RENTAL PROPERTY MANAGEMENT SYSTEM

Student Name: Vishnu Girish Nujaralla Iyappan

Executive Summary:

The primary objective of this study was to design and implement a relational database system to ease the process of Rental Management. It helps to track the profit margin using revenue generated and customers to find the apartments based on budget preferences. It also helps to track the maintenance requests for the apartments to find out which apartment required more maintenance.

The database was designed with the data fields that customers and landlords need. The conceptual model was mapped to a relational model with the necessary primary and foreign keys after the EER and UML diagrams were modelled. This database was then implemented in MySQL and implemented in MongoDB NoSQL database to study the feasibility of this database in a NoSQL environment.

The created database is connected to R and visualization of analyzed data helps landlord, customer, and our firm to improve the standards of Rental Management System. This also helps us to predict and manage the resources needed to customers without delays.

I. Introduction

Finding a rental home and furniture for people who are moving to a new place is really challenging these days. Additionally, it might be challenging for landlords to locate tenants that meet their requirements. The search for a property that fits their budget and is hassle-free takes a lot of time and effort. As there are numerous methods to generate data in today's fast paced world, it is now crucial that the data generated is gathered and handled in a systematic manner so that it can be transformed into significant information. It needs a reliable database management system that enables consumers to be connected with the ideal property for their needs.

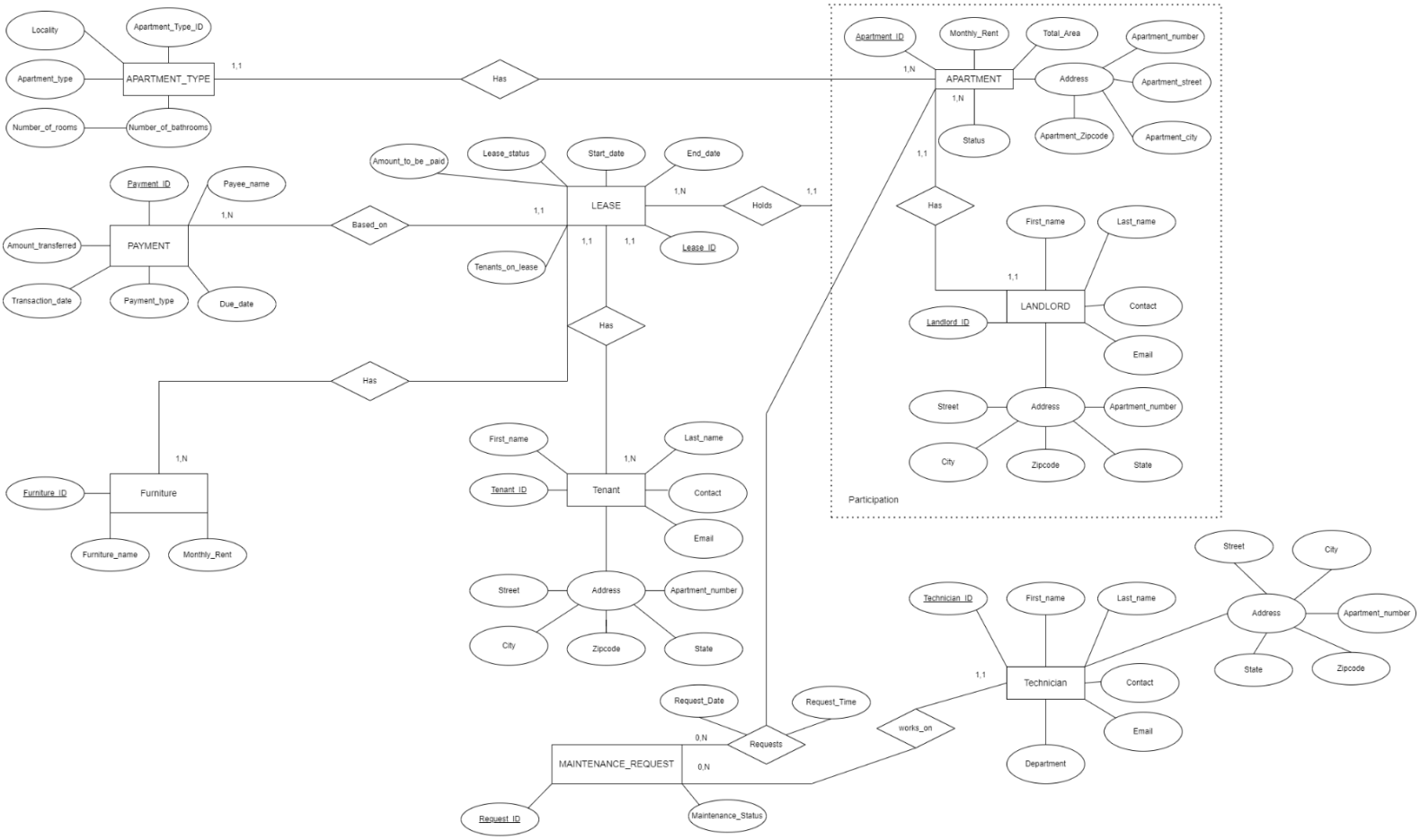
Our database management solution provides a single window that addresses all the requirements of tenants and landlords. Database is created which stores data in normalized format by identifying dependencies and splitting the tables appropriately with proper Primary and Foreign key relationship. Landlord has access to view where he can check all the tenants who are currently living or who used to live in his apartment. Customers can search for apartments depending on their preferences for neighborhood and price range. The property’s maintenance requests can be tracked to determine which apartment needed more work. Rent paid by tenants each month can be used to calculate revenues. This database is also used to monitor the company's earnings, which helps the management system get better.

Our database needs the information of apartment, landlord, customer, employee, furniture, employee roles, maintenance requests, utility, lease details, payment details. For each apartment we need to record the apartment ID, landlord ID, apartment type, address. For each landlord we need to record the name, contact information and address. For each customer we need to record the customer ID, lease ID, name and address. For each employee we need to have employee ID, name and address. For the furniture we need to have furniture ID, cost and type. For maintenance requests we need to have request ID, apartment ID, date of request and date of completion. For lease details we need to have lease ID, start date, end date, members, status. For each payment details we need to have lease ID, amount, transaction type and date. For employee roles we need to have employee roles and employee ID.

Other requirements:
1. One landlord can have many apartments. Whereas one apartment must be maintained by one landlord
2. One apartment can have many customers. Whereas one customer can have only one apartment.
3. One apartment has many leases, and one lease can hold one apartment at a time.
4. One lease can have many customers. Whereas one customer can have only one lease.
5. One employee can have only one role. Whereas one role can have many employees

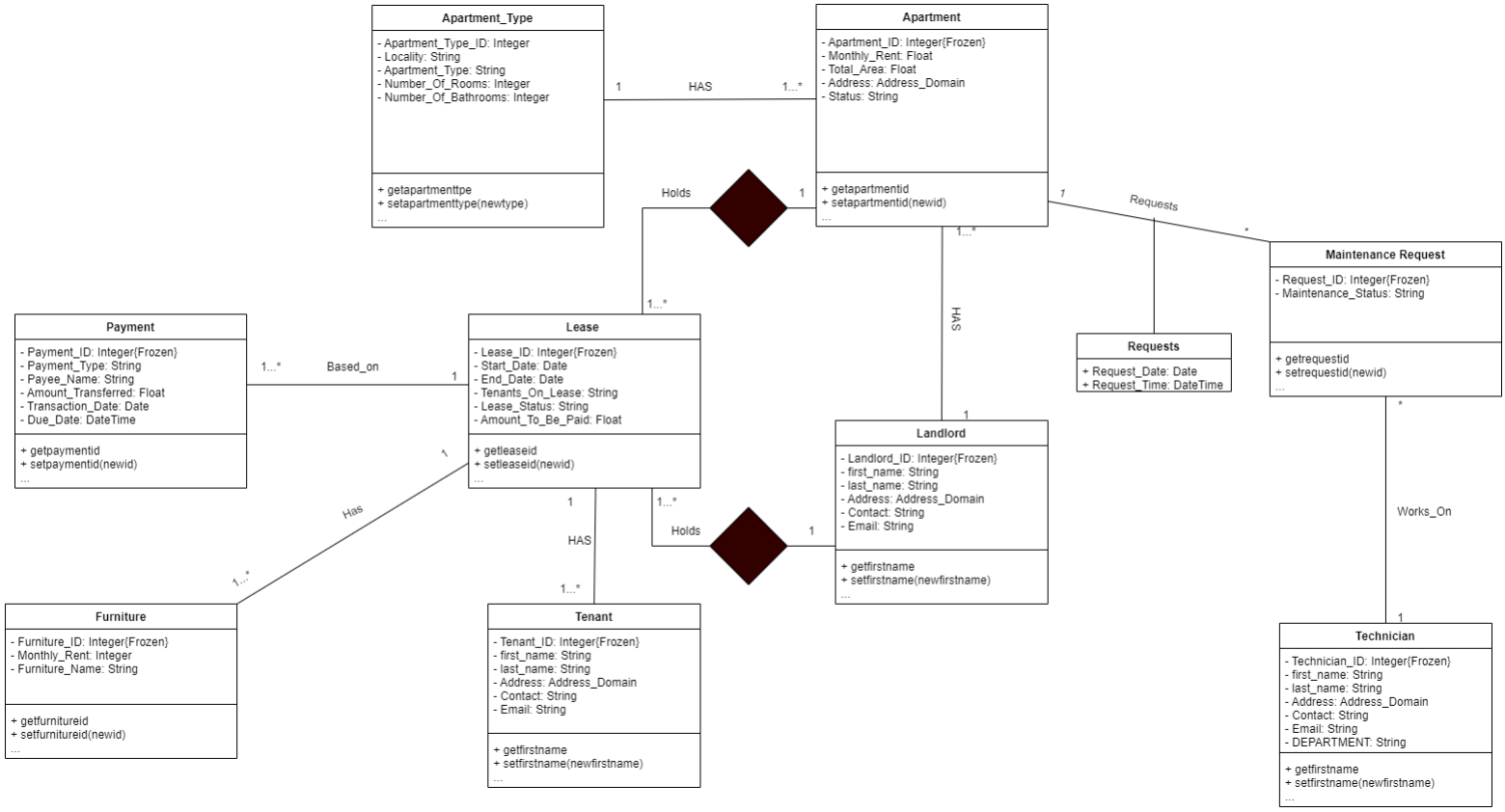
II. Conceptual Data Modeling

1. EER Diagram



.

2. UML Diagram



III. Mapping Conceptual Model to Relational Model

Primary Key- Underlined **Foreign Key- *Italicized***

APARTMENT_TYPE(**Apartment_type_ID,** Apartment_type**,** Number_of_rooms, Number_of_bathrooms, Locality)

APARTMENT(**Apartment_ID**, *Apartment_type_ID(Not Null),*, *Landlord_ID(Not Null),,* Monthly_Rent, Total_Area, Apartment_number, Apartment_street, Apartment_city, Apartment_Zipcode, Status)

LANDLORD(**Landlord_ID**, First_name, Last_name, Email, Contact, Apartment_number, street, city, State, Zipcode)

MAINTENANCE_REQUEST(**Request_ID**, *Apartment_ID, Technician_ID*(Not Null), Request_Close_date, Maintenance_Status, Request_Date, Request_time)

LEASE(**Lease_ID**, *Apartment_ID(Not Null)*, *Landlord_ID(Not Null)*, Tenants_on_lease, Lease_status, Start_date, End_date, Amount_to_be_paid)

PAYMENT(**Payment_ID**, *Lease_ID(Not Null*), Payee_name, Amount_Transferred, Transaction_date, Payment_type, Due_date)

FURNITURE(**Furniture_ID**, *Lease_ID(Not Null)*, Furniture_name, Monthly_Rent)

TENANT(**Tenant_ID**, *Lease_ID(Not Null)*, First_name, Last_name, Email, Contact, Apartment_number, Street, City, State, Zipcode)

Technician(**Technician_ID**, First_Name, Last_Name, Email, Contact, Apt_number, Street, City, State, Zipcode, Department)

IV. Implementation of Relation Model via MySQL and NoSQL

MySQL Implementation:

The database was created in MySQL and the following queries were performed:

QUERY 1: FIND Apartment ID, Apartment city, Landlord ID, Landlord Email where Monthly rent is less than $4000 and Apartment Status is Vacant

```
USE RENTAL_MANAGEMENT_SYSTEM;
SELECT A.APARTMENT_ID, A.APARTMENT_CITY, A.LANDLORD_ID, L.LANDLORD_EMAIL
FROM APARTMENT A, LANDLORD L
WHERE A.LANDLORD_ID = L.LANDLORD_ID AND A.MONTHLY_RENT < 4000 AND
A.APARTMENT_STATUS = "VACANT";
```

Output:

APARTMENT_ID	APARTMENT_CITY	LANDLORD_ID	LANDLORD_EMAIL
100067	Long Beach	688492234	bmccriel@google.pl
100180	Cincinnati	213839298	mbenediktovicho@tinypic.com
100234	Anaheim	450116064	brimells@baidu.com
100244	Salt Lake City	648583973	efritchleyi@umn.edu
100246	Bismarck	485904594	dghirigori14@bloglovin.com
100277	Fort Wayne	649638434	avant25@w3.org
100296	Milwaukee	532814810	tlakelandv@w3.org
100332	Valdosta	723933230	dmouser13@alibaba.com
100348	Miami	101791286	adechelette28@tinyurl.com
100437	Fort Pierce	356036515	dbrickell1s@amazon.de

QUERY 2: FIND NUMBER OF APARTMENTS UNDER EACH APARTMENT CATEGORY

```
SELECT APARTMENT_TYPE, NUMBER_OF_ROOMS, NUMBER_OF_BATHROOMS,
COUNT(APARTMENT_TYPE)
FROM APARTMENT_TYPE
GROUP BY APARTMENT_TYPE, NUMBER_OF_ROOMS, NUMBER_OF_BATHROOMS
ORDER BY NUMBER_OF_ROOMS, NUMBER_OF_BATHROOMS;
```

Output:

APARTMENT_TYPE	NUMBER_OF_ROOMS	NUMBER_OF_BATHROOMS	COUNT(APARTMENT_TYPE)
Apartment	1	1	2
Condo	1	1	4
Apartment	1	2	5
Condo	1	2	1
Apartment	1	3	3
Condo	1	3	4
Condo	2	1	2
Apartment	2	1	3
Apartment	2	2	3
Condo	2	2	1

QUERY 3: FIND THE APARTMENT ID AND CITY WHICH HAS MORE THAN 2 MAINTEMANCE REQUESTS

```
SELECT A.APARTMENT_ID, A.APARTMENT_CITY
FROM APARTMENT A
WHERE 2 <
    (SELECT COUNT(APARTMENT_ID)
    FROM MAINTENANCE_REQUEST M
    WHERE M.APARTMENT_ID = A.APARTMENT_ID);
```

Output:

APARTMENT_ID	APARTMENT_CITY
100050	Springfield
100234	Anaheim
100332	Valdosta
100367	Stockton
NULL	NULL

QUERY 4: FIND THE EMAIL OF LANDLORD AND TENANT WHOSE LEASE END DATE IS BEFORE 2022-03-31

```
SELECT L.LEASE_ID, L.END_DATE, L.AMOUNT_TO_BE_PAID, T.TENANT_EMAIL,
LA.LANDLORD_EMAIL
FROM LEASE L, TENANT T, LANDLORD LA
WHERE L.LEASE_ID = T.LEASE_ID AND L.LANDLORD_ID = LA.LANDLORD_ID AND END_DATE <
'2022-03-31';
```

Output:

LEASE_ID	END_DATE	AMOUNT_TO_BE_PAID	TENANT_EMAIL	LANDLORD_EMAIL
11158	2022-02-15	575	jpischof8@state.gov	ldonaldson8@cloudflare.com
11183	2022-02-18	1430	dklimowski1q@ucsd.edu	cbougourd1q@drupal.org
11189	2022-02-26	725	bvennardv@amazon.co.jp	tlakelandv@w3.org
12122	2022-02-21	426	mpley2a@theglobeandmail.com	agieraths2a@tripod.com
12140	2022-02-01	1294	dmousby1@state.gov	qsiddons1@hostgator.com
13120	2022-02-03	679	dminally@nba.com	nfidgetty@columbia.edu
13194	2022-03-23	1609	egodard1i@prweb.com	ecroker1i@rakuten.co.jp
13197	2022-02-20	1551	hgariochw@msn.com	amccolleyw@vinaora.com
14108	2022-02-28	723	dknowlton2q@bbb.org	lpedlow2n@nhs.uk
14108	2022-02-28	723	gwhistlecraft2n@google.co.jp	lpedlow2n@nhs.uk

QUERY 5: FIND THE REVENUE GENERATED BY THE RENTAL MANAGEMNET FIRM FROM EACH PAYMENT TYPE (ASSUMING 15% OF TOTAL AMOUNT TRANSFERRED BY EACH TYPE IS THE REVENUE FOR RENTAL MANAGENT FIRM)

```
WITH REVENUE AS
        (SELECT PAYMENT_TYPE, AMOUNT_TRANSFERRED FROM PAYMENT)
                SELECT PAYMENT_TYPE, SUM(AMOUNT_TRANSFERRED)*0.15 AS
REVENUE_GENERATED
                FROM REVENUE
                GROUP BY PAYMENT_TYPE;
```

Output:

PAYMENT_TYPE	REVENUE_GENERATED
jcb	41173.50
maestro	11956.05
diners-club-enroute	3402.90
instapayment	1738.05
visa-electron	4391.70
switch	8153.40
americanexpress	5043.90
solo	1603.50
laser	4749.45
diners-club-carte-blanche	1200.30

QUERY 6: FIND TECHNICIAN ID'S AND NUMBER OF TASKS ASSIGNED TO THE TECHNICIANS WHO IS IN MECHNICAL DEPARTMENT

```
SELECT TECHNICIAN_ID, COUNT('TECHNICIAN_ID') AS NUMBER_OF_TASKS_ASSIGNED
FROM MAINTENANCE_REQUEST
 WHERE TECHNICIAN_ID IN
                (SELECT TECHNICIAN_ID
                FROM TECHNICIAN
                WHERE DEPARTMENT='MECHANICAL')
GROUP BY TECHNICIAN_ID;
```

Output:

TECHNICIAN_ID	NUMBER_OF_TASKS_ASSIGNED
167926243	3
183375379	4
254143025	3
268420261	3
420941053	3
449893677	4
638718979	4
726847614	4
801041922	4
868736327	4

QUERY 7: RETRIVE THE ADDRESS OF THE APARTMENTS WHICH RENTED THE BED

```
SELECT APARTMENT_NUMBER, APARTMENT_STREET_NAME,
APARTMENT_CITY,APARTMENT_ZIPCODE
FROM APARTMENT
WHERE APARTMENT_ID IN
                (SELECT APARTMENT_ID
                FROM LEASE
                WHERE LEASE_ID IN
                                (SELECT LEASE_ID
                                 FROM FURNITURE
                                  WHERE FURNITURE_NAME = 'BED'));
```

Output:

APARTMENT_NUMBER	APARTMENT_STREET_NAME	APARTMENT_CITY	APARTMENT_ZIPCODE
7	Daystar	Springfield	80255
2252	Elgar	Las Vegas	85010
3600	Bluestem	Long Beach	89135
557	Almo	Watertown	73179
10	Mayer	Washington	37220
222	Leroy	Cincinnati	33884
7098	Bobwhite	Fort Worth	45454
95	Eggendart	Anaheim	75074
11621	Arrowood	Cincinnati	35487
596	Dryden	Valdosta	24040

QUERY 8: FIND APARTMENT_ID AND ASSIGNED TECHNICIAN ID AND EMAIL WHERE MAINTENANCE STATUS IS PENDING

```
select M.APARTMENT_ID, M. TECHNICIAN_ID, T.TECHNICIAN_EMAIL, M.MAINTENANCE_STATUS
FROM MAINTENANCE_REQUEST AS M LEFT OUTER JOIN TECHNICIAN AS T
ON (M.TECHNICIAN_ID=T.TECHNICIAN_ID)
WHERE M.MAINTENANCE_STATUS = "PENDING";
```

Output:

APARTMENT_ID	TECHNICIAN_ID	TECHNICIAN_EMAIL	MAINTENANCE_STATUS
100163	753381026	larendsenr@shoppro.jp	PENDING
100629	647390232	ahartfieldl@yale.edu	PENDING
100331	532636092	jpadell6@scientificamerican.com	PENDING
100839	272208127	asessionsv@freewebs.com	PENDING
100755	693942487	vodlinj@google.cn	PENDING
100449	294467632	mbrabham2@simplemachines.org	PENDING
100016	449893677	fdutteridge11@buzzfeed.com	PENDING
100987	880402839	ccoulex@ezinearticles.com	PENDING
100109	638718979	tkeighers@jugem.jp	PENDING
100263	156049554	bkiggel5@cornell.edu	PENDING

QUERY 9: CREATE VIEW OF AREA OF APARTMENT LESS THAN 2000

```
CREATE VIEW AREA AS SELECT APARTMENT_ID, TOTAL_AREA, APARTMENT_CITY
FROM APARTMENT WHERE TOTAL_AREA < 2000;
SELECT * FROM AREA;
```

Output:

APARTMENT_ID	TOTAL_AREA	APARTMENT_CITY
100016	1500	Youngstown
100067	1800	Long Beach
100076	1800	Huntington Beach
100089	1800	Salt Lake City
100094	1500	Roanoke
100128	1500	Washington
100141	1500	Huntington
100166	1800	Washington
100331	1800	Brooklyn
100367	1800	Stockton

QUERY 10: RETRIVE APARTMENT_ID, APARTMENT_CITY AND MONTHLY_RENT FOR EACH APARTMENT WHICH HAS THE HIGHEST RENT OF SAME TOTAL_AREA.

```
SELECT A.APARTMENT_ID, A.APARTMENT_CITY, A.MONTHLY_RENT
FROM APARTMENT A
WHERE A.MONTHLY_RENT>=ALL
(SELECT A1.MONTHLY_RENT FROM APARTMENT A1 WHERE A1.TOTAL_AREA=A.TOTAL_AREA);
```

Output:

APARTMENT_ID	APARTMENT_CITY	MONTHLY_RENT
100141	Huntington	5928
100264	Cincinnati	5655
100370	Hartford	5295
100478	Warren	5952
100772	Honolulu	5700
100808	Wichita	5996
100993	Fullerton	5645
NULL	NULL	NULL

NoSQL Implementation:

The following queries were done in MongoDB Online Playground:

QUERY 1: FIND NUMBER OF APARTMENTS UNDER EACH APARTMENT CATEGORY

Configuration: bson

Sample data:

```
[
  {
    "APARTMENT_TYPE_ID": 953,
    "APARTMENT_TYPE": "Apartment",
    "NUMBER_OF_ROOMS": 4,
    "NUMBER_OF_BATHROOMS": 3,
    "LOCALITY": "Saint Paul"
  },
  {
    "APARTMENT_TYPE_ID": 6139,
    "APARTMENT_TYPE": "Apartment",
    "NUMBER_OF_ROOMS": 2,
    "NUMBER_OF_BATHROOMS": 3,
    "LOCALITY": "Manchester"
  },
  {
    "APARTMENT_TYPE_ID": 16188,
    "APARTMENT_TYPE": "Condo",
    "NUMBER_OF_ROOMS": 3,
    "NUMBER_OF_BATHROOMS": 2,
    "LOCALITY": "Irvine"
  },
  {
    "APARTMENT_TYPE_ID": 19762,
    "APARTMENT_TYPE": "Condo",
    "NUMBER_OF_ROOMS": 3,
    "NUMBER_OF_BATHROOMS": 1,
    "LOCALITY": "Louisville"
  },
  {
    "APARTMENT_TYPE_ID": 29748,
```

Query:

```
//FIND NUMBER OF APARTMENTS UNDER EACH APARTMENT CATEGORY
db.collection.aggregate([
  {
    $group: {
      _id: {
        "Apartment_Type": "$APARTMENT_TYPE",
        "NUMBER_OF_ROOMS": "$NUMBER_OF_ROOMS",
        "NUMBER_OF_BATHROOMS": "$NUMBER_OF_BATHROOMS"
      },
      "count": {
        "$sum": 1
      }
    }
  }
])
```

Result:

```
[
  {
    "_id": {
      "Apartment_Type": "Apartment",
      "NUMBER_OF_BATHROOMS": 1,
      "NUMBER_OF_ROOMS": 5
    },
    "count": 2
  },
  {
    "_id": {
      "Apartment_Type": "Condo",
      "NUMBER_OF_BATHROOMS": 3,
      "NUMBER_OF_ROOMS": 5
    },
    "count": 1
  },
  {
    "_id": {
      "Apartment_Type": "Condo",
      "NUMBER_OF_BATHROOMS": 2,
      "NUMBER_OF_ROOMS": 4
    },
    "count": 3
  },
  {
    "_id": {
      "Apartment_Type": "Condo",
      "NUMBER_OF_BATHROOMS": 1,
      "NUMBER_OF_ROOMS": 3
    },
```

MongoDB version 6.0.3 - Report an issue - About this playground

QUERY 2: FIND THE TOTAL AMOUNT SPENT USING EACH PAYMENT TYPE IN ASCENDING ORDER

Configuration: bson

Sample data:

```
[
  {
    "PAYMENT_ID": 600000,
    "LEASE_ID": 14170,
    "PAYEE_NAME": "Maxie",
    "AMOUNT_TRANSFERRED": 1842,
    "TRANSACTION_DATE": "2021-12-17",
    "PAYMENT_TYPE": "jcb",
    "DUE_DATE": "2022-10-21"
  },
  {
    "PAYMENT_ID": 600006,
    "LEASE_ID": 14108,
    "PAYEE_NAME": "Gilemette",
    "AMOUNT_TRANSFERRED": 2124,
    "TRANSACTION_DATE": "2021-12-21",
    "PAYMENT_TYPE": "jcb",
    "DUE_DATE": "2023-01-10"
  },
  {
    "PAYMENT_ID": 600010,
    "LEASE_ID": 13142,
    "PAYEE_NAME": "Tiebout",
    "AMOUNT_TRANSFERRED": 4287,
    "TRANSACTION_DATE": "2021-11-22",
    "PAYMENT_TYPE": "jcb",
    "DUE_DATE": "2022-11-21"
  },
  {
    "PAYMENT_ID": 600012,
    "LEASE_ID": 14123,
```

Query:

```
//gives the total amount spent using each
//payment type in ascending order
db.collection.aggregate([
  {
    "$group": {
      "_id": "$PAYMENT_TYPE",
      totalAmount: {
        "$sum": "$AMOUNT_TRANSFERRED"
      }
    }
  },
  {
    $sort: {
      _id: 1
    }
  }
])
```

Result:

```
[
  {
    "_id": "americanexpress",
    "totalAmount": 23969
  },
  {
    "_id": "bankcard",
    "totalAmount": 4241
  },
  {
    "_id": "china-unionpay",
    "totalAmount": 18105
  },
  {
    "_id": "diners-club-carte-blanche",
    "totalAmount": 1516
  },
  {
    "_id": "diners-club-enroute",
    "totalAmount": 15934
  },
  {
    "_id": "instapayment",
    "totalAmount": 7815
  },
  {
    "_id": "jcb",
    "totalAmount": 155275
  },
  {
    "_id": "laser",
```

MongoDB version 6.0.3 - Report an issue - About this playground

QUERY 3: FIND APARTMENTS LEFT WITH THE UNCLEARED DUE AMOUNT AND VACANT

QUERY 4: FIND VACANT APARTMENT OF AREA GREATER THEAN AND EQUAL TO 1500 AND RENT LESSER THAN AND EQUAL TO 3000$

QUERY 5: FIND TRANSACTION DATA OF 27TH OF DECEMBER 2021 HAVING AMOUNT TRANSFERRED GREATER THAN 1000 LESS THAN 2000

Configuration bson

```
[
  {
    "PAYMENT_ID": 600000,
    "LEASE_ID": 14170,
    "PAYEE_NAME": "Maxie",
    "AMOUNT_TRANSFERRED": 1842,
    "TRANSACTION_DATE": "2021-12-17",
    "PAYMENT_TYPE": "jcb",
    "DUE_DATE": "2022-10-21"
  },
  {
    "PAYMENT_ID": 600006,
    "LEASE_ID": 14108,
    "PAYEE_NAME": "Gilemette",
    "AMOUNT_TRANSFERRED": 2124,
    "TRANSACTION_DATE": "2021-12-21",
    "PAYMENT_TYPE": "jcb",
    "DUE_DATE": "2023-01-10"
  },
  {
    "PAYMENT_ID": 600010,
    "LEASE_ID": 13142,
    "PAYEE_NAME": "Tiebout",
    "AMOUNT_TRANSFERRED": 4287,
    "TRANSACTION_DATE": "2021-11-22",
    "PAYMENT_TYPE": "jcb",
    "DUE_DATE": "2022-11-21"
  },
  {
    "PAYMENT_ID": 600012,
    "LEASE_ID": 14123,
```

Query

```
//gives transaction data of 27th of december 2021
//having amount transferred greater than 1000 and less than 2000
db.collection.find({
  "AMOUNT_TRANSFERRED": {
    "$gt": 1000,
    "$lt": 2000
  },
  "TRANSACTION_DATE": {
    "$eq": "2021-12-27"
  }
},
{
  "_id": 0
})
```

Result

```
[
  {
    "AMOUNT_TRANSFERRED": 1358,
    "DUE_DATE": "2023-01-25",
    "LEASE_ID": 12115,
    "PAYEE_NAME": "Codee",
    "PAYMENT_ID": 600049,
    "PAYMENT_TYPE": "americanexpress",
    "TRANSACTION_DATE": "2021-12-27"
  },
  {
    "AMOUNT_TRANSFERRED": 1484,
    "DUE_DATE": "2022-11-08",
    "LEASE_ID": 10139,
    "PAYEE_NAME": "Sheridan",
    "PAYMENT_ID": 600057,
    "PAYMENT_TYPE": "jcb",
    "TRANSACTION_DATE": "2021-12-27"
  }
]
```

MongoDB version 6.0.3 - Report an issue - About this playground

V. Database Access via R

The database is accessed using R and visualization of analyzed data is shown below. The connection of MySQL to R is done using dbconnect function. ggplot2 library is used to plot the graphs for the analytics.

CONNECTION TO DATABASE

```r
con <- dbConnect(odbc::odbc(), .connection_string = "Driver={MySQL ODBC 8.0 Unicode Driver};", server="localhost",
Database = "rental_management_system", UID = "root", PWD = 'Srinivas@1', Port = 3306)
```

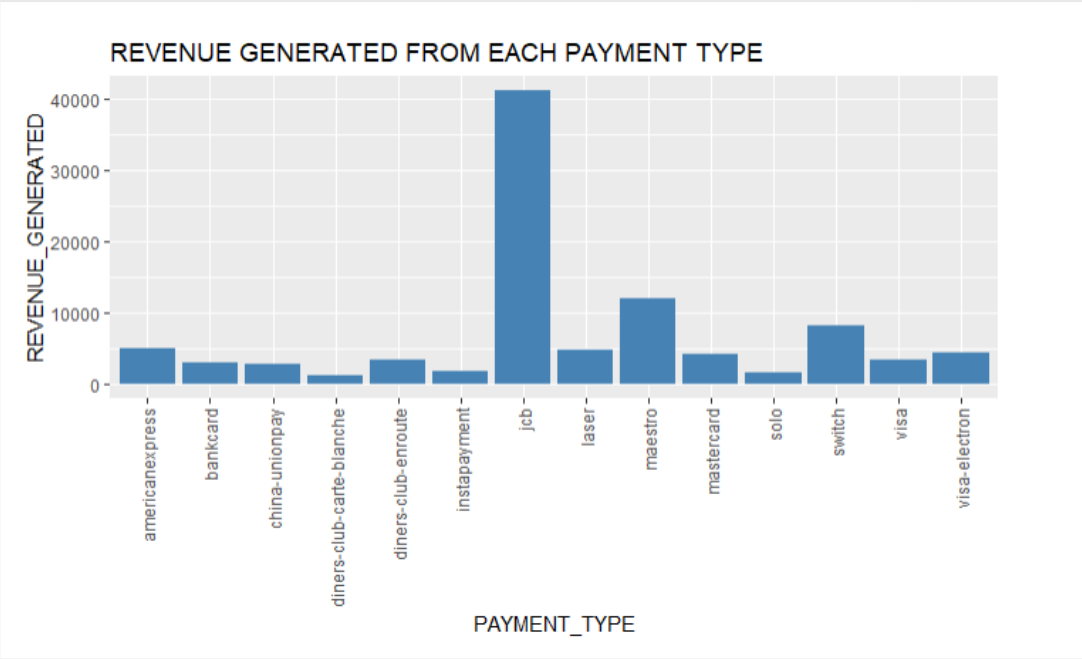
GRAPH 1: FIND THE REVENUE GENERATED BY THE RENTAL MANAGEMNET FIRM FROM EACH PAYMENT TYPE (ASSUMING 15% OF TOTAL AMOUNT TRANSFERRED BY EACH TYPE IS THE REVENUE FOR RENTAL MANAGENT FIRM)

```sql
#1 FIND THE REVENUE GENERATED BY THE RENTAL MANAGEMNET FIRM FROM EACH PAYMENT TYPE ( ASSUMING 15% OF TOTAL AMOUNT
TRANSFERRED BY EACH TYPE IS THE REVENUE FOR RENTAL MANAGENT FIRM)

   WITH REVENUE AS
            (SELECT PAYMENT_TYPE, AMOUNT_TRANSFERRED
             FROM PAYMENT)
                        SELECT PAYMENT_TYPE, SUM(AMOUNT_TRANSFERRED)*0.15 AS REVENUE_GENERATED
                        FROM REVENUE
                        GROUP BY PAYMENT_TYPE;
```

```r
ggplot(mydataframe, aes(x=PAYMENT_TYPE, y= REVENUE_GENERATED))+ geom_bar(stat="identity", fill="steelblue")+
theme(axis.text.x = element_text(angle = 90, vjust = 0.5, hjust=1)) + ggtitle("REVENUE GENERATED FROM EACH PAYMENT
TYPE")
```

OUTPUT:



GRAPH 2: FIND THE PERCENTAGE OF TECHNICIANS IN EACH DEPARTMENT

```
```{sql connection=con, output.var = "department"}

#2 Find the percentage of technicians in each department.

SELECT DEPARTMENT, COUNT(*) AS NUMBER FROM TECHNICIAN GROUP BY DEPARTMENT;


```
```

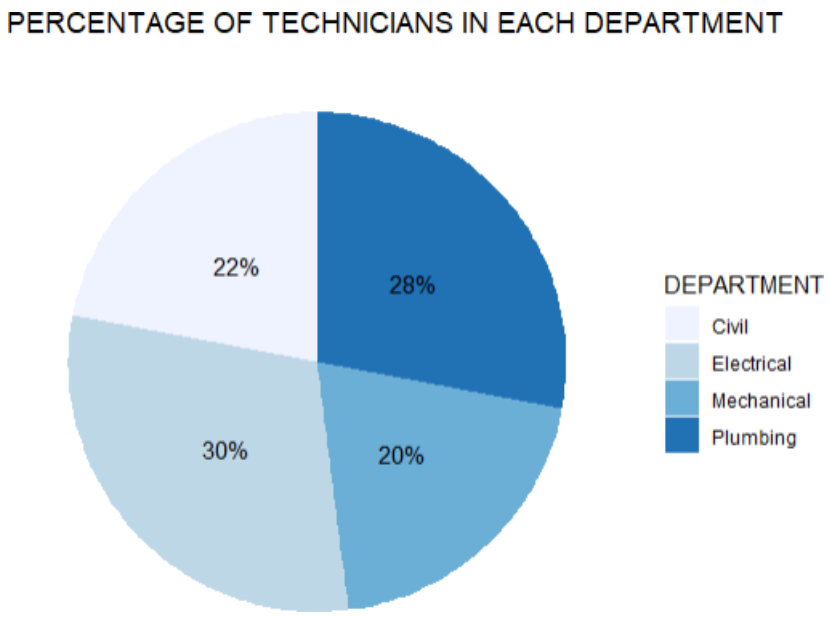
```
```{r}

department$NUMBER <- as.integer(department$NUMBER)
df <- department %>%
  group_by(NUMBER) %>%
  ungroup() %>%
  mutate(perc = `NUMBER` / sum(`NUMBER`)) %>%
  arrange(perc) %>%
  mutate(labels = scales::percent(perc))

ggplot(df, aes(x="", y= NUMBER , fill=DEPARTMENT)) +
  geom_bar(stat="identity", width=1) +
  coord_polar("y", start=0) +  geom_text(aes(label = labels), position = position_stack(vjust=0.5)) +
  labs(x = NULL, y = NULL) +
  theme_classic() +
  theme(axis.line = element_blank(),
          axis.text = element_blank(),
          axis.ticks = element_blank()) +
  scale_fill_brewer(palette="Blues")  + ggtitle("PERCENTAGE OF TECHNICIANS IN EACH DEPARTMENT")


```
```

OUTPUT:



GRAPH 3: FIND NUMBER OF MAINTENANCE REQUESTS IN PARTICULAR TIME OF THE DAY

```
```{sql connection=con, output.var = "time"}

# Find number of maintenance requests in particular time of the day

SELECT DATE_FORMAT(REQUEST_TIME, '%H') AS TIME, count(*) AS NUMBER FROM MAINTENANCE_REQUEST GROUP BY TIME ORDER BY
TIME ;



```

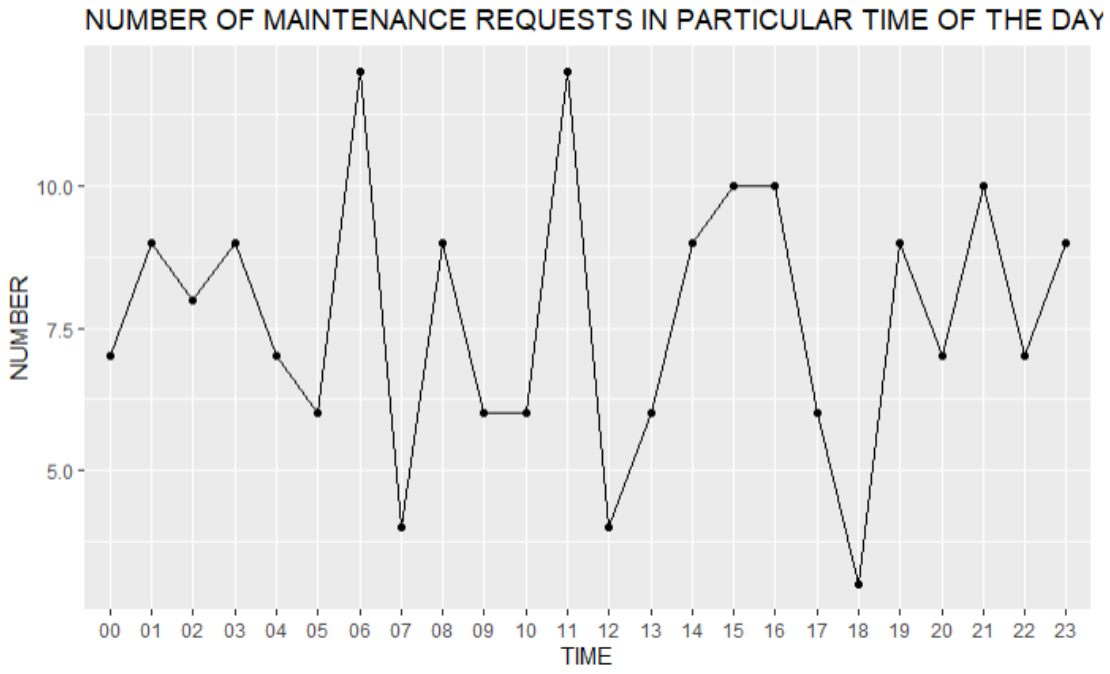
```
```{r}

time$NUMBER <- as.integer(time$NUMBER)

ggplot(time, aes(x = TIME, y = NUMBER,group=1)) + geom_line() + geom_point()  + ggtitle("NUMBER OF MAINTENANCE
REQUESTS IN PARTICULAR TIME OF THE DAY")

```
```
```

OUTPUT:


NUMBER OF MAINTENANCE REQUESTS IN PARTICULAR TIME OF THE DAY
NUMBER
10.0
7.5
5.0
00 01 02 03 04 05 06 07 08 09 10 11 12 13 14 15 16 17 18 19 20 21 22 23
TIME

VII. Summary and Recommendation

The Rental Property Management System database designed on MYSQL is an industry ready relational database that can be implemented in realty firms to ease their process to keep the track of maintenance requests, tenants and landlord information and it will result in great cost cutting and provide a better profit margin at ease and provides great analytics capabilities, a sample of which is shown in this report utilizing R Studio.

The database designed in MYSQL can also be implemented as a website by designing the front end UX wireframes for the database as this will benefit customers and landlords to do self-based search of their needs and will reduce the workload from realty firms.

Further the NoSQL implementations of this database on MONGODB, having implemented on few tables to query on it, this can certainly be used to build the database but the whole data input process benefits of using a relational database.