GROUP ASSIGNMENT:
POLYGLOT PERSISTENCE

Topic:
Nottingham Student Rental Application

Course: MSc Business Analytics
Module Code: BEMM459
Module Name: Database Technologies for Business Analytics

Group Q

Name	Student ID
Yu Hsuan Yang	730030726
Khanh Le Thi Minh	730072786
Trang Quynh Nguyen	730017134
Thanh Phuong Ngoc Pham	740015714

- Youtube link : https://youtu.be/rc1V9eO8yX0
- Github link : https://github.com/UniversityExeterBusinessSchool/bemm459-cw-2024-group_q

TABLE OF CONTENT

Outline **Page**

1. Introduction:

1.1. Problem Background:

Financial strains are a significant factor driving the rapid growth of UK house rentals, affecting both private providers and students. Rising energy prices, labor costs, building expenses, supply chain disruptions, and regulatory pressures compound challenges for rental providers.

Based on the "*Demand for Higher Education to 2035"* reported by Rachel Hewitt from HEPI[1] (Table 1), the average incremental demand for higher education reaches the

[1] *Demand for Higher Education to 2035*. (2020, October 21). HEPI. https://www.hepi.ac.uk/2020/10/22/demand-for-higher-education-to-2035/

highest number (270 per provider) in East Midlands, raising concerns about the shortage of student accommodation here.

Region	Number of higher education providers	Projected need for places per provider in 2035 (based on 28% increase)
North East	13	170
North West	48	175
Yorkshire and The Humber	32	145
East Midlands	27	270
West Midlands	34	240
East	36	220
London	111	210
South East	66	190
South West	40	145

Table 1: Places required per higher education provider, based on increased participation rate

A recent analysis from Unipol and HEPI[2] reveals a 14.6% increase of the average rent for student housing over the past two academic years. This means that rent now nearly covers the entire amount of the typical maintenance loan available to students. Consequently, students are experiencing heightened financial pressure in affording accommodation while pursuing their studies.

City	***Annual average rent 2023/24***	***Average percentage increase since 2021/22***
Bournemouth	£7,396	11.2%
Bristol	£9,200	9.0%
Cardiff	£6,632	11.1%
Exeter	£8,559	16.1%
Glasgow	£7,548	20.4%
Leeds	£7,627	14.7%
Liverpool	£6,467	6.7%
Nottingham	£8,427	15.5%
Portsmouth	£7,183	9.4%
Sheffield	£6,451	10.2%
10 cities	**£7,475**	**14.6%**

Table 2: Average rent level for 2023/24 and Average increase in 2 years in 10 UK cities

[2] *Student accommodation costs across 10 cities in the UK | HEPI. (n.d.).* https://www.hepi.ac.uk/wp-content/uploads/2023/10/Student-accommodation-costs-across-10-cities-in-the-UK.pdf

Understanding students' difficulties, universities guide them on accommodation matters to enrich their educational journey and support academic achievement by connecting them with the "Nottingham Student Rentals," a real estate company, specializes in student accommodation rental services, aiming to expand accommodation options (by integrating institution and private providers), address housing concerns and commit to enhance student well-being.

1.2. Business Functions:

Not only standing out with the highest absolute annual rent (£8,427) and average percentage increase (15.5%), but Nottingham also emerges as an attractive option with a balance between educational opportunities and manageable living costs amid housing market pressures. With the East Midlands region expected to experience a surge in demand for student accommodation, providing housing advisory services in Nottingham is becoming essential. This highlights the potential to assist Nottingham students in navigating the competitive and scarce housing market.

Currently, Nottingham hosts two main universities: the University of Nottingham (with 6 campuses) and Nottingham Trent University (with 4 campuses). Initially, this project focuses on one campus of Nottingham Trent University, with plans to expand in the future.

Overall, our business functions can be described as follows:

- **Property Management**: Managing rental properties, including listings, maintenance, and inspections.
- **Tenant Management**: Handling tenant applications, leases, rent collection, and maintenance requests.
- **Customer Service**: Providing support and assistance to tenants throughout their rental period.
- **Booking System:** storing booking details, room availability, and contracts. This allows for efficient management of property bookings and facilitates the online booking process for prospective tenants.
- **Analytics and Insights**: Analyzing market trends, occupancy rates, and rental performance.

1.3. Use Case Scenario:

Taking advantage of the peak season for students to find accommodation for their upcoming academic years, the Nottingham Student Rentals platform not only assists students in applying for housing, but also maintains the habitable living conditions by concentrating on solving maintenance issues and welcome all feedback throughout their

stay. This extensive interaction data undergoes thorough analysis to glean marketing insights, refine property listings, and enrich user experiences by leveraging historical preferences and feedback trends. Here are our conducted steps to ensure a tailored experience for each student:

- We gather vacant rooms from both university accommodation teams and private property owners.
- Students browse available properties on the Nottingham Student Rentals website, filtering listings by criteria like contract length, room type, and rental price.
- After finding a suitable property, they submit an online application form with personal details, selecting up to 3 preferred options and input prioritizing numbers.
- The property agents review applications, check the room quality, working with landlords, and approve the tenancy agreement.
- The students sign the E-contract, pay the security deposit and first month's rent online, and receive booking confirmation.
- Throughout their stay, students can conveniently report maintenance issues through our website. Upon moving out, they are encouraged to provide feedback on their accommodation experience.

1.4. Choice of Databases:

In Nottingham Student Rentals' application, integrating both RDBMS and NoSQL databases can offer significant advantages.

- *RDBMS (e.g., MySQL):* Ideal for managing structured data such as property listings, tenant information, lease agreements, and financial transactions.

  – Assumptions: Rental properties and tenant records require structured storage for efficient querying and management.

- *NoSQL database (e.g., MongoDB):* Useful for storing unstructured or semi-structured data such as maintenance logs, feedback from tenants, and property photos.

  – Assumptions: Unstructured data like tenant feedback, property photos, and maintenance requests should be stored and analyzed for improving services and property management.

2. Relational Database Management System (RDBMS):

In our case, RDBMS proves invaluable for efficiently managing related data. It enables the storage of student information, residence details, lease contracts, landlord information and rental agents in structured tables, facilitating easy retrieval and manipulation. RDBMS ensures data integrity, supports complicated queries, and streamlines tasks such as residence management and student option tracking for “Nottingham Student Rentals”. By maintaining consistency and enabling quick response to issues, utilizing RDBMS, we can gain a thorough understanding of student needs and preferences, therefore, give our users helpful advice and optimize difficulties in choosing an accommodation for them.

RDBMS facilitates the implementation of relationships between entities through multiplicity constraints, including one-one, one-many, many-many relationships based on the use case as described in the Entity-Relationship (ER) diagram below:

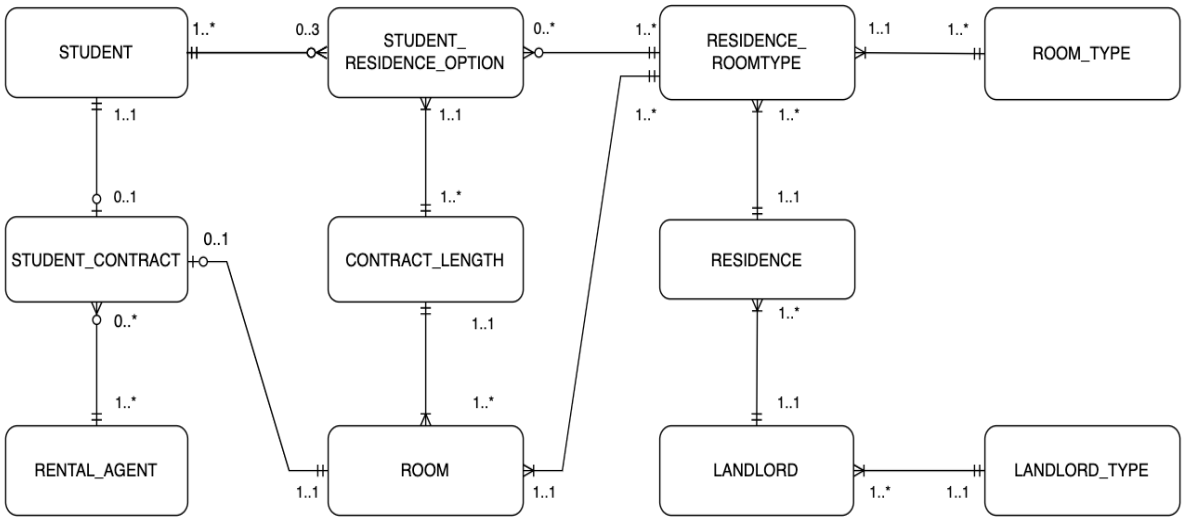


Table 3: The Entity-Relationship (ER) diagram with the entities and relationship of accommodations in Nottingham

This diagram depicts the entities and relationships associated with accommodations in Nottingham. This outlines the structural framework of the data model, illustrating how different entities interact within the accommodation domain.

Entity	Attribute Name	Description	Domain Attribute or/and Datatype	Null	Multi Values
Student	**student_id**	**Uniquely identifies for the student**	**Number**	**no**	**no**
	student_first_name	First name of the student	Text (30 characters)	no	no
	student_last_name	Last name of the student	Text (30 characters)	no	no
	student_number	University Student ID	7-digit number	no	no
	education_level	Level of education of the student	Value can be "Presession", "Undergraduate" or "Postgraduate"	no	no
	personal_email	Personal email address of the student	Text (30 characters) and must include @	no	no
	school_email	School email address of the student	Text (30 characters) and must include @	no	no
	student_gender	Gender of the student	Value can be either "Male", "Female" or "Other"	no	no
	student_DOB	Date of birth of the student	Date format	no	no
	student_nationality	Nationality of the student	Text (30 characters)	no	no
Student Residence Option	**residence_option_id**	**Uniquely identifies for the student residence**	**Number**	**no**	**no**
	residence_priority	Priority of the residence option	Value can be either 1,2 or 3	no	no
	date_created	Timestamp of when the student's options were created	Datetime format	no	no
Landlord	**landlord_id**	**Uniquely identifies for the landlord**	**Number**	**no**	**no**
	landlord_first_name	First name of the landlord	Text (30 characters)	no	no
	landlord_last_name	Last name of the landlord	Text (30 characters)	no	no
	landlord_gender	Gender of the landlord	Value can be either "Male", "Female" or "Other"	yes	no
	landlord_email	Email address of the landlord	Text (30 characters) and must include @	no	no
	landlord_phone_number	Phone number of the landlord	Maximum of 12 characters	**no**	**no**
Landlord Type	**landlord_type_id**	**Uniquely identifies for the landlord type**	**Number**	**no**	**no**
	landlord_type_name	Name of the landlord type	Values are "University accommodation" and "Private properties"	no	no
Residence_Room_Type	**residence_roomtype_id**	**Uniquely identifies for each association betw**	**Number**	no	no
	fee	Fee per week of each room type of each reside	Number	no	no
Residence	**residence_id**	**Uniquely identifies for the residence**	**Number**	**no**	**no**
	is_catered	Indicates if the residence provides catering services	Value can be either True (have) or False (do not have)	no	no
	residence_name	Name of the residence	Text (50 characters)	no	no
	total_rooms_available	Total number of rooms available in the	Number	**no**	**no**
	is_central_social_space	Indicates if the residence has a central social space	Value can be either True (have) or False (do not have)	no	no
	is_cleaning_provision	Indicates if cleaning services are provided	Value can be either True (have) or False (do not have)	no	no
	residence_postcode	Postcode of the residence	Maximum of 6 characters	**no**	**no**
Room	**room_id**	**Uniquely identifies for the room**	**Number**	**no**	**no**
	room_number	Room number	Number	no	no
Room_Type	**room_type_id**	**Uniquely identifies for the room type**	**Number**	**no**	**no**
	room_type	Type of the room	Values are "Standard", "En-suite" and "Studio"	no	no
Contract_Length	**contract_length_id**	**Uniquely identifies for the contract length**	**Number**	**no**	**no**
	contract_length	Length of the rental contract for the room	Values are "40", "44" and "51"	no	no
Student_Contract	**student_contract_id**	**Uniquely identifies for the student contract**	**Number**	**no**	**no**
	offer_paid_on	Timestamp of when the offer was paid for	Datetime format	no	no
	arrival_date	Timestamp of student's arrival	Datetime format	no	no
	departure_date	Timestamp of student's departure	Datetime format	no	no
Rental_Agent	**rental_agent_id**	**Uniquely identifies for the rental agent**	**Number**	**no**	**no**
	rental_agent_first_name	First name of the rental agent	Text (30 characters)	no	no
	rental_agent_last_name	Last name of the rental agent	Text (30 characters)	no	no
	rental_agent_gender	Gender of the rental agent	Value can be either "Male", "Female" or "Other"	no	no
	rental_agent_email	Email address of the rental agent	Text (30 characters) and must include @	no	no
	rental_agent_phone_number	Phone number of the rental agent	Maximum of 12 characters	no	no

Table 4: The Dictionary of the ERD

Meanwhile, Table 4 complements the ERD by providing a comprehensive dictionary elucidating the entity attributes and primary keys, further enhancing our understanding of the data schema. Together, these tables offer a detailed insight into the relational structure and attributes of accommodations in Nottingham the process of database generation, facilitating efficient data management and analysis within this domain.

DDL (Data Definition Language) codes in SQL are used to define and modify the structure of database objects:

- Create Schema

```
CREATE SCHEMA ACCOMMODATION;

GO
```

- Create a table

```
CREATE TABLE ACCOMMODATION.Landlord (
  landlord_id INT PRIMARY KEY NOT NULL,
  landlord_type_id INT NOT NULL,
  landlord_first_name VARCHAR(255) NOT NULL,
  landlord_last_name VARCHAR(255) NOT NULL,
  landlord_gender VARCHAR(255) NULL,
  landlord_email VARCHAR(255) NOT NULL,
  landlord_phone_number VARCHAR(255) NOT NULL,
  FOREIGN KEY (landlord_type_id) REFERENCES ACCOMMODATION.Landlord_Type(landlord_type_id)
);
```

- Add a FOREIGN KEY constraint to the table using ALTER TABLE command

```
ALTER TABLE ACCOMMODATION.Landlord
ADD CONSTRAINT FK_Landlord_Landlord_Type FOREIGN KEY (landlord_type_id) REFERENCES ACCOMMODATION.Landlord_Type(landlord_type_id);
```

By using Python to perform CRUD operations into SQL databases, programmers can flexibly and easily create, read, update, and delete data. Python provides libraries such as PyODBC - a Python library used to connect to databases via ODBC (Open Database Connectivity) that help interact with SQL databases in a convenient and flexible way, helping to increase performance and reduce errors during application development. The data was generated by Mockaroo.

For CRUD operations using Python, here are the steps we write on Jupyter Notebook:

- Import pyodbc and connect to SQL Server

```
import pyodbc

server = 'tcp:mcruebs04.isad.isadroot.ex.ac.uk'
database = 'BEMM459_GroupQ'
username = 'GroupQ'
password = 'XcwH643*Dq'

# Establish a connection to the SQL Server
cnxn = pyodbc.connect('DRIVER={/usr/local/lib/libmsodbcsql.18.dylib};SERVER='+server+';DATABASE='+database+'; UID='+username+';PWD='+ password+';TrustServerCertificate=yes;Encrypt=no;')

# Create a cursor object to execute SQL queries
cursor = cnxn.cursor()
```

- READ the data and print out the table

```
# Select data from all tables in ACCOMMODATION
select_query = '''
SELECT * FROM ACCOMMODATION.Landlord_Type
SELECT * FROM ACCOMMODATION.Landlord
SELECT * FROM ACCOMMODATION.Residence
SELECT * FROM ACCOMMODATION.Student
SELECT * FROM ACCOMMODATION.Student_Contract
SELECT * FROM ACCOMMODATION.Room
SELECT * FROM ACCOMMODATION.Rental_Agent
SELECT * FROM ACCOMMODATION.Residence_RoomType
SELECT * FROM ACCOMMODATION.Student_Residence_Option
SELECT * FROM ACCOMMODATION.Room_Type
SELECT * FROM ACCOMMODATION.ContractLength
'''

# Make the SQL run on the server
cursor.execute(select_query)

# Print the selected data
for row in cursor:
    print(row)
```

- INSERT data into the table and execute the SQL query

```
# Insert data into SQL tables
createTsql = '''

INSERT INTO ACCOMMODATION.Landlord_Type (landlord_type_id, landlord_type_name)
VALUES
    (1, 'Private properties'),
    (2, 'University accommodation');

INSERT INTO ACCOMMODATION.Landlord (landlord_id, landlord_type_id, landlord_first_name, landlord_last_name, landlord_gender, landlord_email, landlord_phone_number)
VALUES
    (1, 1, 'Levi', 'Sheen', 'Male', 'lsheen0@gmail.com', '07415123456'),
    (2, 1, 'Shem', 'Crecy', 'Male', 'screcy1@gmail.com', '07720987654'),
    (3, 1, 'Tiler', 'Mangham', 'Male', 'tmangham2@gmail.com', '07903456789'),
    (4, 1, 'Barnabas', 'Grassot', 'Male', 'bgrassot3@gmail.com', '07856234567'),
    (5, 2, 'Nottingham Trent', 'University', '', 'admin@ntu.ac.uk', '07540876543'),
    (6, 1, 'Maurizia', 'Cathersides', 'Female', 'mcathersides5@gmail.com', '07624345678'),
    (7, 1, 'Worden', 'Poyzer', 'Male', 'wpoyzer6@gmail.com', '07891654321'),
    (8, 1, 'Zacharia', 'Mustill', 'Male', 'zmustill7@gmail.com', '07980123456'),
    (9, 1, 'Donia', 'Deniscke', 'Female', 'ddeniscke8@gmail.com', '07739987654'),
    (10, 1, 'Van', 'Hand', 'Male', 'vhand9@gmail.com', '07460234567');

'''

# Execute the SQL query
cursor.execute(createTsql)


# Use commit to get it to finish
cnxn.commit()
```

- SELECT the data to see the updated table using pandas library and creating dataframe

```
import pandas as pd

# Establish a connection to database

conn = pyodbc.connect('DRIVER={/usr/local/lib/libmsodbcsql.18.dylib};SERVER='+server+';DATABASE='+database+';UID='+username+';PWD='+ password+';TrustServerCertificate=yes;Encrypt=no;')

# Set the display.max_rows option to None
pd.set_option('display.max_rows', None)

# Create DataFrame from SQL tables
Student_Contract_df = pd.read_sql_query("SELECT * FROM ACCOMMODATION.Student_Contract", conn)

# Print information for each table"
print("Student_Contract:")
print(Student_Contract_df)

# Close the connection
conn.close()
```

```
Student_Contract:
    student_contract_id  rental_agent_id  student_id  room_id  \
0                     1                1           1        1
1                     2                1           3        8
2                     3                2           9       13
3                     4                2          14       23
4                     5                2          20       38
5                     6                4          28       21
6                     7                4          30        7
7                     8                5          34       25
8                     9                8          34        2
9                    10               10          55        5
10                   11               10          60       33
11                   12               11          61        6
12                   13               11          64       10
13                   14               11          73       47
14                   15               12          79       17

          offer_paid_on arrival_date departure_date
0  2024-02-25 11:20:00   2024-09-27     2024-10-18
1  2024-02-25 16:35:00   2024-09-29     2024-10-18
2  2024-02-25 21:50:00   2024-09-25     2024-10-18
3  2024-02-26 03:05:00   2024-09-30     2024-10-18
4  2024-02-26 08:20:00   2024-09-26     2024-10-25
5  2024-02-26 18:50:00   2024-09-28     2024-10-18
...
11 2024-02-27 21:20:00   2024-09-27     2024-10-18
12 2024-02-28 02:35:00   2024-09-29     2024-10-25
13 2024-03-17 23:20:00   2024-09-28     2024-10-25
14 2024-03-24 15:50:00   2024-09-30     2024-10-18
```

- Following our use case, one student must have only one contract. From the result table above, we can see that there are two contracts from one student with student_id = 34, which conflicts with our use case. To double check if there is any duplicate (no student with more than one contract), we use CRUD to delete based on the latest offer_paid_on.

```
testsql='''
-- Delete duplicate student contract (choose the one with the latest offer_paid_on)
WITH LatestOffers AS (
  SELECT student_id, MAX(offer_paid_on) AS LatestOfferDate
  FROM ACCOMMODATION.Student_Contract
  GROUP BY student_id
)
DELETE T
FROM (
  SELECT *
  , DupRank = ROW_NUMBER() OVER (
    PARTITION BY student_id
    ORDER BY offer_paid_on DESC
  )
  FROM ACCOMMODATION.Student_Contract
) AS T
INNER JOIN LatestOffers ON T.student_id = LatestOffers.student_id
WHERE DupRank > 1
AND T.offer_paid_on < LatestOffers.LatestOfferDate;
'''

# Make the SQL run on the server
cursor.execute(testsql)

# Use commit to get it to finish
cnxn.commit()
```

```
import pandas as pd

# Establish a connection to database

conn = pyodbc.connect('DRIVER={/usr/local/lib/libmsodbcsql.18.dylib};SERVER='+server+';DATABASE='+database+';UID='+username+';PWD='+ password+';TrustServerCertificate=yes;Encrypt=no;')

# Set the display.max_rows option to None
pd.set_option('display.max_rows', None)

# Create DataFrame from SQL tables
Student_Contract_df = pd.read_sql_query("SELECT * FROM ACCOMMODATION.Student_Contract", conn)

# Print information for each table"
print("Student_Contract:")
print(Student_Contract_df)

# Close the connection
conn.close()
```

In this updated Student_Contract table, there no longer exists the duplicate contract.

```
Student_Contract:
    student_contract_id  rental_agent_id  student_id  room_id  \
0                     1                1           1        1
1                     2                1           3        8
2                     3                2           9       13
3                     4                2          14       23
4                     5                2          20       38
5                     6                4          28       21
6                     7                4          30        7
7                     8                5          34       25
8                    10               10          55        5
9                    11               10          60       33
10                   12               11          61        6
11                   13               11          64       10
12                   14               11          73       47
13                   15               12          79       17

         offer_paid_on arrival_date departure_date
0  2024-02-25 11:20:00   2024-09-27     2024-10-18
1  2024-02-25 16:35:00   2024-09-29     2024-10-18
2  2024-02-25 21:50:00   2024-09-25     2024-10-18
3  2024-02-26 03:05:00   2024-09-30     2024-10-18
4  2024-02-26 08:20:00   2024-09-26     2024-10-25
5  2024-02-26 18:50:00   2024-09-28     2024-10-18
6  2024-02-27 00:05:00   2024-09-30     2024-10-18
...
10 2024-02-27 21:20:00   2024-09-27     2024-10-18
11 2024-02-28 02:35:00   2024-09-29     2024-10-25
12 2024-03-17 23:20:00   2024-09-28     2024-10-25
13 2024-03-24 15:50:00   2024-09-30     2024-10-18
```

- Due to the false fee for the room type id =6 from the landlord, we use UPDATE command to change the fee from 250 to 240.

```
update='''
-- Update fee for residence_roomtype_id = 6
UPDATE ACCOMMODATION.Residence_RoomType
SET fee = 240
WHERE residence_roomtype_id = 6;
'''
# Make the SQL run on the server
cursor.execute(update)

# Use commit to get it to finish
cnxn.commit()
```

```
# Establish a connection to database
conn = pyodbc.connect('DRIVER={/usr/local/lib/libmsodbcsql.18.dylib};SERVER='+server+';DATABASE='+database+';UID='+username+';PWD='+ password+';TrustServerCertificate=yes;Encrypt=no;')

# Set the display.max_rows option to None
pd.set_option('display.max_rows', None)

# Create DataFrame from SQL tables
Residence_RoomType_df = pd.read_sql_query("SELECT * FROM ACCOMMODATION.Residence_RoomType", conn)

# Print information for each table"
print("Residence_RoomType:")
print(Residence_RoomType_df)

# Close the connection
conn.close()
```

```
Residence_RoomType:
    residence_roomtype_id  room_type_id  residence_id  fee
0                       1             1             1  120
1                       2             2             1  150
2                       3             2             2  200
3                       4             3             2  300
4                       5             1             3  150
5                       6             2             3  240
6                       7             3             3  350
7                       8             1             4  120
8                       9             3             5  200
9                      10             1             6  150
10                     11             2             6  180
11                     12             3             6  260
12                     13             1             7  140
13                     14             2             7  180
14                     15             2             8  200
15                     16             1             9  150
16                     17             3             9  300
17                     18             3            10  300
```

In essence, CRUD is considered to be data-oriented functions for retrieval and manipulation, especially in a large dataset.

3. NoSQL database:

3.1. Why we choose MongoDB:

- **Schema Flexibility:** MongoDB's schema-less design allows to easily store and manage unstructured and semi-structured data, accommodating diverse data types encountered in rental management, such as tenant feedback, property media, maintenance requests, and historical search data.
- **Scalability**: MongoDB's horizontal scaling capability ensures efficient handling of increasing data volumes and user requests. Scalability is truly crucial for maintaining stability during peak usage or portfolio expansion.
- **Efficient Querying**: MongoDB's powerful querying features (ad-hoc queries, indexing, and aggregation pipelines) enable quick retrieval and analysis of rental-related data. This supports the seamless operation and decision-making on our rental websites.
- **High Availability and Fault Tolerance**: MongoDB's built-in features (replica sets and automatic recovery) ensure an uninterrupted system operation. Data replication across servers can minimize downtime, critical for landlords, tenants, and rental agents to access services continuously.
- **Document-Oriented Data Model**: The MongoDB's document-oriented approach strongly matches with rental management data, often represented as JSON-like documents for properties managing, tenants' feedback, and more.

3.2. How we create database

At Nottingham Student Rentals, we harness the power of NoSQL databases to streamline the management of student accommodation rentals across four crucial aspects as below.

3.2.1. Tenant Feedback Management

Tenants' feedback plays a pivotal role in refining our services and serving as references for prospective tenants. Our NoSQL database – MongoDB flexible set up lets us store different types of data, capturing comprehensive feedback, including textual reviews, ratings, and multimedia content.

```
{
  _id: 15,
  student_contract_id: 2,
  feedback_text: 'Convenient location, good wifi connection but hygiene is a problem.',
  rating:
      {
              location: 4,
              cleaning: 2,
              wifi: 5,
              social_spaces: 3,
              property_management: 5
      },
  media_type: 'image',
  media_data: 'https://dummyimage.com/202x194',
  timestamp: '2024-02-19T10:57:48'
}
```

3.2.2. Property Media Management

Media content is crucial for both property owners and tenants because it assists the owners in not only setting realistic expectation for students, but also saving time and resources for prospective tenants to visit, together with facilitating remote decision-making, especially in international students. By leveraging MongoDB technology, we are enabled to approach the scalable storage and flexible management of multimedia assets associated with rental properties, including photos, videos, and 360-degree virtual tours, facilitating the ability of tracking and accessing information pertaining to a particular property.

```
{
  _id: 9,
  residence_id: 5,
  media_type: 'video',
  media_data: 'http://www.howard-webb.com/main/explorelogin.php.mp4',
  description: 'An apartment with a stunning view from large balcony.',
  timestamp: '2024-02-29T14:18:09'
}
```

3.2.3. Maintenance Request Tracking

Our online reporting system swiftly resolves maintenance issues in rental properties to ensure tenant satisfaction and manage risk. Each document in MongoDB contains key details such as request ID and contract ID, synced with student and residence IDs, alongside additional information like maintenance tasks and media data. MongoDB's indexing feature truly facilitates quick retrieval of records and efficiently handles growing data volumes, maintaining system efficiency, so that a large amount of maintenance records is quickly stored and retrieved, enhancing the managing experiences.

```
{
  _id: 189,
  student_contract_id: 12,
  maintenance_task: 'Refinish scratched tabletop',
  timestamp: '2024-03-09T18:23:06',
  status: 'pending',
  media_type: 'image',
  media_data: 'https://placekitten.com/520/801'
}
```

3.2.4. Historical Search Data Analysis for Marketing Insights

Storing extensive historical search data enables us to conduct statistical analysis and integrate crucial marketing insights, optimizing website and listing content. Utilizing MongoDB, key fields like search logs, user behavior data, engagement metrics, demographics, timestamps, and session information are efficiently indexed and stored. MongoDB's aggregation features facilitate running complex queries for valuable historical insights, complemented by horizontal scaling via its sharding capabilities.

```
{
    _id: 244,
    user_id: '75652de7-71a1-4486-af51-8ead92a33095',
    search_query: 'studio 51 weeks Sherwood Lodge',
    timestamp: '2024-03-12T17:45:58',
    search_results:
        [
                {
                        room_type: 'studio',
                        contract_length: '40 weeks',
                        residence_name: ‘Sherwood Lodge'
                },
                {
                        room_type: 'studio',
                        contract_length: '51 weeks',
                        residence_name: 'Nottingham Student Haven'
                }
        ],
    engagement_metrics:
        {
                clicks: 12,
                time_on_page_min: 286,
                scroll_depth: 0.37248941295923643
        },
    user_demographics:
        {
                age: 61,
                gender: 'Other',
                location: 'Mauritius'
        },
    session_information:
        {
                device: 'Tablet',
                browser: 'Opera/9.28.(Windows NT 10.0; tn-ZA) Version/11.0'
        }
}
```

3.3. Coding logic and CRUD operations:

Here are our steps executing on Jupyter Notebook file:

- **Connectivity / Install package**

– Install pymongo package.
– Set the correct MongoDB port.
– Install Faker library to generate dummy records.

- **Create**

– Create a new database: “groupQ_BEMM459_coursework”.

```
In [4]: #Create Group Q database
        mongoclient = pymongo.MongoClient("mongodb://localhost:7000/")
        mydb = mongoclient["groupQ_BEMM459_coursework"]
```

– Create 4 collections for the 4 aspects: “Tenant_Feedback”, “Property_Media, “Maintenance_Request, “Historical_Search”.

```
In [6]: #Create a new collection named Tenant_Feedback:
        tenant_feedback_col = mydb["Tenant_Feedback"]
        print(type(tenant_feedback_col))
```

```
<class 'pymongo.collection.Collection'>
```

– Use faker library to generate dummy data as an array for each collection. In each array, documents/embedded documents are constructed based on the purpose of each aspect.
– Define functions for CRUD operations.
– Generate documents and insert them into collections.

```
In [12]: # Insert the records into the collection, as documents in MongoDB
         doc1 = tenant_feedback_col.insert_many(tenant_feedback)

         # Print list of the _id values of the inserted doc1:
         print(doc1.inserted_ids)
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16]
```

- **Read**

– Retrieve all documents from the collection but only show specific fields.

In [13]:
```
# Query all documents but retrieve only fields of feedback_text and rating for location (projection):
myquery = tenant_feedback_col.find({},{'_id':0,'student_contract_id':1,'rating.location':1})
for x in myquery:
    print(x)
```

```
{'student_contract_id': 7, 'rating': {'location': 2}}
{'student_contract_id': 3, 'rating': {'location': 3}}
{'student_contract_id': 4, 'rating': {'location': 1}}
{'student_contract_id': 15, 'rating': {'location': 3}}
{'student_contract_id': 14, 'rating': {'location': 2}}
{'student_contract_id': 11, 'rating': {'location': 2}}
{'student_contract_id': 5, 'rating': {'location': 1}}
{'student_contract_id': 9, 'rating': {'location': 4}}
{'student_contract_id': 13, 'rating': {'location': 2}}
{'student_contract_id': 16, 'rating': {'location': 1}}
{'student_contract_id': 10, 'rating': {'location': 5}}
{'student_contract_id': 2, 'rating': {'location': 1}}
{'student_contract_id': 12, 'rating': {'location': 2}}
{'student_contract_id': 6, 'rating': {'location': 4}}
{'student_contract_id': 8, 'rating': {'location': 2}}
{'student_contract_id': 1, 'rating': {'location': 2}}
```

– Limit the result to a specific number of documents, using limit() method.

In [56]:
```
#Limit the result to only return 3 documents and print results:
for x in tenant_feedback_col.find().limit(3):
    print(x)
```

```
{'_id': 1, 'student_contract_id': 7, 'feedback_text': 'Reach wear sort play impact direction. Assume
home amount argue.', 'rating': {'location': 2, 'cleaning': 3, 'wifi': 5, 'social_spaces': 2, 'propert
y_management': 5}, 'media_type': 'image', 'media_data': 'https://dummyimage.com/363x471', 'timestam
p': '2024-02-12T03:39:43'}
{'_id': 2, 'student_contract_id': 3, 'feedback_text': 'Agree career star price artist value. Movement
myself baby career.', 'rating': {'location': 3, 'cleaning': 5, 'wifi': 3, 'social_spaces': 2, 'proper
ty_management': 2}, 'media_type': 'image', 'media_data': 'https://s3.eu-west-2.amazonaws.com/assets.c
rm-students.com/2018/10/Russell-View-Nottingham-Large-Studio.jpg', 'timestamp': '2024-03-05T20:05:2
0'}
{'_id': 3, 'student_contract_id': 4, 'feedback_text': 'Moment as win check draw day a red. Eight redu
ce himself head financial reality manager. And common ready network public.', 'rating': {'location':
1, 'cleaning': 3, 'wifi': 2, 'social_spaces': 5, 'property_management': 5}, 'media_type': 'image', 'm
edia_data': 'https://dummyimage.com/363x471', 'timestamp': '2024-03-17T04:23:01'}
```

– Show the first occurrence in the selection, using find_one() method.

In [15]:
```
#Show the first occurrence in the selection.
print(tenant_feedback_col.find_one())
```

```
{'_id': 1, 'student_contract_id': 7, 'feedback_text': 'Reach wear sort play impact direction. Assume
home amount argue.', 'rating': {'location': 2, 'cleaning': 3, 'wifi': 5, 'social_spaces': 2, 'propert
y_management': 5}, 'media_type': 'image', 'media_data': 'https://dummyimage.com/363x471', 'timestam
p': '2024-02-12T03:39:43'}
```

– Filter documents and limit the query results.

In [62]:
```
#Filter Results: media_type = image and rating for location = 5:
myquery = tenant_feedback_col.find({ 'media_type': 'image', 'rating.location': 5 })

for x in myquery:
    print(x)
```

```
{'_id': 11, 'student_contract_id': 10, 'feedback_text': 'Morning a both citizen price. Agreement nort
h door act.', 'rating': {'location': 5, 'cleaning': 2, 'wifi': 3, 'social_spaces': 5, 'property_manag
ement': 5}, 'media_type': 'image', 'media_data': 'https://dummyimage.com/363x471', 'timestamp': '2024
-03-02T17:43:00'}
```

– Sorting documents by using sort() method.

In [42]:
```
#Sorting documents by rating of wifi, from highest to lowest number, filter results :

for x in tenant_feedback_col.find({},{'student_contract_id':1,'rating.wifi':1}).sort('rating.wifi', -1):
    print() #each element is printed on a new line afterward.
    print(x)
```

```
{'_id': 2, 'student_contract_id': 2, 'rating': {'wifi': 5}}

{'_id': 7, 'student_contract_id': 6, 'rating': {'wifi': 5}}

{'_id': 11, 'student_contract_id': 14, 'rating': {'wifi': 5}}

{'_id': 12, 'student_contract_id': 10, 'rating': {'wifi': 5}}

{'_id': 14, 'student_contract_id': 9, 'rating': {'wifi': 5}}

{'_id': 3, 'student_contract_id': 8, 'rating': {'wifi': 4}}

{'_id': 6, 'student_contract_id': 12, 'rating': {'wifi': 4}}

{'_id': 8, 'student_contract_id': 5, 'rating': {'wifi': 4}}

{'_id': 1, 'student_contract_id': 11, 'rating': {'wifi': 3}}

{'_id': 4, 'student_contract_id': 15, 'rating': {'wifi': 3}}

{'_id': 15, 'student_contract_id': 16, 'rating': {'wifi': 2}}
```

- **Update**

– Update one documents.

In [63]:
```
#Update a record, or document.
#Update an image link in the feedback of student_contract_id:3
myquery = {'student_contract_id':3}
newvalues = {"$set": {'media_data':'https://s3.eu-west-2.amazonaws.com/Russell-View-Nottingham-Large-Studio.jpg'}}

tenant_feedback_col.update_one(myquery, newvalues)

#print only updated document after updating:
for x in tenant_feedback_col.find({'student_contract_id':3},{'_id':0,'student_contract_id':1,'media_data':1}):
    print(x)
```

```
{'student_contract_id': 3, 'media_data': 'https://s3.eu-west-2.amazonaws.com/assets.crm-students.com/2018/10/Russell-View-Notti
ngham-Large-Studio.jpg'}
```

– Update multiple documents.

In [66]:
```
#Update many documents:
#changing all value '1' in rating.wifi into '1: (need to contact the tenant)':

myquery = {'rating.wifi': 1}
newvalues = {'$set': {'rating.wifi': '1: (need to contact the tenant)'}}

#Print all documents, retrieve only '_id' and 'rating.wifi':
for i in tenant_feedback_col.find({},{'_id':1,'rating.wifi':1}):
    print(i)
```

```
{'_id': 1, 'rating': {'wifi': 5}}
{'_id': 2, 'rating': {'wifi': 3}}
{'_id': 3, 'rating': {'wifi': 2}}
{'_id': 4, 'rating': {'wifi': 5}}
{'_id': 5, 'rating': {'wifi': 4}}
{'_id': 6, 'rating': {'wifi': '1: (need to contact the tenant)'}}
{'_id': 7, 'rating': {'wifi': 4}}
{'_id': 8, 'rating': {'wifi': '1: (need to contact the tenant)'}}
{'_id': 9, 'rating': {'wifi': 3}}
{'_id': 11, 'rating': {'wifi': 3}}
{'_id': 12, 'rating': {'wifi': 3}}
{'_id': 13, 'rating': {'wifi': '1: (need to contact the tenant)'}}
{'_id': 14, 'rating': {'wifi': 5}}
{'_id': 15, 'rating': {'wifi': 2}}
{'_id': 16, 'rating': {'wifi': 3}}
```

- **Delete**

– Delete one document.

In [70]:
```
#Deleting the redundant record which is student_contract_id:16
tenant_feedback_col.delete_one({'student_contract_id':16})

# Query all documents:
for x in tenant_feedback_col.find({},{'student_contract_id':1}):
    print(x)
```

```
{'_id': 1, 'student_contract_id': 7}
{'_id': 2, 'student_contract_id': 3}
{'_id': 3, 'student_contract_id': 4}
{'_id': 4, 'student_contract_id': 15}
{'_id': 5, 'student_contract_id': 14}
{'_id': 6, 'student_contract_id': 11}
{'_id': 7, 'student_contract_id': 5}
{'_id': 8, 'student_contract_id': 9}
{'_id': 9, 'student_contract_id': 13}
{'_id': 11, 'student_contract_id': 10}
{'_id': 12, 'student_contract_id': 2}
{'_id': 13, 'student_contract_id': 12}
{'_id': 14, 'student_contract_id': 6}
{'_id': 15, 'student_contract_id': 8}
{'_id': 16, 'student_contract_id': 1}
```

– Delete all documents.

In [ ]:
```
#Deleting all documents in the collection:
x = tenant_feedback_col.delete_many({})

print(x.deleted_count, " documents deleted.")
```

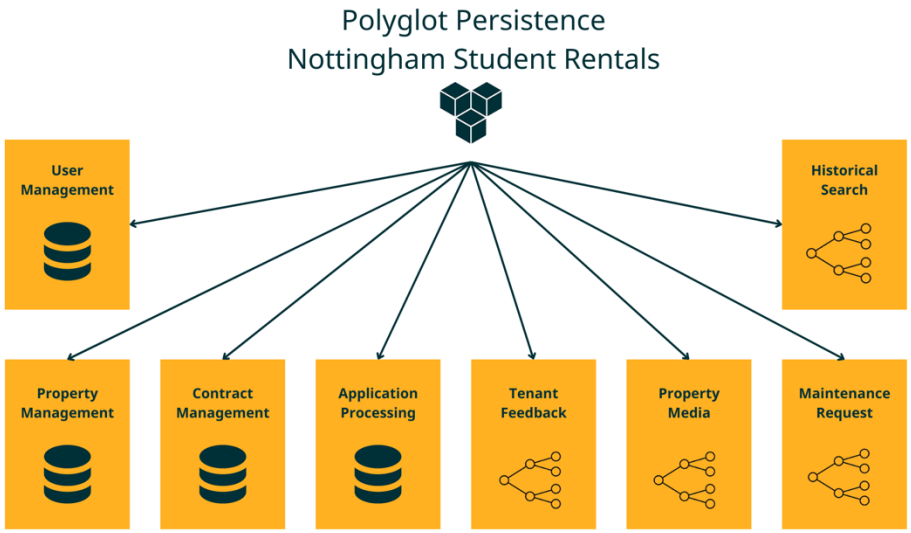
4. Opportunities for Polyglot Persistence

Throughout the process of developing Nottingham Student Rental's application, we have taken advantage of polyglot persistence, due to its innovative opportunity to leverage the advantages of both SQL and NoSQL databases. According to Wang, Z. (2019), it is obvious that integrating SQL and NoSQL databases is becoming an appropriate way for the different components, and "is prevalent in data-intensive big data systems, as they are distributed and parallel by nature". This approach is especially effective in the case of our application that has both structured and unstructured data and demands a data management solution specific to the domain.

Domain-Oriented Approach to Polyglot Persistence

An approach we applied for our polyglot persistence database is domain-oriented approach, which we have determined the functions that need structured data management, and those need the flexible handling of unstructured data. This method, as Wang, Z. (2019) suggests, enables the selection of the most suitable one based on the functional demands of different parts of the application. This approach makes it possible to streamline data management techniques, maintain scalability and flexibility and align with the distributed nature of big data systems.

The interaction between SQL and NoSQL addresses the application's complex data handling needs. While SQL databases make sure of the transactional integrity of structured data management, NoSQL have scalability and flexibility that are essential for managing unstructured data and creating actionable insights from big data analytics.



SQL for Structured Data Management

For structured data such as student profiles, property details, student contract, and application processing workflows, those are essential for ensuring the integrity and accuracy of transactions within the application. Our application's core functions of matching student preferences with available accommodations and managing lease agreements, are in need of SQL database since it supports complex queries and transactions. It follows the direction of Wang, Z. (2019)'s, regarding the necessity of maintaining ACID (Atomicity, Consistency, Isolation, Durability) properties for transactional integrity in distributed systems, ensuring that operations are processed reliably and consistently. For instance, when dealing with the relationships between rental agents, students, and property contracts, SQL databases offer the most suitable data integrity and reliability, ensuring that all transactions are processed accurately and consistently.

NoSQL for Unstructured Data Flexibility

On the other hand, MongoDB is excellent for handling unstructured data, such as Tenant Feedback Management, Property Media Management, and Maintenance Request Tracking. This NoSQL database's schema-less nature and BASE (Basically Available, Soft State, Eventual Consistency) properties provide the dynamic and flexible handling of a variety of data formats, ranging from reviews, ratings to multimedia content like photos and videos of properties (Wang, Z. (2019)). Additionally, MongoDB is strong in its ability to scale horizontally, allowing our platform to handle the huge data volumes of unstructured data such as users' search history.

However, it can be seen that integrating these databases within a polyglot persistence framework presents challenges particularly in maintaining data consistency and managing the complexity of data pipelines. As Wang, Z. (2019) discusses, it is a challenging task to establish data pipelines that can accommodate the demanding and specific data requirements for different data storage. This can be achieved with a proper planning and implementation that will prevent the formation of complex structures and ensure smooth data flow and processing.

5. Conclusion:

In conclusion, the project of the Nottingham Student Rental's application has addressed the housing problem of many students. Looking into the growing demand for student housing and the increase in rental costs, we have developed a platform to connect students with accommodation services. Our project has utilized polyglot persistence in which we have carefully selected RDBMS for structured data such as student profiles, student contract and NoSQL for unstructured data such as tenant review, users' search history. Moreover, this approach of polyglot persistence has made us well equipped to handle diverse data types in an efficient manner and thereby allows us to scale up and provide flexibility. Through this project, we had an opportunity to acquire database management skills, which provide a great contribution to the improvement of students' accommodation in Nottingham.

References

- Coronel, C., Blewitt, C., Crockett, K., & Morris, S. (2020). *Database Principles: Fundamentals of design, implementation, and management.* Cengage.
- H. (2023, October 25). *Rent now swallows up virtually all of the average maintenance loan as the student accommodation market reaches "crisis point" - HEPI.* HEPI. https://www.hepi.ac.uk/2023/10/26/student-rents-now-swallow-up-virtually-all-of-the-of-the-average-maintenance-loan-as-market-reaches-crisis-point-in-affordability/
- Hewitt, R. (2020, October 20). *Demand for Higher Education to 2035 - HEPI*. HEPI. https://www.hepi.ac.uk/2020/10/22/demand-for-higher-education-to-2035/
- Wang, Z. (2019). A Review of Polyglot Persistence in the Big Data World. *Information, 10(4), 141.* https://doi.org/10.3390/info10040141
- *Student accommodation costs across 10 cities in the UK* | HEPI. (n.d.). https://www.hepi.ac.uk/wp-content/uploads/2023/10/Student-accommodation-costs-across-10-cities-in-the-UK.pdf
- Sullivan, D. (2015). *NoSQL for Mere Mortals.* Addison-Wesley Professional.