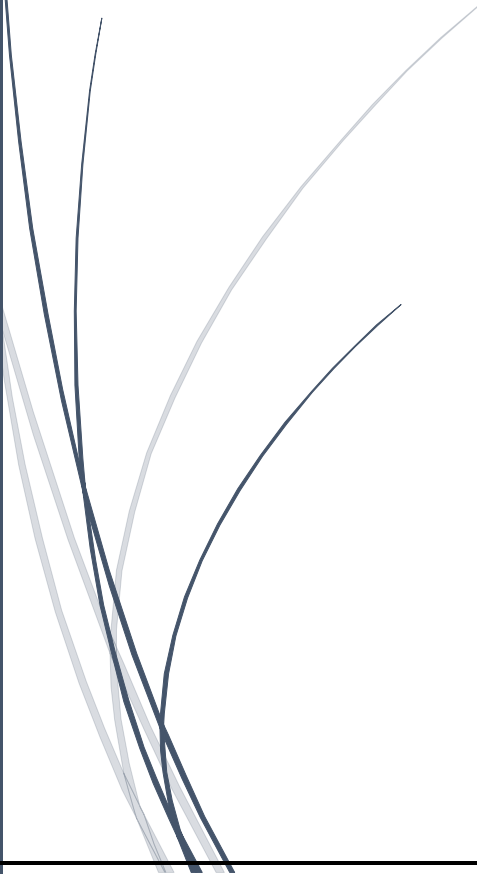


12/6/2018

# Housing Management System



## **Contents**

• <b>Current System</b>	<b>2</b>
• <b>Proposed System</b>	<b>2</b>
• <b>Requirement</b>	<b>2-3</b>
• <b>ER Diagram</b>	<b>4</b>
• <b>Relational Database Schema</b>	<b>5</b>
• <b>SQL Create Schema</b>	<b>6-10</b>
• <b>SQL Queries and output</b>	<b>11-19</b>
• <b>Discussion points</b>	<b>20-21</b>

## **Current System:**

The current system at most of the places around the country are either completely manual or semi-automated. The manual system starts all the way by calling the leasing office for availability and enquiring about the price and signing the lease, making payment and other manual process. This could be good for small housing company but for a company which has properties based in multiple locations this becomes tough, Also the manual process introduces lot of labor cost and window for potential errors. The semi-automated procedure also involves manual procedures but some things like payments are automated.

## **Proposed Solution:**

We are proposing a solution that maintains centralized database to store data about the company's properties located in different places and total number of apartments under each house and details about properties that are already sold or available for sale, leased or vacant apartments. The system would store all the details about each and every property and minimize the manual efforts reducing the overcall costs. The system also contains the capability to receive and respond to complaints and intimating proper concerned department for appropriate solution. The system will also have the capability for the storing and retrieving historical data on demand.

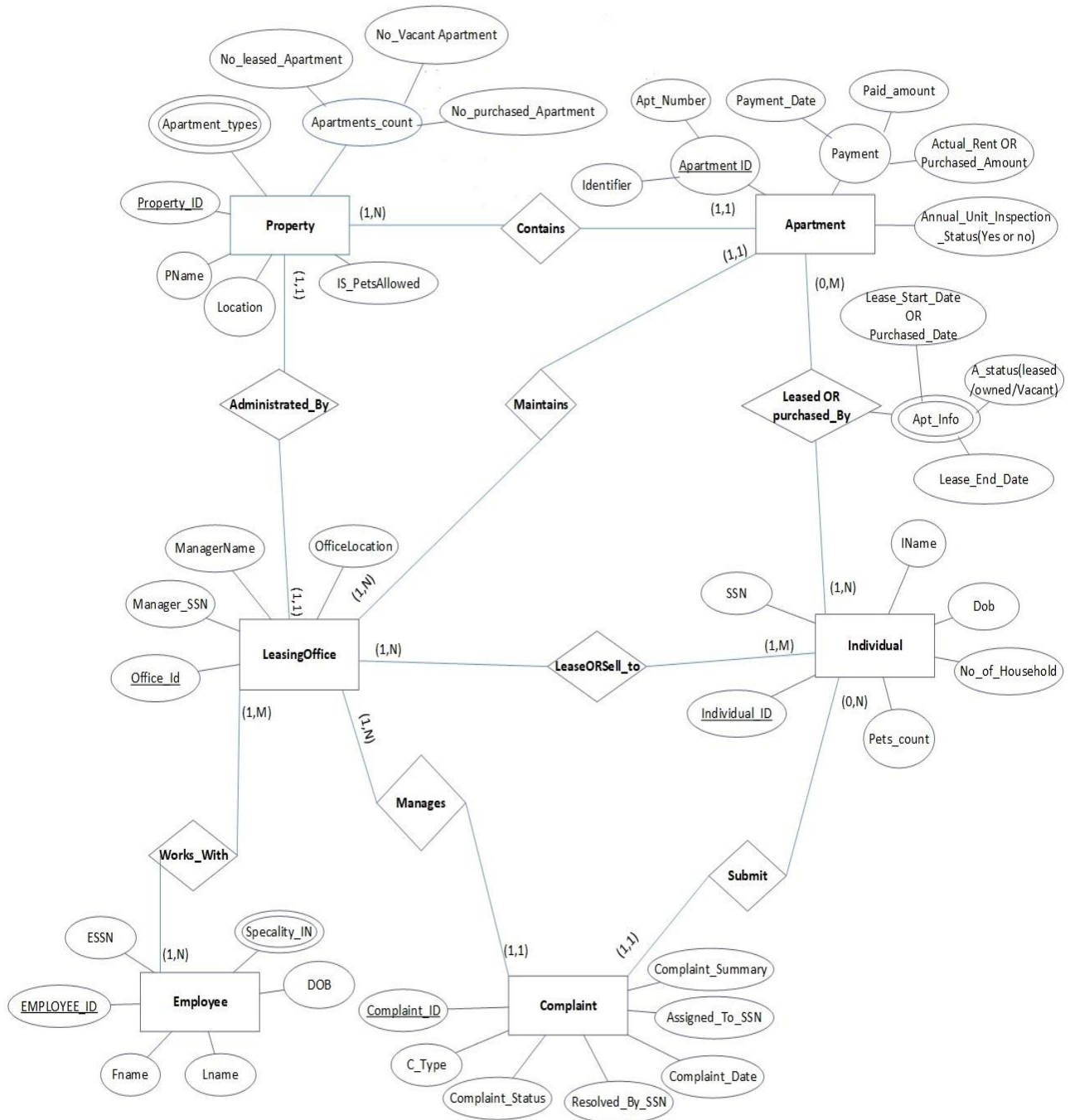
## **Requirement:**

Housing companies wants to maintain a database that contains details of each of its properties at different locations and all their details. The property must contain property Id, name, location, apartment types associated with each property (2BHK,3BHK,1BHK etc.) apartment count which is a composite attribute of leased, vacant and owned apartment along with details on pet's restriction (pets allowed or not).

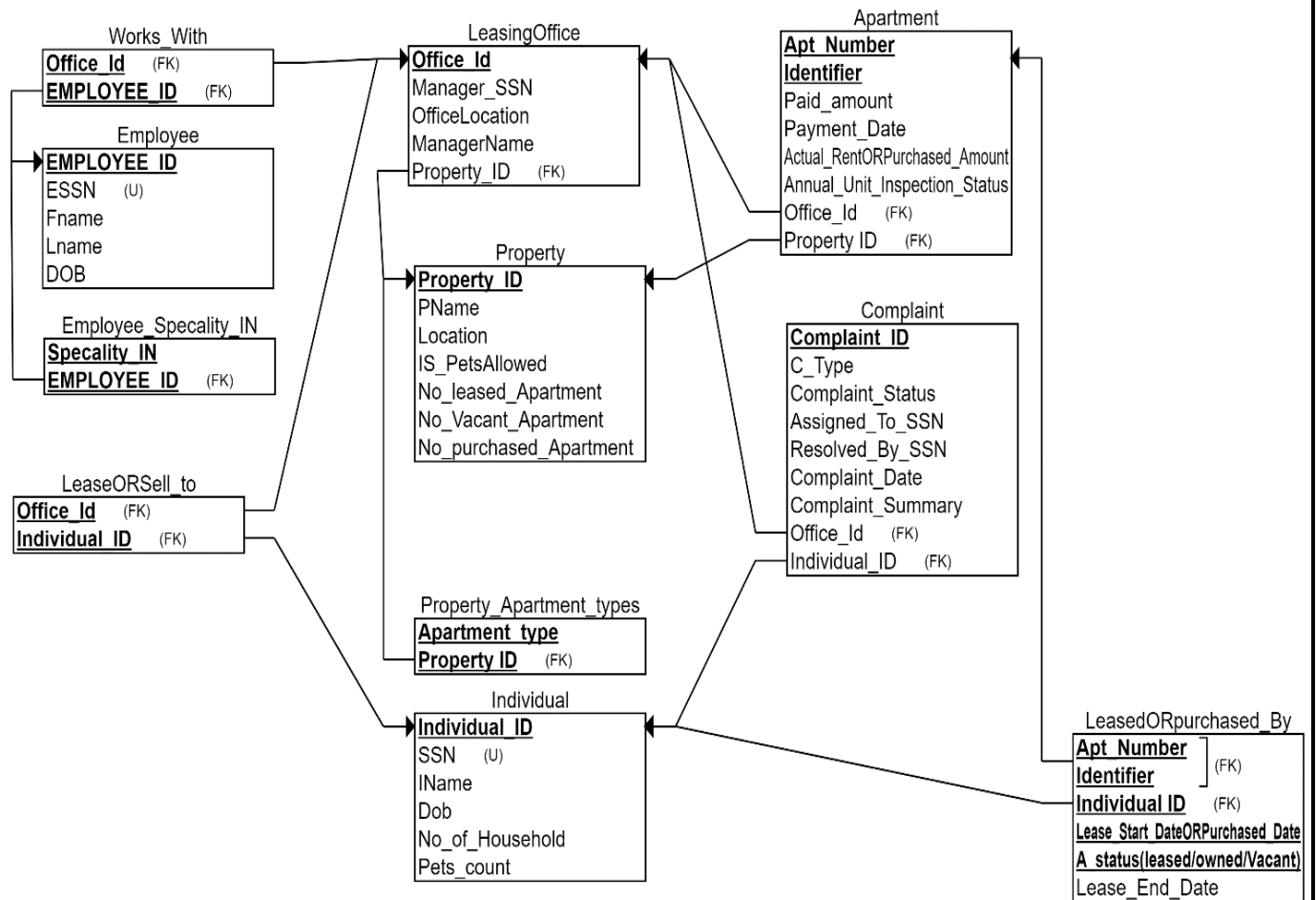
1. Each property contains many apartments built under it and each apartment contains Apartment ID which is a combination of Apartment number (apartment number) and identifier (building ID), details about rent called payment which is composite attribute of payment date, paid amount and actual annual lease or purchased amount where all attributes can be understood based of apartment current status.
2. Each property is administered by at most one Leasing Office. Each leasing office has Office ID, Manager\_SSN, name who manages the office and office location.
3. Leasing office leases or sells apartment to individuals and stores details about Apartment current status like (leased, owned or vacant), lease start date or purchased date wr.t to each

apartment and individual. Leasing office can lease apartments to any number of individuals.

4. Apartments are leased or owned by the individuals whose details contains data about individual ID(SSN), Individual name, DOB, number of household and pets stay in leased or owned apartment. One individual can lease one or more apartments and one apartment can be leased or purchased by one or more individuals who live in that apartment. Individual who leases the apartment can purchase same or different apartment. In addition to this each apartment has the annual inspection status populated.
5. An individual can submit number of complaints which will be managed by the leasing office of his resident apartment. Every complaint has complaint id, complaint type, status, assigned person SSN and resolved person SSN who actually resolved the complaint along with date of complaint registered. One complaint can be logged by one individual and managed or resolved by any one the leasing office, but an individual can log any number of complaints and leasing office can manage any number of complaints.
6. Every leasing office has employees who work with them. An employee has employee id, SSN, fname, lname, DOB and specialization. An employee can work for one or more leasing office but should work for at least at one leasing office. A leasing office can contain more than one employee.

**ER Diagram**

## Relational Database Schema



## **SQL Create Schema**

I have created a schema in MySQL named “HMS” technically the name of the database in which I will be creating the tables using the below queries.

### **Property Table:**

```
use HMS;
```

```
CREATE TABLE Property
```

```
(
```

```
Property_ID INT AUTO_INCREMENT PRIMARY KEY,
```

```
PName VARCHAR(15) NOT NULL UNIQUE ,
```

```
Location VARCHAR(30)NOT NULL,
```

```
IS_PetsAllowed CHAR(3) NOT NULL,
```

```
No_leased_Apartment INT NOT NULL DEFAULT '0',
```

```
No_Vacant_Apartment INT NOT NULL DEFAULT '0',
```

```
No_purchased_Apartment INT NOT NULL DEFAULT '0'
```

```
);
```

### **Leasing Office Table:**

```
CREATE TABLE LeasingOffice
```

```
(
```

```
Office_Id INT AUTO_INCREMENT PRIMARY KEY,
```

```
Manager_SSN VARCHAR(9) UNIQUE,
```

```
OfficeLocation VARCHAR(30)NOT NULL,
```

```
ManagerName VARCHAR(15)NOT NULL,
```

```
Property_ID INT NOT NULL ,
```

```
FOREIGN KEY (Property_ID) REFERENCES Property(Property_ID)ON DELETE CASCADE ON UPDATE  
CASCADE
```

```
);
```

**Apartment Table:**

```

CREATE TABLE Apartment
(
    Apt_Number INT AUTO_INCREMENT,
    Identifier varchar(5),
    Payment_Date DATE,
    Paid_amount FLOAT ,
    Actual_Annual_RentORPurchased_Amount FLOAT NOT NULL ,
    Annual_Unit_Inspection_Status CHAR(3) NOT NULL DEFAULT 'NO', # yes or No
    Office_Id INT NOT NULL,
    Property_ID INT NOT NULL,
    PRIMARY KEY(Apt_Number,Identifier),
    FOREIGN KEY (Office_Id) REFERENCES LeasingOffice(Office_Id) ON DELETE CASCADE ON UPDATE
    CASCADE,
    FOREIGN KEY (Property_ID) REFERENCES Property(Property_ID)ON DELETE CASCADE ON UPDATE
    CASCADE
);

```

**Individual Table:**

```

CREATE TABLE Individual
(
    Individual_ID INT AUTO_INCREMENT PRIMARY KEY,
    SSN VARCHAR(9) UNIQUE,
    IName VARCHAR(15) NOT NULL,
    Dob DATE NOT NULL,
    No_of_Household INT NOT NULL DEFAULT '0',
    Pets_count INT NOT NULL DEFAULT '0'
);

```



**Employee Table:**

```
CREATE TABLE Employee
(
  EMPLOYEE_ID INT AUTO_INCREMENT PRIMARY KEY,
  ESSN VARCHAR(9) UNIQUE,
  FName VARCHAR(15) NOT NULL,
  LName VARCHAR(15) NOT NULL,
  DOB DATE NOT NULL
);
```

**Employee Speciality IN Table:**

```
CREATE TABLE Employee_Speciality_IN
(
  Speciality_IN VARCHAR(15),
  EMPLOYEE_ID INT,
  PRIMARY KEY(Speciality_IN,EMPLOYEE_ID),
  FOREIGN KEY (EMPLOYEE_ID) REFERENCES Employee(EMPLOYEE_ID)ON DELETE CASCADE ON UPDATE
  CASCADE
);
```

**Complaint Table:**

```
CREATE TABLE Complaint
(
  Complaint_ID INT PRIMARY KEY AUTO_INCREMENT,
  C_Type VARCHAR(10),
  Complaint_Status VARCHAR(10) NOT NULL DEFAULT 'OPEN' CHECK (Complaint_Status='OPEN' OR
  Complaint_Status='ASSIGNED' OR Complaint_Status='INPROGRESS' OR Complaint_Status='RESOLVED'),
  Assigned_To_SSN VARCHAR(9),
  Resolved_By_SSN VARCHAR(9),
```

```

Complaint_Date DATE,

Complaint_Summary TEXT NOT NULL, #TEXT has no limit of characters MySQL dont support CLOB
BLOB.

Office_Id INT NOT NULL,

FOREIGN KEY (Office_Id) REFERENCES LeasingOffice(Office_Id) ON DELETE CASCADE ON UPDATE
CASCADE,

Individual_ID INT NOT NULL,

FOREIGN KEY(Individual_ID) REFERENCES Individual(Individual_ID) On DELETE CASCADE ON UPDATE
CASCADE

);

```

### **Works With Table:**

```

CREATE TABLE Works_With
(
    Office_Id INT,
    EMPLOYEE_ID INT,
    PRIMARY KEY (Office_Id, EMPLOYEE_ID),
    FOREIGN KEY (Office_Id) REFERENCES LeasingOffice(Office_Id) ON DELETE CASCADE ON UPDATE
    CASCADE,
    FOREIGN KEY (EMPLOYEE_ID) REFERENCES Employee(EMPLOYEE_ID) ON DELETE CASCADE ON UPDATE
    CASCADE
);

```

### **LeaseORSell to Table:**

```

CREATE TABLE LeaseORSell_to
(
    Individual_ID INT ,
    Office_Id INT,
    PRIMARY KEY (Office_Id, Individual_ID),
    FOREIGN KEY (Office_Id) REFERENCES LeasingOffice(Office_Id) ON DELETE CASCADE ON UPDATE
    CASCADE,

```

FOREIGN KEY (Individual\_ID) REFERENCES Individual(Individual\_ID) ON DELETE CASCADE ON UPDATE CASCADE

);

### **LeasedORpurchased By Table**

CREATE TABLE LeasedORpurchased\_By

(

Apt\_Number INT,

Identifier varchar(5),

Individual\_ID INT,

A\_status VARCHAR(8) NOT NULL, #leased or owned or vacant

Lease\_Start\_DateORPurchased\_Date DATE NOT NULL,

Lease\_End\_Date DATE,

PRIMARY KEY (Apt\_Number,Identifier, Individual\_ID,A\_status,Lease\_Start\_DateORPurchased\_Date),

FOREIGN KEY (Apt\_Number,Identifier) REFERENCES Apartment(Apt\_Number,Identifier) ON DELETE CASCADE ON UPDATE CASCADE,

FOREIGN KEY (Individual\_ID) REFERENCES Individual(Individual\_ID) ON DELETE CASCADE ON UPDATE CASCADE

);

### **Property Apartment type Table:**

CREATE TABLE Property\_Apartment\_type

(

Property\_ID INT,

Apartment\_type VARCHAR(5) CHECK(Apartment\_type='2BHK' OR Apartment\_type='3BHK' OR Apartment\_type='1BHK'),

PRIMARY KEY(Property\_ID,Apartment\_type),

FOREIGN KEY (Property\_ID) REFERENCES Property(Property\_ID) ON DELETE CASCADE ON UPDATE CASCADE

);

## SQL Queries

**##list vacant apartments(Apartment ID available at Dayton location for peppertree and along with property ID, Pname and location and Apartment ID.**

```
select P.Property_ID,PName,P. Location,A.Office_Id,A.Apt_Number,A.Identifier
```

```
from Apartment A , Property P
```

```
where (A.Apt_Number,A.Identifier) NOT IN (select L.Apt_Number,L.Identifier
```

```
from Apartment A,LeasedORpurchased_By L
```

```
where A.Apt_Number = L.Apt_Number
```

```
AND A.Identifier=L.Identifier
```

```
AND L.A_status IN("Leased","Owned"))
```

```
AND A.Property_ID=P.Property_ID
```

```
AND Location="Dayton" and PName="Peppertree";
```

**Result:**

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

```

101 # list vacant apartments(Apartment ID available at dayton location for peppertree
102 #and along with property ID,Pname and location and Apartment ID
103
104 select P.Property_ID,PName,P. Location,A.Office_Id,A.Apt_Number,A.Identifier
105 from Apartment A ,Property P
106 where (A.Apt_Number,A.Identifier) NOT IN
107 (select L.Apt_Number,L.Identifier
108 from Apartment A,LeasedORpurchased_By L
109 where A.Apt_Number = L.Apt_Number and A.Identifier=L.Identifier
110 AND L.A_status IN("Leased","Owned"))
111
112 AND A.Property_ID=P.Property_ID and
113 Location="Dayton" and PName="Peppertree";
114
115
116

```

The query is executed, and the result is displayed in the 'Result Grid' tab. The result shows one row of data:

Property_ID	PName	Location	Office_Id	Apt_Number	Identifier
1	Peppertree	Dayton	1	47	E1

The 'Output' tab shows the execution details:

#	Time	Action	Message	Duration / Fetch
1	09:41:45	select P.Property_ID,PName,P. Location,A.Office_Id,A.Apt_Number,A.Identifier from Apa...	1 row(s) returned	0.031 sec / 0.000 sec

The status bar at the bottom indicates 'Query Completed'.

**##Retrive All apartment Details(Apartemnt ID,OfficeID,Property Id) along with their Apartmnet vacany status (Note status will be null for vacant apartment.)**

```
select DISTINCT A.Apt_Number,A.Identifier ,L.A_status,Office_Id,Property_ID
```

```
from (Apartment A Left OUTER JOIN LeasedORpurchased_By L
```

```
on A.Apt_Number = L.Apt_Number and A.Identifier=L.Identifier );
```

**Result:**

The screenshot displays the SQL Server Enterprise Manager interface. The query editor shows the following SQL statement:

```
#Retrive All apartment Details(Apartemnt ID,OfficeID,Property Id) along with their Apartmnet vacany status
#Note status will be null for vacant apartment.
select DISTINCT A.Apt_Number,A.Identifier ,L.A_status,Office_Id,Property_ID
from (Apartment A Left OUTER JOIN LeasedORpurchased_By L
on A.Apt_Number = L.Apt_Number and A.Identifier=L.Identifier );
```

The result grid shows the following data:

Apt_Number	Identifier	A_status	Office_Id	Property_ID
43	A1	Leased	1	1
44	B1	Leased	1	1
45	C1	Leased	1	1
46	D1	Owned	1	1
47	E1	NULL	1	1
48	F1	Owned	1	1
49	AA1	Leased	2	2
50	AB1	Leased	2	2
51	AC1	Owned	2	2
52	BA1	NULL	3	3

The output pane at the bottom shows the execution details:

#	Time	Action	Message	Duration / Fetch
1	09:47:53	select DISTINCT A.Apt_Number,A.Identifier ,L.A_status,Office_Id,Property_ID from (Apt...	42 row(s) returned	0.031 sec / 0.000 sec

Query Completed

**## Retrieve apartment ID and Payment attributes for which the actual amount is not paid (actual amount – paid amount) due amount to be paid.**

```
select DISTINCT A.Apt_Number,A.Identifier,Payment_date,
Paid_amount, Actual_RentORPurchased_Amount ,
(Actual_RentORPurchased_Amount-Paid_amount) AS Tobepaid
from (Apartment A JOIN LeasedORpurchased_By L ON A.Apt_Number=L.Apt_Number)
JOIN Individual I ON L.Individual_ID = I.Individual_ID where (Actual_RentORPurchased_Amount-
Paid_amount)>0;
```

### Result:

The screenshot displays a database management interface with a left sidebar containing navigation menus for MANAGEMENT, INSTANCE, PERFORMANCE, and SCHEMAS. The main area shows a SQL query editor with the following text:

```
1 #Retrieve apartment ID and Payment attributes for which the actual amount is not paid
2 #i.e.(actual amount - paid amount) due amount to be paid.
3 select DISTINCT A.Apt_Number,A.Identifier,Payment_date,
4 Paid_amount ,Actual_RentORPurchased_Amount ,
5 (Actual_RentORPurchased_Amount-Paid_amount) AS Tobepaid
6 from (Apartment A JOIN LeasedORpurchased_By L ON A.Apt_Number=L.Apt_Number)
7 JOIN Individual I ON L.Individual_ID = I.Individual_ID
8 where (Actual_RentORPurchased_Amount-Paid_amount)>0;
```

Below the query editor, the 'Result Grid' shows two rows of data:

Apt_Number	Identifier	Payment_date	Paid_amount	Actual_RentORPurchased_Amount	Tobepaid
63	DB1	2018-10-31	2000	2410	410
49	AA1	2018-10-29	1260	2000	740

At the bottom, the 'Output' section shows the execution details for 'Action Output':

#	Time	Action	Message	Duration / Fetch
1	18:36:49	select DISTINCT A.Apt_Number,A.Identifier,Payment_date, Paid_amount ,Actual_RentORPur...	2 row(s) returned	0.047 sec / 0.000 sec

The status bar at the bottom indicates 'Query Completed'.

## ##Get the details of individual ID,SSN, Name and leasing office ID ,location who purchased apartment from property "Et Waters" with their date of purchase

Select I.Individual\_ID,SSN,IName,Lo.Office\_Id,

Lo.OfficeLocation,Lease\_Start\_DateORPurchased\_Date,Lp.A\_status,P.PName

from Individual I ,LeasingOffice Lo,Property P,LeaseORSell\_to Ls,LeasedORpurchased\_By Lp

where I.Individual\_ID = Lp.Individual\_ID AND Lp.A\_status = "Owned"

AND Ls.Individual\_ID=I.Individual\_ID AND Ls.Office\_Id = Lo.Office\_Id

AND Lo.Property\_ID =P.Property\_ID AND P.PName="Et Waters";

### Result:

The screenshot displays a database management interface with a left sidebar containing navigation menus for MANAGEMENT, INSTANCE, PERFORMANCE, and SCHEMAS. The main window shows a SQL query editor with the following text:

```

1  #get the details of individual ID,SSN, Name and leasing office ID ,location
2  #who purchased apartment from property "Et Waters" with their date of purchase
3  • select I.Individual_ID,SSN,IName,Lo.Office_Id,Lo.OfficeLocation,Lease_Start_DateORPurchased_Date,Lp.A_status,
4  from Individual I ,LeasingOffice Lo,Property P,LeaseORSell_to Ls,LeasedORpurchased_By Lp
5  where I.Individual_ID = Lp.Individual_ID AND Lp.A_status = "Owned"
6  AND Ls.Individual_ID=I.Individual_ID AND Ls.Office_Id = Lo.Office_Id
7  AND Lo.Property_ID =P.Property_ID AND P.PName="Et Waters";
8
9
10
11
12
13

```

Below the query editor, a "Result Grid" shows the query results. The grid has columns: Individual\_ID, SSN, IName, Office\_Id, OfficeLocation, Lease\_Start\_DateORPurchased\_Date, A\_status, and PName. One row is displayed:

Individual_ID	SSN	IName	Office_Id	OfficeLocation	Lease_Start_DateORPurchased_Date	A_status	PName
18	155671254	fewqwe	7	Phonixville	2018-11-05	Owned	Et Waters

At the bottom, an "Output" pane shows the execution details:

#	Time	Action	Message	Duration / Fetch
1	19:34:02	select I.Individual_ID,SSN,IName,Lo.Office_Id,Lo.OfficeLocation,Lease_Start_DateORP...	1 row(s) returned	0.047 sec / 0.000 sec

**# fetch the Complaint ID,status along with Individual ID,Name with their apartment ID who submitted compaints earlier than 7 days whose status is not resolved.**

```
select Complaint_ID,Complaint_Status,Complaint_Date ,DATEDIFF(CURDATE(),Complaint_Date) As Days,
I.Individual_ID ,IName,A.Apt_Number,A.Identifier
from Individual I,Complaint C , LeasedORpurchased_By L,Apartment A
where C.Individual_ID = I.Individual_ID and I.Individual_ID = L.Individual_ID
and L.Apt_Number=A.Apt_Number and L.Identifier=A.Identifier
and DATEDIFF(CURDATE(),Complaint_Date) >=7 and Complaint_Status !='RESOLVED' order by
Complaint_ID;
```

### **Result:**

The screenshot displays a database management interface with a SQL query editor and a results grid. The query is as follows:

```
43 # fetch the Complaint ID,status along with Individual ID,Name with their apartment ID
44 #who submitted compaints earlier than 7 days whose status is not resolved
45
46 select Complaint_ID,Complaint_Status,Complaint_Date ,DATEDIFF(CURDATE(),Complaint_Date) As Days,
47 I.Individual_ID ,IName,A.Apt_Number,A.Identifier
48 from Individual I,Complaint C , LeasedORpurchased_By L,Apartment A
49 where C.Individual_ID = I.Individual_ID and I.Individual_ID = L.Individual_ID
50 and L.Apt_Number=A.Apt_Number and L.Identifier=A.Identifier
51 and DATEDIFF(CURDATE(),Complaint_Date) >=7 and Complaint_Status !='RESOLVED' order by Complaint_ID;
52
53
```

The results grid shows the following data:

Complaint_ID	Complaint_Status	Complaint_Date	Days	Individual_ID	IName	Apt_Number	Identifier
3	INPROGRESS	2018-11-22	14	12	favr	56	CA1
4	INPROGRESS	2018-11-20	16	10	wer	50	AB1
5	ASSIGNED	2018-11-23	13	25	yhnt	55	BD1
6	ASSIGNED	2018-11-23	13	25	yhnt	55	BD1
7	ASSIGNED	2018-11-22	14	22	wegr	70	ED1

The interface also shows a left sidebar with navigation options like 'MANAGEMENT', 'INSTANCE', 'PERFORMANCE', and 'SCHEMAS'. The bottom section displays the execution output, indicating that the query was successful and returned 5 rows.



### ## Retrive names of individual whose are currently on lease.

select DISTINCT I.Individual\_ID,Iname,Apt\_Number,Identifier from Individual I, LeasedORpurchased\_By  
where (Apt\_Number,Identifier,I.Individual\_ID) IN (select Apt\_Number,Identifier,Individual\_ID from  
LeasedORpurchased\_By where A\_Status ="Leased" and Lease\_End\_Date IS NULL);

### Result:

The screenshot shows the SQL Server Enterprise Manager interface. The left pane displays the 'SCHEMAS' tree with 'LeasedORpurchased\_By' selected. The central pane shows the following SQL query:

```
# Retrive names of individual along with apartment ID whose are currently on lease
select DISTINCT I.Individual_ID,Iname,Apt_Number,Identifier
from Individual I, LeasedORpurchased_By
where (Apt_Number,Identifier,I.Individual_ID) IN (select Apt_Number,Identifier,Individual_ID
from LeasedORpurchased_By
where A_Status ="Leased" and Lease_End_Date IS NULL);
```

The 'Result Grid' displays the following data:

Individual_ID	Iname	Apt_Number	Identifier
1	Ahdi	43	A1
15	fewcqw	43	A1
2	grrg	44	B1
17	wer3	44	B1
24	efw	44	B1
5	grfe	45	C1

The 'Output' pane shows the execution message: '1 22:43:12 select DISTINCT I.Individual\_ID,Iname,Apt\_Number,Identifier from Individual I, LeasedORpurchased\_By where (Apt\_Number,Identifier,I.Individual\_ID) IN (select Apt\_Number,Identifier,Individual\_ID from LeasedORpurchased\_By where A\_Status ="Leased" and Lease\_End\_Date IS NULL); 15 row(s) returned'. The duration is 0.047 sec / 0.000 sec.

### ## update lease end date of any apartment whose Lease start date is 2014 to "2018-01-07" and lease\_end\_date is null for Apatment ID(69,EC1)

update LeasedORpurchased\_By set Lease\_End\_Date = '2018-06-07' where Lease\_End\_Date IS NULL and Lease\_Start\_DateORPurchased\_Date and YEAR(Lease\_Start\_DateORPurchased\_Date) =2014 and Apt\_Number='69' and Identifier='EC1';

### Result:

The screenshot shows the SQL Server Enterprise Manager interface. The left pane displays the 'SCHEMAS' tree with 'LeasedORpurchased\_By' selected. The central pane shows the following SQL query:

```
#update lease end date of any apartment whose Lease start date is 2014
#to "2018-01-07" and lease_end_date is null for Apatment ID(69,EC1)
update LeasedORpurchased_By
set Lease_End_Date = '2018-06-07'
where Lease_End_Date IS NULL and Lease_Start_DateORPurchased_Date and
YEAR(Lease_Start_DateORPurchased_Date) =2014 and Apt_Number='69' and Identifier='EC1';
```

The 'Output' pane shows the execution message: '1 22:40:28 update LeasedORpurchased\_By set Lease\_End\_Date = '2018-06-07' where Lease\_End\_Date IS NULL and Lease\_Start\_DateORPurchased\_Date and YEAR(Lease\_Start\_DateORPurchased\_Date) =2014 and Apt\_Number='69' and Identifier='EC1'; 1 row(s) affected Rows matched: 1 Changed: 1 Warnings: 0'. The duration is 0.110 sec.

## ## Insert values into table Property

```
INSERT INTO Property(
```

```
PName ,Location,IS_PetsAllowed ,No_leased_Apartment ,No_Vacant_Apartment ,
```

```
No_purchased_Apartment )
```

```
values
```

```
("West Ghoshen","Downington","No",2,5,1),
```

```
("UNC Homes","Exton","Yes",2,6,1),
```

```
("Harrison Hill","West Chester","Yes",2,3,1);
```

## Result:

The screenshot displays the SQL Server Enterprise Manager interface. The left pane shows the 'SCHEMAS' tree with 'Employee\_Specialty\_IN' expanded. The main query window contains the following SQL script:

```
#insert values into table Property
INSERT INTO Property(
  PName ,
  Location,
  IS_PetsAllowed ,
  No_leased_Apartment ,
  No_Vacant_Apartment ,
  No_purchased_Apartment )
values
('West Ghoshen','Downington','No',2,5,1),
('UNC Homes','Exton','Yes',2,6,1),
('Harrison Hill','West Chester','Yes',2,3,1);
```

The 'Output' pane at the bottom shows the execution results:

#	Time	Action	Message	Duration / Fetch
1	23:04:10	INSERT INTO Property( PName , Location, IS_PetsAllowed , No_leased_Apartment , No_Vacant_Apartment , No_purchased_Apartment )	3 row(s) affected Records: 3 Duplicates: 0 Warnings: 0	0.094 sec

The status bar at the bottom indicates 'Query Completed'.

**## Retrive Employee details who work for more then one leasing office along with number of office he/she work with.**

```
select E.EMPLOYEE_ID,ESSN,FName,LName,DOB,count(*)
```

```
from Works_With W,Employee E
```

```
where W.EMPLOYEE_ID=E.EMPLOYEE_ID
```

```
Group by EMPLOYEE_ID
```

```
having Count(*)>1;
```

### **Result:**

The screenshot displays the SQL Server Enterprise Manager interface. The left pane shows the 'SCHEMAS' tree with 'Complaint\_Summary', 'Office\_Id', and 'Individual\_Id' selected. The main pane shows a query window with the following SQL code:

```
#retrive Employee details who work for more then one leasing office
#along with number of office he/she work with .

select E.EMPLOYEE_ID,ESSN,FName,LName,DOB,count(*)
from Works_With W,Employee E
where W.EMPLOYEE_ID=E.EMPLOYEE_ID
Group by EMPLOYEE_ID
having Count(*)>1;
```

The query results are displayed in a table with the following columns: EMPLOYEE\_ID, ESSN, FName, LName, DOB, and count(\*). The results show three rows of data:

EMPLOYEE_ID	ESSN	FName	LName	DOB	count(*)
1	5455682	Dqwa	wfefive	1948-06-13	4
2	2455385	ffs	wsfe	2014-08-19	2
5	2745472	fsd	sf	2012-03-17	2

The bottom pane shows the 'Output' tab with the following information:

#	Time	Action	Message	Duration / Fetch
1	23:29:23	select E.EMPLOYEE_ID,ESSN,FName,LName,DOB,count(*) from Works_With W,Empl...	3 row(s) returned	0.031 sec / 0.000 sec

The status bar at the bottom indicates 'Query Completed'.

## Alter Table:

Alter table Apartment change Actual\_Annual\_RentORPurchased\_Amount  
Actual\_RentORPurchased\_Amount FLOAT NOT NULL;

## Result:

The screenshot shows the SQL Server Enterprise Manager interface. The left pane displays the 'SCHEMAS' tree with 'Works\_With' selected. The main pane shows the execution of the following SQL statement:

```
Alter table Apartment change Actual_Annual_RentORPurchased_Amount Actual_RentORPurchased_Amount FLOAT NOT NULL;
```

The 'Output' pane shows the execution results:

#	Time	Action	Message	Duration / Fetch
1	18:31:51	Alter table Apartment change Actual_Annual_RentORPurchased_Amount Actual_RentORPurchased_Amount FLOAT NOT NULL;	0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0	0.125 sec

The 'Query Completed' message is displayed at the bottom.

The screenshot shows the SQL Server Enterprise Manager interface. The left pane displays the 'SCHEMAS' tree with 'Works\_With' selected. The main pane shows the execution of the following SQL statement:

```
select * from Apartment;
```

The 'Result Grid' displays the following data:

Apt_Number	Identifier	Payment_Date	Paid_amount	Actual_RentORPurchased_Amount	Annual_Unit_Inspection_Status	Office_Id	Property_ID
43	A1	2018-12-05	1200	1200	YES	1	1
44	B1	2018-11-20	1210	1140	YES	1	1
45	C1	2018-11-25	1100	1100	NO	1	1
46	D1	2018-10-29	1100	1100	YES	1	1
47	E1	2018-10-28	1000	1200	NO	1	1
48	F1	2018-10-29	1100	1100	YES	1	1

The 'Output' pane shows the execution results:

#	Time	Action	Message	Duration / Fetch
1	18:31:51	Alter table Apartment change Actual_Annual_RentORPurchased_Amount Actual_RentORPurchased_Amount FLOAT NOT NULL;	0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0	0.125 sec
2	18:33:11	select * from Apartment LIMIT 0, 1000	42 row(s) returned	0.031 sec / 0.000 sec

The 'Query Completed' message is displayed at the bottom.

## **Discussion points**

I initially started this project with the question what database I can design, eventually I came across designing a database for Housing Management System (HMS). While doing this project starting from requirement gathering it has given me an exposure to the real-world experience of entity, attribute and relation along with structural constraints. While converting requirements to ER diagram and to relational schema theoretically everything looks fine until we actually implement it. While implementing the relational database schema I came across some situations when I needed to revisit the ER diagram and realized the mistake of my existing problem about the relation between two entities. My existing knowledge about relation was while defining structural constraint I have defined them in ER diagram with respect to relation itself instead of considering constraint w.r.t to two entity with action item w.r.t to relation.

For example: In the initial ER diagram I initially defined cardinality between individual and Apartment w.r.t leasing information or purchase information as many to many relation as individual can lease or own any number of apartments. But the question comes when we think about the status of an apartment along with purchaser leasing date. Which can be multivalued and composite data as we need to store historical data about on which date who purchased or leased the apartment along with lease end date

With this mistake I had to update my ER diagram and relational schema which resulted in rework which made me clear that clear requirements and correct understanding of the requirement is essential to avoid rework that may occur huge costs in real time projects.

Apart from this I also learned about some datatypes in MySQL. MySQL uses TEXT datatype rather than BLOB/CLOB and we can't use TEXT as primary key so instead we need VARCHAR. Initially When I designed ER diagram with Complaint ID and Complaint summary as relation attributes I came across situation of added TEXT data type as primary key but encountered SQL error, that is when I realized TEXT can't be used as primary key since the size is not fixed. Later after through analysis of the requirement the attribute "Complaint\_ID" was being used with Entity rather than an attribute on Relation which uniquely identify the tuple in Database instance. I also came to know about the normalization and de-normalization process and their importance. In my project most of the tables are normalized and that has given the advantages but also it made me realize about the big queries that need to be written to get the data. Also, when the table size grows, and joins increase execution time becomes the issue and that is when de normalization is required. I came to know that not always normalization is helpful.

I have used Amazon AWS EC2 instance to run the MySQL database and used the MySQL workbench to connect to the database. This was a learning curve which gave me exposure to AWS and hosting and running DB instances on AWS.

I also learned that in real world scenario there is no physical delete operations that are performed on the data. Instead the delete operation are actually update on a column (like Active\_Indicator which is either 0 or 1) and which identifies a row as active or inactive. This helps to maintain the history of the data. However I have not implemented Active indicators in the current Database because it was late by the time I realized it and did not want to make changes at last moment due to time constraints.

Finally by concluding the discussion I would like to add future scope that this system can be further enhanced to support the features like scheduling visits, notifying the future residents about schedules, reminding tenants for their rent payments and a system for notifying users who subscribe about the availability and other details.