Individual Assignment on Polyglot persistence

Module code: BEMM459
Database Technologies for Business Analytics
Word count: 2500

Part A1 2

- Choosing an organization and describing the business function 2
- The use case for the project 3
- Justification of the database choice 3
- List of assumptions 4

Part A2 5

- Constructing the ER diagram based on the use case (A1) 5
- Determining domain attributes and primary key {PK} / alternate keys [AK] 5

Part A3 6

- Logical Database Design 6
- Normalization 7
- Referential Integrity 9

Part A4 11

- How data will be stored in the document database 11

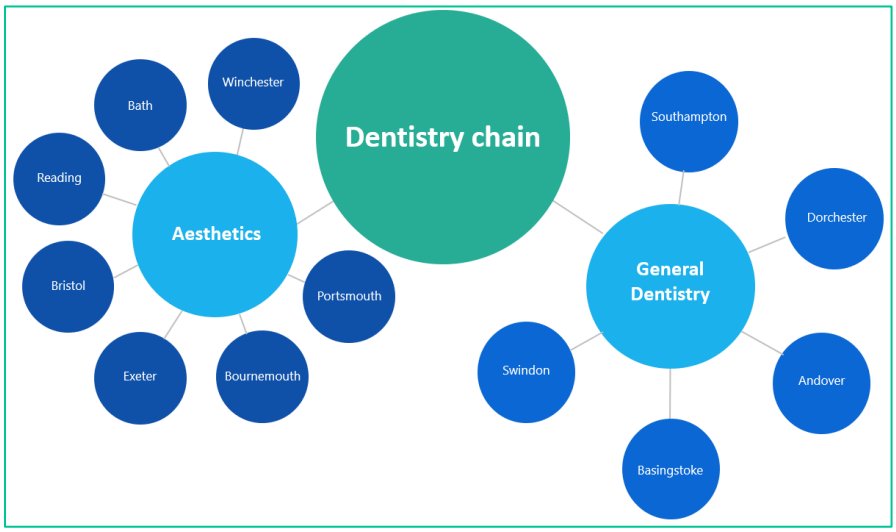
Part A5 12

- SQLite operations: 12
- MongoDB operations: 15
- Polyglot persistence 15

Part A1

➢ Choosing an organization and describing the business function

A Dentistry in Southampton, that has opened in the late 90s has been keeping track of all its bookings, patient information, payments and inventory using notebooks. The amount of data saved on paper became too huge to be handled and the Dentistry considered storing its data in a proper database. Another reason for such transition is the fact that the Dentistry has been performing extremely well and expanded in multiple cities in South England so that all these qualified dentists would be able to do their best to fulfil patients' requirements and take care of their smiles. The expansion decision was based on massive market research that pointed out that in some cities there is a scarcity of aesthetics dentistries, while at the same time the market is flooded by general dentistries and in other cities, the situation is right the opposite. Therefore, the Dentistry operates in two main areas – Aesthetic Dentistry in some cities and General Dentistry in other cities.



The transition towards a proper database is also driven by the increasing number of requirements from the Department of Health and Social Care regarding dentist therapies provided, namely clear description of all patient's needs, materials needed, the price paid by the patient, etc. so that corruption to be avoided. Furthermore, the Dentistry should collect patient contact data in order to collect feedback from patients and use it to improve its services, become more patient-focused and build a sustainable competitive advantage over other dentistries. Moreover, both Dentistry chains offer a huge range of services including tooth extraction, aligners therapy, whitening procedures and many others. For each of these therapies, the Dentistry should keep a log of the duration of the therapies for better appointment management and it is also crucial to keep track of the equipment and materials needed for each therapy for better pricing. On top of the booking services record, the Dentistry will need all the expenses in the database including medical equipment and materials (medicines, medical appliances) to monitor their costs and spending. Moreover, the dentistry would need to keep track of the dentists' personal information such as contact details, wage and qualifications in order to be able to recommend the right specialist for the right medical case and to contact the dentists when needed, add bonuses to the wages when extraordinary patient satisfaction is present. A record of each appointment should be kept as well, that includes the date and time of the appointment, the patient and the therapy made.

➢ The use case for the project

The dentistry wants to establish many aspects of the database such as expenditures tracking, online booking system, employee data storage, equipment, and materials availability, patient data, etc. However, the initial stage of the project will consider an **expenditure tracking system** which will be used as a base for a complex dashboard with many filters that will enable the Management team to make better data-driven decisions, by monitoring the expenditures of both Dentistry chains. If needed, further improvements could be made in the future. The main requirements for the initial stage of the project are as follows:

1. The Dentistry will have two chains (Aesthetics and General Dentistry) and each Dentistry will be connected to one and only one of the two main chains.
2. Up to now, the Dentistry operates in 13 cities, "Aesthetics" chain is in 8 of them, the other 5 are "General Dentistry". The database should store the contact details of each Dentistry such as phone number, address, and email.
3. Each Dentistry will provide at least 5 types of therapy and a maximum of 16 and one therapy could be provided in multiple dentistries. Because the Dentistry chain operates in different cities, the price for different therapies in different cities may vary and should be stored in the database.
4. One dentistry can serve from 1 to a maximum of 5 patients at once and contact information for each patient will be stored.
5. Each patient can make from 1 to many appointments at once.
6. Each appointment will include only one therapy and the date and time of the appointment will be stored in the database.
7. The database should allow staff to add appointments, patients and therapies to the database, update the date or time of the appointments, once the patients notify them for such changes and delete them, update therapy and patient data.
8. The database should keep track of all equipment and materials needed for a single appointment (number of equipment and materials for single appointment is **not fixed**), the quantity of the materials and their price, so that the Dentistry chain can constantly monitor its expenditures.

➢ Justification of the database choice

Rather than using either RDBMS or NoSQL, the best choice of storage architecture for the Dentistry will be to benefit from the strengths of both types of databases so that the final product will perfectly fit the needs of the company. Based on the needed functionalities, the kind of expected queries and the variability vs regularity of the data, it would be impossible to

choose only one database that could fulfil all requirements of the Dentistry and the better decision would be to combine both RDBMS and NoSQL by implementing a polyglot persistence so that all advantages associated with both databases would offer solutions for the dentistry requirements.

The database that is going to be created will support an information system that staff members are going to use for creating appointments that include patient data, therapy data and equipment and materials needed. Considering the large number of appointments from both dentistry chains and the plans of the Dentistry for further expansion, a database that can support great volumes of data will be needed. Another feature that the database should support is fast response time as many 'write' operations will be executed on daily basis and at the same time, the monitoring of the current activity of the Dentistry will be required from the Management team regularly. Such requirements will be a challenge for an RDBMS because RDBMS can not provide that high availability while at the same time supporting large volumes of reading and writing operations. These disadvantages of the RDBMS could be amended by using a NoSQL database. The further expansion of the Dentistry promises an immense number of appointments and therapies that should be stored, and a NoSQL database can deal perfectly with this amount of data, and it could also provide prompt response time. However, NoSQL databases have their disadvantages. Data integrity and ACID compliance are important database features that should be supported by the database when personal data is stored and retrieved from it.

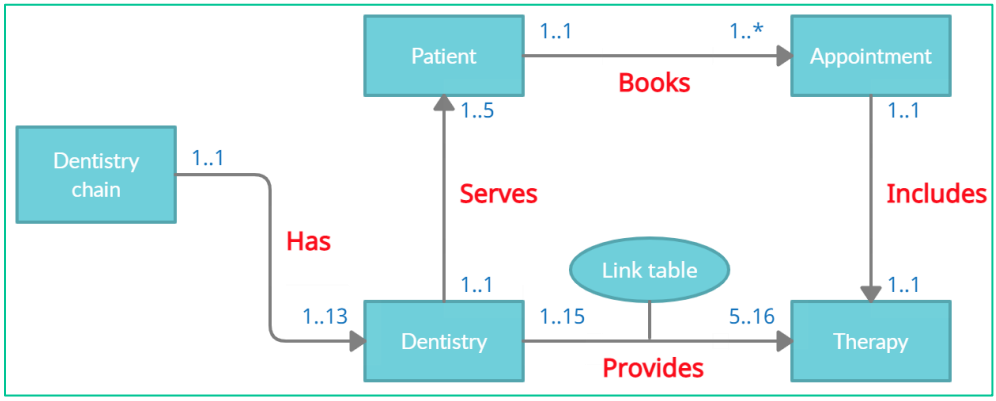
The most appropriate RDBMS that could be used for the Dentistry database is SQLite. In comparison to Microsoft Access, SQLite is more powerful, can handle larger amounts of data, has simpler syntax, and works faster across multiple platforms. On the other hand, the most appropriate NoSQL for the Dentistry database will be a document database. Advantageous features of document databases are their flexibility in terms of structure and their scalability. They do not require a fixed predefined structure, and this makes them the perfect fit for the Dentistry needs because the number of equipment and materials needed for each appointment is not fixed and could vary from just a few mirrors and tweezers to 200 tiny pieces for orthodontic therapy. Another feature of document databases that the Dentistry will benefit from is their ability to query and filter collections and documents extremely quickly. However, for the particular use case of the Dentistry, key-value databases would not be appropriate as they store key-value pairs in namespaces, and it would be tough for each appointment to be a namespace. On the other hand, graph databases are not appropriate for the use case of the Dentistry as well, because no relationship between appointments is present.

➢ List of assumptions

• Advanced data validation will be implemented in the application layer (for example patient shouldn't be able to book aesthetic therapies in General dentistry)
• GUI will be responsible for the visibility of booked and free appointments

Part A2

- Constructing the ER diagram based on the use case (A1)



- Determining domain attributes and primary key {PK} / alternate keys [AK]

Entity	Attribute Name	Description	Domain Attribute and/or Data type	Null	Multi-Values
Dentistry Chain	DentistryChainID {PK}	Unique identification number	Numeric	No	No
	DentistryChainName	Dentistry name	Text	No	No
	DentistryChainPhone [AK]	Dentistry chain's phone number	Numeric	No	No
	DentistryChainEmail [AK]	Contact email	Text (30 characters) and must include '@'.	No	No
Dentistry	DentistryID {PK}	Unique identification number	Numeric	No	No
	DentistryName	Dentistry name	Text	No	No
	DentistryAddress	Dentistry address	Text	No	No
	DentistryPhone [AK]	Dentistry's phone number	Numeric	No	No
	DentistryEmail [AK]	Contact email	Text (30 characters) and must include '@'.	No	No
Patient	PatientID {PK}	Unique number for each patient generated after booking first appointment	Numeric	No	No
	PatientFirstName	Patient's first name	Text	No	No
	PatientLastName	Patient's last name	Text	No	No
	PatientDOB	Patient's date of birth	DateTime		
	PatientAddress	Patient's address	Text	No	No
	PatientPhone [AK]	Patient's phone number	Numeric	No	No
	PatientEmail [AK]	Patient's email	Text (30 characters) and must include '@'.	No	No
Appointment	AppointmentID {PK}	Unique number to identify a distinct appointment	Numeric	No	No
	AppointmentDate [AK]	Date of the appointment	DateTime	No	No
	AppointmentTime [AK]	Time of the appointment	DateTime	No	No
Therapy	TherapyID {PK}	Unique number to identify a distinct therapy	Numeric	No	No
	TherapyName [AK]	Name of the therapy	Text	No	No
	TherapyDuration	Duration the therapy takes (in minutes)	Numeric	No	No

Part A3

- Logical Database Design

Dentistry Chain (Has) Dentistry – 1:* relationship

Dentistry Chain
(DentistryChainID, DentistryChainName, DentistryChainPhone, DentistryChainEmail)

Primary key: DentistryChainID
Alternate key: DentistryChainEmail, DentistryChainPhone

Dentistry
(DentistryID, DentistryName, DentistryAddress, DentistryPhone, DentistryEmail, DentistryChainID)

Primary key: DentistryID
Alternate key: DentistryEmail, DentistryPhone
Foreign key: DentistryChainID references **DentistryChain**(DentistryChainID)

Dentistry (Serves) Patient – 1:* relationship

Dentistry
(DentistryID, DentistryName, DentistryAddress, DentistryPhone, DentistryEmail, DentistryChainID)

Primary key: DentistryID
Alternate key: DentistryEmail, DentistryPhone
Foreign key: DentistryChainID references **DentistryChain**(DentistryChainID)

Patient
(PatientID, PatientFirstName, PatientLastName, PatientDOB, PatientAddress, PatientPhone, PatientEmail, DentistryID)

Primary key: PatientID
Alternate key: PatientPhone, PatientEmail
Foreign key: DentistryID references **Dentistry**(DentistryID)

Patient (Books) Appointment – 1:* relationship

Patient
(PatientID, PatientFirstName, PatientLastName, PatientDOB, PatientAddress, PatientPhone, PatientEmail, DentistryID)

Primary key: PatientID
Alternate key: PatientPhone, PatientEmail
Foreign key: DentistryID references **Dentistry**(DentistryID)

Appointment
(AppointmentID, AppointmentDate, AppointmentTime, PatientID)

Primary key: AppointmentID
Alternate key: AppointmentDate, AppointmentTime
Foreign key: PatientID references **Patient**(PatientID)

Appointment (Includes) Therapy – 1:* relationship

Therapy
(TherapyID, TherapyName, TherapyDuration)

Primary key: TherapyID
Alternate key: TherapyName

Appointment
(AppointmentID, AppointmentDate, AppointmentTime, PatientID, TherapyID)

Primary key: AppointmentID
Alternate key: AppointmentDate, AppointmentTime
Foreign key: PatientID references **Patient**(PatientID)
Foreign key: TherapyID references **Therapy**(TherapyID)

Dentistry (Provides) Therapy – *:* relationship

Dentistry
(DentistryID, DentistryName, DentistryAddress, DentistryPhone, DentistryEmail, DentistryChainID)

Primary key: DentistryID
Alternate key: DentristryPhone, DentsitryEmail
Foreign key: DentistryChainID references **DentistryChain**(DentistryChainID)

Therapy
(TherapyID, TherapyName, TherapyDuration)

Primary key: TherapyID
Alternate key: TherapyName

Link Table Identified

DentistryTherapy
(TherapyID, DentistryID, dtLabourPrice)
Primary key: TherapyID, DentistryID
Foreign key: TherapyID references **Therapy**(TherapyID)
Foreign key: DentistryID references **Dentistry**(DentistryID)

➢ Normalization

Table 1:

DentistryChain (DentistryChainID, DentistryChainName, DentistryChainPhone, DentistryChainEmail)

Primary key: DentistryChainID
Alternate key: DentistryChainEmail, DentistryChainPhone

- There are no multi-valued attributes in any column, hence the table is in **1NF**.
- There are no composite primary keys, hence the table is in **2NF**.
- All the values in the non-PK columns can be defined using the PK column or the candidate key column(s) and no other columns. Hence, the table is in **3NF**. **(note: DentistryChainEmail, DentistryChainPhone are candidate keys)**

Table 2:

Dentistry (DentistryID, DentistryName, DentistryAddress, DentistryPhone, DentistryEmail, DentistryChainID)

Primary key: DentistryChainID
Alternate key: DentistryEmail, DentistryPhone
Foreign key: DentistryChainID references DentistryChain(DentistryChainID)

- There are no multi-valued attributes in any column, hence the table is in **1NF**.
- There are no composite primary keys, hence the table is in **2NF**.
- All the values in the non-PK columns can be defined using the PK column or the candidate key column(s) and no other columns. Hence, the table is in **3NF**. **(note: DentistryEmail, DentistryPhone are candidate keys)**

Table 3:

Patient (PatientID, PatientFirstName, PatientLastName, PatientDOB, PatientAddress, PatientPhone, PatientEmail, DentistryID)

Primary key: PatientID
Alternate key: PatientEmail, PatientPhone
Foreign key: DentistryID references Dentistry(DentistryID)

- There are no multi-valued attributes in any column, hence the table is in **1NF**.
- There are no composite primary keys, hence the table is in **2NF**.
- All the values in the non-PK columns can be defined using the PK column or the candidate key column(s) and no other columns. Hence, the table is in **3NF**. **(note: PatientEmail, PatientPhone are candidate keys)**

Table 4:

Therapy (TherapyID, TherapyName, TherapyDuration)

Primary key: TherapyID
Alternate key: TherapyName

- There are no multi-valued attributes in any column, hence the table is in **1NF**.
- There are no composite primary keys, hence the table is in **2NF**.
- All the values in the non-PK columns can be defined using the PK column or the candidate key column(s) and no other columns. Hence, the table is in **3NF**. **(note: TherapyName is candidate key)**

Table 5:

Appointment (AppointmentID, AppointmentDate, AppointmentTime, PatientID, TherapyID)

Primary key: AppointmentID
Alternate key: AppointmentDate, AppointmentTime
Foreign key: PatientID references **Patient**(DentistryID)
Foreign key: TherapyID references **Therapy**(TherapyID)

- There are no multi-valued attributes in any column, hence the table is in **1NF**.
- There are no composite primary keys, hence the table is in **2NF**.
- All the values in the non-PK columns can be defined using the PK column or the candidate key column(s) and no other columns. Hence, the table is in **3NF**. **(note: AppointmentDate, AppointmentTime are candidate keys)**

Table 6:

DentistryTherapy (TherapyID, DentistryID, dtLabourPrice)

Primary key: TherapyID, DentistryID
Foreign key: TherapyID references **Therapy**(TherapyID)
Foreign key: DentistryID references **Dentistry**(DentistryID)

- There are no multi-valued attributes in any column, hence the table is in **1NF**.
- There are no composite primary keys, hence the table is in **2NF**.
- All the values in the non-PK columns can be defined using the PK column or the candidate key column(s) and no other columns. Hence, the table is in **3NF**.

➢ Referential Integrity

Table 1

Dentistry Chain (DentistryChainID, DentistryChainName, DentistryChainPhone, DentistryChainEmail)

Primary key: DentistryChainID
Alternate key: DentistryChainEmail, DentistryChainPhone

Table 2

Dentistry (DentistryID, DentistryName, DentistryAddress, DentistryPhone, DentistryEmail, DentistryChainID)

Primary key: DentistryID
Alternate key: DentistryEmail, DentistryPhone
Foreign key: DentistryChainID references **DentistryChain**(DentistryChainID) on Update Cascade on Delete No Action

Table 3

Patient (PatientID, PatientFirstName, PatientLastName, PatientDOB, PatientAddress, PatientPhone, PatientEmail, DentistryID)

Primary key: PatientID
Alternate key: PatientPhone, PatientEmail
Foreign key: DentistryID references **Dentistry**(DentistryID) on Update Cascade on Delete Restrict

Table 4

Therapy (TherapyID, TherapyName, TherapyDuration)

Primary key: TherapyID
Alternate key: TherapyName

Table 5

Appointment (AppointmentID, AppointmentDate, AppointmentTime, PatientID, TherapyID)

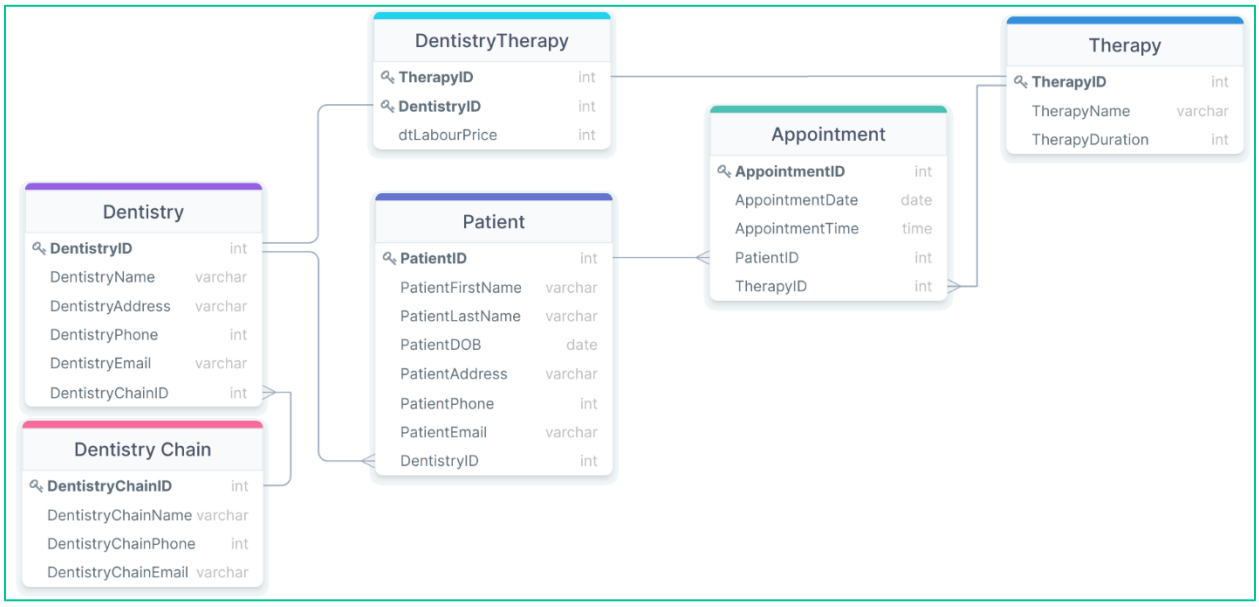
Primary key: AppointmentID
Alternate key: AppointmentDate, AppointmentTime
Foreign key: PatientID references **Patient**(PatientID) on Update Cascade on Delete Restrict
Foreign key: TherapyID references **Therapy**(TherapyID) on Update Cascade on Delete Restrict

Table 6

DentistryTherapy (TherapyID, DentistryID, dtLabourPrice)

Primary key: TherapyID, DentistryID
Foreign key: TherapyID references **Therapy**(TherapyID) on Update Cascade on Delete Restrict
Foreign key: DentistryID references **Dentistry**(DentistryID) on Update Cascade on Delete Restrict

The tables above describe how entities in A2 have been converted into tables with specific attributes. The relationships between entities are implemented by foreign keys and updating or deleting data in a table, cascadely updates or deletes data in all tables related to it. Cascading delete is a dangerous operation and it is used with high attention in the tables above.



Part A4

➢ How data will be stored in the document database

As mentioned in A1, the NoSQL database that will be used to fulfil the increasing requirements of the Dentistry is a document database, namely MongoDB. Document databases are the most "structured" NoSQL databases and provide great flexibility in terms of structure while at the same time it is advisable that documents in the same collection have a similar structure. In the case of the Dentistry database, flexibility is a crucial feature of the database as each appointment will include numerous pieces of equipment which may vary from 2-3 to 200. At the same time, all documents will have the same structure as noted in the figure below.

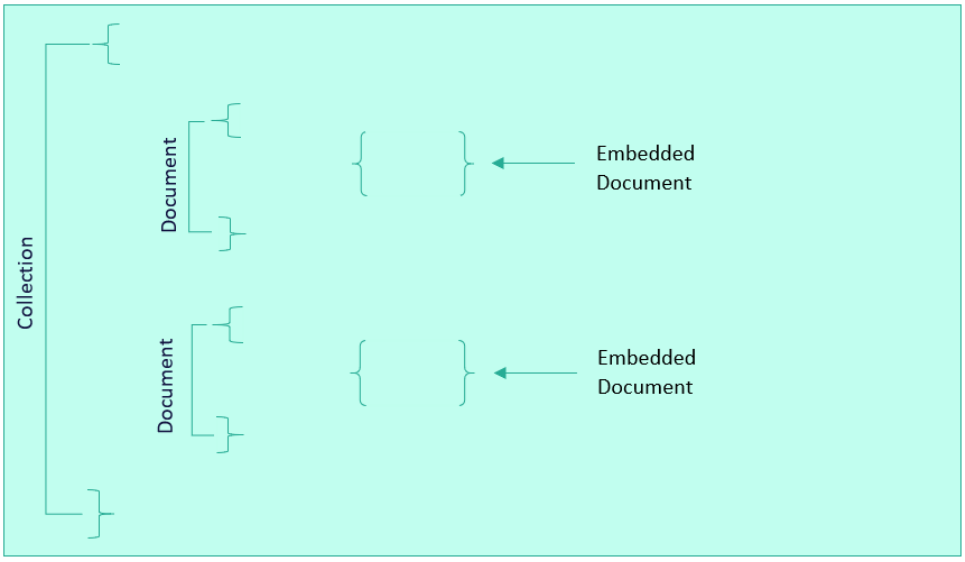


Figure 1: Document database schema

For the purpose of this project, only one collection will be used. It will contain all appointments as documents, where the document id will correspond to the id of the appointment in the RDBMS so that the data could be merged later and further explored. Each document stores as many embedded documents as the number of equipment and materials used and each embedded document consist of 3 key-value pairs, namely the name of equipment, the quantity used and the price of the equipment. The design decision for using embedded documents is based on the queries that will be made. It has been taken into consideration that all the data in the embedded documents will be used together for calculation purposes. Therefore, these assumptions have justified the design decision.

Another option for the database schema is storing each appointment as a different collection and each piece of equipment used as an individual document in this collection. Such schema was not chosen because of the possibility of performance issues. Querying multiple collections at once is much slower than querying a single collection with similar documents and this fact has been taken into account during the design process.

Part A5

According to the use case in A1, the staff members and the Management team will need the database for their everyday tasks and also for performance evaluation of the dentistries in both chains. The following operations have been enabled by both RDBMS and NoSQL and in the end, polyglot persistence has been implemented.

➢ SQLite operations:

1. Adding new patient details

```
# A staff member can manually add a new patient

inputPatientID = int(input('Enter ID of the patient: '))
inputPatientFirstName = input('Enter first name of the patient: ')
inputPatientLastName = input('Enter last name of the patient: ')
inputPatientDOB = input('Enter the date of birth of the patient: ')
inputPatientAddress = input('Enter the address of the patient: ')
inputPatientPhone = int(input('Enter the phone of the patient: '))
inputPatientEmail = input('Enter the email of the patient: ')
inputDentistryID = int(input('Enter the ID of the dentistry that the patient will visit: '))

qry="insert into Patient values (?,?,?,?,?,?,?,?);"

try:
        cur.execute(qry, \
                    (inputPatientID, inputPatientFirstName, inputPatientLastName, inputPatientDOB, inputPatientAddress, \
                    inputPatientPhone, inputPatientEmail, inputDentistryID))
        print ('New patient added!')
        conn.commit()
except:
        print ('Error in adding patient .. rollback')
        conn.rollback()
```

```
Enter ID of the patient: 21
Enter first name of the patient: Jane
Enter last name of the patient: Rose
Enter the date of birth of the patient: 13/08/1999
Enter the address of the patient: 13 High Street RG4 3LF
Enter the phone of the patient: 7598596852
Enter the email of the patient: janerose@yahoo.com
Enter the ID of the dentistry that the patient will visit: 11
New patient added!
```

2. Assign an appointment to a patient

```
# A staff member can manually assign an appointment to the new patient:

inputAppointmentID = int(input('Enter ID of the appointment: '))
inputAppointmentDate = input('Enter date of the appointment: ')
inputAppointmnetTime = input('Enter time of the appointmnet: ')
inputPatientID = int(input('Enter ID of the patient: '))
inputTherapyID = int(input('Enter ID of the therapy: '))

qry="insert into Appointment values (?,?,?,?,?);"

try:
        cur.execute(qry, \
                    (inputAppointmentID, inputAppointmentDate, inputAppointmnetTime, inputPatientID, inputTherapyID))
        print ('New appointment added!')
        conn.commit()
except:
        print ('Error in adding appoitnment .. rollback')
        conn.rollback()
```

```
Enter ID of the appointment: 23
Enter date of the appointment: 08/03/2021
Enter time of the appointmnet: 12:00
Enter ID of the patient: 21
Enter ID of the therapy: 15
New appointment added!
```

3. Update patient details

```
# Update a patient address based on user input

inputPatientID = int(input('Enter PatientID: '))
inputPatientAddress = input("Enter address to be updated: ")

qry = 'UPDATE Patient SET PatientAddress=? WHERE PatientID=?'

try:
        cur.execute(qry, (inputPatientAddress,inputPatientID))
        print ('Patient address updated!')
        conn.commit()
except:
        print ('Error in update operation .. rollback')
        conn.rollback()
```

```
Enter PatientID: 5
Enter address to be updated: 2 High Street SO50 6RG
Patient address updated!
```

4. Update appointment details

```
# Update an appointment therapy

inputAppointmentID = int(input('Enter AppointmentID: '))
inputThereapyID = input("Enter TherapyID to be updated: ")

qry = 'UPDATE Appointment SET TherapyID=? WHERE AppointmentID=?'

try:
        cur.execute(qry, (inputTherapyID, inputAppointmentID))
        print ('Appointmnent therapy updated!')
        conn.commit()
except:
        print ('Error in update operation .. rollback')
        conn.rollback()
```

```
Enter AppointmentID: 16
Enter TherapyID to be updated: 7
Appointmnent therapy updated!
```

5. Delete an appointment

```
# Delete appointment with ID=? (manual input)

inputAppointmentID = int(input('Enter AppointmentID to be deleted: '))

qry = 'DELETE from Appointment where AppointmnetID =?'

try:
        cur.execute(qry, (inputAppointmentID,))
        print ('Appointmnent deleted!')
        conn.commit()
except:
        print ('Error in delete operation .. rollback')
        conn.rollback()
```

```
Enter AppointmentID to be deleted: 20
Error in delete operation .. rollback
```

On top of these functions, the RDBMS is capable of running queries that will ease the work of staff members and will help them make business insights based on the present data.

1. Display all patients who are served by a particular Dentistry

```
# Query to display all patients of the Brighton Dentistry (DentistryID = 3)

qry = "SELECT * FROM Patient WHERE DentistryID = 3"

cur. execute(qry)

rows = cur.fetchall()

for row in rows:
    print(row)
```

```
(18, 'Suzan', 'Scavo', '13/03/2000', '6 Padwell Road BN18 9WM', 7598969887, 'suzans@yahoo.com', 3)
(20, 'Hugo', 'Lopez', '07/11/1992', '18 Padwell Road BN18 9WM', 7598696355, 'hugolopez@yahoo.com', 3)
```

2. Display the appointments for a particular day

```
# Query to display all appointments on 01/03/2021

qry = "SELECT * FROM Appointment WHERE AppointmentDate = '01/03/2021' "

cur. execute(qry)

rows = cur.fetchall()

for row in rows:
    print(row)
```

```
(1, '01/03/2021', '10:00', 1, 12)
(2, '01/03/2021', '12:00', 2, 13)
(3, '01/03/2021', '15:00', 3, 1)
```

3. Display all appointments made, first and last name of the patient and the labour price

```
# Query to didsplay all appointments made, the patient first and last name and the labour price for each appointment

qry = '''

    SELECT AppointmentID, PatientFirstName, PatientLastName, dtLabourPrice
    FROM Appointment
    JOIN Patient ON Appointment.PatientID = Patient.PatientID
    JOIN DentistryTherapy ON Appointment.TherapyID = DentistryTherapy.TherapyID AND
         Patient.DentistryID = DentistryTherapy.DentistryID;
    '''

cur.execute(qry)

rows = cur.fetchall()

for row in rows:
    print(row)
```

```
(1, 'Oliver', 'West', 190)
(2, 'Heather', 'Peterson', 250)
(3, 'Roger', 'Davidson', 100)
(4, 'Olivia', 'Rose', 300)
(5, 'Ellie', 'Collins', 120)
(6, 'David', 'Millar', 150)
(7, 'Peter', 'Audric', 120)
(8, 'Polly', 'Shelby', 200)
(9, 'Thomas', 'Gravy', 120)
(10, 'Owen', 'West', 310)
(11, 'Kevin', 'Hart', 100)
(12, 'Rebecca', 'Rose', 130)
(13, 'Daniel', 'Buxton', 270)
(14, 'Adam', 'Rayan', 120)
(15, 'Katie', 'Millar', 200)
(16, 'Donald', 'Duck', 100)
(17, 'Margharet', 'Alexander', 100)
(18, 'Suzan', 'Scavo', 210)
(19, 'Alexander', 'Millar', 200)
(20, 'Hugo', 'Lopez', 300)
(21, 'Adam', 'Rayan', 110)
(22, 'Owen', 'West', 210)
```

➢ MongoDB operations:

1. Add appointment equipment

```
# Insert single document

mydoc = {"_id":25, "equipment": [{"name": "Prong Plier", "quantity": 2, "price": 15},
                                 {"name": "Beak Plier", "quantity": 1, "price": 9}]
        }

var = mycol.insert_one(mydoc)

print(var)
```

```
<pymongo.results.InsertOneResult object at 0x000001A4C75A8340>
```

2. Update equipment for a particular appointment

```
# Update equipment data in a document

myquery = { "name": "Cavit"}
newvalues = { "$set": {"name": "Implant"}}

x = mycol.update_one(myquery, newvalues)

print(x.modified_count, "documents updated.")
```

```
0 documents updated.
```

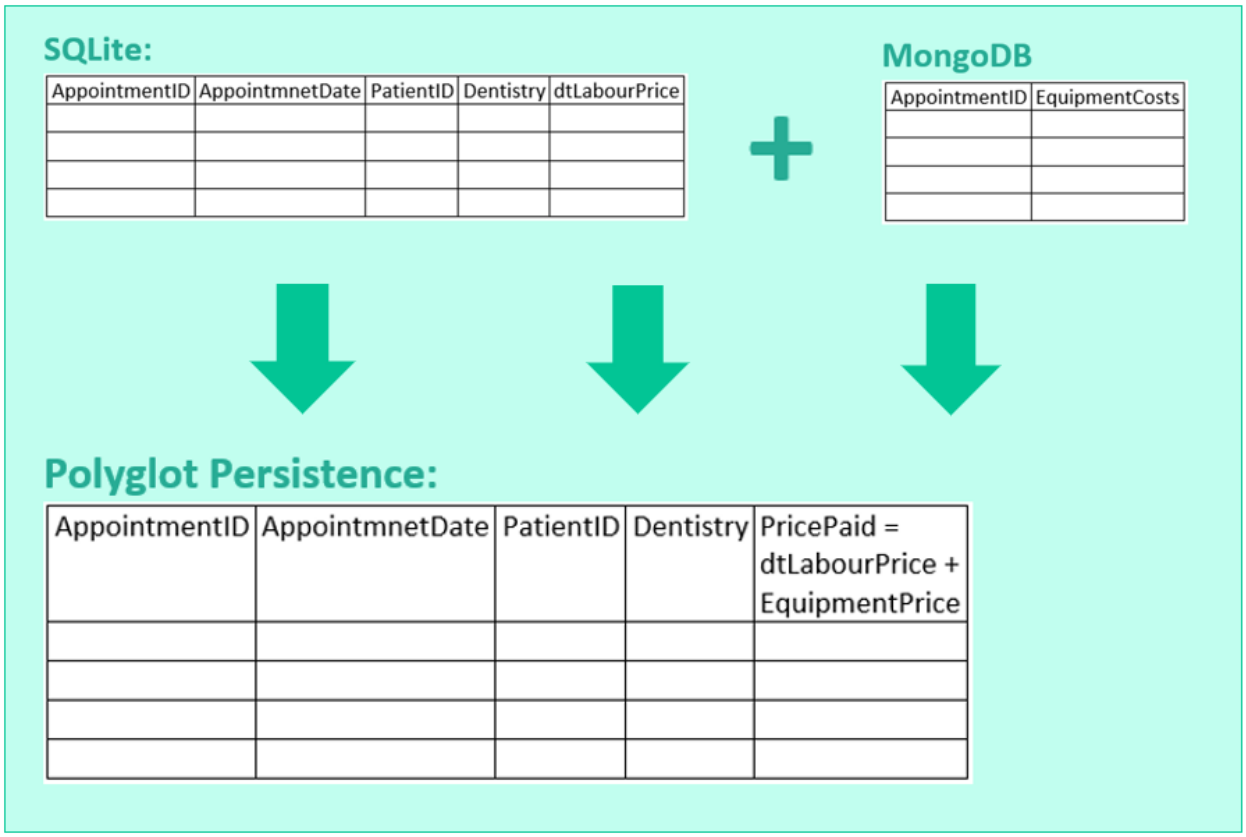
3. Update the price of particular equipment in all appointments it is present

```
# Update the price of all equipment that starts with "Bur"

myquery = { "name": { "$regex": "^Bur" } }
newvalues = { "$set": { "price": 12 } }

x = mycol.update_many(myquery, newvalues)

print(x.modified_count, "documents updated.")
```

```
0 documents updated.
```

➢ Polyglot persistence

Polyglot persistence has been implemented by merging data from both relational and non-relational databases. The reasons for storing data separately in different databases have been discussed in A1. One of the requirements of the use case in A1 is the ability of the database to monitor the price of each appointment. This requirement could be fulfilled by querying the relational database for all appointments, the dentistry where the appointment takes place and the price of the labour of the dentist. After querying the relational database, some calculations in MongoDB are performed and no matter how much equipment has been used, the costs are easily calculated by multiplying the quantity of the equipment and the price. Afterwards, both dataframes are merged, based on the ID of the appointment. The following picture describes graphically how this operation is performed so that detailed business insights to be derived.


SQLite:
AppointmentID
AppointmnetDate
PatientID
Dentistry
dtLabourPrice
MongoDB
AppointmentID
EquipmentCosts
Polyglot Persistence:
AppointmentID
AppointmnetDate
PatientID
Dentistry
PricePaid = dtLabourPrice + EquipmentPrice