Step 1- *Download and Install MonetDB and PySpark*

MonetDB:

For the installation of MonetDB the following steps were applied:

- Install python 3 that monetDB supports
- Install MonetDB
- Find and copy the path of PythonCore of python 3
- Paste the path in setKEY_NAME="… " that exists in the pyapi_locatepython3 inside of MonetDB folder
- Pip install numpy in python 3
- Run M5server
- Run mclient with username monetdb and password monetdb

PySpark:

For the installation of PySpark, we had to install Java in (C:). Afterwards, I pip installed pyspark and imported all the necessary packaged and modules. Lastly, the following code was used:

```
os.environ['PYSPARK_PYTHON'] = sys.executable

os.environ['PYSPARK_DRIVER_PYTHON'] = sys.executable

spark = SparkSession.builder.appName("zillow").getOrCreate()
```

Step 2-*Loading the Data*

PySpark:

The data were imported into a DataFrame through "spark.read.csv(…/zillow.csv)"

```
+--------------+-------+------------+-----+-----------+----------+--------------------+--------------------+
|         title|address|        city|state|postal_code|     price|  facts and features|real estate provider|
+--------------+-------+------------+-----+-----------+----------+--------------------+--------------------+
|Condo for sale|   null|  Somerville|   MA|      02145|  $342,000|2 bds, 1.0 ba ,70...|William Raveis R....|
|Condo for sale|   null|      Boston|   MA|      02116|$1,700,000|2 bds, 2.0 ba ,12...|Century 21 North ...|
|Condo for sale|   null|      Boston|   MA|      02118|  $336,500|1 bds, 1.0 ba ,10...|Maloney Propertie...|
|House for sale|   null|      Boston|   MA|      02118|$9,950,000|4 bds, 7.0 ba ,68...|Campion & Company...|
|Condo for sale|   null|      Boston|   MA|      02128|  $479,000|2 bds, 3.0 ba ,10...|Berkshire Hathawa...|
|House for sale|   null| East Boston|   MA|      02128|  $899,000|3 bds, 3.0 ba ,23...|Berkshire Hathawa...|
|Condo for sale|   null|  Somerville|   MA|      02145|  $397,300|2 bds, 1.0 ba ,78...|William Raveis R....|
|Condo for sale|   null|South Boston|   MA|      02127|  $619,900|2 bds, 1.0 ba ,85...|Bento Real Estate...|
|Condo for sale|   null|      Boston|   MA|      02116|  $850,000|1 bds, 1.0 ba ,67...|Engel & Volkers B...|
|Condo for sale|   null|      Boston|   MA|      02114|  $649,900|2 bds, 1.0 ba ,51...|Leading Edge Real...|
+--------------+-------+------------+-----+-----------+----------+--------------------+--------------------+
only showing top 10 rows
```

MonetDB:

The data were imported in monetDB, through the "LOADER". The "LOADER" functions can only be written in Python and are created like user-defined functions, without the specification of return types. "Another difference between user-defined functions and loader functions is that you do not need to load all the data at once. Instead, the data can be passed to MonetDB incrementally. This is done using the emit.emit() function. This function can be called to hand over a set of rows to MonetDB. This way, you can load your data into MonetDB even if it does not fit into main memory. The emit function expects a dictionary as parameter. This dictionary should contain the column names as keys, and the values of the columns as values." The code of the "LOADER" function is provided below:

```
create or replace LOADER myloader(filename string) LANGUAGE PYTHON {
import pandas as pd
willow = pd.read_csv(filename,sep=',').to_dict('list')
_emit.emit(willow)};
```

Hence, we created the "LOADER" function that is named "myloader" and it expects the path of the willow.csv file as an input. Then, we specify the python programming language and we write the python code inside the curly brackets. The _emit.emit() function expects a dictionary as parameter. Hence, we provide the appropriate dictionary with the name of each column and the corresponding values of each column in a list.

The origin of the data is from Zillow. It consists of a dataset including ~100K listings from the BOSTON, MA area in CSV format. The query that is needed is the following:

```
CREATE TABLE willow FROM LOADER myloader('…/zillow.csv' );
```

The schema of the table created through the loader is the following:

Grid Text

	title	city	state	postal_code	price	facts and features	real estate provider
1	Condo for sale	Somerville	MA	2,145	$342,000	2 bds, 1.0 ba ,705 sqft	William Raveis R.E. & Home Services
2	Condo for sale	Boston	MA	2,116	$1,700,000	2 bds, 2.0 ba ,1228 sqft	Century 21 North East
3	Condo for sale	Boston	MA	2,118	$336,500	1 bds, 1.0 ba ,1000 sqft	Maloney Properties, Inc.
4	House for sale	Boston	MA	2,118	$9,950,000	4 bds, 7.0 ba ,6836 sqft	Campion & Company Fine Homes Real Estate
5	Condo for sale	Boston	MA	2,128	$479,000	2 bds, 3.0 ba ,1000 sqft	Berkshire Hathaway HomeServices Commonwealth Real Estate
6	House for sale	East Boston	MA	2,128	$899,000	3 bds, 3.0 ba ,2313 sqft	Berkshire Hathaway HomeServices Warren Residential
7	Condo for sale	Somerville	MA	2,145	$397,300	2 bds, 1.0 ba ,780 sqft	William Raveis R.E. & Home Services
8	Condo for sale	South Bostor	MA	2,127	$619,900	2 bds, 1.0 ba ,856 sqft	Bento Real Estate Group, Inc.
9	Condo for sale	Boston	MA	2,116	$850,000	1 bds, 1.0 ba ,675 sqft	Engel & Volkers Boston
10	Condo for sale	Boston	MA	2,114	$649,900	2 bds, 1.0 ba ,511 sqft	Leading Edge Real Estate

The table contains the columns tile, address (which is not shown due to only null values), city, state, postal code, price, facts and features, real estate provider and url (which is not shown due to lack of space). Each column contains string values.

Step 3-*Analysis*

Question 1- *Extract number of bedrooms. You will implement a UDF that processes the "facts_and_features" column and extracts the number of bedrooms.*

PySpark:

We need to create a function which extracts the number of bedrooms from a string. We observed that the “facts and features” column displays the number of bedrooms, bathrooms and sqft in this specific order. Hence, we obtain the integer of the first word of the string in this column, which is the number of bedrooms. If the first word is None, it returns zero.

```
def beds(str):
    if str.split()[0]=='None':
        return 0
    else:
        return int(str.split()[0])
```

Then, we create the UDF with a lambda function which will apply the above “beds” function in each row.

```
bedsUDF=udf(lambda z:beds(z),StringType())
```

Finally, we use the withColumn (found in documentation of PySpark) function that applies the UDF in a specific column and creates a new column in the DataFrame with the number of bedrooms.

```
df = df.withColumn('beds',bedsUDF('facts and features'))
```

MonetDB:

We need to create a function which extracts a list that contains the number of bedrooms. This function needs a column as an input and returns another column that contains the number of bedrooms. Hence, each function in MonetDB needs a loop. The string manipulation in python code follows the same logic as the PySpark.

```
create or replace FUNCTION get_bedrooms(i STRING)
RETURNS INTEGER
LANGUAGE PYTHON {
x=[]
for j in range(len(i)):
    if i[j].split()[0]=="None":
        x.append(0)
    else:
        x.append(int(i[j].split()[0]))
  return x};
```

Question 2- *Extract number of bathrooms. You will implement a UDF that processes the "facts_and_features" column and extracts the number of bathrooms.*

PySpark:

We need to create a function which extracts the number of bathrooms from a string. We observed that the “facts and features” column displays the number of bedrooms, bathrooms and sqft in this specific order. Hence, we obtain the integer of the third word of the string in this column, which is the number of bathrooms. If this word is None, it returns zero.

```
def baths(str):
    if str.split()[2]=='None':
        return 0
    else:
        return int(float(str.split()[2]))
```

Then, we create the UDF with a lambda function which will apply the above “baths” function in each row.

```
bathsUDF=udf(lambda z:baths(z),StringType())
```

Finally, we use the withColumn (found in documentation of PySpark) function that applies the UDF in a specific column and creates a new column in the DataFrame with the number of bathrooms.

```
df = df.withColumn('baths',bathsUDF('facts and features'))
```

MonetDB:

We need to create a function which extracts a list that contains the number of bathrooms. This function needs a column as an input and returns another column that contains the number of bathrooms. Hence, each function in MonetDB needs a loop. The string manipulation in python code follows the same logic as the PySpark.

```
create or replace FUNCTION get_bathrooms(i STRING)
RETURNS INTEGER
LANGUAGE PYTHON {
x=[]
for j in range(len(i)):
    if i[j].split()[2]=="None":
        x.append(0)
    else:
        x.append(int(float(i[j].split()[2])))
return x};
```

Question 3- *Extract sqft. You will implement a UDF that processes the "facts_and_features" column and extracts the sqft.*

PySpark:

We need to create a function which extracts the number of sqft from a string. We observed that the “facts and features” column displays the number of bedrooms, bathrooms and sqft in this specific order. We need to separate the third part of the string after the commas and obtain the integer of the first word of the string in, which is the number of sqft. If this number is None, it returns zero.

```
def sqft(str):
    if str.split(',')[2].split()[0]=="None":
        return 0
    else:
        return int(str.split(',')[2].split()[0])
```

Then, we create the UDF with a lambda function which will apply the above “sqft” function in each row.

```
sqftUDF=udf(lambda z:sqft(z),StringType())
```

Finally, we use the withColumn (found in documentation of PySpark) function that applies the UDF in a specific column and creates a new column in the DataFrame with the number of sqft.

```
df = df.withColumn('sqft',sqftUDF('facts and features'))
```

MonetDB:

We need to create a function which extracts a list that contains the number of sqft. This function needs a column as an input and returns another column that contains the number of sqft. Hence, each function in MonetDB needs a loop. The string manipulation in python code follows the same logic as the PySpark.

```
create or replace FUNCTION get_sqft(i STRING)
RETURNS INTEGER
LANGUAGE PYTHON {
x=[]
for j in range(len(i)):
      if i[j].split(',')[2].split()[0]=="None":
            x.append(0)
      else:
            x.append(int(i[j].split(',')[2].split()[0]))
return x};
```

Question 4- *Extract type. You will implement a UDF that processes the "title" column and returns the type of the listing (e.g. condo, house, apartment)*

PySpark:

We need to create a function which extracts the type of listing from a string. We observed that the “title” column contains specific words. We assume based on that, that there are five types of listings. These types are Condo, House, Lot/Land, New Construction, Family Home (we could include that in House Category) and Other type. Based on that we convert the string in lower letters and categorize it in one of these categories.

```
def types(str):
    if 'condo' in str.lower():
        return "Condo"
    elif "house" in str.lower():
```

```
        return "House"
    elif "land" in str.lower():
        return "Lot/Land"
    elif "construction" in str.lower():
        return "New Construction"
    elif "home" in str.lower():
        return "Family Home"
    else:
        return "Other type"
typesUDF=udf(lambda z:types(z),StringType())
df = df.withColumn('type',typesUDF('title'))
```

MonetDB:

We need to create a function which extracts a list that contains the type of listing. This function needs a column as an input and returns another column that contains the type of listing. Hence, each function in MonetDB needs a loop. The string manipulation in python code follows the same logic as the PySpark.

```
CREATE OR REPLACE FUNCTION get_type(i STRING)
RETURNS string
LANGUAGE PYTHON {
x=[]
for j in range(len(i)):
      if 'condo' in i[j].lower():
            x.append('Condo')
      elif 'house' in i[j].lower():
            x.append('House')
      elif 'land' in  i[j].lower():
            x.append('Lot/Land')
      elif 'construction' in i[j].lower():
            x.append('New Construction')
      elif 'home' in i[j].lower():
            x.append('Family Home')
      else:
            x.append('Other Type')
return x};
```

Question 5- *Extract offer. You will implement a UDF that processes the title column and returns the type of offer. This can be sale, rent, sold, forclose.*

PySpark:

We need to create a function which extracts the type of offer from a string. We observed that the “title” column contains specific words. We assume based on that, that there are four types of offers. These types are sale, rent,

sold, foreclose and Other state if any exists. Based on that we convert the string in lower letters and categorize it in one of these categories.

```
def offers(str):
    if 'sale' in str.lower():
        return "sale"
    elif "rent" in str.lower():
        return "rent"
    elif "sold" in str.lower():
        return "sold"
    elif "foreclosure" in str.lower():
        return "foreclose"
    else:
        return "Other offer"
offersUDF=udf(lambda z:offers(z),StringType())
df = df.withColumn('offers',offersUDF('title'))
```

MonetDB:

We need to create a function which extracts a list that contains the type of offer. This function needs a column as an input and returns another column that contains the type of offer. Hence, each function in MonetDB needs a loop. The string manipulation in python code follows the same logic as the PySpark.

```
CREATE OR REPLACE FUNCTION get_offer(i STRING)
RETURNS string
LANGUAGE PYTHON {
x=[]
for j in range(len(i)):
      if 'sale' in i[j].lower():
            x.append('sale')
      elif 'rent' in i[j].lower():
            x.append('rent')
      elif 'sold' in  i[j].lower():
            x.append('sold')
      elif 'foreclosure' in i[j].lower():
            x.append('foreclose')
      else:
            x.append('Other State')
return x};
```

Question 6- *Filter out listings that are not for sale.*

PySpark:

We filter out listing that are not for sale with:

```
df=df[df.offers=='sale']
```

We keep all the data in one DataFrame during the filtering, so there is no excessive memory used.

MonetDB:

We create the final table with all the extra columns that the UDFs create.

```
create table final_willow as select *, get_bedrooms("facts and features") as
bedrooms, get_bathrooms("facts and features") as bathrooms, get_sqft("facts and
features") as sqft, get_type("title") as type, get_offer("title") as offer from
willow;
```

Finally, we create a table that filter out listings that are not for sale with:

```
create table filtered_sales as select * from final_willow where (offer= 'sale');
```

Question 7- *Extract price. You will implement a UDF that processes the price column and extract the price. Prices are stored as strings in the CSV. This UDF parses the string and returns the price as an integer.*

PySpark:

We need to create a function which extracts the price from a string. We observed that the "price" column contains a string that starts with '$' and there are some cases that include '+'. Based on that we convert the string in integer after we exclude '$', ',', '+'.

```
def price(str):
    return int(float(str[1:].replace(',','').replace('+','')))
priceUDF=udf(lambda z:price(z),StringType())
df = df.withColumn('prices',priceUDF('price'))
```

MonetDB:

We need to create a function which extracts a list that contains price. This function needs a column as an input and returns another column that contains the price. Hence, each function in MonetDB needs a loop. The string manipulation in python code follows the same logic as the PySpark.

```
CREATE OR REPLACE FUNCTION get_price(i STRING)
RETURNS integer
LANGUAGE PYTHON {
x=[]
for j in range(len(i)):
      x.append(int(float(i[j][1:].replace(',','').replace('+',''))))
return x};
```

Question 8- *Filter out listings with more than 10 bedrooms*

Question 9- *Filter out listings with price greater than 20000000 and lower than 100000*

Question 10- *Filter out listings that are not houses*

The questions 8, 9, 10 follow the same logic.

PySpark:

We filter out listing that have more than 10 bedrooms, price greater than 20000000 and lower than 100000 and those that are not houses with:

```
df=df[df.beds<=10]

df=df[(df.prices<20000000) & (df.prices>100000)]

df=df[df.type=="House"]
```

We keep all the data in one DataFrame during the filtering, so there is no excessive memory used.

```
+--------------+------------+-----+-----------+----------+--------------------+--------------------+----+-----+----+-----+-----
-+-------+
|         title|        city|state|postal_code|     price|  facts and features|real estate provider|beds|baths|sqft| type|offer
s| prices|
+--------------+------------+-----+-----------+----------+--------------------+--------------------+----+-----+----+-----+-----
-+-------+
|House for sale|      Boston|   MA|      02118|$9,950,000|4 bds, 7.0 ba ,68...|Campion & Company...|   4|    7|6836|House|  sal
e|9950000|
|House for sale| East Boston|   MA|      02128|  $899,000|3 bds, 3.0 ba ,23...|Berkshire Hathawa...|   3|    3|2313|House|  sal
e| 899000|
|House for sale|      Boston|   MA|      02113|$1,200,000|2 bds, 1.0 ba ,11...|       CL Properties|   2|    1|1165|House|  sal
e|1200000|
|House for sale|      Boston|   MA|      02129|$1,119,000|3 bds, 4.0 ba ,16...|All Star Realty, ...|   3|    4|1680|House|  sal
e|1119000|
|House for sale|South Boston|   MA|      02127|$1,699,000|4 bds, 3.0 ba ,20...|             Compass|   4|    3|2043|House|  sal
e|1699000|
|House for sale|      Boston|   MA|      02128|  $589,000|2 bds, 2.0 ba ,12...|Gibson Sotheby's ...|   2|    2|1200|House|  sal
e| 589000|
|House for sale|      Boston|   MA|      02115|$9,750,000|4 bds, 7.0 ba ,47...|MGS Group Real Es...|   4|    7|4754|House|  sal
e|9750000|
|House for sale|  Somerville|   MA|      02145|$2,075,000|4 bds, 4.0 ba ,31...|Gibson Sotheby's ...|   4|    4|3121|House|  sal
e|2075000|
|House for sale|      Boston|   MA|      02114|$3,200,000|3 bds, 3.0 ba ,20...|William Raveis R....|   3|    3|2048|House|  sal
e|3200000|
|House for sale|South Boston|   MA|      02127|$1,175,000|3 bds, 3.0 ba ,18...|Coldwell Banker R...|   3|    3|1867|House|  sal
e|1175000|
+--------------+------------+-----+-----------+----------+--------------------+--------------------+----+-----+----+-----+-----
-+-------+
only showing top 10 rows
```

MonetDB:

We create the final filtered table with all the columns from filtered_sales table and the addition of column that the get_price UDF creates.

```
create table filtered as select * from (select *, get_price("price") as prices from
filtered_sales) as price_table where (bedrooms <=10) and (prices > 100000) and
(prices < 20000000) and (type= 'House');
```

	title	addres	city		posta	price	facts and features	real estate pro	url	bedroom	bathrooms	sqft	type	offer	price
1	House for sale	[NULL]	Boston	MA	2,118	$9,950,00(	4 bds, 7.0 ba ,6836 sq	Campion & Comp	https:/	4	7	6,836	House	sale	9,950,000
2	House for sale	[NULL]	East Boston	MA	2,128	$899,000	3 bds, 3.0 ba ,2313 sq	Berkshire Hathaw	https:/	3	3	2,313	House	sale	899,000
3	House for sale	[NULL]	Boston	MA	2,113	$1,200,00(	2 bds, 1.0 ba ,1165 sq	CL Properties	https:/	2	1	1,165	House	sale	1,200,000
4	House for sale	[NULL]	Boston	MA	2,129	$1,119,00(	3 bds, 4.0 ba ,1680 sq	All Star Realty, Inc	https:/	3	4	1,680	House	sale	1,119,000
5	House for sale	[NULL]	South Boston	MA	2,127	$1,699,00(	4 bds, 3.0 ba ,2043 sq	Compass	https:/	4	3	2,043	House	sale	1,699,000
6	House for sale	[NULL]	Boston	MA	2,128	$589,000	2 bds, 2.0 ba ,1200 sq	Gibson Sotheby's	https:/	2	2	1,200	House	sale	589,000
7	House for sale	[NULL]	Boston	MA	2,115	$9,750,00(	4 bds, 7.0 ba ,4754 sq	MGS Group Real I	https:/	4	7	4,754	House	sale	9,750,000
8	House for sale	[NULL]	Somerville	MA	2,145	$2,075,00(	4 bds, 4.0 ba ,3121 sq	Gibson Sotheby's	https:/	4	4	3,121	House	sale	2,075,000
9	House for sale	[NULL]	Boston	MA	2,114	$3,200,00(	3 bds, 3.0 ba ,2048 sq	William Raveis R.I	https:/	3	3	2,048	House	sale	3,200,000
10	House for sale	[NULL]	South Boston	MA	2,127	$1,175,00(	3 bds, 3.0 ba ,1867 sq	Coldwell Banker F	https:/	3	3	1,867	House	sale	1,175,000

Question 11- *Calculate average price per sqft for houses for sale grouping them by the number of bedrooms*

PySpark:

We use “groupBy” with “beds” in the DataFrame and then we apply the pyspark sql average function in prices divided with sqft in each group.

```
df.groupBy("beds").agg(pyspark.sql.functions.avg(df.prices/df.sqft)).show()
```

```
+----+--------------------+
|beds|avg((prices / sqft))|
+----+--------------------+
|   7|  1126.0252348993286|
|   3|   687.2172712125251|
|   8|   1567.647058823529|
|   0|              1250.0|
|   5|   733.9532477532824|
|   6|   422.3111656297147|
|   9|  1108.1412183984849|
|   4|   909.1473996440552|
|   2|   716.0381965996941|
+----+--------------------+
```

MonetDB:

The query for this question is straight forward. We use the average function the same way as in PySpark in grouped by bedrooms

```
select avg(prices/sqft) as avg_price_per_sqft  ,bedrooms from filtered group by
bedrooms;
```

	avg_price_per_sqft	bedrooms
1	908.5857436766	4
2	686.789516129	3
3	715.6112156568	2
4	733.4645502646	5
5	421.5555555556	6
6	1,250	0
7	1,108	9
8	1,126	7
9	1,567	8

We observe that there are some differences in the results of the average price per sqft between PySpark and MonetDB. It can be estimated that there are some rounding errors.