# PROPERTY – VENDOR MANAGEMENT SYSTEM

# FINAL REPORT

**IE 6700 Data Management for Analytics**

**Instructor:** Prof. Venkat Krishnamurthy

**GROUP 9:**

Medhavi Uday Pande

Param Madan

Rashmi Daga

**INDEX-**

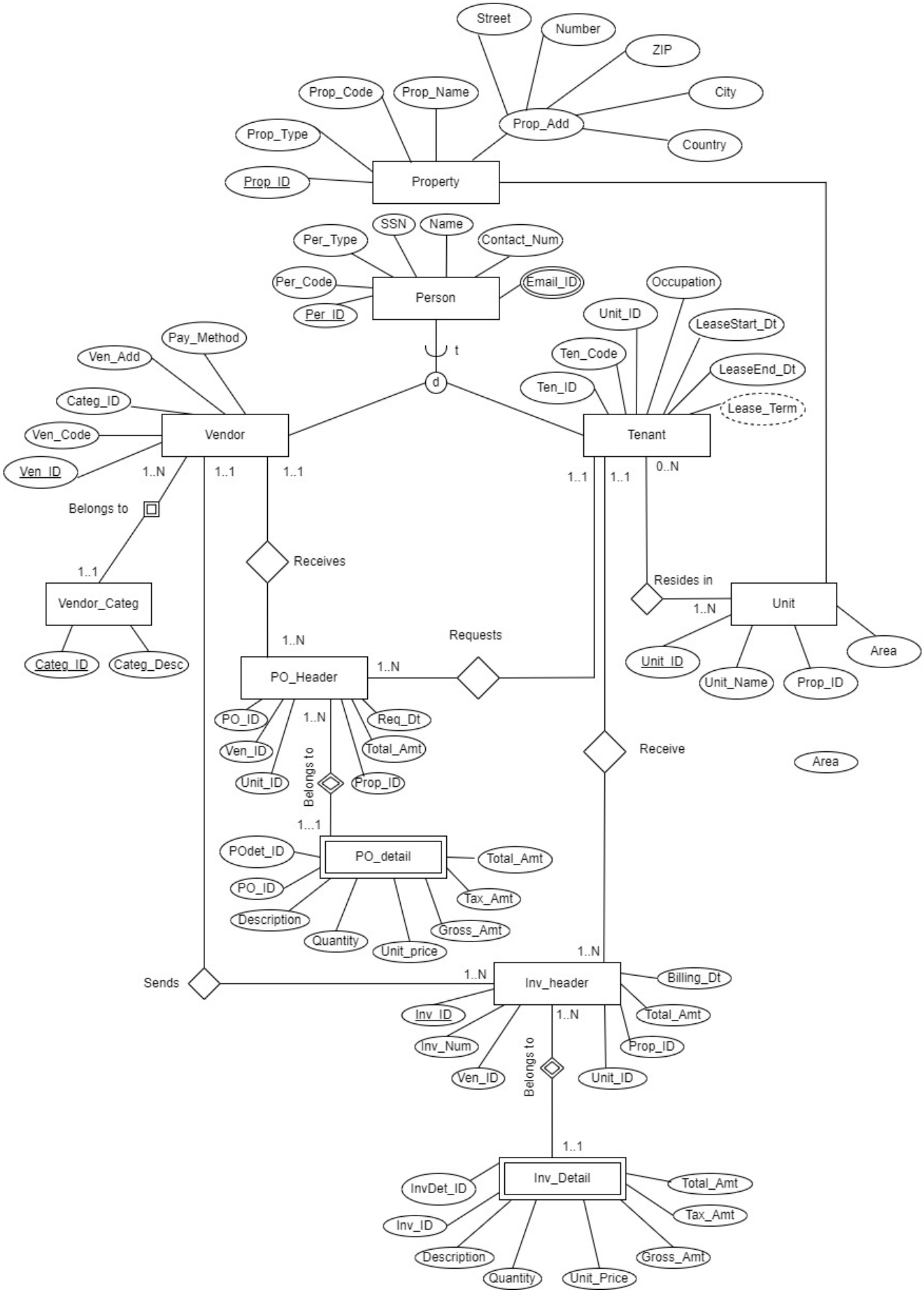
## BUSINESS PROBLEM DEFINITION

As a property management company, we are constantly seeking innovative solutions to enhance our property management processes. To this end, we have undertaken the task of developing a comprehensive Property Management System. The system is designed to streamline and optimize various aspects of property management, including property, units, tenants, vendor management, purchase orders, invoices, and more. We need you to design the database for this system. Your mission is to create an Enhanced Entity-Relationship Diagram (EERD) that effectively represents the entire property management ecosystem. The ERD should cover the following key components:

1. Properties and Units: We manage multiple properties, each of which contains various units. Each unit can be unique in terms of its characteristics and availability. Design a schema that efficiently tracks these properties and their associated units.
2. Tenants: Tenants reside in specific units. Create a mechanism to manage tenant information and their associations with units.
3. Vendors: We engage various vendors to provide goods and services such as maintenance, repair, cleaning, electricity and more. These vendors need to be organized based on their types and categories.
4. Vendor Categories: We need to classify the vendors based on Vendor Category. Each vendor should belong to one category, helping us manage various vendor services efficiently.
5. Purchase Orders: Tenants can request Purchase orders (POs) for specific services required at their units. The POs must include the necessary information such as date of request, total amount, etc.
6. Purchase Order Details: Each purchase order will have specific details regarding the requested service, including its expected description, quantity, unit price, tax, total price, etc. and other related information.
7. Invoices: After vendors provide services, they send invoices to our property management company. The database should be able to record these Invoices including the amount, invoice number, billing date, etc.
8. Invoice Details: Invoices will contain detailed information about the services or goods provided, their costs, quantity, tax, description and other relevant details.

Construct a comprehensive ERD that incorporates all these entity types, attribute types, and their relationships.

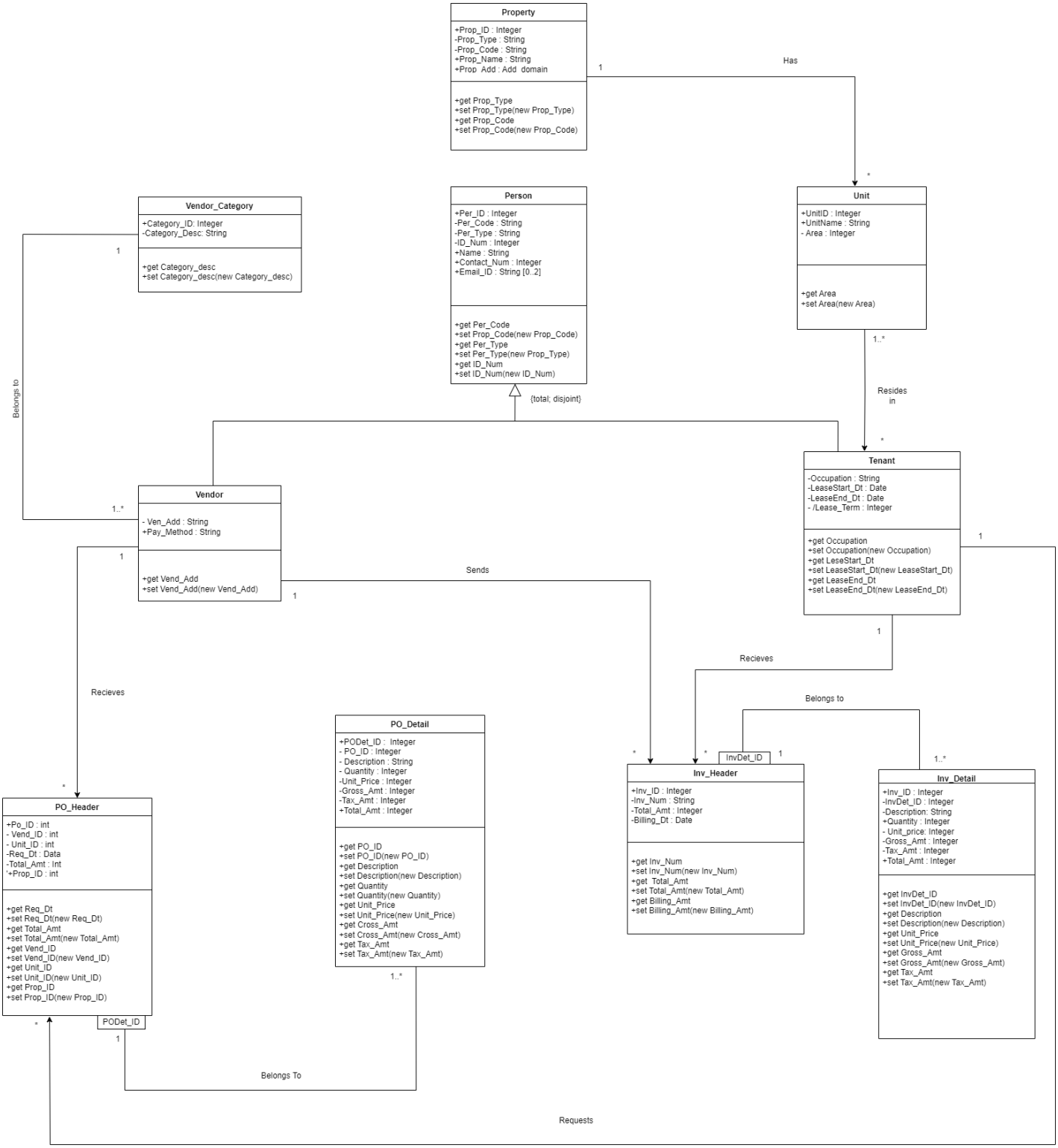
# ENHANCED ENTITY- RELATIONSHIP DIAGRAM



# REFERENCE DATA

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| **REFERENCE DATA** | | | | | | | | |
| | | | | | | | | |
| **Abbreviations** | | | | | | | | |
| PK= Primary Key | | | | | | | | |
| AT= Attribute Type | | | | | | | | |
| FK= Foreign Key | | | | | | | | |
| | | | | | | | | |
| | | **Property** | | **Person** | | **Vendor** | | **Tenant** |
| | PK, Auto Increment | Prop_ID | PK, Auto Increment | Per_ID | FK = PK of Person | Ven_ID | FK = PK of Person | Ten_ID |
| | | Prop_Code | | Per_Code | | Ven_Code | | Ten_Code |
| | | Prop_Name | | Per_Type (Vendor/Tenant) | FK = PK of Vendor_Categ | Categ_ID | FK = PK of Unit | Unit_ID |
| | | Prop_Type | | SSN | | Ven_Add | | Occupation |
| | Composite AT | Prop_Add | | Name | | Pay_Method | | LeaseStart_Dt |
| | | Street | | Contact_Num | | | | LeaseEnd_Dt |
| | | Number | Multi-valued AT | Email ID | | | Derived AT = LeaseEnd_Dt - LeaseStart_Dt | LeaseTerm |
| | | ZIP | | | | | | |
| | | City | | | | | | |
| | | Country | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | **Unit** | | **Vendor_Categ** | | | | |
| | PK, Auto Increment | Unit_ID | PK, Auto Increment | Categ_ID | | | | |
| | | Unit_Name | | Categ_Desc | | | | |
| | FK = PK of Property | Prop_ID | | | | | | |
| | | Area | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | **PO_Header** | | **PO_Detail** | | **Inv_Header** | | **Inv_Detail** |
| | PK, Auto Increment | PO_ID | Auto Increment | POdet_ID | Auto Increment | Inv_ID | Auto Increment | InvDet_ID |
| | FK = PK of Person | Ven_ID | PK of PO_Header | PO_ID | | Inv_Num | PK of Inv_Detail | Inv_ID |
| | FK = PK of Unit | Unit_ID | | Description | FK = PK of Person | Ven_ID | | Description |
| | FK = PK of Property | Prop_ID | | Quantity | FK = PK of Unit | Unit_ID | | Quantity |
| | | Total_Amt | | Unit_Price | FK = PK of Property | Prop_ID | | Unit_Price |
| | | Req_Dt | | Gross_Amt | | Total_Amt | | Gross_Amt |
| | | | | Tax_Amt | | Billing_Dt | | Tax_Amt |
| | | | | Total_Amt | | | | Total_Amt |

# UML Class Diagram



## RELATIONAL MODEL MAPPED FROM CONCEPTUAL MODEL

Note- Primary Keys are written in red and underlined. Foreign Keys are written in green and italics.

- Property (Prop_ID, Prop_Code, Prop_Name, Prop_Address)

- Unit (Unit_ID, Unit_Name, Area, *Prop_ID*)
  Prop_ID is a Foreign Key Referencing to 'Prop_ID' of relation 'Property'; NULL is not allowed.

- Person (Per_ID, Per_Code, Per_Type, ID_Num, Per_Name, Contact_Num)

- Person_Email (*Per_ID*, Email)
  Per_ID and Email together make the primary key.
  Per_ID is the foreign key referencing to Per_ID in the relation 'Person'; NULL is not allowed.

- Tenant (*Per_ID*, Occupation, LeaseStart_Dt, LeaseEnd_Dt, LeaseTerm, Prop_ID)
  Per_ID serves as the primary key as well as the Foreign Key for this relation. It references the Per_ID of the relation 'Person': NULL is not allowed.

- Vendor (*Per_ID*, Ven_Add, Pay_Method, *Categ_ID*)
  Per_ID serves as the Primary Key as well as the Foreign Key for this relation. It references the Per_ID of the relation 'PersGron': NULL is not allowed.
  Categ_ID is a Foreign Key referencing to the Categ_ID of the relation 'Vendor_Categ': NULL is not allowed.

- Vendor_Categ (Categ_ID, Categ_Desc)

- PO_Header (PO_ID, TotalAmt, ReqDt, *Ven_ID*, *Ten_ID*)
  Ven_ID is the Foreign Key referencing to 'Ven_ID' in the relation 'Vendor'; NULL is not allowed.
  Ten_ID is the Foreign Key referencing to 'Ten_ID' in the relation 'Tenant'; NULL is not allowed.

- PO_Detail (PODet_ID, *PO_ID*, Description, Quantity, UnitPrice, GrossAmt, TaxAmt, TotalAmt)
  PODet_ID and PO_ID together make the Primary Key.
  PO_ID is Foreign Key referencing to 'PO_ID' in the relation 'PO_Header'; NULL is not allowed.

- Inv_Header (Inv_ID, Inv_Num, TotalAmt, BillingDt, *Ven_ID*, *Ten_ID*)
  Ven_ID is the Foreign Key referencing to 'Ven_ID' in the relation 'Vendor'; NULL is not allowed.
  Ten_ID is the Foreign Key referencing to 'Ten_ID' in the relation 'Tenant'; NULL is not allowed.

- Inv_Detail (InvDet_ID, *Inv_ID*, Description, Quantity, UnitPrice, GrossAmt, TaxAmt, TotalAmt)
  InvDet_ID and Inv_ID together make the Primary Key.
  Inv_ID is Foreign Key referencing to 'Inv_ID' in the relation 'Inv_Header'; NULL is not allowed.

## NORMALIZATION UP TO AT LEAST 3.5 NF

The above relation is normalized and cannot be normalized any further. Further stated are some assumptions, considering which we can conclude the following-

1. No multi-valued and composite attribute types exist.
2. All non-prime attribute types are fully functionally dependent on a key.
3. No non-prime attribute types are transitively dependent on the primary key.
4. For each non-trivial functional dependency X -> Y, X is a super key.
5. For non-trivial multi-valued dependencies, X -> Y - > Z, X is a super key.

**Assumptions-**

1. Prop_Code, Prop_Name, Prop_Add depend on Prop_ID in the relation Property.
2. Unit_Name and Area depend on Unit_ID in the relation Unit.
3. Per_Code, Per_Type, ID_Num, Per_Name, Contact_Num depend on Person.
4. Email is multi-value attribute type for a Person, and depends on Per_ID.
5. Occupation, LeaseStart_Dt. LeaseEnd_Dt, LeaseTerm depend on Per_ID in the relation Tenant.
6. Ven_Add, Pay_Method depend on Per_ID in Vendor.
7. Categ_Desc in Vendor_Categ depends on Categ_ID.
8. TotalAmt, ReqDt, Ven_ID, Ten_ID in the PO_Header depend on the PO_ID.
9. Description, Qunaitity, UnitPrice, GrossAmt, TaxAmt, TotalAmt depend on PODet_ID and PO_ID in the relation PO_Detail.
10. TotalAmt, BillingDt, Ven_ID, Ten_ID in the Inv_Header depend on the Inv_ID.
11. Description, Qunaitity, UnitPrice, GrossAmt, TaxAmt, TotalAmt depend on InvDet_ID and Inv_ID in the relation Inv_Detail.

# DDL QUERIES (TABLE CREATION, TRIGGERS, STORED PROCEDURES)

1. **DDL Query for creating procedure:** Created procedure to calculate the amounts in the PO related tables.
   1) Calculated the Gross Amount based on the Quantity and Unit Price in the PO Detail lines.
   2) Calculated the Tax Amount i.e. 10 % of the Gross Amount in the PO Detail lines.
   3) Calculated Total Amount as the sum of the Gross and Tax Amount in the PO Detail lines.

```
CREATE PROCEDURE UpdatePOAmt (PO_ID INT)
BEGIN
UPDATE PO_Detail SET GrossAmt = Quantity * UnitPrice WHERE PO_ID = @PO_ID;
UPDATE PO_Detail SET TaxAmt = GrossAmt*0.1 WHERE PO_ID = @PO_ID;
UPDATE PO_Detail SET TotalAmt = GrossAmt + TaxAmt WHERE PO_ID = @PO_ID;
UPDATE PO_Header PO SET PO.TotalAmt= (SELECT sum(d.totalamt)
                                      FROM po_detail d
                                      WHERE d.po_id = po.po_id
                                      GROUP BY d.po_id)
WHERE PO.PO_ID = @PO_ID;
END;
CALL UpdatePOAmt(3728);
```

2. **DDL Query for creating procedure**: Created procedure to calculate the amounts in the PO related tables.
   1) Calculated the Gross Amount based on the Quantity and Unit Price in the PO Detail lines.
   2) Calculated the Tax Amount i.e. 10 % of the Gross Amount in the PO Detail lines.
   3) Calculated Total Amount as the sum of the Gross and Tax Amount in the PO Detail lines.

```
CREATE PROCEDURE UpdateInvAmt (Inv_ID INT)
BEGIN
UPDATE Inv_Detail SET GrossAmt = Quantity * UnitPrice WHERE Inv_ID = @Inv_ID;
UPDATE Inv_Detail SET TaxAmt = GrossAmt*0.1 WHERE Inv_ID = @Inv_ID;
UPDATE Inv_Detail SET TotalAmt = GrossAmt + TaxAmt WHERE Inv_ID = @Inv_ID;
UPDATE Inv_Header ih SET ih.TotalAmt = (SELECT sum(d.totalamt)
                                        FROM inv_detail d
                                        WHERE d.inv_id = ih.inv_id
                                        GROUP BY d.inv_id)
 WHERE ih.inv_ID = @Inv_ID;
 END;
CALL UpdateInvAmt(1577);
```

3. **DDL Queries for creating tables:** Create table queries are listed below.

```
CREATE TABLE Property (
   Prop_ID int NOT NULL AUTO_INCREMENT,
   Prop_Code varchar(255) NOT NULL,
   Prop_Name varchar(255) NOT NULL,
   Prop_Address varchar(255),
   Prop_City varchar(255),
   Prop_State varchar(255),
   Prop_Zip int,
   PRIMARY KEY (Prop_ID)
);
```

```
CREATE TABLE Unit (
        Unit_ID int NOT NULL AUTO_INCREMENT,
   Unit_Name varchar(255) NOT NULL,
   Unit_Area int,
   Prop_ID int,
   PRIMARY KEY (Unit_ID),
   FOREIGN KEY (Prop_ID) REFERENCES Property(Prop_ID)
);


CREATE TABLE Person (
        Per_ID int NOT NULL AUTO_INCREMENT,
   Per_Code varchar(255) NOT NULL,
   Per_Type varchar(255) NOT NULL,
   ID_Num bigint(8) NOT NULL,
   Per_Name varchar(255),
   Contact_Num bigint(8),
   PRIMARY KEY (Per_ID)
);

CREATE TABLE Person_Email (
        Per_ID int NOT NULL,
   Email varchar(255) NOT NULL,
   CONSTRAINT PK_Person_Email PRIMARY KEY (Per_ID, Email),
   FOREIGN KEY (Per_ID) REFERENCES Person(Per_ID)
);

CREATE TABLE Tenant (
        Per_ID int NOT NULL,
   Occupation varchar(255),
   LeaseStart_Dt datetime,
   LeaseEnd_Dt datetime,
   LeaseTerm int,
   Prop_ID int,
   PRIMARY KEY (Per_ID),
        FOREIGN KEY (Per_ID) REFERENCES Person(Per_ID),
        FOREIGN KEY (Unit_ID) REFERENCES Unit(Unit_ID)
);

CREATE TABLE Vendor_Categ (
        Categ_ID int NOT NULL AUTO_INCREMENT,
   Categ_Desc varchar(255) NOT NULL,
   PRIMARY KEY (Categ_ID)
);

CREATE TABLE Vendor (
        Per_ID int NOT NULL,
   Ven_Add varchar(255),
   Pay_Method varchar(255),
   Categ_ID int,
   PRIMARY KEY (Per_ID),
        FOREIGN KEY (Per_ID) REFERENCES Person(Per_ID),
        FOREIGN KEY (Categ_ID) REFERENCES Vendor_Categ(Categ_ID)
);
```

```
CREATE TABLE PO_Header (
        PO_ID int NOT NULL AUTO_INCREMENT,
   TotalAmt int,
   ReqDt datetime,
   Ven_ID int NOT NULL,
   Ten_ID int NOT NULL,
   PRIMARY KEY (PO_ID),
        FOREIGN KEY (Ven_ID) REFERENCES Vendor(Per_ID),
        FOREIGN KEY (Ten_ID) REFERENCES Tenant(Per_ID)
);

CREATE TABLE PO_Detail (
        PODet_ID int NOT NULL AUTO_INCREMENT,
   PO_ID int NOT NULL,
   Det_Desc varchar(255),
   Quantity int,
   UnitPrice int,
   GrossAmt int,
   TaxAmt int,
   TotalAmt int,
   CONSTRAINT PK_PO_Detail PRIMARY KEY (PODet_ID, PO_ID),
   FOREIGN KEY (PO_ID) REFERENCES PO_Header(PO_ID)
);

CREATE TABLE Inv_Header (
        Inv_ID int NOT NULL AUTO_INCREMENT,
   Inv_Num int,
   TotalAmt int,
   BillingDt datetime,
   Ven_ID int NOT NULL,
   Ten_ID int NOT NULL,
   PRIMARY KEY (Inv_ID),
        FOREIGN KEY (Ven_ID) REFERENCES Vendor(Per_ID),
        FOREIGN KEY (Ten_ID) REFERENCES Tenant(Per_ID)
);

CREATE TABLE Inv_Detail (
        InvDet_ID int NOT NULL AUTO_INCREMENT,
   Inv_ID int NOT NULL,
   Det_Desc varchar(255),
   Quantity int,
   UnitPrice int,
   GrossAmt int,
   TaxAmt int,
   TotalAmt int,
   CONSTRAINT PK_Inv_Detail PRIMARY KEY (InvDet_ID, Inv_ID),
   FOREIGN KEY (Inv_ID) REFERENCES Inv_Header(Inv_ID)
);
```

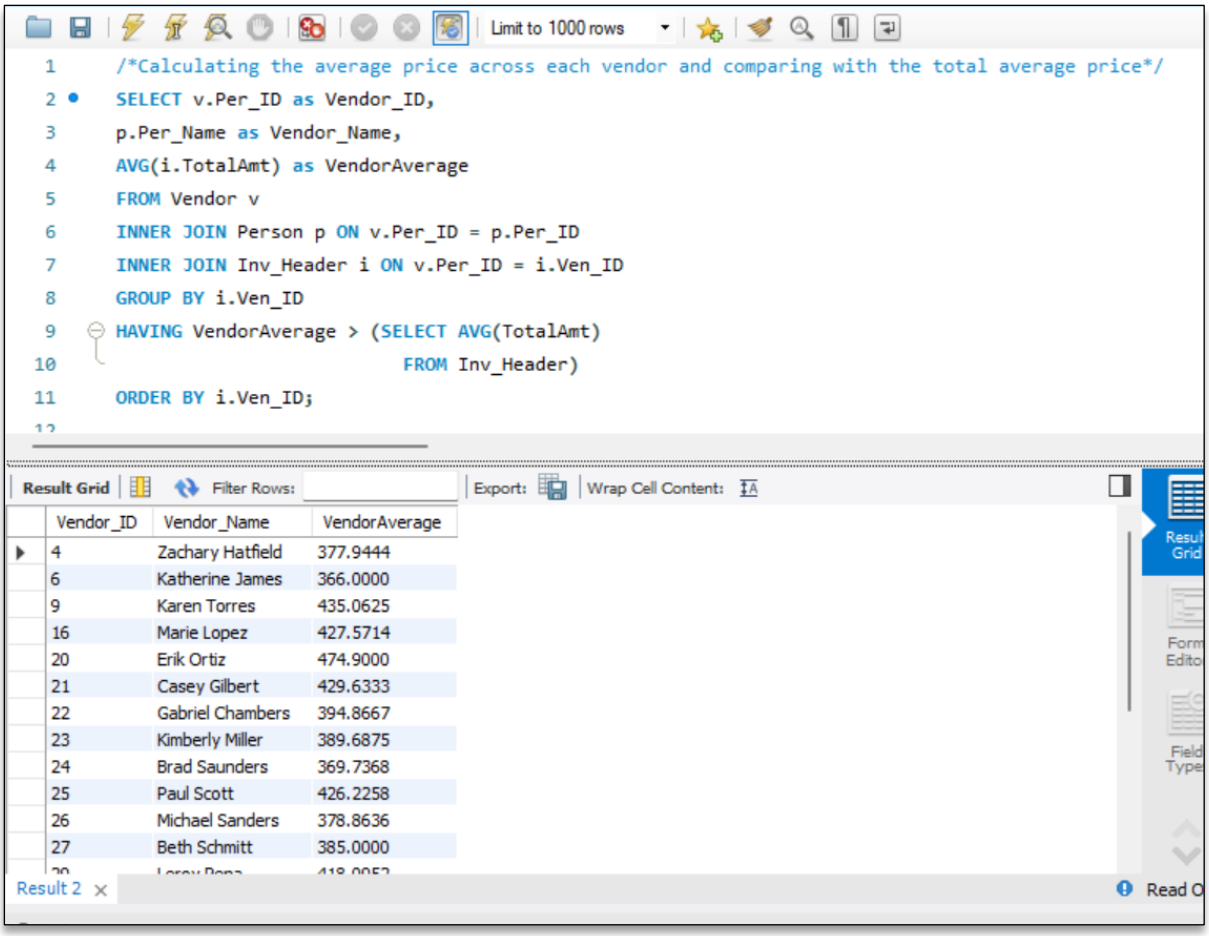
## DML QUERIES, ANALYTICAL PURPOSES, OUTPUTS

1. **DDL Query for creating triggers**: Created Triggers to calculate the lease term, which is derived from lease start date and lease end date. One trigger has been made to update lease term on inserting new lease. Another has been made to update lease term on modifying the lease dates of an existing lease.

```
DELIMITER //
-- Trigger for BEFORE INSERT
CREATE TRIGGER CalculateLeaseTerm_Insert
BEFORE INSERT ON Tenant
FOR EACH ROW
BEGIN
-- Calculate LeaseTerm
SET NEW.LeaseTerm = TIMESTAMPDIFF(MONTH, NEW.LeaseStart_Dt, NEW.LeaseEnd_Dt);
END;
//
-- Trigger for BEFORE UPDATE
CREATE TRIGGER CalculateLeaseTerm_Update
BEFORE UPDATE ON Tenant
FOR EACH ROW
BEGIN
 -- Calculate LeaseTerm
SET NEW.LeaseTerm = TIMESTAMPDIFF(MONTH, NEW.LeaseStart_Dt, NEW.LeaseEnd_Dt);
END;
//
DELIMITER ;
```

2. **Vendor Prices Analysis:** Fetching all the vendors whose total invoice amount is greater than the average total invoice amount.
   **Analytical purpose:** This will help the property managers contact such vendors and negotiate on the basis of statistics to charge less. Eventually, they can reduce business with such vendors or use these statistics to negotiate with the new vendors.

```
SELECT v.Per_ID as Vendor_ID,
p.Per_Name as Vendor_Name,
AVG(i.TotalAmt) as VendorAverage
FROM Vendor v
INNER JOIN Person p ON v.Per_ID = p.Per_ID
INNER JOIN Inv_Header i ON v.Per_ID = i.Ven_ID
GROUP BY i.Ven_ID
HAVING VendorAverage > (SELECT AVG(TotalAmt)
                        FROM Inv_Header)
                    ORDER BY i.Ven_ID;
```



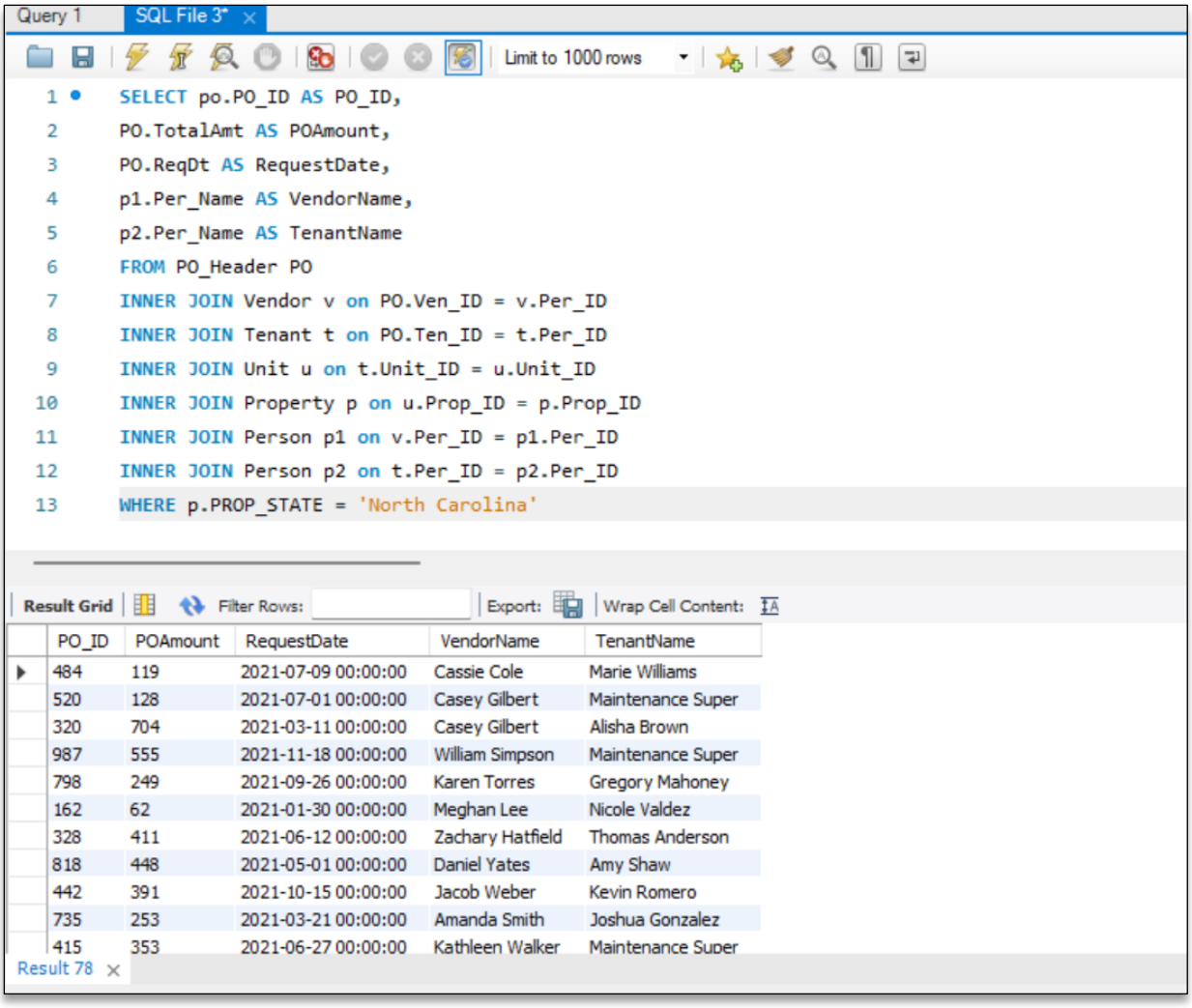
```
/*Calculating the average price across each vendor and comparing with the total average price*/
SELECT v.Per_ID as Vendor_ID,
p.Per_Name as Vendor_Name,
AVG(i.TotalAmt) as VendorAverage
FROM Vendor v
INNER JOIN Person p ON v.Per_ID = p.Per_ID
INNER JOIN Inv_Header i ON v.Per_ID = i.Ven_ID
GROUP BY i.Ven_ID
HAVING VendorAverage > (SELECT AVG(TotalAmt)
                        FROM Inv_Header)
ORDER BY i.Ven_ID;
```

Result Grid

| Vendor_ID | Vendor_Name | VendorAverage |
|---|---|---|
| 4 | Zachary Hatfield | 377.9444 |
| 6 | Katherine James | 366.0000 |
| 9 | Karen Torres | 435.0625 |
| 16 | Marie Lopez | 427.5714 |
| 20 | Erik Ortiz | 474.9000 |
| 21 | Casey Gilbert | 429.6333 |
| 22 | Gabriel Chambers | 394.8667 |
| 23 | Kimberly Miller | 389.6875 |
| 24 | Brad Saunders | 369.7368 |
| 25 | Paul Scott | 426.2258 |
| 26 | Michael Sanders | 378.8636 |
| 27 | Beth Schmitt | 385.0000 |

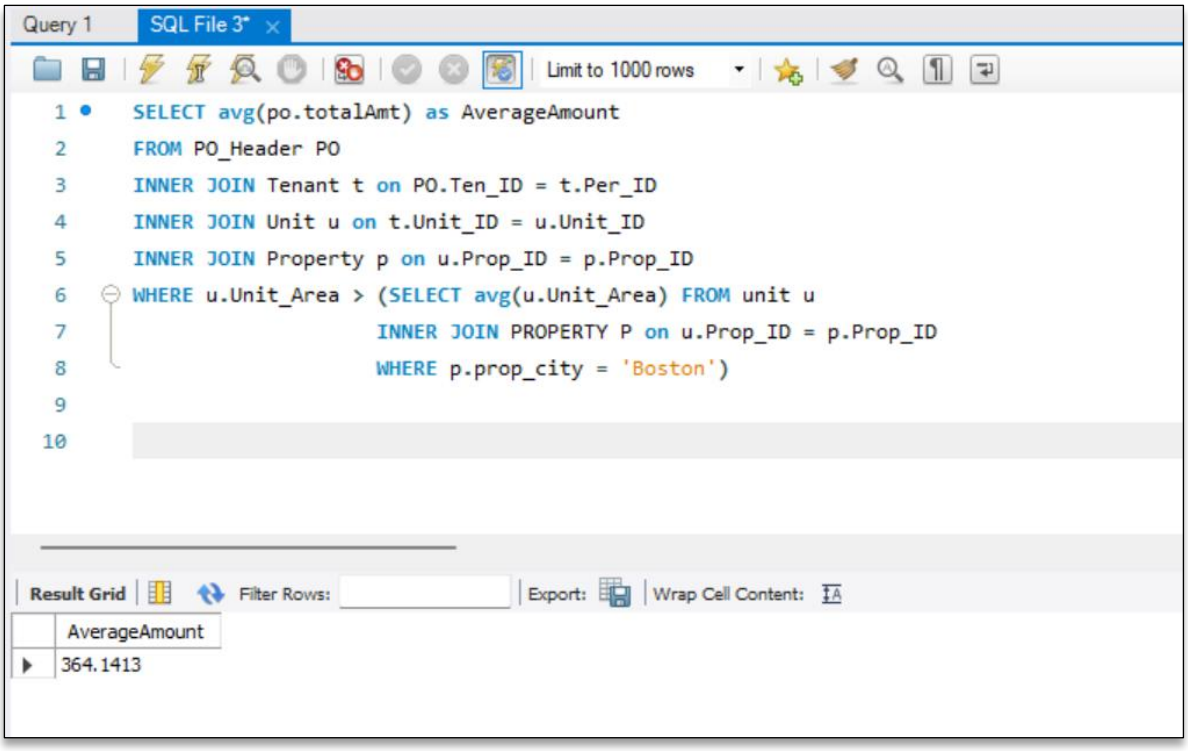
3. **Services Requested across the state Analysis**: Fetch the PO ID, Amount, Request Date, Vendor Name, Tenant Name of Tenants with Pos created and residing in properties of the state North Carolina.
   **Analytical Purpose:** This will help the property managers do a analysis for their state as to how many purchase orders are being requested by the tenants. Additionally, they query can be extended on advanced analytical platforms to compare the purchase orders for their property with the overall average across the state and suggest long term changes on the properties instead of repeatedly fixing small issues.

```
SELECT po.PO_ID AS PO_ID,
PO.TotalAmt AS POAmount,
PO.ReqDt AS RequestDate,
p1.Per_Name AS VendorName,
p2.Per_Name AS TenantName
FROM PO_Header PO
INNER JOIN Vendor v on PO.Ven_ID = v.Per_ID
INNER JOIN Tenant t on PO.Ten_ID = t.Per_ID
INNER JOIN Unit u on t.Unit_ID = u.Unit_ID
INNER JOIN Property p on u.Prop_ID = p.Prop_ID
INNER JOIN Person p1 on v.Per_ID = p1.Per_ID
INNER JOIN Person p2 on t.Per_ID = p2.Per_ID
WHERE p.PROP_STATE = 'North Carolina'
```



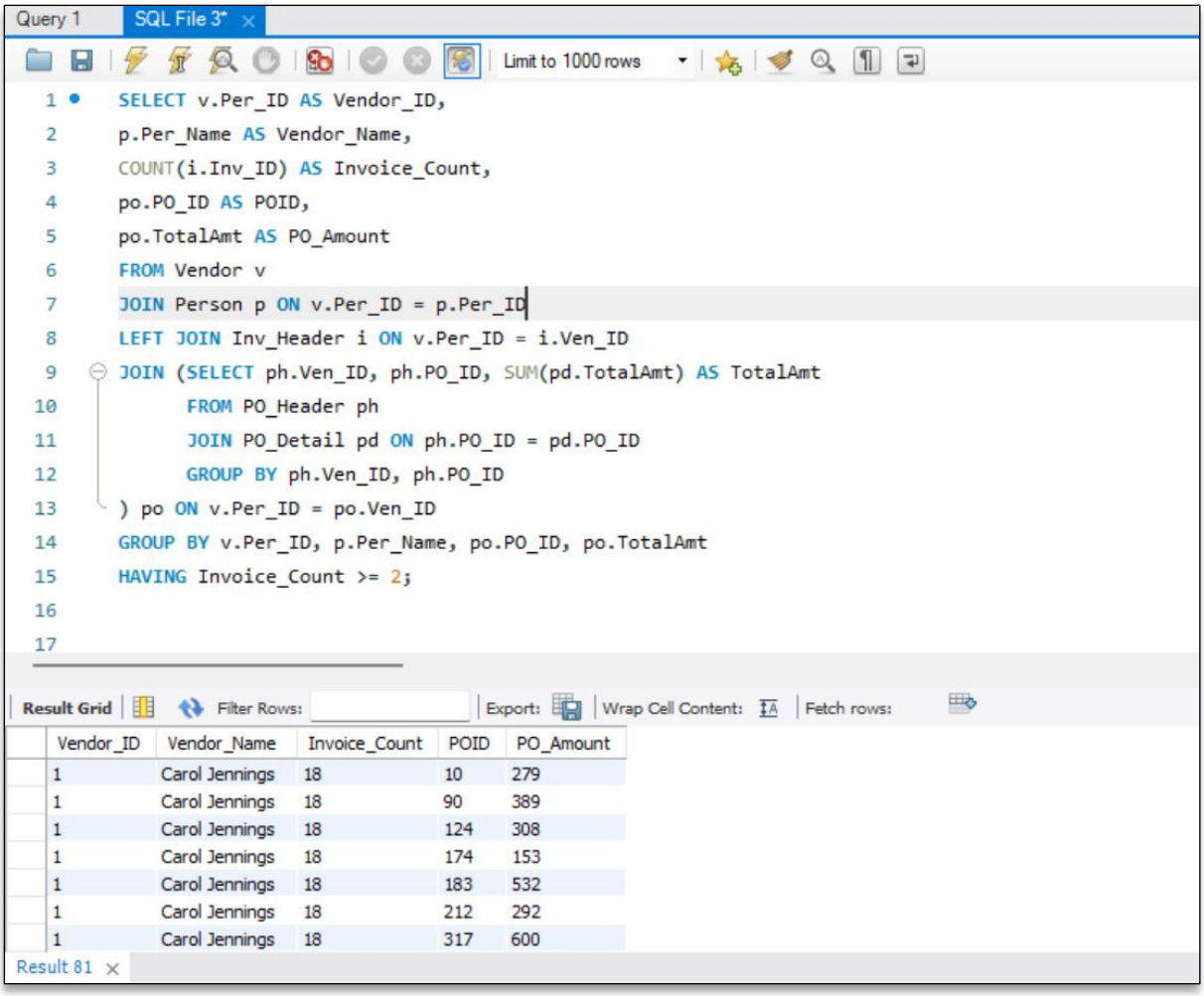

4. **PO and Area Analysis:** Calculate the average PO amount of the POs that belong to the Tenants residing in units with area which is greater than the average area of units of the properties in Boston.
   **Analytical Purpose:** This analysis can help the property managers to fetch the average PO amounts of houses in a particular city which reside in houses greater than the average area of a unit across the city. This can help the property managers analyse if it is the size of the houses which is having an effect on the number and amount of services being requested.

```
SELECT avg(po.totalAmt) as AverageAmount
FROM PO_Header PO
INNER JOIN Tenant t on PO.Ten_ID = t.Per_ID
INNER JOIN Unit u on t.Unit_ID = u.Unit_ID
INNER JOIN Property p on u.Prop_ID = p.Prop_ID
WHERE u.Unit_Area > (SELECT avg(u.Unit_Area) FROM unit u
                                    INNER JOIN PROPERTY P on u.Prop_ID = p.Prop_ID
                WHERE p.prop_city = 'Boston')
```




5. **Purchase Order Price per Vendor:** Fetch vendors with at least two invoices and their corresponding purchase orders.
   **Analytical Purpose:** This can help the property managers keep a track of all the invoices and Pos per vendor. It can been noted how many of the Pos were converted and the all the requested services were actually provided.

```
SELECT v.Per_ID AS Vendor_ID,
p.Per_Name AS Vendor_Name,
COUNT(i.Inv_ID) AS Invoice_Count,
po.PO_ID AS POID,
po.TotalAmt AS PO_Amount
FROM Vendor v
JOIN Person p ON v.Per_ID = p.Per_ID
LEFT JOIN Inv_Header i ON v.Per_ID = i.Ven_ID
JOIN (SELECT ph.Ven_ID, ph.PO_ID, SUM(pd.TotalAmt) AS TotalAmt
        FROM PO_Header ph
    JOIN PO_Detail pd ON ph.PO_ID = pd.PO_ID
    GROUP BY ph.Ven_ID, ph.PO_ID
) po ON v.Per_ID = po.Ven_ID
GROUP BY v.Per_ID, p.Per_Name, po.PO_ID, po.TotalAmt
HAVING Invoice_Count >= 2;
```




6. **Area Analysis:** Fetch the count of units based on areas. Term them as common spaces, less area, medium area, more area.
**Analytical Purpose:** If analytics are drawn and it is concluded that a particular service is being requested repeatedly, for example, flooring for a particular unit or ceiling for particular floor, then the owner can be suggested to undergo renovation.

```
SELECT AreaAnalysis, Count(x.AreaAnalysis) AS Counts
FROM (SELECT u.unit_name as UnitName,
                    p.prop_name as PropertyName,
        CASE WHEN u.unit_area < 1000 THEN 'Common Spaces'
                            WHEN u.unit_area BETWEEN 1000 AND 1250 THEN 'Less Area'
           WHEN u.unit_area BETWEEN 1251 AND 1350 THEN 'Medium Area'
           WHEN u.unit_area > 1351 THEN 'More Area'
                    END as AreaAnalysis
         FROM Unit u
     INNER JOIN Tenant t ON u.Unit_ID = t.Unit_ID
     INNER JOIN PO_Header po ON t.Per_ID = po.Ten_ID
     INNER JOIN PO_Detail pd ON po.PO_ID = pd.PO_ID
     INNER JOIN Property p ON u.Prop_ID = p.Prop_ID
     WHERE pd.Det_Desc = 'Flooring Service') x
GROUP BY AreaAnalysis
ORDER BY Counts;
```

```sql
/*Calculating the count of area and bifurcating them into less, medium, more area for flooring*/
SELECT AreaAnalysis, Count(x.AreaAnalysis) AS Counts
FROM (SELECT u.unit_name as UnitName,
             p.prop_name as PropertyName,
             CASE WHEN u.unit_area < 1000 THEN 'Common Spaces'
                  WHEN u.unit_area BETWEEN 1000 AND 1250 THEN 'Less Area'
                  WHEN u.unit_area BETWEEN 1251 AND 1350 THEN 'Medium Area'
                  WHEN u.unit_area > 1351 THEN 'More Area'
             END as AreaAnalysis
      FROM Unit u
      INNER JOIN Tenant t ON u.Unit_ID = t.Unit_ID
      INNER JOIN PO_Header po ON t.Per_ID = po.Ten_ID
      INNER JOIN PO_Detail pd ON po.PO_ID = pd.PO_ID
      INNER JOIN Property p ON u.Prop_ID = p.Prop_ID
      WHERE pd.Det_Desc = 'Flooring Service') x
GROUP BY AreaAnalysis
ORDER BY Counts;
```

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

| AreaAnalysis | Counts |
|---|---|
| Medium Area | 14 |
| Common Spaces | 85 |
| More Area | 93 |
| Less Area | 159 |

7. **Service and Number of Services Requested Analysis:** Calculate the sum of quantities requested per service.
**Analytical Purpose:** The services that are being requested often can be noted. More vendors with these services can be found to make the process more efficient. Else, the existing vendors can be asked to provide faster services since they have the opportunity to build more business with us.

SELECT Det_Desc as DescReq, Sum(Quantity) as TotalReq
FROM PO_Detail
GROUP BY Det_Desc;

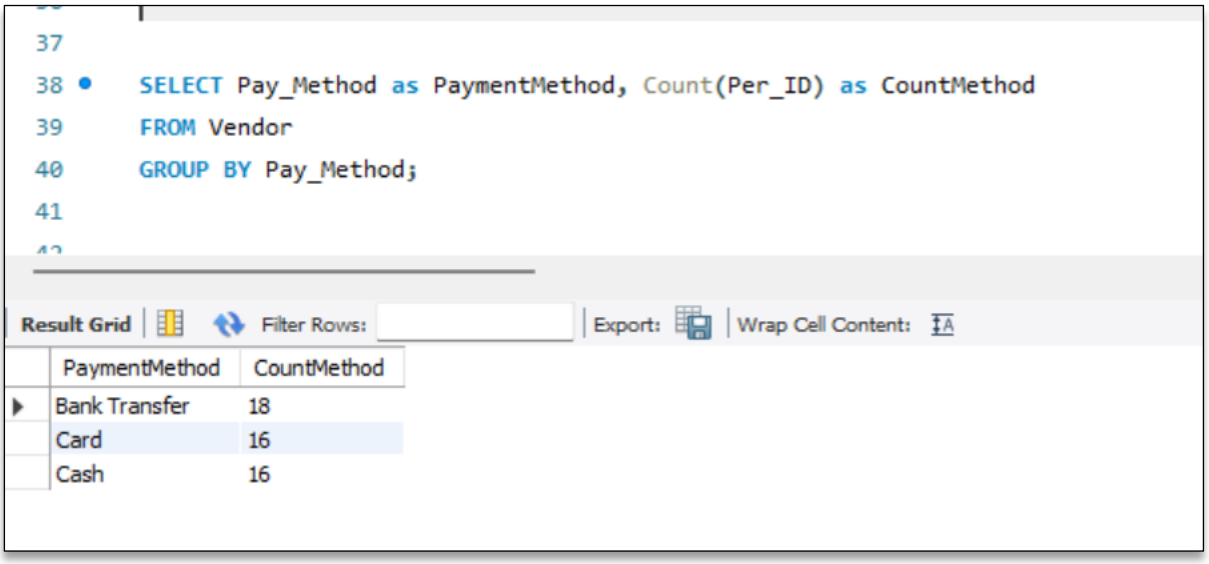
```sql
SELECT Det_Desc as DescReq, Sum(Quantity) as TotalReq
FROM PO_Detail
GROUP BY Det_Desc;
```

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

| DescReq | TotalReq |
|---|---|
| Plumber Service | 1753 |
| Electricity Service | 1906 |
| Window Cleaning Service | 1950 |
| Gas Service | 1782 |
| Flooring Service | 1934 |
| AC Maintenance Service | 1668 |

8. **Payment Method Analysis:** Fetch the count of vendors providing each of the payment methods. **Analytical Purpose:** Payment Method Analysis is done to check what percentage of transactions are being done in which kind of method. This can be utilized to minimize the methods that have higher service charges. In the future, the company can also consider developing its own payment gateway for its own vendors, tenants and can also act as a third-party provider to other companies.

```
SELECT Pay_Method as PaymentMethod, Count(Per_ID) as CountMethod
FROM Vendor
GROUP BY Pay_Method;
```



# PYTHON VISUALIZATIONS

Python to MySQL Connection-

**SQL to Python Connection**

In [1]:
```
pip install mysql-connector-python
```

```
Requirement already satisfied: mysql-connector-python in c:\users\hp\anaconda3\lib\site-packages (8.2.0)
Requirement already satisfied: protobuf<=4.21.12,>=4.21.1 in c:\users\hp\anaconda3\lib\site-packages (from mysql-connector-p
ython) (4.21.12)
Note: you may need to restart the kernel to use updated packages.
```

In [2]:
```
import mysql.connector
from mysql.connector import Error
import warnings
warnings.filterwarnings("ignore")

try:
    connection = mysql.connector.connect(host='localhost',
                                         database='venmgmt',
                                         user='root',
                                         password= )
    if connection.is_connected():
        db_Info = connection.get_server_info()
        print("Connected to MySQL Server version ", db_Info)
        cursor = connection.cursor()
        cursor.execute("select database();")
        record = cursor.fetchone()
        print("You're connected to database: ", record)

except Error as e:
    print("Error while connecting to MySQL", e)
```

```
Connected to MySQL Server version  8.0.33
You're connected to database:  ('venmgmt',)
```

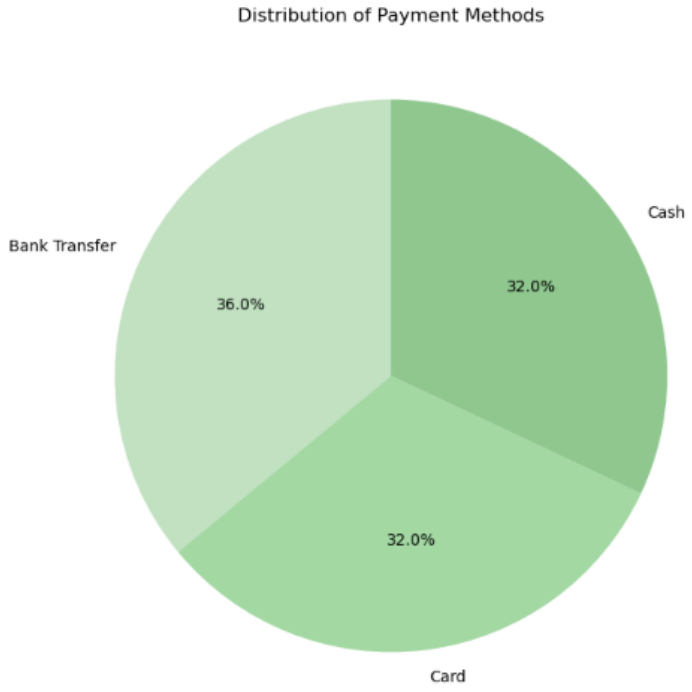
In [3]:
```
import pandas as pd
import matplotlib.pyplot as plt
```

1. Payment Method Analysis (Analytical Purpose mentioned above with the SQL Query)-

**1. Payment Method Analysis**

In [4]:
```
query1 = """ SELECT Pay_Method as PaymentMethod, Count(Per_ID) as CountMethod
FROM Vendor
GROUP BY Pay_Method """

query1_df = pd.read_sql_query(query1, connection)
```

In [5]:
```
import pandas as pd
import matplotlib.pyplot as plt

colors = ['#C1E1C1', '#A3D8A3', '#8FC78F']

plt.figure(figsize=(8, 8))
plt.pie(query1_df['CountMethod'], labels=query1_df['PaymentMethod'], autopct='%1.1f%%', startangle=90, colors=colors)
plt.title('Distribution of Payment Methods')
plt.show()
```

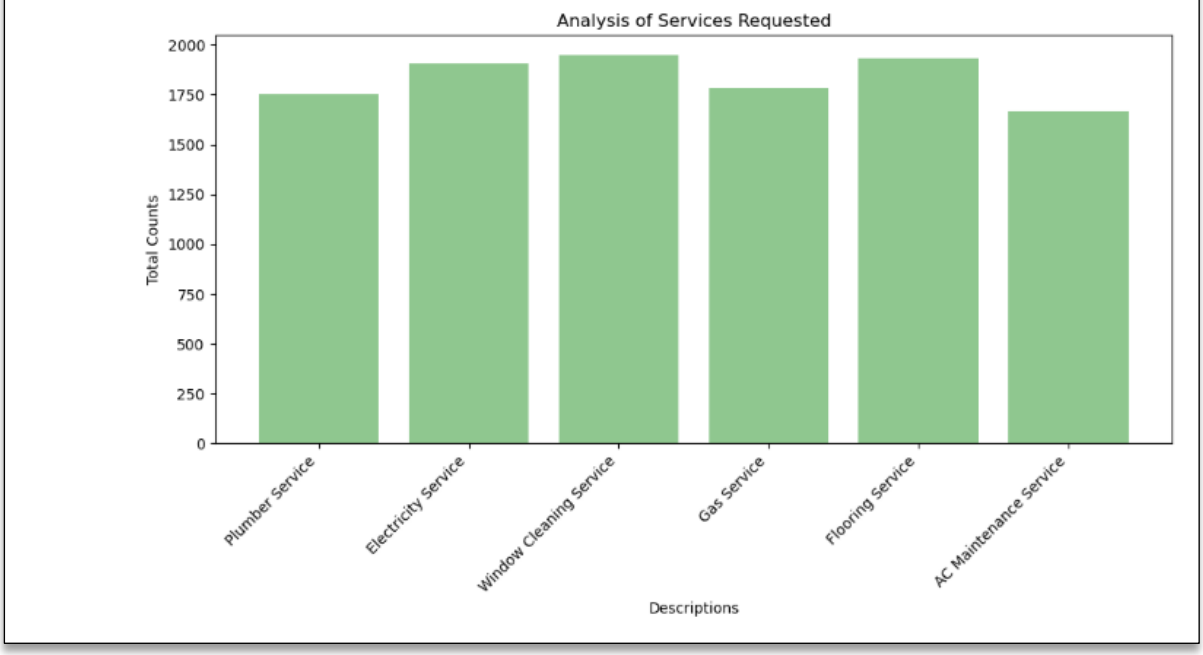


2. Analysis of Quantity of Services Requested (Analytical Purpose mentioned above with the SQL Query)-

**2. Analysis of Quantity of Services Requested**

In [6]:
```
query2 = """SELECT Det_Desc as DescReq, Sum(Quantity) as TotalReq
FROM PO_Detail
GROUP BY Det_Desc
"""

query2_df = pd.read_sql_query(query2, connection)
```

In [7]:
```
import pandas as pd
import matplotlib.pyplot as plt

plt.figure(figsize=(10, 6))
plt.bar(query2_df['DescReq'], query2_df['TotalReq'], color='#8FC78F')

plt.xlabel('Descriptions')
plt.ylabel('Total Counts')
plt.title('Analysis of Services Requested')

plt.xticks(rotation=45, ha='right')
plt.tight_layout()
plt.show()
```

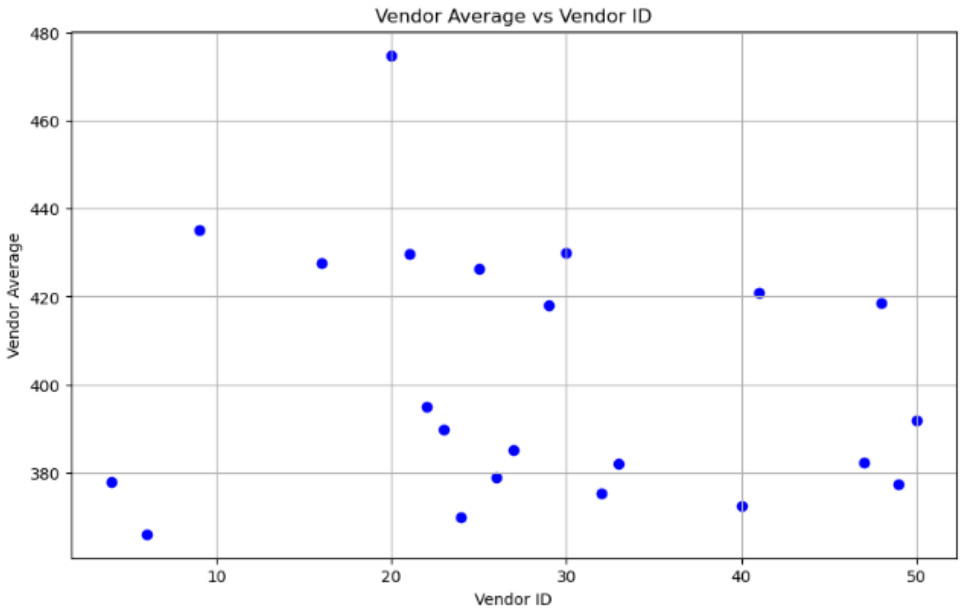


3. Comparison of Average price per Vendor with overall average price (Analytical Purpose mentioned above with the SQL Query)-

**3. Comparison of Average price per Vendor with overall average price**

In [8]:
```
query3 = """ SELECT v.Per_ID as Vendor_ID,
p.Per_Name as Vendor_Name,
AVG(i.TotalAmt) as VendorAverage
FROM Vendor v
INNER JOIN Person p ON v.Per_ID = p.Per_ID
INNER JOIN Inv_Header i ON v.Per_ID = i.Ven_ID
GROUP BY i.Ven_ID
HAVING VendorAverage > (SELECT AVG(TotalAmt) FROM Inv_Header)
ORDER BY i.Ven_ID;
"""

query3_df = pd.read_sql_query(query3, connection)
```

In [9]:
```
import matplotlib.pyplot as plt

plt.figure(figsize=(10, 6))
plt.scatter(query3_df['Vendor_ID'], query3_df['VendorAverage'], color='blue')
plt.title('Vendor Average vs Vendor ID')
plt.xlabel('Vendor ID')
plt.ylabel('Vendor Average')
plt.grid(True)
plt.show()
```

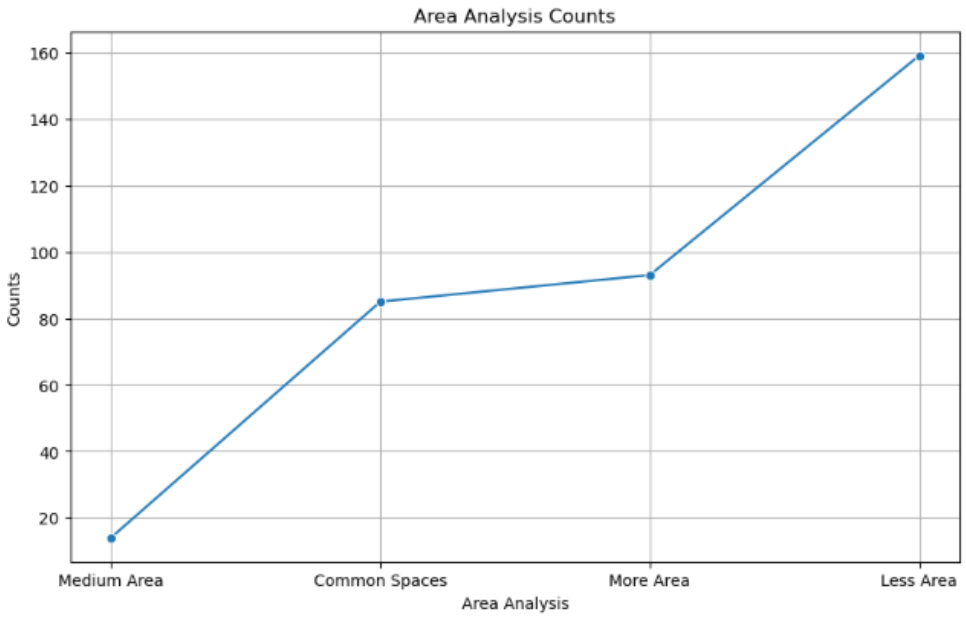


4. Area Analysis (Analytical Purpose mentioned above with the SQL Query)-

**4. Area Analysis**

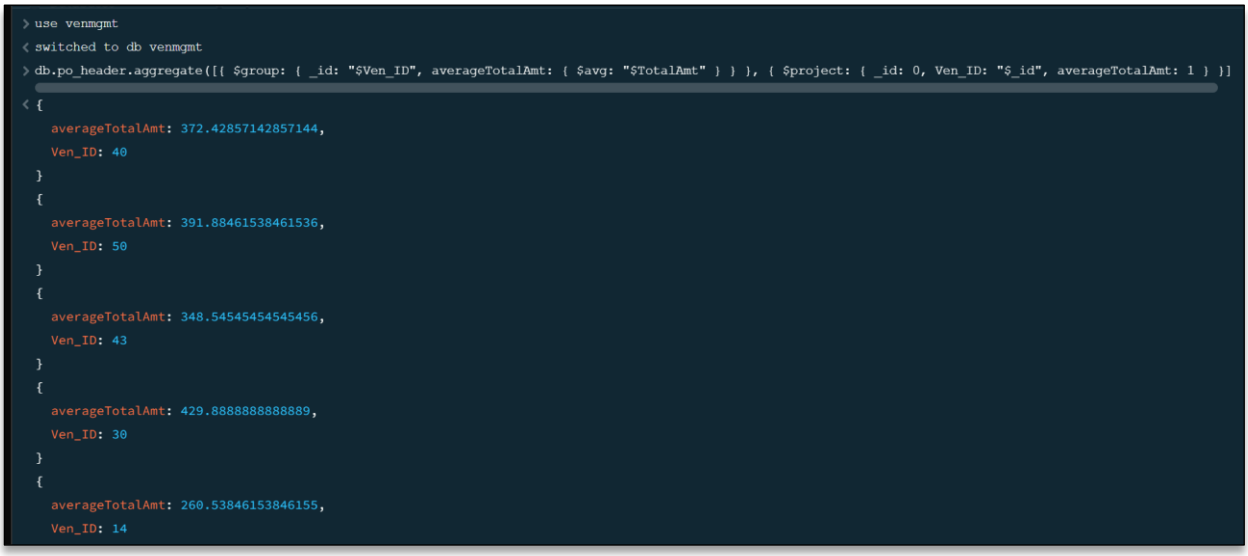
In [10]:
```
query4 = """ SELECT AreaAnalysis, Count(x.AreaAnalysis) AS Counts
FROM (SELECT u.unit_name as UnitName,
             p.prop_name as PropertyName,
             CASE WHEN u.unit_area < 1000 THEN 'Common Spaces'
             WHEN u.unit_area BETWEEN 1000 AND 1250 THEN 'Less Area'
                  WHEN u.unit_area BETWEEN 1251 AND 1350 THEN 'Medium Area'
                  WHEN u.unit_area > 1351 THEN 'More Area'
            END as AreaAnalysis
      FROM Unit u
      INNER JOIN Tenant t ON u.Unit_ID = t.Unit_ID
      INNER JOIN PO_Header po ON t.Per_ID = po.Ten_ID
      INNER JOIN PO_Detail pd ON po.PO_ID = pd.PO_ID
      INNER JOIN Property p ON u.Prop_ID = p.Prop_ID
      WHERE pd.Det_Desc = 'Flooring Service') x
GROUP BY AreaAnalysis
ORDER BY Counts;
"""

query4_df = pd.read_sql_query(query4, connection)
```

In [11]:
```
import matplotlib.pyplot as plt
import seaborn as sns

# Line plot using seaborn
plt.figure(figsize=(10, 6))
sns.lineplot(x='AreaAnalysis', y='Counts', data=query4_df, marker='o')
plt.title('Area Analysis Counts')
plt.xlabel('Area Analysis')
plt.ylabel('Counts')
plt.grid(True)
plt.show()
```



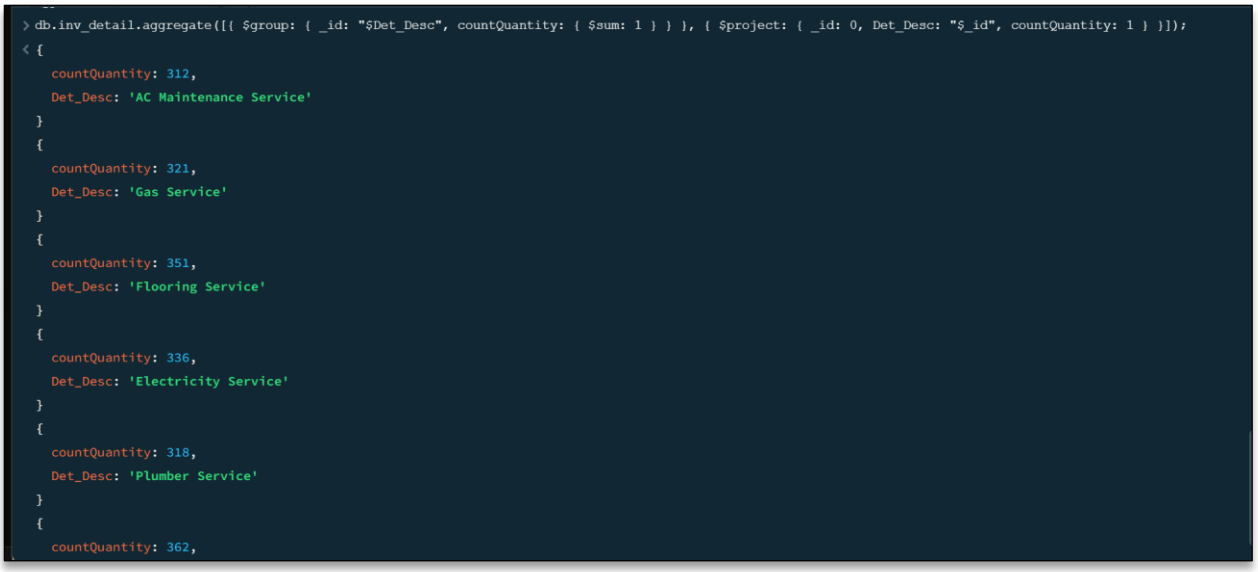
1. **Vendor-PO Analysis:** Calculate the average price of the purchase orders per vendor.
   **Analytical Purpose:** This will help the property managers analyse how many services are being requested per vendor. Further analysis can be carried out to map these services based on the categories of the vendors. Eventually, Pos being requested for a specific purpose to a specific vendor can be considered. Vendors can be reinforced with positive or negative feedback based on this analysis. This will boost their motivation to work well and provide discounts.

```
db.po_header.aggregate([
  {
    $group: {
      _id: "$Ven_ID",
      averageTotalAmt: { $avg: "$TotalAmt" }
    }
  },
  {
    $project: {
      _id: 0,
      Ven_ID: "$_id",
      averageTotalAmt: 1
    }
  }
]);
```



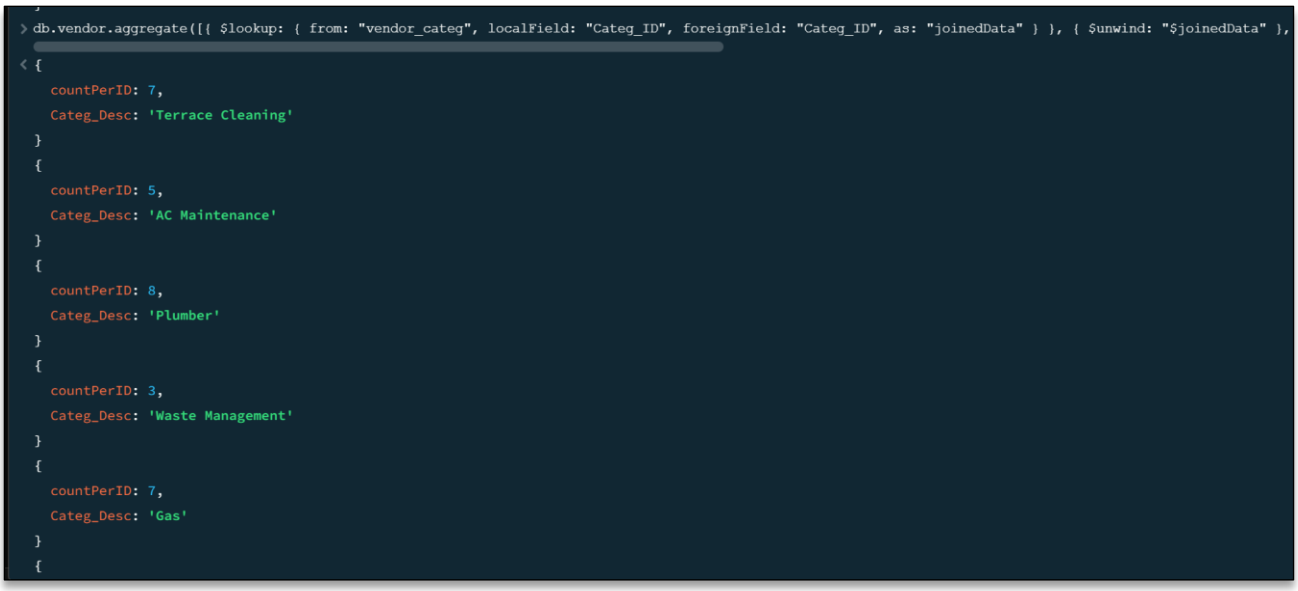
2. **Quantity-Service Analysis:** Calculate the quantity of each service billed.
   **Analytical Purpose:** Quantity of each of the services billed can be visualized. This can be used to further boost the working of the managers on seeking efficient vendors providing services that are requested more often. Furthermore, analysis can be extended to do a detailed analysis of the sum of these quantities over each region or property.

```
db.inv_detail.aggregate([
  {
    $group: {
      _id: "$Det_Desc",
      countQuantity: { $sum: 1 }
    }
  },
  {
    $project: {
      _id: 0,
      Det_Desc: "$_id",
      countQuantity: 1
    }
  }
]);
```



3. **Vendor-Categroy Analysis:** Calculate the count of vendor in each category.
   **Analytical Purpose**: Vendor per category have been calculated. This can be further collaborated with analysis of services requested per category to draw the need to come up with a new vendors.

```
db.vendor.aggregate([
  {
    $lookup: {
      from: "vendor_categ",
      localField: "Categ_ID",
      foreignField: "Categ_ID",
      as: "joinedData"
    }
  },
  {
    $unwind: "$joinedData"
  },
  {
    $group: {
      _id: "$joinedData.Categ_Desc",
      countPerID: { $sum: 1 }
    }
  },
  {
    $project: {
      _id: 0,
      Categ_Desc: "$_id",
      countPerID: 1
    }
  }
]);
```



# SUMMARY AND CONCLUSION

Throughout the semester, we worked on creating an efficient database. We started with observing and coming up with a problem statement and goal. We worked on the Enhanced Entity Relationship diagram after decided the entities that should be a part of the database. We mapped cardinalities between all the entities. There was inclusion of concepts of composite attribute types (for example, address), multivalued attribute types (for example, email ID), derived attribute types (for example, lease term). We included triggers and stored procedures to calculate the derived attribute type from 2 other attribute types. We included the concept of specialization to map each “person” as either a tenant or a vendor. The concept of weak entity type was used in mapping each of the PO Detail Lines to its header. Likewise, the same concept was used in Invoice Detail Lines and its header.

We started with DDL Queries to create the database i.e. tables, triggers, stored procedures. The data in our database was created by us from the scratch and imported. Over 4000 transactions were created to do real time analysis. We then moved forward to DML Queries. Performed analysis using complex SQL queries. We used aggregate functions, subqueries, case statements, and some other concepts to develop queries.

These queries were then visualized on Python. Upon learning the connectivity of python with SQL, we started creating graphs. We used our data to create pie chart, bar graph, scatter plot. Line graph.

Lastly, we implemented some of our queries in NoSQL using MongoDB. We ran these analytical queries on MongoDB Compass.

In conclusion, the database created by us included most of the SQL Concepts that we learnt throughout the semester. We generated several analytical outputs that could be used by a property management company to particularly manage their Accounts Payable side i.e. Vendor Management.