Modern Data Management

&

Business Intelligence

Assignment 1

Students

Eftychia Gkini (p2822108)

Maria Skoli (p2822131)

Class

Part Time

Table of Contents

Introduction

For the purpose of this Assignment, we are going to delve into to a real estate dataset based on the site E-Properties.

The line we will follow to perform our analysis is the following. First of all, we are going to design the Entity- Relationship Diagram which will be a great indicator of the existing relationship between the database. After that, we are going to Create our Tables in SQL and we will insert our data in each table.

Afterwards, we are going to present how to extract the information we want from our database using the appropriate queries based on specific cases.

Finally, we will present how to connect our database in SQL Server Management Studio with the programming language R in order to perform certain processes.

Entity-Relationship Model

The first step of our analysis is to design the Entity-Relationship Diagram of our data. This will be the guide we will follow to perform our analysis. The E-R Diagram consists of entities, relationships, attributes and constraints. Due to the nature of our assignment, we observed that our data can be divided in 5 main entities: Agent, Estimation, Property and Region. Each entity has several attributes as we can see in the E-R Diagram below.

More specifically, the attributes of the entity Agent are: Name, Last Name, Gender, Age, Address and the ID of the Agent. The attributes for the entity Estimation are the ID, the Date, and the Price of the real estate estimation. The attributes for the entity Property are the ID, Address, Floor, Size, Year of Construction and the Type of the apartment. The attribute Type has also the AFM and the Identity Number as attributes. Finally, Region entity has as attributes: the ID of the Region, the Name of the Region, the population and the average income.

To create related tables, we first define a relationship between two tables. The relationship that connects our entities is the following:

An Agent calculates one or more Estimations. An estimation is based on a specific property, but a property can have multiples estimations. Finally, one or more properties are located in a specific Region.

Therefore, the connections between the tables are “one to many” (1→*) . Because relationships work both ways, there are also “many-to-one” relationships.

The E-R Diagram is shown below. The Drawio (app.diagrams.net) is used for the design of the Diagram.

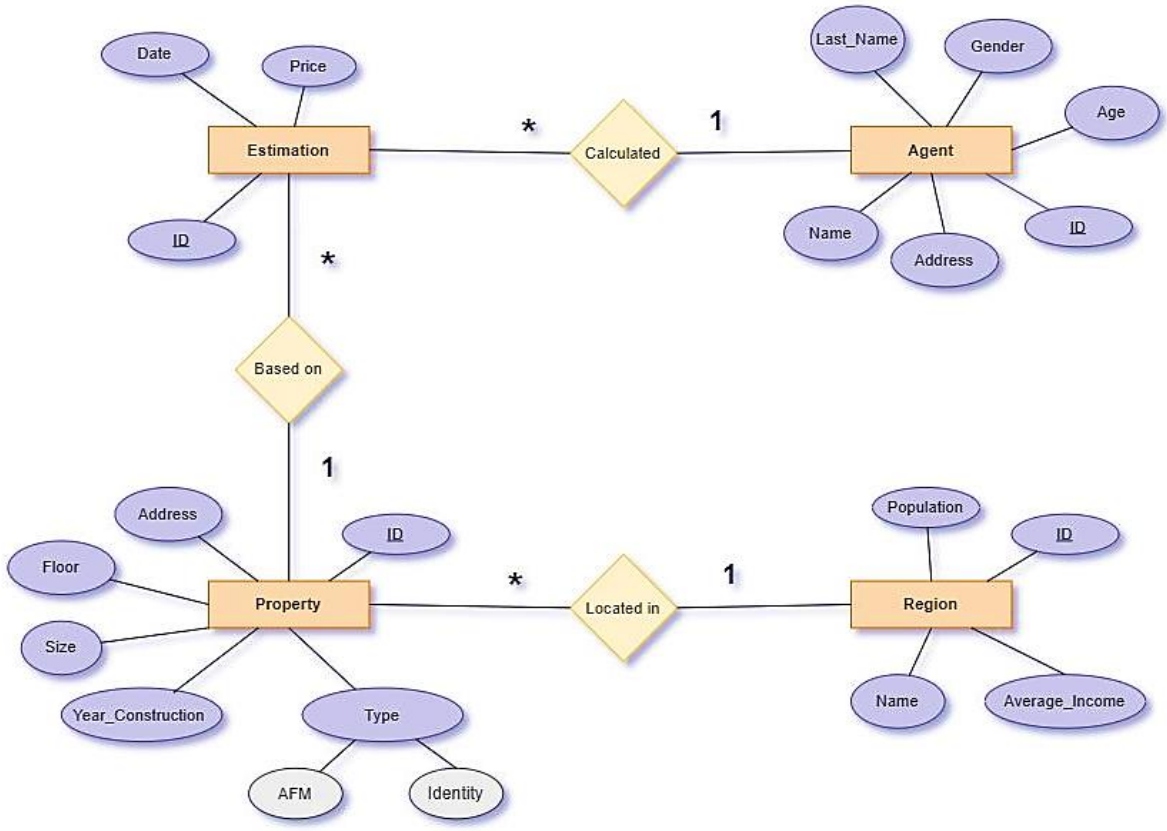

FIGURE 1: ENTITY- RELATIONSHIP MODEL

Create Tables

The first step is to create a new database named *"E-Property"* in Microsoft SQL Server Management Studio. Then, we are going to create a table for each entity. These tables have as columns the attributes of the corresponding entity.

- The following SQL query creates a new table called **Agent**.

```
CREATE TABLE Agent
(
ID int NOT NULL primary key,
Name varchar(25),
Last_Name varchar(25),
Gender varchar(15),
Age int,
Address varchar(255)
)
```

The columns ID and gender are of type int and will hold an integer. Also, ID has been declared as not null since it is a primary key and it must be not null. The columns: Name, Last_Name, Gender and address are of type varchar and will hold characters.

- The following SQL query creates a new table called **Region**

```
CREATE TABLE Region
(
ID int NOT NULL primary key,
Name varchar(25),
Population int,
Average_Income decimal(18,2)
)
```

We declare the column Name as varchar, the column Population as integer and the average income as decimal. The ID is the primary key of this table.

- The following SQL query creates a new table called **Property**

```
CREATE TABLE Property
(
ID int NOT NULL primary key,
Address varchar(255),
Floor_Property int,
Size int,
Year_of_Construction int,
Id_Region int NOT NULL REFERENCES Region(ID) ON DELETE CASCADE,
Type varchar(255),
AFM int,
Identity_Number varchar(25)
)
```

We declare the columns: ID, Floor property, Size, AFM and Year of Construction as integers and the columns Address, Type, Identity_Number as varchar because they will hold characters. Also, we add a new variable: Id_Region which is going to be the foreign key that connects the table Property with the table Region. We used the function "References … on

delete cascade" in order to delete the referencing rows in the Property table when the referenced row is deleted in the Region table which has a primary key.

- The following SQL query creates a new table called **Estimation.**

```
CREATE TABLE Estimation
(
ID int NOT NULL primary key,
Date date,
Price  int,
Id_Agent int NOT NULL REFERENCES Agent(ID) ON DELETE CASCADE,
Id_Property int NOT NULL REFERENCES Property(ID) ON DELETE CASCADE
)
```

We declare the columns ID and Price as integers and the column Date as date type. Also, we insert two more attributes the Id_Agent and the Id_Property which are going to be the foreign keys that connect the table Estimation with the tables Agent and Property respectively.

QUERY 1[1]

Create-Tables.sql

[1] By clicking the below image, you will automatically open the query in the SQL Server Management Studio

Relational Schema

In order to have a better perspective of the connection of the tables we created in Microsoft SQL Server Management Studio the relational diagram below. In this diagram we can understand better the one-to-many connection. For instance, the primary key in the Property table, ID is designed to contain unique values. The foreign key in the Estimation table, Property_ID is designed to allow multiple instances of the same value.

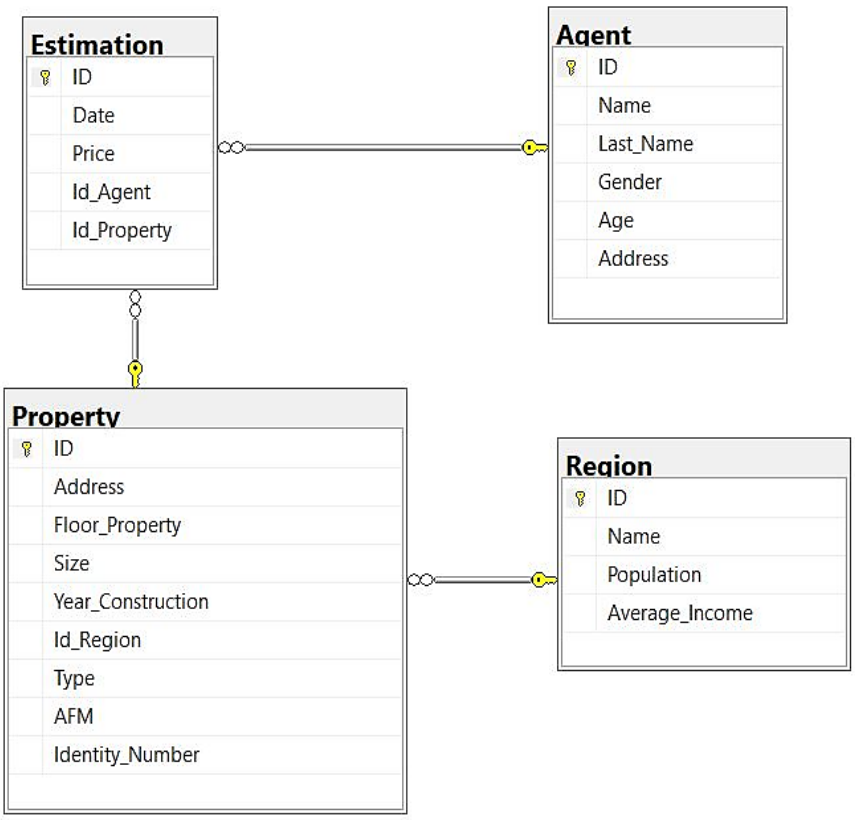

FIGURE 2: RELATIONAL SCHEMA

Insert Values in Tables

The next step is to insert our data to the previous created tables. We use the statement *"INSERT INTO"* to insert a new record in the corresponding table.

It is important to note that we should follow the order we used when we created the tables because of the existing relations between them. For example, before we insert values into the Estimation table, we should first insert values into the Property table because the Estimation table contains the Property ID as a foreign key.

Following this reasoning path, we firstly inserted values into the tables Agent and Region, then into the Property table and last in the Estimation table.

Our database contains 20 Agents, 35 Properties in 20 Regions and the number of Estimations is 50.

An example of this process for the table Region is shown below.

```
INSERT INTO Region
(ID, [Name], [Population], [Average_Income])
VALUES
(1, 'Agios Dimitrios', 71294, 37987),
(2, 'Nea Smurnh', 63076, 35003),
(3, 'Agia Paraskeui', 57193, 24205),
(4, 'Xalandri', 74192, 25013),
(5, 'Glifada', 82305, 24508),
(6, 'Ilion', 71793, 27803),
(7, 'Voula', 44179, 42317),
(8, 'Lagonisi', 24129 , 22330),
(9, 'Kalivia', 15793, 43157),
(10, 'Ilioupoli', 34187, 25933),
(11, 'Gkizi', 54798, 39917),
(12, 'Dionisos', 14781, 14787),
(13, 'Perissos', 42163, 26516),
(14, 'Ampelokipoi', 52489, 32512),
(15, 'Ekali', 32448, 45129),
(16, 'Elliniko', 42161, 55116),
(17, 'Argiroupoli', 51587, 23384),
(18, 'Kifisia', 32369, 57987),
(19, 'Koukaki', 25168, 45123),
(20, 'Athina', 53492, 37331)
GO
```

The selection of the first 5 rows from Region Table will now look like this:

	ID	Name	Population	Average_Income
1	1	Agios Dimitrios	71294	37987.00
2	2	Nea Smurnh	63076	35003.00
3	3	Agia Paraskeui	57193	24205.00
4	4	Xalandri	74192	25013.00
5	5	Glifada	82305	24508.00


Agent Table.sql


Region Table.sql


Property Table.sql


Estimation Table.sql

Retrieve Data from E-Properties based on certain cases.

Case 1: *Properties from Regions with average income > 40.000 and estimation period between 24.12.20 and 31.12.20.*

For this problem we used the following query:

```
-- Δείξε τον κωδικό και τη διεύθυνση των ακινήτων που ανήκουν σε περιοχή με μέσο εισόδημα μεγαλύτερο
-- των 40.000€ και έχουν εκτιμηθεί μεταξύ 24/12/2020 και 31/12/2020.

select P.ID,Address from Property as P
inner join Region as R on R.ID=P.Id_Region
inner join Estimation as E on  E.Id_Property=P.ID
where R.Average_Income>=40000
        and Date>='2020-12-24' and Date<='2020-12-31'
```

FIGURE 3: QUESTION 1

Query 1.sql

The INNER JOIN keyword selects all rows from both tables when there is a match between the columns. For instance, as far for the first inner join if we have rows in the table “Region” that do not have matches in the "Property" table, these rows will not be shown.

The WHERE keyword filters the rows and displays only those which satisfy the given conditions.

The combination of *Inner joins* and *Where* clause gives us the requested output which is shown below:

	ID	Address
1	29	Platwnos 276
2	32	Axarnwn 26

FIGURE 4: OUTPUT QUESTION 1

As we can see the properties that fulfill those conditions are the property with ID: 29 and Address Platwnos 276 and the property with ID: 32 and address Axarnwn 26.

Case 2: *Number of Estimations for the year 2020 per Agent*

For this case we used the following query.

```
1   -- Για κάθε εκτιμητή δείξε το πλήθος των εκτιμήσεων που έχει πραγματοποιήσει το 2020.
2
3   select A.ID,A.name,A.Last_Name ,count(E.ID) as NumberOfEstimation from Agent as A
4   inner join Estimation as E on E.Id_Agent=A.ID
5   where YEAR(E.Date)=2020
6   group by A.ID,A.name, A.Last_Name
```

FIGURE 5: QUESTION 2

Query 2.sql

The aggregate function COUNT() returns the number of rows that matches specified criteria of the where clause.

GROUP BY statement groups the estimations for each Agent and displays them as a summary number in a single row.

This query would produce the following result:

	ID	name	Last_Name	NumberOfEstimation
1	1	Maria	Skoli	1
2	4	John	Papadopoulos	1
3	5	Nick	Oikonomopoulos	1
4	6	Jason	Williams	2
5	7	Natalia	Abatzidou	1
6	8	Barbara	Panagopoulou	2
7	9	Jack	Jones	1
8	10	Constantine	Papakostas	3
9	11	Michael	Koutsoumaris	1
10	12	Peter	Smith	1
11	15	Alice	Plati	2
12	16	George	Gaveras	1
13	17	Alex	Skondras	2
14	18	George	Vasileiou	2
15	19	Joanne	Andreopoulou	3
16	20	Constantine	Alamanis	3

FIGURE 6: OUTPUT QUESTION 2

Case 3: *Number of Properties that have been estimated more than two times during 2020*

For this case we used the following query.

```
1  --Δείξε τον κωδικό των ακινήτων που έχουν εκτιμηθεί περισσότερες από δύο φορές μέσα στο 2020.
2
3  Select E.Id_Property from Estimation as E
4  where  Year(E.Date)=2020
5  group by E.Id_Property
6  having count(E.Id_Property) >= 2
```

FIGURE 7: QUESTION 3

Query 3.sql

We used the HAVING () statement because we wanted to add an aggregated function in it. This statement can be used instead of the WHERE after a GROUP BY statement.

In this code first we used the WHERE statement to select the estimations of the properties that have been executed in 2020. Then, the HAVING statement filters the previous result and displays only the ID of the properties that have been estimated 2 or more times.

This query would produce the following output:

	Id_Property
1	24
2	27
3	32

FIGURE 8: OUTPUT QUESTION 3

The properties with ID: 24, 27 and 32 have been estimated two or more times during 2020.

Case 4: *Number of estimations in regions where the average income > 25.000€*

For this case we used the following query.

```
--Χρησιμοποιώντας εμφωλευμένα ερωτήματα, δείξτε τον κωδικό των εκτιμήσεων που έχουν πραγματοποιηθεί
--σε περιοχές με μέσο εισόδημα μεγαλύτερο των 25.000€

SELECT E.ID
FROM Estimation as E
INNER JOIN Property as P
ON P.ID = E.Id_Property
WHERE P.Id_Region IN (SELECT R.ID FROM Region as R WHERE (R.Average_Income > 25000) )
```

FIGURE 9: QUESTION 4

Query 4.sql

In this query, WHERE IN clause is used to filter records and returns list of all estimations where their average income is more than 25000.

At first, we create a selection from the table Region of all Regions' IDs where the average income of those is greater than 25000. After that we select the properties that their ID_Region is into the previous selection. At the final selection we display the Estimations' IDs which are referenced to properties' IDs into the second selection.

This query would produce the following output:

	Estimation_ID
1	1
2	3
3	4
4	8
5	10
6	11
7	12
8	13
9	14
10	17
11	18
12	20
13	21
14	22
15	23
16	24
17	26
18	27

19	28
20	29
21	30
22	31
23	32
24	33
25	34
26	36
27	38
28	39
29	40
30	41
31	42
32	43
33	45
34	46
35	47
36	48
37	49

FIGURE 10: OUTPUT QUESTION 4

Case 5: *Number of properties estimations in regions with population > 50.000 in 2020*

For this case we used the following query.

```
1  -- Δείξε το πλήθος των εκτιμήσεων του 2020 για ακίνητα που ανήκουν σε περιοχές με πληθυσμό > 50.000.
2
3  Select count(E.id) Number_Of_Estimation from Estimation as E
4  inner join Property as P on P.ID=E.Id_Property
5  inner join Region as R on R.ID= P.Id_Region
6  where Year(E.Date)=2020 and R.Population>50000
```

FIGURE 11: QUESTION 5

Query 5.sql

In this query, WHERE clause is used to filter records and extract only rows where the year is equal to 2020 and population greater than 50000. The aggregated function COUNT is used to return the number of these rows.

This query would produce the following output:

	Number_Of_Estimation
1	13

FIGURE 12: OUTPUT QUESTION 5

13 estimations have been calculated in 2020 for properties that are located into regions with population greater than 50.000.

Case 6: *Average estimation per square meter for each Region.*

For this case we used the following query.

```
-- Για κάθε κωδικό περιοχής, δείξε τον κωδικό της περιοχής και τη μέση τιμή εκτίμησης ανά τμ της περιοχής,
--σε αύξουσα σειρά της μέσης τιμής εκτίμησης.

select R.ID ,R.Name,cast(avg(E.Price/P.Size) as decimal(18,2)) as Average_Price_per_square_meter
from Estimation as E
inner join Property as P on E.Id_Property=P.ID
inner join Region as R on R.ID=P.Id_Region
group by R.ID,R.Name
order by Average_Price_per_square_meter
```

FIGURE 13: QUESTION 6

Query 6.sql

The main purpose of this query is to estimate the average of the division of price with the meters squared (m²) of each property. We used the cast function to convert the outcome. Specifically, the outcome of final division:

$$(cast(avg(E.Price/P.Size)\ as\ decimal(18,2)))$$

is decimal with two digits after the decimal place.

The statement ORDER BY is used to display in ascending order the average estimations we calculated before. This query would produce the following output:

	ID	Name	Average_Price_per_square_meter
1	16	Elliniko	894.00
2	6	Ilion	1723.00
3	13	Perissos	2170.00
4	8	Lagonisi	2211.00
5	4	Xalandri	2350.00
6	3	Agia Paraskeui	2359.00
7	17	Argiroupoli	2443.00
8	2	Nea Smurnh	2843.00
9	11	Gkizi	2897.00
10	7	Voula	3204.00
11	14	Ampelokipoi	3591.00
12	9	Kalivia	4201.00
13	19	Koukaki	4297.00
14	5	Glifada	5205.00
15	12	Dionisos	6302.00
16	15	Ekali	7050.00
17	10	Ilioupoli	7735.00
18	18	Kifisia	7952.00
19	20	Athina	8440.00
20	1	Agios Dimitrios	10955.00

FIGURE 14: OUTPUT QUESTION 6

For example, the average estimation price for a property in the Region with ID 4 is 2350€ per 1m²

Case 7: *Number of Apartments and Number of Offices each Agent has estimated.*

For this case we used the following query.

```sql
-- Για κάθε εκτιμητή και για το 2020, δείξε τον κωδικό του εκτιμητή, το πλήθος των εκτιμήσεων κατοικιών
-- που έχει πραγματοποιήσει, και το πλήθος των εκτιμήσεων γραφείων που έχει πραγματοποιήσει (3 στήλες)

SELECT A.ID,
    sum(CASE
        when p.Type='Apartments'
            then 1
        else
            0
        end)  as 'Apartments' ,

    sum(CASE
        when p.Type='Office'
            then 1
        else 0
        end )  as 'Office'
from Agent as A
inner join Estimation as E on E.Id_Agent=A.ID
inner join Property as P on e.Id_Property=p.ID
where year(E.Date)=2020
group by A.ID
```

FIGURE 15: QUESTION 7

Query 7.sql

In this query we calculate the number of offices and apartments per assessor. We use two different functions to achieve this (SUM function and CASE function).

The SUM function returns the total sum of Apartments and Offices that have been estimated by a specific agent. By this way we can have only one line per assessor.

The CASE function is used to create two different columns. The first column counts the number of apartments and second the number of offices. When an Agent has estimated an Apartment, we add 1 apartment to the sum function that counts the Apartments. In the same way when an Agent estimates an Office, we add 1 office to the sum function that counts the Offices.

The query above would produce the following output:

	ID	Apartments	Office
1	1	0	1
2	4	1	0
3	5	1	0
4	6	1	1
5	7	1	0
6	8	2	0
7	9	1	0
8	10	2	1
9	11	1	0
10	12	0	1
11	15	1	1
12	16	1	0
13	17	2	0
14	18	2	0
15	19	1	2
16	20	3	0

FIGURE 16: OUTPUT QUESTION 7

For example, the agent with ID 8 has estimated 2 apartments and 0 offices.

Case 8: *The change in the average Price of estimation per m^2 from the year 2019 to the year 2020*

For this case we used the following query.

```sql
-- Για κάθε κωδικό περιοχής, δείξε τη μεταβολή της μέσης τιμής εκτίμησης ανά τμ μεταξύ 2020 και 2019

WITH region2019 as
(
select R.Name ,avg(E.Price/P.Size) as avg2019,YEAR(E.Date) as year19 from Region as R
inner join Property as P on r.ID=P.Id_Region
inner join Estimation as E on E.Id_Property=p.ID
group by r.ID,R.Name, YEAR(E.Date)
having YEAR(E.Date)=2019
),

region2020 as
(
select R.Name ,avg(E.Price/P.Size) as avg2020,YEAR(E.Date) as year20 from Region as R
inner join Property as P on r.ID=P.Id_Region
inner join Estimation as E on E.Id_Property=p.ID
group by r.ID,R.Name, YEAR(E.Date)
having YEAR(E.Date)=2020
)

------------------------------------------- Main Query -------------------------------------------

select
    case
    when r19.Name is null and r20.name is not null
            then    r20.Name
    when r20.Name is null and r19.name is not null
            then r19.Name
    when r20.Name is not null and r19.name is not null
            then r20.Name
    end as Region
    ,
    CASE WHEN r19.avg2019 is not null and r20.avg2020 is not null
        then
        cast(r20.avg2020-r19.avg2019 as varchar)
        WHEN r19.avg2019 IS NULL
        then 'No information 2019'
        When r20.avg2020 IS NULL
        then 'No information 2020'
    end as 'Mesh timh Ektimhshs 2019-2020'
    from region2020 as r20

full outer join region2019 as r19 on r20.Name=r19.name
order by Region
```

FIGURE 17: QUESTION 8

Query 8.sql

In this query we have created WITH clauses. SQL WITH clause allow us to create a temporary query block, which can be referenced in several places within our main SQL query. The first WITH query is equal to second query and the only different of them is the year of reference.

It is also necessary to mention the use of CASE WHEN function, which is used in the Main Query. In this question we estimate the average change in price per region between years 2019 and 2020. If we have available data for both 2020 and 2019 the query returns the appropriate result. However, if there is an absence of information about 2019 or 2020 estimations, the code returns a default message declaring the absence of information.

This query would produce the following output:

	Region	Mesh timh Ektimhshs 2019-2020
1	Agia Paraskeui	No information 2020
2	Agios Dimitrios	-4244
3	Ampelokipoi	3784
4	Argiroupoli	No information 2019
5	Athina	9146
6	Dionisos	615
7	Ekali	-698
8	Elliniko	No information 2019
9	Gkizi	-80
10	Glifada	-1252
11	Ilion	-196
12	Ilioupoli	526
13	Kalivia	4296
14	Kifisia	7538
15	Koukaki	1882
16	Lagonisi	-158
17	Nea Smurnh	449
18	Perissos	22
19	Voula	-187
20	Xalandri	-153

FIGURE 18: OUTPUT QUESTION 8

For Example, as we can see for Voula Region the price decreased per 187€ per m^2**.**

For the Argiroupoli Region we have no estimations during 2019 so we can not calculate the difference in price, and the “no information 2019” note is shown in the results.

Case 9: *The percentage of estimations and the percentage of population per Region during 2020*

For this case we calculate the estimations as a percentage of the total estimations calculated from the regions that have been estimated during 2020. The second thing to calculate is the population as a percentage of the total population calculated from all the available regions in the dataset.

```sql
-- Για κάθε κωδικό περιοχής και για το 2020, δείξε το πλήθος των εκτιμήσεων της περιοχής σαν ποσοστό του
-- συνολικού πλήθος εκτιμήσεων του 2020 (μία στήλη), και τον πληθυσμό της περιοχής σαν ποσοστό του
-- συνολικού πληθυσμού όλων των περιοχών.

with percentages as (

select distinct r.ID,count(e.ID) over() as total_es,
count(e.ID) over(partition by r.id) as region_es
    from  Estimation as e
    inner join Property as p on E.Id_Property=p.ID
    inner join Region as R on R.ID=p.Id_Region
    where YEAR(E.Date)=2020
    )
,
 totpop as (
 select r.ID,r.Population as pop_per_region, sum(R.population) over() as total_population
 from Region as R
 )

Select  per.id,cast(cast(cast(per.region_es as decimal(7,2))/per.total_es as decimal(4,2))*100 as varchar(25)) + '%' as Percantages_Estimations,
cast(cast(cast(totpop.pop_per_region as decimal(10,2))/totpop.total_population as decimal(10,2))*100 as varchar(25)) + '%' as Percantages_Population
from percentages as per
inner join totpop on totpop.id = per.ID
```

FIGURE 19: QUESTION 9

Query 9.sql

For this query we used the OVER () and the PARTITION BY statements in order to create a new column where we need to perform an aggregation. Specifically, the first OVER() statement creates a new column in which the count aggregation is performed in order to calculate the number of total estimations happened in 2020. The second OVER PARTITION BY statement performs the count aggregation on estimation IDs which are grouped by (use of partition by) the Region ID.

The SELECT DISTINCT statement is used to avoid having multiple rows for a single region.

The CAST statement is used in order to convert the calculation into the form of percentage (%).

The query above would produce the following output.

	id	Percantages_Estimations	Percantages_Population
1	1	7.00%	8.00%
2	2	4.00%	7.00%
3	4	4.00%	8.00%
4	5	7.00%	9.00%
5	6	4.00%	8.00%
6	7	4.00%	5.00%
7	8	4.00%	3.00%
8	9	4.00%	2.00%
9	10	7.00%	4.00%
10	11	4.00%	6.00%
11	12	7.00%	2.00%
12	13	4.00%	4.00%
13	14	4.00%	6.00%
14	15	7.00%	3.00%
15	16	4.00%	4.00%
16	17	7.00%	5.00%
17	18	7.00%	3.00%
18	19	4.00%	3.00%
19	20	7.00%	6.00%

FIGURE 20: OUTPUT QUESTION 9

Connecting R with the database in SQL Server Management Studio

Case 9: *The percentage of estimations and the percentage of population per region during 2020 (using R)*

For the last case, we establish a connection between the programming language R and our database in Microsoft SQL Server Management Studio (SSMS).

Firstly, the installation of RODBC package and the odbcDriverConnect function allow us to plug in our database “E-Properties”.

```
4  install.packages("RODBC")
5  library(RODBC)
6
7  Connection=odbcDriverConnect('driver={SQL Server};server=LAPTOP-U0DI431G;
8                                database=E-Properties;trusted_connection=true')
```

FIGURE 21: CONNECTION SQL WITH R

Secondly, by using the function sqlSave we create a new table in our database named “TemporaryTable “. Before the use of function sqlQuery this table did not include data, it was a “blank” table with three columns and zero rows.

After the use of function sqlQuery data was inserted in our table, by using our first SQL query. To be more specific, we use the while loop in SQL to insert our records in “TemporaryTable” table.

We declare variables @codeRegion, @EstPer and @EstPop and initialize them with 0 or 1 based on the use of each one in our query. Then, a while loop is executed until the value of the @codeRegion becomes twenty (Twenty is the number of regions). In the body of the while loop, the insert query is being used to insert one record into the “TemporaryTable” table. In the ID column the @codeRegion variable is appended and for the ValueRegion and ValuePop, correspondingly, the @EstPer and @EstPop variables are appended.

Finally, by using, again, the sqlSave function we select all the records from “TemporaryTable” to extract our result.

```
df <- data.frame(matrix(ncol = 3, nrow = 0))
names(df)=c("ID","ValueRegion","ValuePop")

sqlSave(Connection,df,tablename="TemporaryTable",rownames=F)

sqlSave(Connection,df,tablename="TemporaryTable",rownames=F, append=T)

sqlQuery(Connection,
         "DECLARE @codeRegion int = 1 , @EstPer int =0, @EstPop int =0

WHILE @codeRegion <= 20
  BEGIN
  IF ( @codeRegion in (SELECT r.id as Region FROM Region R  INNER JOIN Property P ON P.Id_Region=R.ID
                       INNER JOIN Estimation E ON E.Id_Property=P.ID WHERE YEAR(E.Date)= 2020))
    BEGIN
    SET @EstPer =( SELECT COUNT(E.ID) FROM Region R INNER JOIN Property P ON P.Id_Region=R.ID
                  INNER JOIN Estimation E ON E.Id_Property=P.ID WHERE R.ID=@codeRegion and YEAR(E.Date)= 2020)
    SET @EstPop=( SELECT sum(r.Population)/count(R.ID) FROM Region R INNER JOIN Property P ON P.Id_Region=R.ID
                  INNER JOIN Estimation E ON E.Id_Property=P.ID WHERE R.ID=@codeRegion)
    INSERT INTO [E-Properties].[dbo].[TemporaryTable]
    VALUES (@codeRegion, @EstPer,@EstPop);
    END

  SET @codeRegion=@codeRegion + 1
  SET @EstPer=0
  SET @EstPop=0
  END;")

tableofpercentages=sqlQuery(Connection, "select  A.ID
    ,cast(cast(cast(A.ValueRegion as decimal(7,2))/b.TotalEst as decimal(4,2))*100 as varchar(25)) + '%' as  Percantages_perRegion
    ,cast(cast(cast(A.ValuePop as decimal(7,2))/C.TotalPop as decimal(4,2))*100 as varchar(25)) + '%' as  Percantages_perPopulation
    FROM (select * from dbo.TemporaryTable as T) as A ,
    ( select count(E.ID) AS TotalEst from Estimation AS E WHERE YEAR(E.Date)=2020) as B ,
    ( select SUM( R.Population)  AS TotalPop from Region R  ) as C")

tableofpercentages
```

FIGURE 22: QUESTION 9 SCRIPT IN R

The argument “tableofpercentages” produces the following output:

```
> tableofpercentages
   ID Percantages_perRegion Percantages_perPopulation
1   1                7.00%                    8.00%
2   2                4.00%                    7.00%
3   4                4.00%                    8.00%
4   5                7.00%                    9.00%
5   6                4.00%                    8.00%
6   7                4.00%                    5.00%
7   8                4.00%                    3.00%
8   9                4.00%                    2.00%
9  10                7.00%                    4.00%
10 11                4.00%                    6.00%
11 12                7.00%                    2.00%
12 13                4.00%                    4.00%
13 14                4.00%                    6.00%
14 15                7.00%                    3.00%
15 16                4.00%                    4.00%
16 17                7.00%                    5.00%
17 18                7.00%                    3.00%
18 19                4.00%                    3.00%
19 20                7.00%                    6.00%
```

FIGURE 23: OUTPUT IN R


Final-Question-in-R.r


Query for R.sql