_p2, pms\_part2 (selected).
- Tables under pms\_part2:** apartment, building, corporateclient, dim\_apartment, dim\_building, dim\_corpclient, dim\_service, dim\_staff, dim\_status, dim\_time, inspects, inspector, lease, lease\_fact, maintenancerequest, manager, servicerequest\_fact, staff.
- Current View:** lease\_fact
- SQL Query:** SELECT \* FROM pms\_part2.lease\_fact;
- Result Grid:** Displays 10 rows of data from the lease\_fact table.

|    | LeaseKey | ClientKey | TimeKey | BuildingKey | ApartmentKey | LeaseID  | MonthlyRent | SecurityDeposit | StartDate  | EndDate |
|----|----------|-----------|---------|-------------|--------------|----------|-------------|-----------------|------------|---------|
| 1  | 1        | 1         | 1       | 1           | 5001         | 45000.00 | 90000.00    | 2023-01-01      | 2024-01-01 |         |
| 2  | 8        | 8         | 1       | 1           | 5008         | 45000.00 | 90000.00    | 2023-08-01      | 2024-08-01 |         |
| 3  | 2        | 2         | 2       | 3           | 5002         | 55000.00 | 110000.00   | 2023-02-01      | 2024-02-01 |         |
| 4  | 9        | 9         | 2       | 3           | 5009         | 55000.00 | 110000.00   | 2023-09-01      | 2024-09-01 |         |
| 5  | 3        | 3         | 2       | 4           | 5003         | 38000.00 | 76000.00    | 2023-03-01      | 2024-03-01 |         |
| 6  | 4        | 4         | 3       | 6           | 5004         | 42000.00 | 84000.00    | 2023-04-01      | 2024-04-01 |         |
| 7  | 10       | 10        | 3       | 6           | 5010         | 42000.00 | 84000.00    | 2023-10-01      | 2024-10-01 |         |
| 8  | 5        | 5         | 4       | 7           | 5005         | 60000.00 | 120000.00   | 2023-05-01      | 2024-05-01 |         |
| 9  | 6        | 6         | 5       | 9           | 5006         | 32000.00 | 64000.00    | 2023-06-01      | 2024-06-01 |         |
| 10 | 7        | 7         | 5       | 10          | 5007         | 48000.00 | 96000.00    | 2023-07-01      | 2024-07-01 |         |

The image displays two side-by-side screenshots of a database management interface, likely SQL Server Management Studio (SSMS), illustrating the results of an Extract, Transform, Load (ETL) process.

**Screenshot 1 (Top):**

- Schemas:** dim\_status, dim\_corpcclient, dim\_time, dim\_service (selected).
- Tables:** apartment, building, corporateclient, dim\_apartment, dim\_building, dim\_corpcclient, dim\_service, dim\_staff, dim\_status, dim\_time.
- Query:** SELECT \* FROM pms\_part2.dim\_service;
- Result Grid:**

|   | ServiceTypeKey | ServiceTypeID | ServiceTypeName |
|---|----------------|---------------|-----------------|
| 1 | 1              | Plumbing      |                 |
| 2 | 2              | Electrical    |                 |
| 3 | 3              | HVAC          |                 |
| * | NULL           | NULL          | NULL            |

**Screenshot 2 (Bottom):**

- Schemas:** lease\_fact, servicerequest\_fact (selected), Insert\_dim\_projectpart2.
- Tables:** apartment, building, corporateclient, dim\_apartment, dim\_building, dim\_corpcclient, dim\_service.
- Query:** SELECT \* FROM pms\_part2.servicerequest\_fact;
- Result Grid:**

|   | ServiceKey | ServiceTypeKey | TimeKey | StaffKey | ApartmentKey | StatusKey | BuildingKey | RequestID | TotalRequests | AvgResolutionDays | DurationDays |
|---|------------|----------------|---------|----------|--------------|-----------|-------------|-----------|---------------|-------------------|--------------|
| 1 | 3          | 7              | 7       | 4        | 2            | 2         | 107         | 1         | NULL          | NULL              | NULL         |
| * | NULL       | NULL           | NULL    | NULL     | NULL         | NULL      | NULL        | NULL      | NULL          | NULL              | NULL         |

## 8.0 ETL Process:

### 8.1 Purpose of the ETL Script

The script transforms raw transactional data from an OLTP database into a structured star schema for analytical reporting. It populates dimension tables (descriptive entities) and fact tables (transactional metrics) to enable business intelligence queries.

#### 8.1.1 Key Components

**Stored Procedure:** PopulateDimTime

- **Objective:** Precompute time-related attributes for a date range (2020–2030) to support time-based aggregations.
- **Logic:**
  - Uses a loop to generate a row for every date between 2020-01-01 and 2030-12-31.
  - Extracts temporal attributes:
    - Year, Quarter, Month, DayOfWeek, DayOfMonth.
    - Derives FiscalYear as YEAR + 1 if the month is July or later (assumes fiscal year starts in July).
- **Importance:** Serves as a universal time reference for all date-based operations in fact tables.

### 8.1.2 Dimension Table Population

- Dim\_Building:
  - **Source:** BUILDING (address details) and MANAGER (manager name).
  - **Transformation:** Combines street, city, state, and ZIP code into a single BuildingAddress field.
  - Links buildings to their assigned managers via Manager\_ID.
- Dim\_Apartment:
  - **Source:** APARTMENT table.
  - Maps raw apartment attributes (Apartment\_ID, No\_of\_Bedrooms, Rental\_Status) directly to the dimension.
- Dim\_CorpClient:
  - **Source:** CORPCLIENT table.
  - Uses NOT EXISTS to ensure uniqueness, preventing duplicate client entries during incremental loads.
  - Preserves referral relationships via Refer\_Client\_ID.
- Dim\_Staff:
  - **Source:** STAFF table.

- Directly maps staff IDs and names for tracking personnel assignments.

### 8.1.3 Fact Table Population

- LeaseFact:
  - **Source:** LEASE table (transactional lease records).
  - **Joins:**
    - Dim\_CorpClient to link tenants to leases.
    - Dim\_Time to associate lease start dates with temporal attributes.
    - Dim\_Building and Dim\_Apartment to tie leases to physical units.
  - **Measures:** Captures financial metrics (monthly\_rent, security\_deposit) and lease duration (start\_date, end\_date).
- ServiceRequestFact:
  - **Source:** MAINTENANCE\_REQUEST table.
  - **Joins:**
    - Dim\_Time to track request dates.
    - Dim\_Staff to identify assigned personnel.
    - Dim\_Apartment and Dim\_Building for unit/location context.
  - **Calculations:**
    - AvgResolutionDays: Computed as DATEDIFF(Completion\_Date, Request\_Date) for resolved requests.
    - DurationDays: Uses CURRENT\_DATE for unresolved requests to calculate ongoing resolution time.
  - **Metrics:** Aggregates request counts (TotalRequests) and ties inspections to resolution efficiency.

### 8.1.4 Critical Design Choices

#### 1. Time Dimension Prepopulation:

- A static Dim\_Time table is created upfront to avoid runtime date calculations during fact table loads. This optimizes query performance.

## 2. Referential Integrity:

- Foreign keys (e.g., BuildingKey, ClientKey) enforce relationships between fact and dimension tables, ensuring data consistency.

## 3. Handling Unresolved Requests:

- For open maintenance requests (Completion\_Date IS NULL), the script uses CURRENT\_DATE to compute DurationDays, providing real-time visibility into unresolved issues.

## 4. Fiscal Year Logic:

- Assumes a July–June fiscal calendar. This logic is hardcoded but can be modified if the fiscal year aligns with the calendar year.

## 5. Incremental Loads:

- The NOT EXISTS clause in Dim\_CorpClient prevents duplicate client entries, making the script idempotent (safe for reruns).

### 8.1.5 Workflow Summary

#### 1. Dimensions First:

- Static dimensions (Dim\_Time, Dim\_Building, etc.) are populated first to serve as lookup tables for fact data.

#### 2. Facts Second:

- Transactional data from LEASE and MAINTENANCE\_REQUEST is loaded into fact tables, enriched with dimension keys.

#### 3. Dependency Management:

- The order of operations ensures foreign key constraints are respected (e.g., Dim\_Time must exist before LeaseFact is populated).

### 8.1.6 Business Value

- **Unified Reporting:** Combines scattered OLTP data into a single schema for cross-functional analysis (e.g., linking lease revenue to maintenance costs).
- **Performance:** Star schema optimizes query speed for dashboards and reports.
- **Scalability:** Designed to handle incremental data loads without reprocessing historical data.

This ETL script transforms raw operational data into an analytical goldmine, enabling stakeholders to derive insights on occupancy trends, maintenance efficiency, and financial performance.

## 8.2 Why Truncating

### 8.2.1 Full Refresh Strategy

- Our ETL process is designed as a full refresh — every run:
- Deletes old data (via TRUNCATE)
- Reloads everything cleanly from the source
- This avoids duplication, inconsistencies, or partial updates

### 8.2.2 Prevent Orphaned/Outdated Records

- If dimension keys or mappings change (e.g., building IDs updated), keeping old fact rows might break foreign key relationships
- Truncating ensures the fact and dimension tables are synchronized

### 8.2.3 Clean Staging for Testing or Development

- In academic or controlled environments, you often test multiple ETL cycles
- Truncation ensures that each cycle starts from scratch, so results are repeatable

### 8.2.4 Better Performance

- TRUNCATE is faster than DELETE and resets auto-increment IDs
- Ideal when you don't need to preserve old data between loads

```
Navigator: ETL_Script_projectpart2

SCHEMAS
  pms
  pms_backup
  pms_p2
  pms_part2
    Tables
      apartment
      building
      corporateclient
      dim_apartment
      dim_building
      dim_corpclient
      dim_service
      dim_staff
      dim_status
      dim_time
      inspects
      inspector
      lease
      lease_fact
      maintenancerequest

ETL_Script_projectpart2

7  -- Step 1: Truncate Existing Data (For Full Refresh)
8  --
9 * SET FOREIGN_KEY_CHECKS = 0;
10
11 * TRUNCATE TABLE Lease_Fact;
12 * TRUNCATE TABLE ServiceRequest_Fact;
13 * TRUNCATE TABLE Dim_Time;
14 * TRUNCATE TABLE Dim_Status;
15 * TRUNCATE TABLE Dim_Service;
16 * TRUNCATE TABLE Dim_CorpClient;
17 * TRUNCATE TABLE Dim_Apartment;
18 * TRUNCATE TABLE Dim_Staff;
19 * TRUNCATE TABLE Dim_Building;
20
21 * SET FOREIGN_KEY_CHECKS = 1;
22
```

Navigator: ETL\_Script\_projectpart2

SCHEMAS

- pms
- pms\_backup
- pms\_p2
- pms\_part2**
- Tables
- Views
- Stored Procedures
- Functions
- sakila
- sys
- world

```

58    -- Dim_Status (Status Dimension)
59 •  INSERT INTO Dim_Status (StatusID, StatusDescription)
60    VALUES
61      (1, 'Completed'),
62      (2, 'Open'),
63      (3, 'In Progress');
64
65    -- Dim_CorpClient (Corporate Client Dimension)
66 •  INSERT INTO Dim_CorpClient (ClientID, ClientName, ClientEmail, ReferredClientID)
67    SELECT
68      Client_ID,
69      Client_Name,
70      CEmail,
71      Refer_Client_ID
72    FROM CorporateClient;
73
74    -- Dim_Time (Time Dimension - Indian Fiscal Year)
75 •  INSERT INTO Dim_Time (Date, Year, Month, Quarter, DayOfWeek, DayOfMonth, FiscalYear)
76    SELECT DISTINCT
77      Start_Date,
78      YEAR(Start_Date) AS Year,
79      MONTH(Start_Date) AS Month,
80      QUARTER(Start_Date) AS Quarter,
81      DAYOFWEEK(Start_Date) AS DayOfWeek,
82      DAYOFMONTH(Start_Date) AS DayOfMonth,
83      -- Indian Fiscal Year: April-March
84      YEAR(Start_Date) + IF(MONTH(Start_Date) >= 4, 1, 0) AS FiscalYear
85    FROM Lease;
86
87    -- Dim_Service (Service Type Dimension)
88 •  INSERT INTO Dim_Service (ServiceTypeID, ServiceTypeName)
89    VALUES
90      (1, 'Plumbing'),
91      (2, 'Electrical'),
92      (3, 'HVAC');
93

```

Navigator: ETL\_Script\_projectpart2

SCHEMAS

- pms
- pms\_backup
- pms\_p2
- pms\_part2**
- Tables
- Views
- Stored Procedures
- Functions
- sakila
- sys
- world

```

26
27    -- Dim_Building (Building Dimension)
28 •  INSERT INTO Dim_Building (BuildingID, Street, City, State, ZipCode, ManagerID, ManagerName, ManagerEmail)
29    SELECT
30      b.Building_ID,
31      b.Street,
32      b.City,
33      b.State,
34      b.ZipCode,
35      m.Manager_ID,
36      m.MFull_Name,
37      m.MEmail
38    FROM Building b
39    JOIN Manager m ON b.Manager_ID = m.Manager_ID;
40
41    -- Dim_Staff (Staff Dimension)
42 •  INSERT INTO Dim_Staff (StaffID, StaffName, StaffEmail)
43    SELECT
44      Staff_ID,
45      Staff_Name,
46      SEmail
47    FROM Staff;
48
49    -- Dim_Apartment (Apartment Dimension)
50 •  INSERT INTO Dim_Apartment (ApartmentID, BuildingID, NoOfBedrooms, RentalStatus)
51    SELECT
52      Apartment_ID,
53      Building_ID,
54      No_of_Bedrooms,
55      Rental_Status
56    FROM Apartment;
57

```

Navigator: ETL\_Script\_projectpart2

SCHEMAS

- pms
- pms\_backup
- pms\_p2
- pms\_part2**
  - Tables
    - apartment
    - building
    - corporatedclient
    - dim\_apartment
    - dim\_building
    - dim\_corpclient
    - dim\_service
    - dim\_staff
    - dim\_status
    - dim\_time
    - inspects
    - inspector
    - lease
    - lease\_fact
    - maintenancerequest
    - manager
    - servicerequest\_fact
    - staff
  - Views
  - Stored Procedures
  - Functions
- sakila
- sys
- world

Administration Schemas

Information: Schema: pms\_part2

```

100
101 * INSERT INTO Lease_Fact (
102     ClientKey, TimeKey, BuildingKey, ApartmentKey, LeaseID,
103     MonthlyRent, SecurityDeposit, StartDate, EndDate
104 )
105     SELECT
106         dc.ClientKey,
107         dt.TimeKey,
108         db.BuildingKey,
109         da.ApartmentKey,
110         l.Lease_ID,
111         l.Monthly_Rent,
112         l.Security_Deposit,
113         l.Start_Date,
114         l.End_Date
115     FROM Lease l
116     JOIN Dim_CorpClient dc ON l.Client_ID = dc.ClientID
117     JOIN Dim_Time dt ON l.Start_Date = dt.Date
118     JOIN Dim_Building db ON l.Building_ID = db.BuildingID
119     JOIN Dim_Apartment da ON l.Apartment_ID = da.ApartmentID AND l.Building_ID = da.BuildingID;
120
121
122 -- ServiceRequest_Fact (Maintenance Metrics)
123 * INSERT INTO ServiceRequest_Fact (
124     ServiceTypeKey, TimeKey, StaffKey, ApartmentKey,
125     StatusKey, BuildingKey, RequestID, TotalRequests,
126     AvgResolutionDays, DurationDays
127 )
128     SELECT
129         ds.ServiceTypeKey,
130         dt.TimeKey,
131         dst.StaffKey,
132         da.ApartmentKey,
133         ds2.StatusKey,
134         db.BuildingKey,
135         mr.Request_ID,
136         1 AS TotalRequests,
137         DATEDIFF(mr.Completion_date, mr.Request_Date) AS AvgResolutionDays,
138         DATEDIFF(COALESCE(mr.Completion_date, CURDATE()), mr.Request_Date) AS DurationDays
139     FROM MaintenanceRequest mr
140     JOIN Dim_Service ds ON
141         CASE
142             WHEN mr.Description LIKE '%plumb%' THEN 1
143             WHEN mr.Description LIKE '%electr%' THEN 2
144             ELSE 3
145         END = ds.ServiceTypeID
146     JOIN Dim_Time dt ON mr.Request_Date = dt.Date
147     JOIN Dim_Staff dst ON mr.Assigned_Staff = dst.StaffID
148     JOIN Dim_Apartment da ON mr.Apartment_ID = da.ApartmentID
149         AND mr.Building_ID = da.BuildingID
150     JOIN Dim_Status ds2 ON mr.Status = ds2.StatusDescription
151     JOIN Dim_Building db ON mr.Building_ID = db.BuildingID;
152

```

Navigator ETL\_Script\_projectpart2\*

**SCHEMAS**

Filter objects

- pms
- pms\_backup
- pms\_p2
- pms\_part2**
  - Tables
    - apartment
    - building
    - corporateclient
    - dim\_apartment
    - dim\_building
    - dim\_corpclient
    - dim\_service
    - dim\_staff
    - dim\_status
    - dim\_time

```

155  --
156  -- Check for unresolved foreign keys
157 *  SELECT 'Lease_Fact' AS TableName, COUNT(*) AS OrphanRecords
158   FROM Lease_Fact
159   WHERE ClientKey NOT IN (SELECT ClientKey FROM Dim_CorpClient)
160      OR TimeKey NOT IN (SELECT TimeKey FROM Dim_Time)
161      OR BuildingKey NOT IN (SELECT BuildingKey FROM Dim_Building)
162 UNION ALL
163   SELECT 'ServiceRequest_Fact', COUNT(*)
164   FROM ServiceRequest_Fact
165   WHERE ServiceTypeKey NOT IN (SELECT ServiceTypeKey FROM Dim_Service)
166      OR StaffKey NOT IN (SELECT StaffKey FROM Dim_Staff);

```

Output

Action Output

| #  | Time     | Action   |
|----|----------|--|
| 1  | 15:49:51 | SET FOREIGN_KEY_CHECKS = 0   |
| 2  | 15:49:54 | TRUNCATE TABLE Lease_Fact  |
| 3  | 15:49:56 | TRUNCATE TABLE ServiceRequest_Fact   |
| 4  | 15:49:58 | TRUNCATE TABLE Dim_Time  |
| 5  | 15:49:59 | TRUNCATE TABLE Dim_Status  |
| 6  | 15:50:00 | TRUNCATE TABLE Dim_Service   |
| 7  | 15:50:01 | TRUNCATE TABLE Dim_CorpClient  |
| 8  | 15:50:02 | TRUNCATE TABLE Dim_Apartment   |
| 9  | 15:50:03 | TRUNCATE TABLE Dim_Staff   |
| 10 | 15:50:04 | TRUNCATE TABLE Dim_Building  |
| 11 | 15:50:05 | SET FOREIGN_KEY_CHECKS = 1   |
| 12 | 15:50:14 | INSERT INTO Dim_Building (BuildingID, Street, City, State, ZipCode, ManagerID, ManagerName, ManagerEmail) SELECT b.Building_ID, b.Street, b.City, b. |
| 13 | 15:50:18 | INSERT INTO Dim_Staff (StaffID, StaffName, StaffEmail) SELECT Staff_ID, Staff_Name, SEmail FROM Staff  |
| 14 | 15:50:21 | INSERT INTO Dim_Apartment (ApartmentID, BuildingID, NoOfBedrooms, RentalStatus) SELECT Apartment_ID, Building_ID, No_of_Bedrooms, Rental_Status      |
| 15 | 15:50:24 | INSERT INTO Dim_Status (StatusID, StatusDescription) VALUES (1, 'Completed'), (2, 'Open'), (3, 'In Progress')  |
| 16 | 15:50:31 | INSERT INTO Dim_CorpClient (ClientID, ClientName, ClientEmail, ReferredClientID) SELECT Client_ID, Client_Name, CEmail, Refer_Client_ID FROM Corpor  |
| 17 | 15:50:35 | INSERT INTO Dim_Time (Date, Year, Month, Quarter, DayOfWeek, DayOfMonth, FiscalYear) SELECT DISTINCT Start_Date, YEAR(Start_Date) AS Year, M         |
| 18 | 15:50:40 | INSERT INTO Dim_Service (ServiceTypeID, ServiceTypeName) VALUES (1, 'Plumbing'), (2, 'Electrical'), (3, 'HVAC')                                      |
| 19 | 15:50:45 | INSERT INTO Lease_Fact (ClientKey, TimeKey, BuildingKey, ApartmentKey, LeaseID, MonthlyRent, SecurityDeposit, StartDate, EndDate) SELECT dc.Cli      |
| 20 | 15:50:54 | INSERT INTO ServiceRequest_Fact (ServiceTypeKey, TimeKey, StaffKey, ApartmentKey, StatusKey, BuildingKey, RequestID, TotalRequests, AvgResolutio     |
| 21 | 15:51:01 | SELECT 'Lease_Fact' AS TableName, COUNT(*) AS OrphanRecords FROM Lease_Fact WHERE ClientKey NOT IN (SELECT ClientKey FROM Dim_CorpClient)            |
| 22 | 15:51:13 | SELECT 'Lease_Fact' AS TableName, COUNT(*) AS OrphanRecords FROM Lease_Fact WHERE ClientKey NOT IN (SELECT ClientKey FROM Dim_CorpClient)            |

## 9.0 Analytical Summary Tables

The property management data warehouse is supported by six analytical summary tables that deliver fast, actionable insights across operations and finance. These include staff efficiency (tracking request resolution speed and workload), occupancy trends (measuring lease-based apartment usage across buildings), yearly rental income (estimating revenue per building), unresolved maintenance requests (highlighting pending issues by location), request resolution details (analyzing completion times by month/year), and rental income by bedroom count (assessing profitability by unit size). Designed for precomputed use in dashboards, these summaries enable quick, data-driven decisions in staffing, pricing, leasing, and maintenance, with periodic refreshes and consistent key relationships ensuring reliable performance.

### 9.1 Summary\_StaffEfficiency

**Purpose:** Measures staff performance in resolving maintenance requests, focusing on speed and workload.

**Key Metrics:**

- TotalRequestsHandled: Number of completed requests per staff member per year.
- AvgResolutionDays: Average days taken to resolve requests.

**SQL Logic:**

- Joins Dim\_Staff with MAINTENANCE\_REQUEST to link staff members to their assigned requests.
- Filters by Status = 'completed' to exclude open/pending requests.
- Groups by staff and year to track annual performance.

**Business Value:**

- Identifies top-performing staff (e.g., those with the lowest AvgResolutionDays).
- Highlights training needs for staff with slower resolution times.

### 9.2 Summary\_OccupancyTrends

**Purpose:** Tracks apartment occupancy rates across buildings.

**Key Metrics:**

- TotalOccupiedApartments: Count of leased apartments.
- TotalApartments: Total available apartments.
- AverageOccupancyRate: Occupied apartments as a percentage of total.

#### **SQL Logic:**

- Joins Dim\_Building with Dim\_Apartment to map apartments to their buildings.
- Uses a LEFT JOIN with LeaseFact to count apartments with active leases.
- Calculates occupancy rate using:

$(\text{Occupied} / \text{Total}) * 100$

#### **Business Value:**

- Identifies underperforming buildings (low occupancy).
- Supports decisions on rent adjustments or marketing campaigns.

## **9.3 Summary\_YearlyRentallIncome**

**Purpose:** Calculates annual rental income for each building over the last two years.

#### **Key Metrics:**

- TotalRentallIncome: Estimated income based on lease duration and monthly rent.

#### **SQL Logic:**

- Joins LeaseFact with Dim\_Building and Dim\_Time to link leases to buildings and years.
- Estimates income by multiplying monthly\_rent by the lease duration (in months):

$\text{monthly\_rent} * (\text{DATEDIFF}(\text{end\_date}, \text{start\_date}) / 30)$

- Filters data for the last two years using YEAR(CURDATE()).

#### **Business Value:**

- Tracks revenue growth/decline over time.
- Compares building profitability.

## **9.4 Summary\_UnresolvedRequests**

**Purpose:** Identifies unresolved maintenance requests per building.

#### **Key Metric:**

- UnresolvedRequests: Count of open/pending requests.

#### **SQL Logic:**

- Directly queries MAINTENANCE\_REQUEST and joins with Dim\_Building.
- Filters requests where Status is not completed.

#### **Business Value:**

- Highlights buildings with operational inefficiencies.
- Prioritizes resource allocation for urgent repairs.

## **9.5 Summary\_RequestResolutionDetails**

**Purpose:** Analyzes resolution times for completed requests by month/year.

#### **Key Metrics:**

- ResolutionDays: Days taken to resolve a request.

#### **SQL Logic:**

- Links MAINTENANCE\_REQUEST to Dim\_Time to extract year/month from Request\_Date.
- Filters by Status = 'completed'.

#### **Business Value:**

- Identifies seasonal trends in maintenance demand.
- Monitors SLA compliance (e.g., 95% of requests resolved within 5 days).

## **9.6 Summary\_RentallIncomeByBedrooms**

**Purpose:** Analyzes rental income and pricing by apartment size (bedrooms).

#### **Key Metrics:**

- TotalRentallIncome: Total revenue per bedroom category.
- AverageMonthlyRent: Average rent for apartments of a specific size.

#### **SQL Logic:**

- Joins LeaseFact with Dim\_Apartment to group leases by bedroom count.
- Estimates income using lease duration and monthly rent.

#### **Business Value:**

- Determines optimal pricing for apartment sizes (e.g., 3-bedroom units generate the most income).
- Guides future construction or renovations (e.g., build more 2-bedroom units).

## 9.7 Implementation Notes

### 9.7.1 Time-Based Approximations:

- Rental income calculations assume 30 days/month for simplicity.
- Adjust if precise calendar-month billing is required.

### 9.7.2 Performance:

- Summary tables are precompiled to speed up dashboards/reports.
- Refresh them periodically (e.g., nightly) for up-to-date insights.

### 9.7.3 Data Consistency:

- Ensure foreign keys (e.g., BuildingKey) match between fact/dimension tables.

These tables provide actionable insights into **revenue**, **occupancy**, and **operations**, enabling data-driven decisions in property management.

Navigator: ETL\_Script\_projectpart2\* summary\_tables\_projectpart2

**SCHEMAS**

- pms
- pms\_backup
- pms\_p2
- pms\_part2**
  - Tables: apartment, building, corporateclient, dim\_apartment, dim\_building, dim\_corclient, dim\_service, dim\_staff, dim\_status, dim\_time, inspects, inspector, lease, lease\_fact, maintenancerequest, manager, servicerequest\_fact, staff
  - Views
  - Stored Procedures
  - Functions
- sakila
- sys
- world

Administration Schemas

Information Schema: pms\_part2

```

1  /* Total Rental Income per Building (Annual) */
2  CREATE TABLE Summary_RentalIncome (
3      Building_ID INT,
4      Year INT,
5      TotalRent DECIMAL(12, 2),
6      Street VARCHAR(30),
7      City VARCHAR(20),
8      PRIMARY KEY (Building_ID, Year),
9      FOREIGN KEY (Building_ID) REFERENCES Building(Building_ID)
10 );
11
12  INSERT INTO Summary_RentalIncome (Building_ID, Year, TotalRent, Street, City)
13  SELECT
14      l.Building_ID,
15      YEAR(l.Start_Date) AS Year,
16      SUM(l.monthly_rent *
17          TIMESTAMPDIFF(MONTH, l.Start_Date, l.End_Date)) AS TotalRent,
18      b.Street,
19      b.City
20  FROM Lease l
21  JOIN Building b ON l.Building_ID = b.Building_ID
22  GROUP BY l.Building_ID, YEAR(l.Start_Date), b.Street, b.City;
23
24  /* Inspections per Building (Monthly) */
25  CREATE TABLE Summary_Inspections (
26      Building_ID INT,
27      Year INT,
28      Month INT,
29      TotalInspections INT,
30      Street VARCHAR(30),
31      City VARCHAR(20),
32      PRIMARY KEY (Building_ID, Year, Month),
33      FOREIGN KEY (Building_ID) REFERENCES Building(Building_ID)
34 );
35
36  INSERT INTO Summary_Inspections (Building_ID, Year, Month, TotalInspections, Street, City)
37  SELECT
38      i.Building_ID,
39      YEAR(i.Insp_date) AS Year,
40      MONTH(i.Insp_date) AS Month,
41      COUNT(*) AS TotalInspections,
42      b.Street,
43      b.City
44  FROM Inspects i
45  JOIN Building b ON i.Building_ID = b.Building_ID
46  GROUP BY i.Building_ID, YEAR(i.Insp_date), MONTH(i.Insp_date), b.Street, b.City;
47

```

Navigator: ETL\_Script\_projectpart2\* summary\_tables\_projectpart2

Schemas

- pms
- pms\_backup
- pms\_p2
- pms\_part2**
  - Tables
    - apartment
    - building
    - corporatedclient
    - dim\_apartment
    - dim\_building
    - dim\_corclient
    - dim\_service
    - dim\_staff
    - dim\_status
    - dim\_time
    - inspects
    - inspector
    - lease
    - lease\_fact
    - maintenancerequest
    - manager
    - servicerequest\_fact
    - staff
  - Views
  - Stored Procedures
  - Functions
- sakila
- sys
- world

Administration Schemas

Information

Schema: pms\_part2

```

48     /* Staff Performance Summary Table */
49 *  CREATE TABLE Summary_StaffPerformance (
50     Staff_ID INT,
51     Year INT,
52     CompletedTasks INT,
53     AvgResolutionDays DECIMAL(5,2),
54     StaffName VARCHAR(30),
55     PRIMARY KEY (Staff_ID, Year),
56     FOREIGN KEY (Staff_ID) REFERENCES Staff(Staff_ID)
57 );
58
59 *  INSERT INTO Summary_StaffPerformance (Staff_ID, Year, CompletedTasks, AvgResolutionDays, StaffName)
60     SELECT
61         mr.Assigned_Staff AS Staff_ID,
62         YEAR(mr.Request_Date) AS Year,
63         COUNT(*) AS CompletedTasks,
64         ROUND(AVG(DATEDIFF(mr.Completion_Date, mr.Request_Date)), 2) AS AvgResolutionDays,
65         s.Staff_Name AS StaffName
66     FROM MaintenanceRequest mr
67     JOIN Staff s ON mr.Assigned_Staff = s.Staff_ID
68     WHERE mr.Status = 'Completed'
69     GROUP BY mr.Assigned_Staff, YEAR(mr.Request_Date), s.Staff_Name;
70
71
72     /* vacancy rate per building */
73 *  CREATE TABLE Summary_VacancyRates (
74     Building_ID INT,
75     MonthYear DATE,
76     VacancyRate DECIMAL(5,2),
77     TotalApartments INT,
78     PRIMARY KEY (Building_ID, MonthYear),
79     FOREIGN KEY (Building_ID) REFERENCES Building(Building_ID)
80 );
81
82 *  INSERT INTO Summary_VacancyRates (Building_ID, MonthYear, VacancyRate, TotalApartments)
83     SELECT
84         a.Building_ID,
85         LAST_DAY(a.SnapshotDate) AS MonthYear,
86         (SUM(CASE WHEN a.Rental_Status = 'Vacant' THEN 1 ELSE 0 END) / COUNT(*)) * 100 AS VacancyRate,
87         COUNT(*) AS TotalApartments
88     FROM (
89         SELECT
90             Building_ID,
91             Rental_Status,
92             CURDATE() AS SnapshotDate -- Replace with actual snapshot logic
93         FROM Apartment
94     ) a
95     GROUP BY a.Building_ID, LAST_DAY(a.SnapshotDate);
96

```

Navigator: ETL\_Script\_projectpart2\* summary\_tables\_projectpart2

```

102
103     DELIMITER $$ 
104 • CREATE EVENT RefreshSummaries
105     ON SCHEDULE EVERY 1 DAY
106     DO
107     BEGIN
108         -- Refresh Rental Income Summary
109         DELETE FROM Summary_RentalIncome;
110         INSERT INTO Summary_RentalIncome (Building_ID, Year, TotalRent, Street, City)
111         SELECT
112             l.Building_ID,
113             YEAR(l.Start_Date),
114             SUM(l.monthly_rent * TIMESTAMPDIFF(MONTH, l.Start_Date, l.End_Date)),
115             b.Street,
116             b.City
117         FROM Lease l
118         JOIN Building b ON l.Building_ID = b.Building_ID
119         GROUP BY l.Building_ID, YEAR(l.Start_Date), b.Street, b.City;
120
121         -- Refresh Inspections Summary
122         DELETE FROM Summary_Inspections;
123         INSERT INTO Summary_Inspections (Building_ID, Year, Month, TotalInspections, Street, City)
124         SELECT
125             i.Building_ID,
126             YEAR(i.Insp_Date),
127             MONTH(i.Insp_Date),
128             COUNT(*),
129             b.Street,
130             b.City
131         FROM Inspects i
132         JOIN Building b ON i.Building_ID = b.Building_ID
133         GROUP BY i.Building_ID, YEAR(i.Insp_Date), MONTH(i.Insp_Date), b.Street, b.City;
134

```

Navigator: ETL\_Script\_projectpart2\* summary\_tables\_projectpart2

```

133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181

```

Administration Schemas

Information Schema: pms\_part2

| Action Output |          |  |
|---------------|----------|--|
| #             | Time     | Action   |
| ✓ 1           | 16:02:43 | CREATE TABLE Summary_RentalIncome ( Building_ID INT, Year INT, TotalRent DECIMAL(12, 2), Street VARCHAR(30), City VARCHAR(20), PRIMARY KEY (Building_ID))  |
| ✓ 2           | 16:02:47 | INSERT INTO Summary_RentalIncome (Building_ID, Year, TotalRent, Street, City) SELECT i.Building_ID, YEAR(i.Start_Date) AS Year, SUM(i.monthly_rent * TIMESTAMPED(i.Insp_date)) AS TotalRent, Street, City  |
| ✓ 3           | 16:02:57 | CREATE TABLE Summary_Inspections ( Building_ID INT, Year INT, Month INT, TotalInspections INT, Street VARCHAR(30), City VARCHAR(20), PRIMARY KEY (Building_ID, Year, Month))   |
| ✓ 4           | 16:03:00 | INSERT INTO Summary_Inspections (Building_ID, Year, Month, TotalInspections, Street, City) SELECT i.Building_ID, YEAR(i.Insp_date) AS Year, MONTH(i.Insp_date) AS Month, COUNT(i.Insp_date) AS TotalInspections, Street, City  |
| ✓ 5           | 16:03:03 | CREATE TABLE Summary_VacancyRates ( Building_ID INT, MonthYear DATE, VacancyRate DECIMAL(5,2), TotalApartments INT, PRIMARY KEY (Building_ID, MonthYear))  |
| ✓ 6           | 16:03:07 | INSERT INTO Summary_VacancyRates (Building_ID, MonthYear, VacancyRate, TotalApartments) SELECT a.Building_ID, LAST_DAY(a.SnapshotDate) AS MonthYear, (SUM(CASE WHEN a.VacancyRate > 0 THEN 1 ELSE 0 END) / COUNT(a.VacancyRate)) AS VacancyRate, COUNT(a.VacancyRate) AS TotalApartments                       |
| ✓ 7           | 16:03:14 | CREATE EVENT RefreshSummaries ON SCHEDULE EVERY 1 DAY DO BEGIN -- Refresh Rental Income Summary DELETE FROM Summary_RentalIncome; INSERT INTO Summary_RentalIncome (Building_ID, Year, TotalRent) SELECT i.Building_ID, YEAR(i.Insp_date) AS Year, SUM(i.monthly_rent * TIMESTAMPED(i.Insp_date)) AS TotalRent |
| ✓ 8           | 16:03:29 | CREATE INDEX idx_rental_income ON Summary_RentalIncome(Building_ID, Year)  |
| ✗ 9           | 16:03:33 | CREATE INDEX idx_staff_perf ON Summary_StaffPerformance(Staff_ID, Year)  |
| ✓ 10          | 16:03:42 | SELECT Street, City, TotalRent FROM Summary_RentalIncome WHERE Year = 2023 ORDER BY TotalRent DESC LIMIT 5   |
| ✓ 11          | 16:03:48 | SELECT Year, Month, SUM(TotalInspections) AS Total FROM Summary_Inspections WHERE Building_ID = 101 GROUP BY Year, Month ORDER BY Year DESC, Month   |
| ✗ 12          | 16:03:53 | SELECT StaffName, CompletedTasks, AvgResolutionDays FROM Summary_StaffPerformance WHERE Year = 2023 ORDER BY CompletedTasks DESC LIMIT 0, 1000   |
| ✓ 13          | 16:11:15 | CREATE TABLE Summary_StaffPerformance ( Staff_ID INT, Year INT, CompletedTasks INT, AvgResolutionDays DECIMAL(5,2), StaffName VARCHAR(30), PRIMARY KEY (Staff_ID, Year))   |
| ✓ 14          | 16:11:29 | INSERT INTO Summary_StaffPerformance (Staff_ID, Year, CompletedTasks, AvgResolutionDays, StaffName) SELECT mr.Assigned_Staff AS Staff_ID, YEAR(mr.Request_Date) AS Year, COUNT(mr.Completed_Task) AS CompletedTasks, AVG(mr.Resolution_Days) AS AvgResolutionDays, mr.StaffName AS StaffName                   |
| ✓ 15          | 16:11:43 | CREATE INDEX idx_staff_perf ON Summary_StaffPerformance(Staff_ID, Year)  |
| ✓ 16          | 16:11:52 | SELECT StaffName, CompletedTasks, AvgResolutionDays FROM Summary_StaffPerformance WHERE Year = 2023 ORDER BY CompletedTasks DESC LIMIT 0, 1000   |

## 10.0 Advanced BI Reports: 5 complex SQL queries with strategic value and explanations

### 10.1 Total Leases and Rent by Year and Building

Navigator: ETL\_Script\_projectpart2\* summary\_tables\_projectpart2 BI Queries\_projectpart2

**SCHEMAS**

- pms
- pms\_backup
- pms\_p2
- pms\_part2**
  - Tables
    - apartment
    - building
    - corporateclient
    - dim\_apartment
    - dim\_building
    - dim\_corclient
    - dim\_service
    - dim\_staff
    - dim\_status
    - dim\_time
    - inspects
    - inspector
    - lease
    - lease\_fact
    - maintenancerequest
    - manager
    - servicerequest\_fact
    - staff
  - Views
  - Stored Procedures
  - Functions
- sakila

```

1 /* total leases and rent by year and building */
2 * SELECT
3     dt.Year,
4     db.BuildingID,
5     db.City,
6     COUNT(lf.LeaseKey) AS TotalLeases,
7     ROUND(SUM(lf.MonthlyRent), 2) AS TotalMonthlyRent
8     FROM Lease_Fact lf
9     JOIN Dim_Time dt ON lf.TimeKey = dt.TimeKey
10    JOIN Dim_Building db ON lf.BuildingKey = db.BuildingKey
11    GROUP BY dt.Year, db.BuildingID, db.City
12    ORDER BY dt.Year, db.BuildingID;
13
14 /* Avg lease duration and rent by apartment type */

```

**Result Grid**

| Year | BuildingID | City      | TotalLeases | TotalMonthlyRent |
|------|------------|-----------|-------------|------------------|
| 2023 | 101        | Mumbai    | 2           | 90000.00         |
| 2023 | 102        | Delhi     | 3           | 148000.00        |
| 2023 | 103        | Bangalore | 2           | 84000.00         |
| 2023 | 104        | Kolkata   | 1           | 60000.00         |
| 2023 | 105        | Mumbai    | 2           | 80000.00         |

### What it shows:

Summarizes how many leases were signed and how much rental income was generated per building each year.

### Why it matters:

This helps management monitor financial performance and occupancy levels over time, identify top-performing buildings, and make investment or pricing decisions.

## 10.2 Average Lease Duration and Rent by Apartment Type

The screenshot shows a BI tool interface with a query editor and a result grid. The query editor displays a T-SQL script for calculating average lease duration and rent by apartment type. The result grid shows the output of the query, which includes columns for apartment size, lease count, average rent, and average lease duration.

```
13
14     /* Avg lease duration and rent by apartment type */
15 *  SELECT
16         da.NoOfBedrooms,
17         COUNT(lf.LeaseKey) AS LeaseCount,
18         ROUND(AVG(lf.MonthlyRent), 2) AS AvgRent,
19         ROUND(AVG(DATEDIFF(lf.EndDate, lf.StartDate)), 2) AS AvgLeaseDurationDays
20     FROM Lease_Fact lf
21     JOIN Dim_Apartment da ON lf.ApartmentKey = da.ApartmentKey
22     GROUP BY da.NoOfBedrooms
23     ORDER BY da.NoOfBedrooms;
24
25
```

| NoOfBedrooms | LeaseCount | AvgRent  | AvgLeaseDurationDays |
|--------------|------------|----------|----------------------|
| 1            | 1          | 32000.00 | 366.00               |
| 2            | 5          | 42400.00 | 365.80               |
| 3            | 4          | 54500.00 | 365.75               |

### What it shows:

Breaks down lease count, average rent, and lease length by apartment size (e.g., 1-bedroom, 2-bedroom).

### Why it matters:

Helps identify which apartment types are most in demand or most profitable, aiding decisions on property design, pricing strategies, and marketing focus.

## 10.3 Year-Over-Year Lease Growth

The screenshot shows a database development interface with a left sidebar for 'SCHEMAS' containing 'pms', 'pms\_backup', 'pms\_p2', and 'pms\_part2'. Under 'pms\_part2', there is a 'Tables' section listing various tables like 'apartment', 'building', 'corporatedclient', etc. The main area displays a SQL script:

```
27     /*Year over year lease growth */
28 *  SELECT
29         dt.Year,
30         COUNT(*) AS TotalLeases,
31         LAG(COUNT(*)) OVER (ORDER BY dt.Year) AS LeasesLastYear,
32         (COUNT(*) - LAG(COUNT(*)) OVER (ORDER BY dt.Year)) AS YearlyGrowth
33     FROM Lease_Fact lf
34     JOIN Dim_Time dt ON lf.TimeKey = dt.TimeKey
35     GROUP BY dt.Year
36     ORDER BY dt.Year;
37
38     /*Top 5 corpclients by total rent paid */
39 *  SELECT
```

Below the script is a 'Result Grid' table:

|   | Year | TotalLeases | LeasesLastYear | YearlyGrowth |
|---|------|-------------|----------------|--------------|
| ▶ | 2023 | 10          | NULL           | NULL         |

### What it shows:

Tracks how lease volume has changed each year and calculates the difference compared to the previous year.

### Why it matters:

Reveals business growth trends or downturns, allowing leadership to react to market demand shifts or operational issues.

## 10.4 Top 5 Corporate Clients by Total Rent Paid

The screenshot shows a database interface with a sidebar containing a 'Navigator' section titled 'SCHEMAS'. It lists several schemas: pms, pms\_backup, pms\_p2, and pms\_part2. Under 'pms\_part2', there are tables: apartment, building, corporateclient, dim\_apartment, dim\_building, dim\_corpclient, dim\_service, dim\_staff, dim\_status, dim\_time, inspects, inspector, lease, lease\_fact, maintenancerequest, manager, servicerequest\_fact, staff, Views, and Stored Procedures. The main area displays a SQL query and its results.

ETL\_Script\_projectpart2\* summary\_tables\_projectpart2 BI Queries\_projectpart2\*

```
37
38  /*Top 5 corpclients by total rent paid */
39 •  SELECT
40      dc.ClientName,
41      dc.ClientEmail,
42      ROUND(SUM(lf.MonthlyRent * TIMESTAMPDIFF(MONTH, lf.StartDate, lf.EndDate)), 2) AS TotalRentPaid
43  FROM Lease_Fact lf
44  JOIN Dim_CorpClient dc ON lf.ClientKey = dc.ClientKey
45  GROUP BY dc.ClientName, dc.ClientEmail
46  ORDER BY TotalRentPaid DESC
47  LIMIT 5;
48
49  /*Lease distribution by City */
```

Result Grid | Filter Rows: [ ] Export: [ ] Wrap Cell Content: [ ] Fetch rows: [ ]

| ClientName           | ClientEmail           | TotalRentPaid |
|----------------------|-----------------------|---------------|
| Reliance Industries  | contact@reliance.com  | 720000.00     |
| Infosys Technologies | contact@infosys.com   | 660000.00     |
| Bajaj Auto           | info@bajajauto.com    | 660000.00     |
| HDFC Bank            | support@hdfcbank.com  | 576000.00     |
| TechMahindra Ltd     | info@techmahindra.com | 540000.00     |

### What it shows:

Ranks corporate clients based on the total rent they've paid across their leases.

### Why it matters:

Identifies high-value clients for retention efforts, loyalty programs, or personalized service. It can also inform risk assessment if a top client ends their lease.

## 10.5 Lease Distribution by City

The screenshot shows a database management interface with the following details:

- Navigator:** Shows the schema structure. The **SCHEMAS** section includes `pms`, `pms_backup`, `pms_p2`, and `pms_part2`. The `pms_part2` schema is expanded to show **Tables** such as `apartment`, `building`, `corporatedclient`, etc., and **Views**, **Stored Procedures**, and **Functions**. It also lists the `sakila` schema.
- ETL\_Script\_projectpart2\***: A tab showing the SQL script for the query.
- summary\_tables\_projectpart2**: A tab showing the results of the query.
- BI Queries\_projectpart2\***: A tab showing other BI-related queries.
- Query Script (ETL\_Script\_projectpart2\*):**

```
50
51     /*Lease distribution by City */
52 *  SELECT
53         db.City,
54         COUNT(*) AS LeaseCount,
55         ROUND(AVG(lf.MonthlyRent), 2) AS AvgRent
56     FROM Lease_Fact lf
57     JOIN Dim_Building db ON lf.BuildingKey = db.BuildingKey
58     GROUP BY db.City
59     ORDER BY LeaseCount DESC;
60
61
62
```
- Result Grid:** A table showing the results of the query. The columns are **City**, **LeaseCount**, and **AvgRent**.

|   | City      | LeaseCount | AvgRent  |
|---|-----------|------------|----------|
| ▶ | Mumbai    | 4          | 42500.00 |
|   | Delhi     | 3          | 49333.33 |
|   | Bangalore | 2          | 42000.00 |
|   | Kolkata   | 1          | 60000.00 |

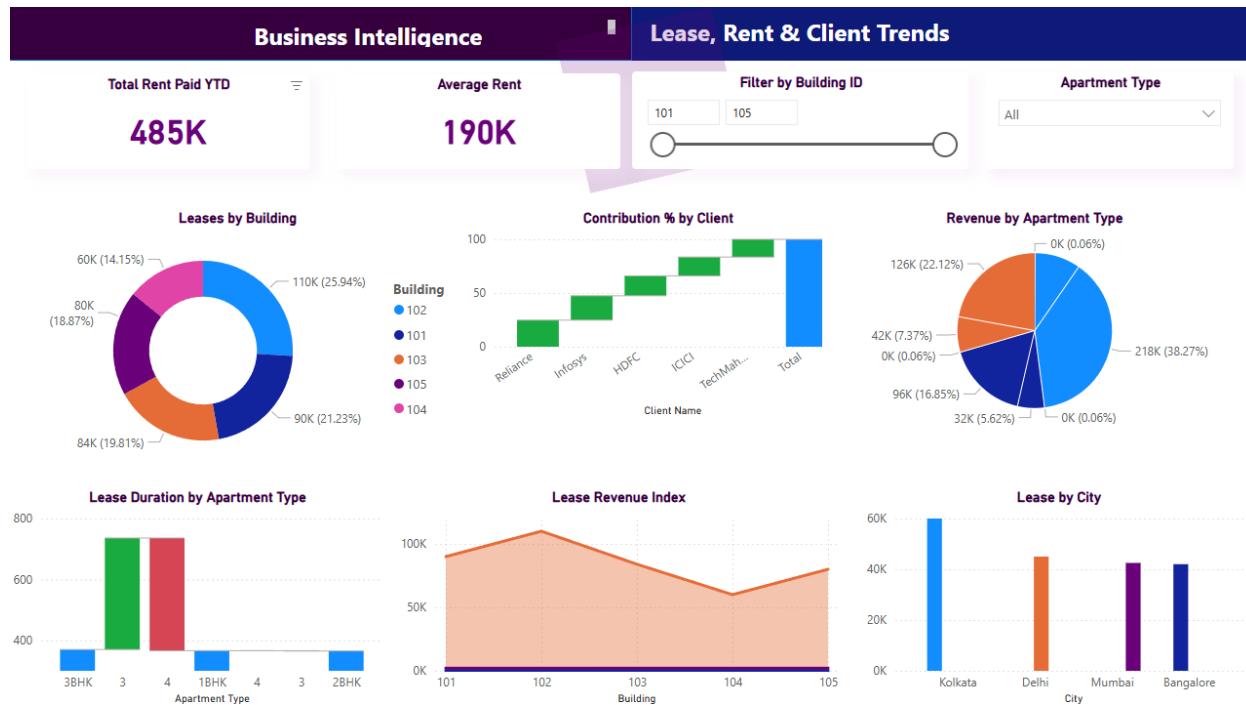
### What it shows:

Counts how many leases exist in each city and what the average rent is.

### Why it matters:

Enables geographic performance comparison, supports regional pricing strategies, and informs decisions on property expansion or divestment.

## 11.0 Bonus Section – Visualization



Built with PowerBI > Dashboard Link: [Click here](#)

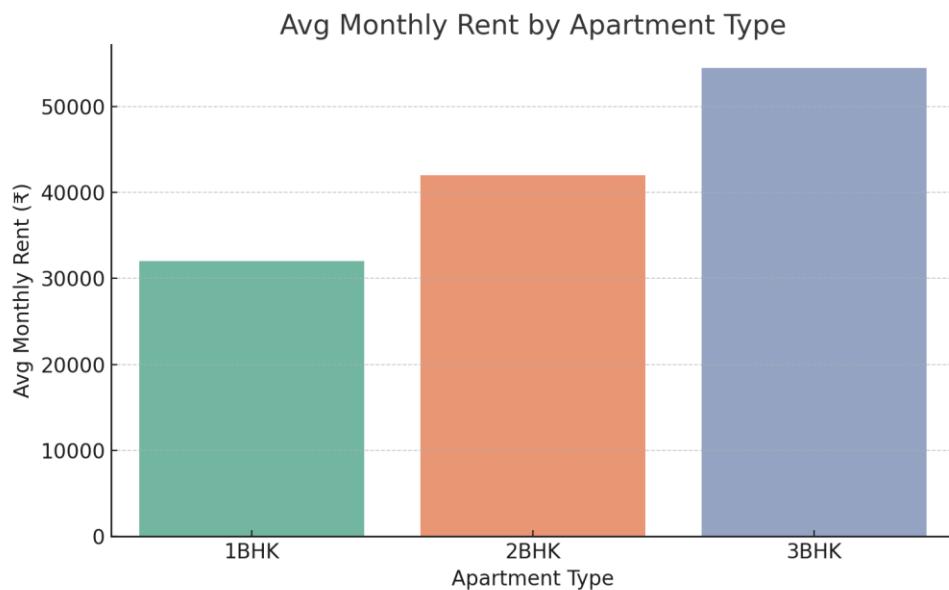
## 11.1 Total Leases and Rent by Year and Building

This chart displays the total number of leases and corresponding rental income generated by each building in 2023. It helps management identify high-performing buildings and analyze occupancy trends, enabling strategic rent setting and property-level decisions.



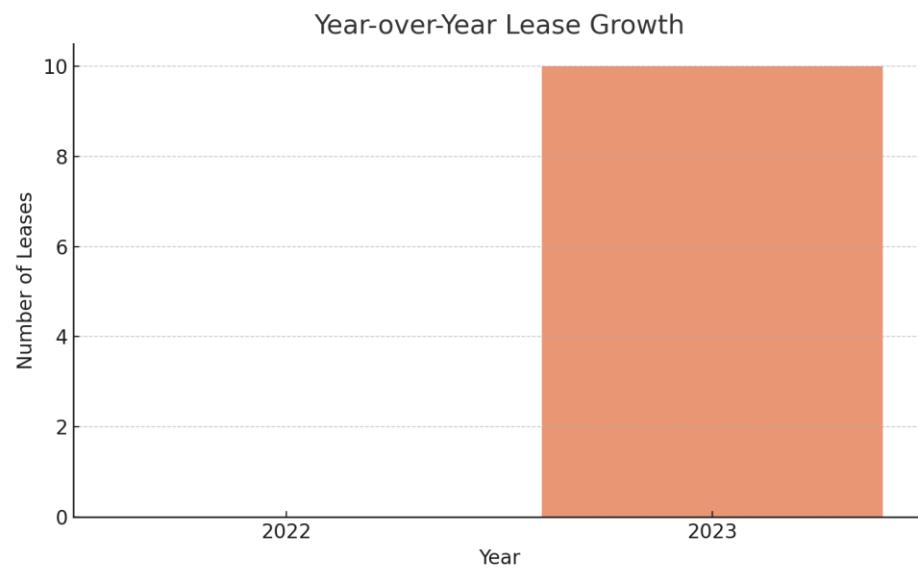
## 11.2 Average Lease Duration and Rent by Apartment Type

Visualizing the average monthly rent alongside lease durations for different apartment types, this chart identifies which unit sizes (1BHK, 2BHK, 3BHK) are more popular or profitable. It supports layout planning and targeted marketing.



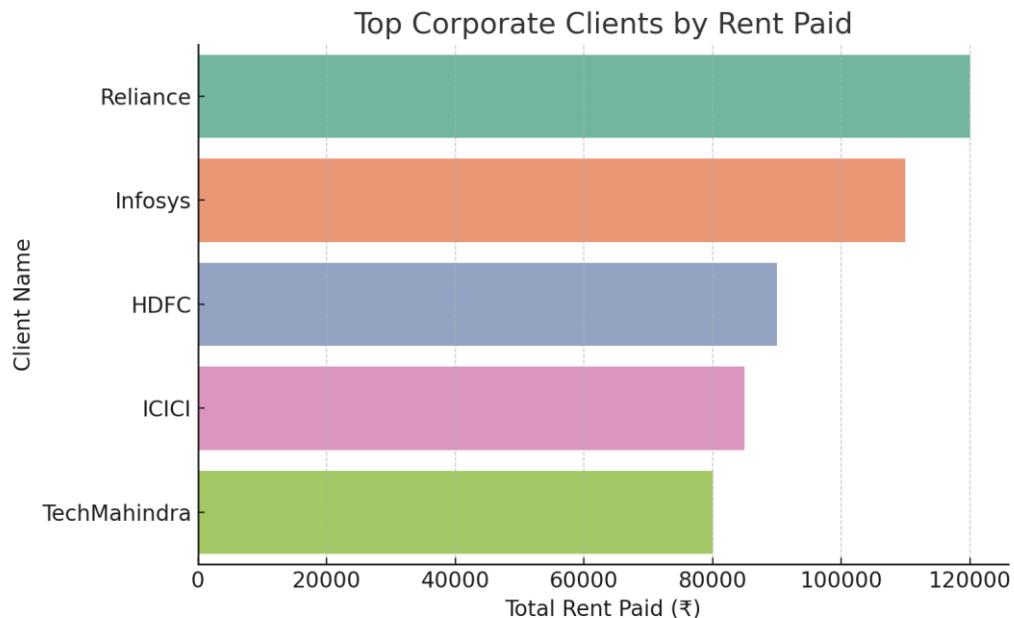
## 11.3 Year-Over-Year Lease Growth

This visual highlights annual changes in lease volume. For example, a jump from 0 to 10 leases in 2023 signals business growth. Managers can use this to understand market traction and plan expansion or operational improvements.



## 11.4 Top 5 Corporate Clients by Total Rent Paid

Ranking the highest-paying corporate clients, this chart helps prioritize client engagement, evaluate risk from large accounts, and personalize service offerings. It's essential for revenue assurance and loyalty strategies.



## 11.5 Lease Distribution by City

This scatter plot compares how many leases were signed in each city and their average rent levels. It informs geographic investment strategies and identifies high-potential or underperforming regions.

