UNIVERSITY of BRADFORD

Large Scale Data Driven Applications	Design and Critical Analysis on SQL and NoSQL on Healthcare No Show Dataset	Version: 4.0 Effective Date: 25-11-2024
Sameer Uddin 20004135		

Contents

Purpose

Missed healthcare appointments distribute clinic workflows. There are many factors influencing this such as demographic, medical conditions, etc… Disruptions can cause problems for patients and for the clinic.

This coursework will utilise an RDBMS, MongoDB and PHP to analyse key factors to provide real-time insights on no-show rates by multiple factors. This will be done by analysing an existing set of data which shows whether patients attend their medical appointments over several months in 2016.

Analyse of Dataset

The original dataset was obtained from Kaggle and contains 106987 rows and has 15 columns.

Column	Description	Type of Data
PatientId	Unique identifier for each patient in the database	Integer
AppointmentID	Unique identifier for each appointment that is booked	Integer
Gender	Gender of patient (male or female)	Character
ScheduledDay	Date of when the appointment was booked over the phone or online	Date
AppointmentDay	The date of the actual appointment	Date
Age	Age of patient	Integer
Neighbourhood	Neighbourhood of the patient	String
Scholarship	Does the patient have a scholarship which can be used to get benefits	Boolean
Hipertension	Does the patient have hypertension	Boolean
Diabetes	Does the patient have diabetes	Boolean
Alcoholism	Does the patient have alcoholism	Boolean
Handcap	Does the patient have handicap	Boolean
SMS_received	Does the patient receive an SMS reminder for appointments	Boolean
Showed_up	Did the patient show up for the appointment	Boolean
Date.diff.	Difference in days between the appointment booking date and appointment date	Integer

Table 1 - Dataset Attributes

From Kaggle we have obtained the data but it is not sorted. If we were to insert all of this data into a relational database straight from the CSV file it will not be organised and queries performed on the database will be inefficient. We need to normalise the data to protect the data and to make the database more flexible for when we add data (Microsoft, 2024).

If we were to add the details of another patient to this database, we would need to add other details such as appointment ID and appointment date, which may not have values as the patient has yet to book them.

First Normal Form

Now that we have cleaned the data and removed attributes that are not required, we can normalise the data so that it can be stored efficiently in the database. The first step is to create tables for each set of related data (Microsoft, 2024). Information about the patient such as “PatientID”, “Gender”, “Age”, “Neighbourhood”, “Scholarship”, “Hipertension”, “Diabetes”, “Alcoholism” and “Handicap”.

The other table of related data would be in regards to the appointments. “AppointmentID”, “PatientID”, “ScheduledDay”, “AppointmentDay”, “SMS_received”, “Showed_Up” and “Date.diff.”.

Each table in this database has been given a primary key to meet the first normal form.

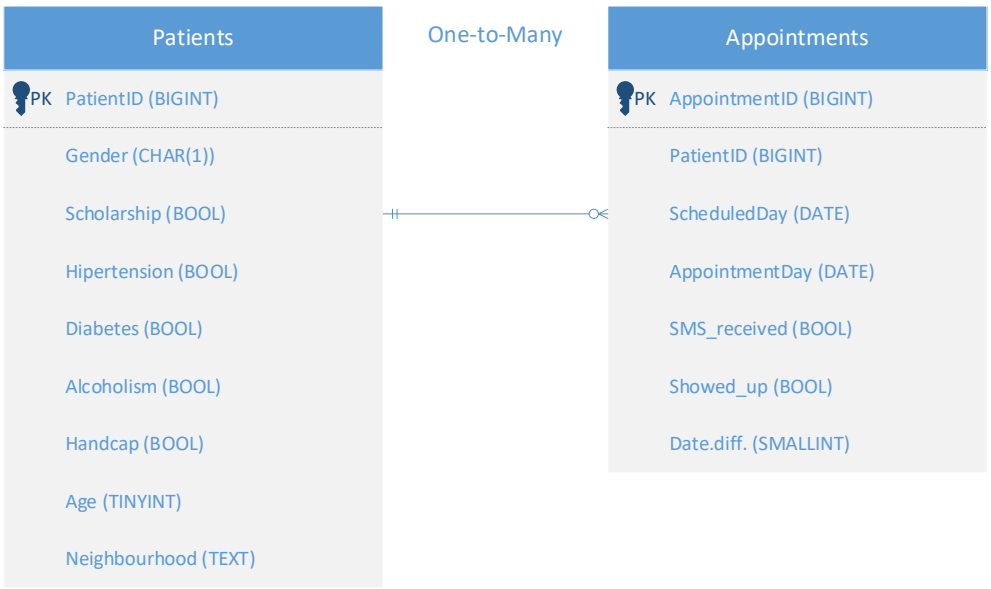


Figure 1 - E-R Diagram (1NF)

Second Normal Form

To apply second normal form to this database we need to add a foreign key and link the two tables together. We should also create separate tables for sets of values that apply to multiple records like “Neighbourhood” and create a new column called “NeighbourhoodID”.

To achieve this using Excel, I created another sheet with Neighbourhood as one column and another with NeighbourhoodID. From the original table I copied all the data and removed duplicate values. There are 81 distinct neighbourhoods.

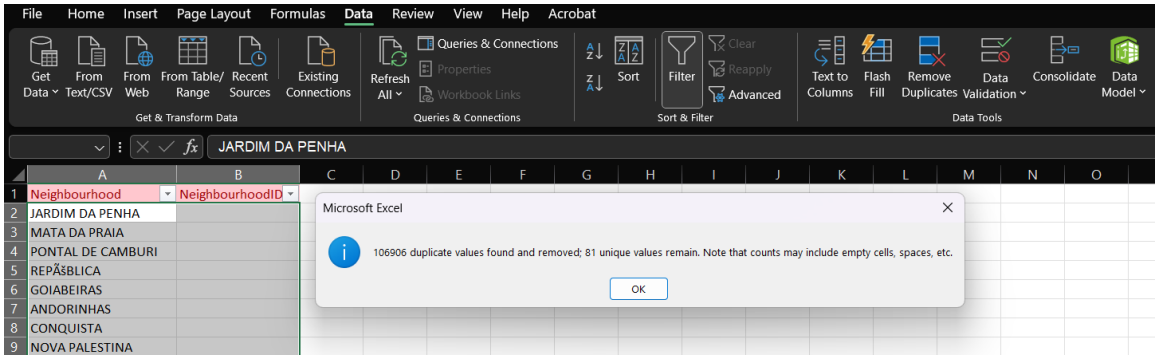


Figure 2 - Removing Duplicate Values

Then the VLOOKUP function is used to add the “NeighbourhoodID” column to the dataset:
Select neighbourhood data in original file, select the table from second sheet, grab data from second column in second sheet, FALSE for an exact match

=VLOOKUP(G2:G106988, Neighbourhood!A2:B82, 2, FALSE)

The next step is to change the ID column by performing a special paste of the values only so that VLOOKUP does not need to be used. The neighbourhood column can now be removed as the names are in another sheet with a corresponding ID.

PatientId	AppointmentID	Gen	ScheduledDay	AppointmentDay	Age	NeighbourhoodID
29872499824296	5642903	F	29/04/2016	29/04/2016	62	1
558997776694438	5642503	M	29/04/2016	29/04/2016	56	1
4262962299951	5642549	F	29/04/2016	29/04/2016	62	2
867951213174	5642828	F	29/04/2016	29/04/2016	8	3
8841186448183	5642494	F	29/04/2016	29/04/2016	56	1

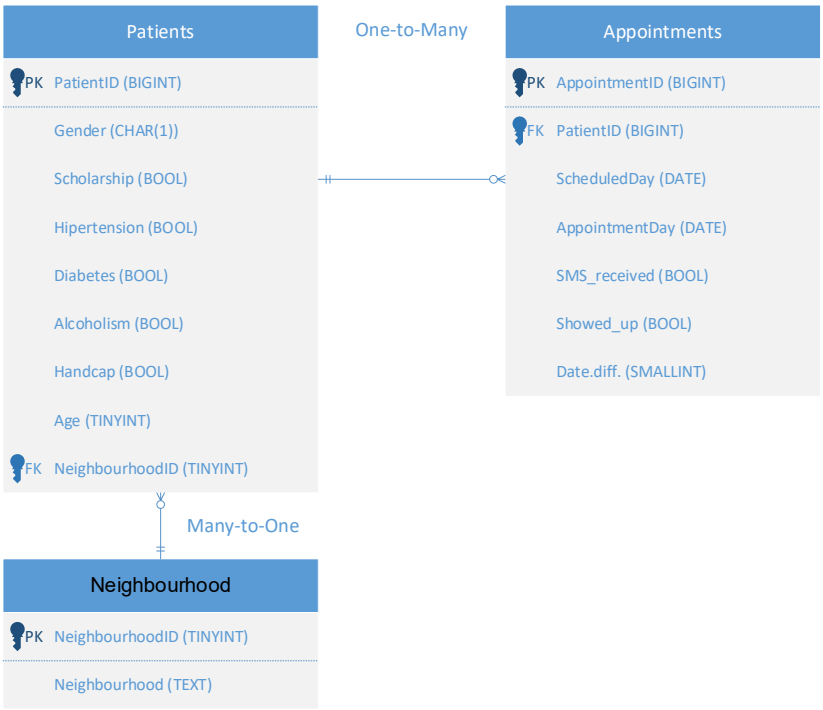


Figure 3 - E-R Diagram (2NF)

Third Normal Form

To apply the third normal form, we need to remove every non-key attribute in the tables that don't depend on the primary key. The date difference depends on other non-keys and does not need to exist as it can be dynamically calculated (Decomplexify, 2021).

Figure 4 - E-R Diagram (3NF)

Now the database is in third normal form. Below is an example of the Excel data.

PatientId	Gender	Age	Neighbour	Scholarshi	Hipertensi	Diabetes	Alcoholism	Handcap
29872499824296	F	62	1	FALSE	TRUE	FALSE	FALSE	FALSE
558997776694438	M	56	1	FALSE	FALSE	FALSE	FALSE	FALSE
4262962299951	F	62	2	FALSE	FALSE	FALSE	FALSE	FALSE
867951213174	F	8	3	FALSE	FALSE	FALSE	FALSE	FALSE

Table 2 - Patients Sheet

Neighbourhood	NeighbourhoodID
JARDIM DA PENHA	1
MATA DA PRAIA	2
PONTAL DE CAMBURI	3
REPÃšBLICA	4

Table 3 - Neighbourhood Sheet

AppointmentID	PatientId	ScheduledDay	AppointmentDay	SMS_received	Showed_up
5642903	29872499824296	29/04/2016	29/04/2016	FALSE	TRUE
5642503	558997776694438	29/04/2016	29/04/2016	FALSE	TRUE
5642549	4262962299951	29/04/2016	29/04/2016	FALSE	TRUE
5642828	867951213174	29/04/2016	29/04/2016	FALSE	TRUE

Table 4 - Appointments Sheet

Cleaning CSV Files

The current CSV files also contain fields like patient ID which are formatted to “general” which makes it harder to read and identify duplicate values.

PatientId	AppointmentID	Gender	ScheduledDay
2.99E+13	5642903	F	29/04/2016

The column “PatientID” converted to a number is below:

PatientId	AppointmentID	Gender	ScheduledDay
29872499824296	5642903	F	29/04/2016

Analysing the CSV file shows that there are duplicate entries for patients, an example being user “9876237815187” who has two entries in row 55636 and in row 104067.

Using the “Remove Duplicates” function in Excel we go from a total of 106987 to 60270 rows, saving storage space and removing redundant data.

Now we can search for users in the appointment sheet in the CSV file and see if there are duplicate attributes in the current dataset.

PatientId	AppointmentID	Gender	ScheduledDay	AppointmentD	Age	Neighbourhood
29872499824296	5642903	F	2016-04-29	2016-04-29	62	JARDIM DA PENHA
29872499824296	5639907	F	2016-04-29	2016-04-29	62	JARDIM DA PENHA

This patient had two appointments in 2016. The data also implies that the user booked two appointments on the same date and even selected the same appointment day.

The other implication is that there was a system or human error that recorded this information two times. The patient also could have changed the time she wanted to book the appointment for, but time is not an attribute that is included in this dataset.

The data does state that the user showed up to both appointments so she could have had an emergency reason to visit the clinic two times that day.

There are no duplicate values for appointment ID so I have not deleted any rows of information for this table.

The date for the appointments also had to be updated to the format YYYY-MM-DD from DD-MM-YYYY as most relational databases have an issue with that date format. To be able to store the date as a date data type it needs to be in the format YYYY-MM-DD. This is done by converting the date to the US format by using Excel and changing the Windows short form date to the correct format for the database.

RDBMS Implementation

Below is the implementation of the relational database using PostgreSQL.

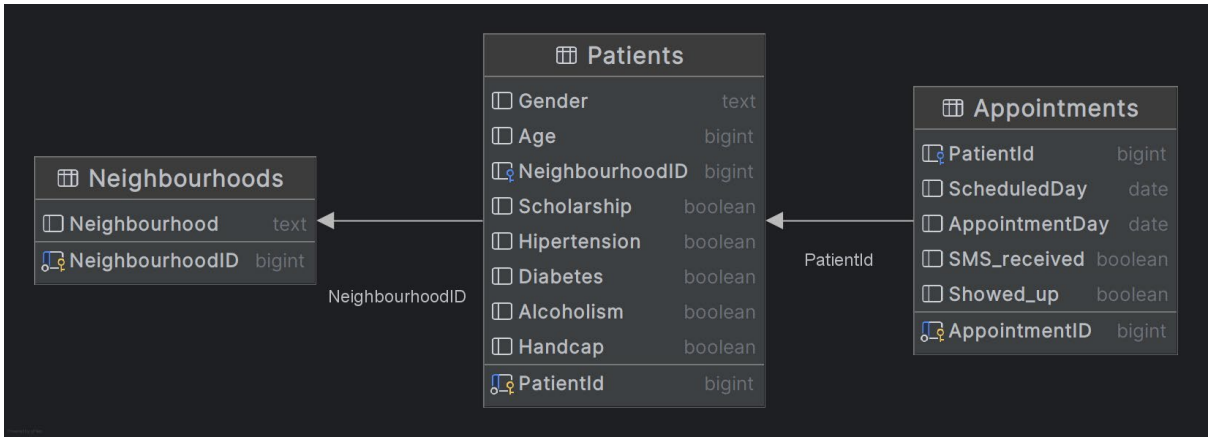


Figure 5 – RDBMS Schema Visualiser

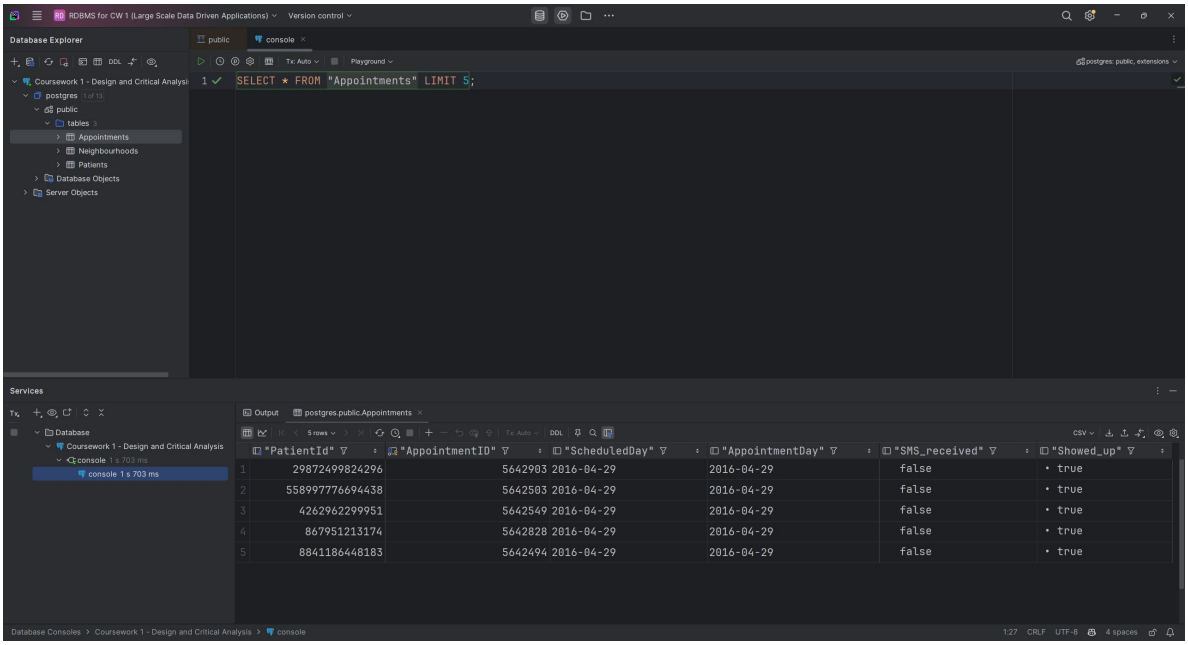


Figure 6 - Appointments Table

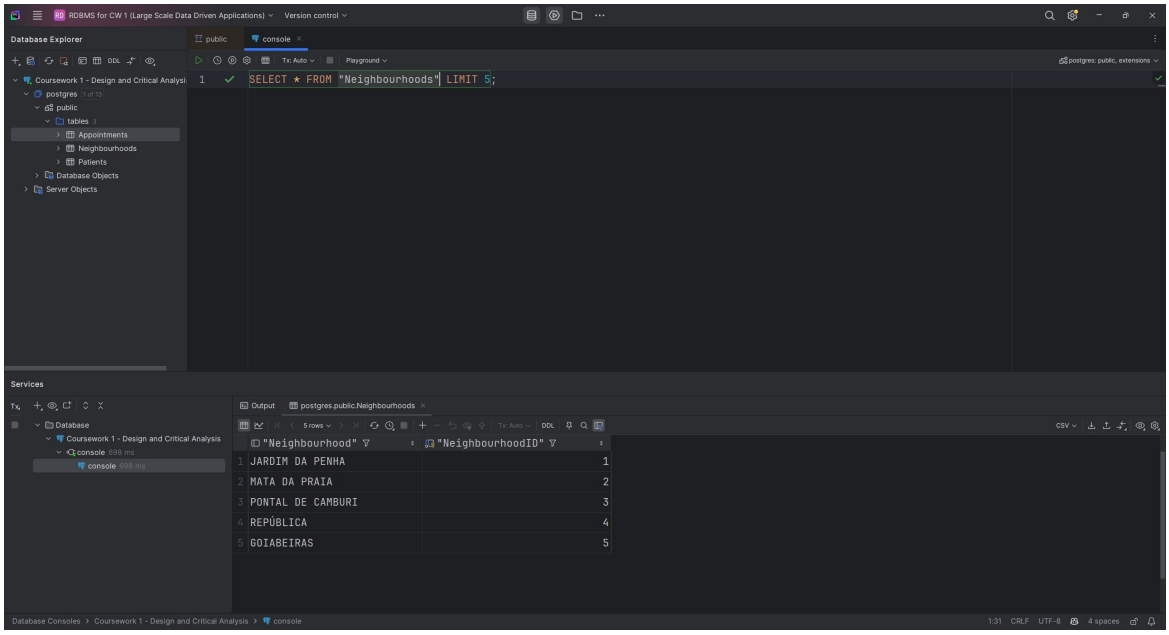


Figure 7 - Neighbourhoods Table

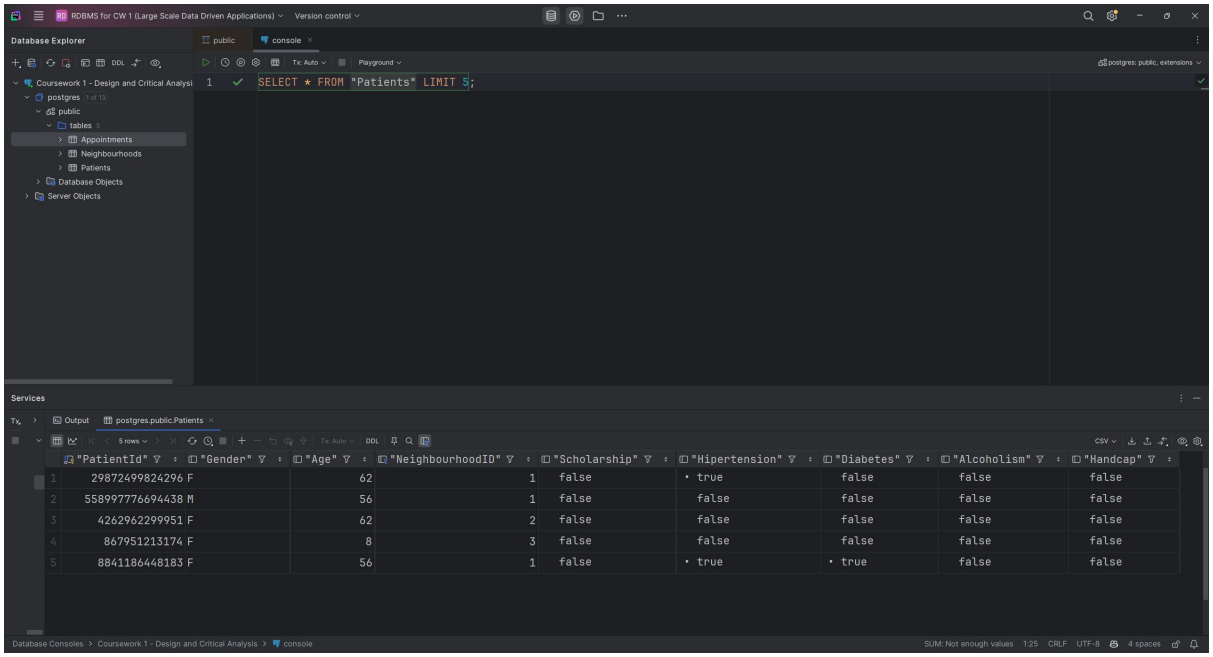


Figure 8 - Patients Table

MongoDB Implementation

For the MongoDB implementation I will be utilising embedding over referencing as the SQL set is normalised and uses a foreign key for referencing. Using embedding for one system and referencing for the other will allow me to analyse the strengths and weaknesses of both systems to determine which is better for a given task. This approach will also mean that queries are simpler. The reads will also be faster which will help with analysing the data.

MongoDB Design

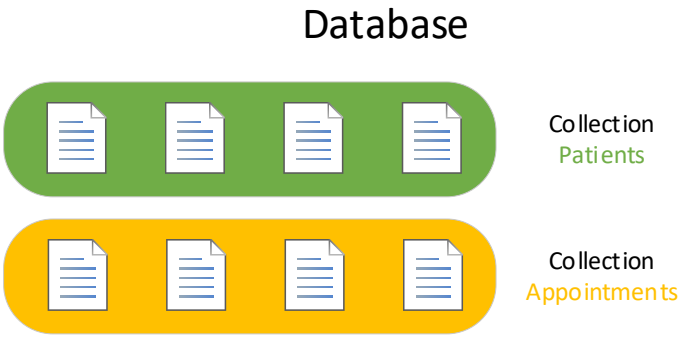


Figure 9 - Database for MongoDB

```
{
    "PatientID": 29872499824296,
    "gender": "M",
    "age": 72,
    "scholarship": false,
    "hipertension": true,
    "diabetes": true,
    "alcoholism": true,
    "handcap": true,
    "neighbourhood": "NOVA PALESTINA"
}
```

```
{
    "AppointmentID": 298724,
    "Patient": {
        "PatientID": 29872499824296,
        "gender": "M",
        "age": 72,
        "scholarship": false,
        "hipertension": true,
        "diabetes": true,
        "alcoholism": true,
        "handcap": true,
        "neighbourhood": "NOVA PALESTINA",
    }
    "ScheduledDay": 2016-12-01,
    "AppointmentDay": 2016-12-02,
    "SMS_received": true,
    "Showed_up": true,
    "Date.diff": 1
}
```

Implementation

The date difference can be calculated using the aggregation pipeline but for this use case I have kept it in the document as it has the potential to improve query performance because it removes the need to perform calculations as it is already pre-calculated.

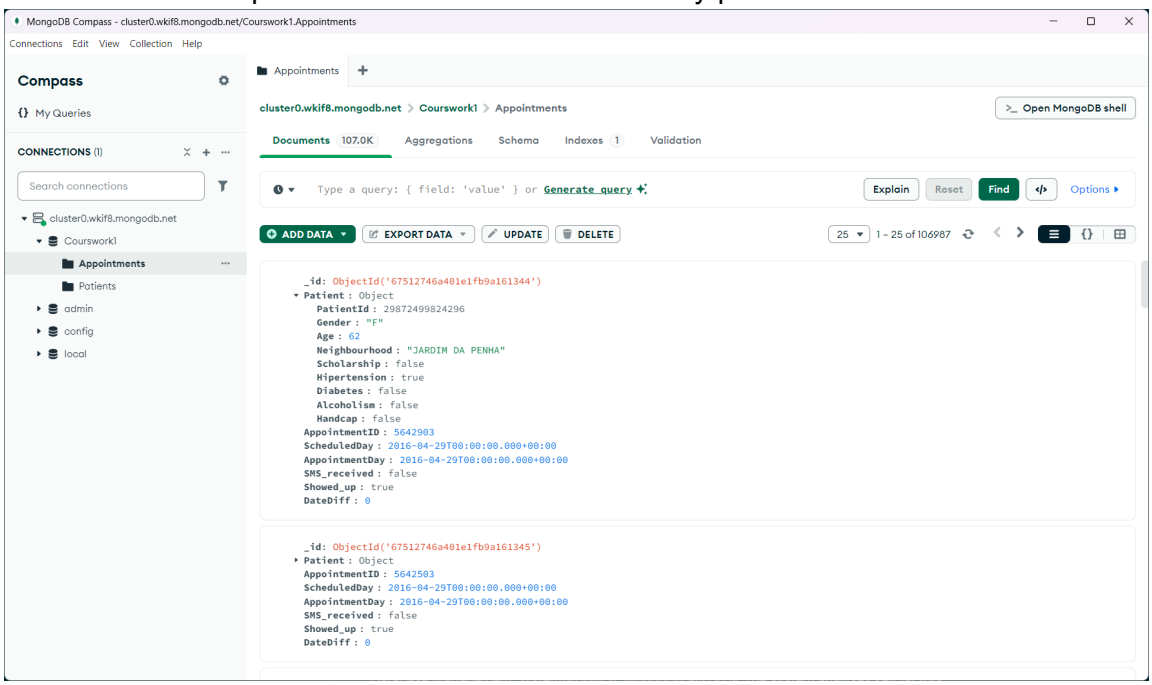


Figure 10 - MongoDB Appointments Collection

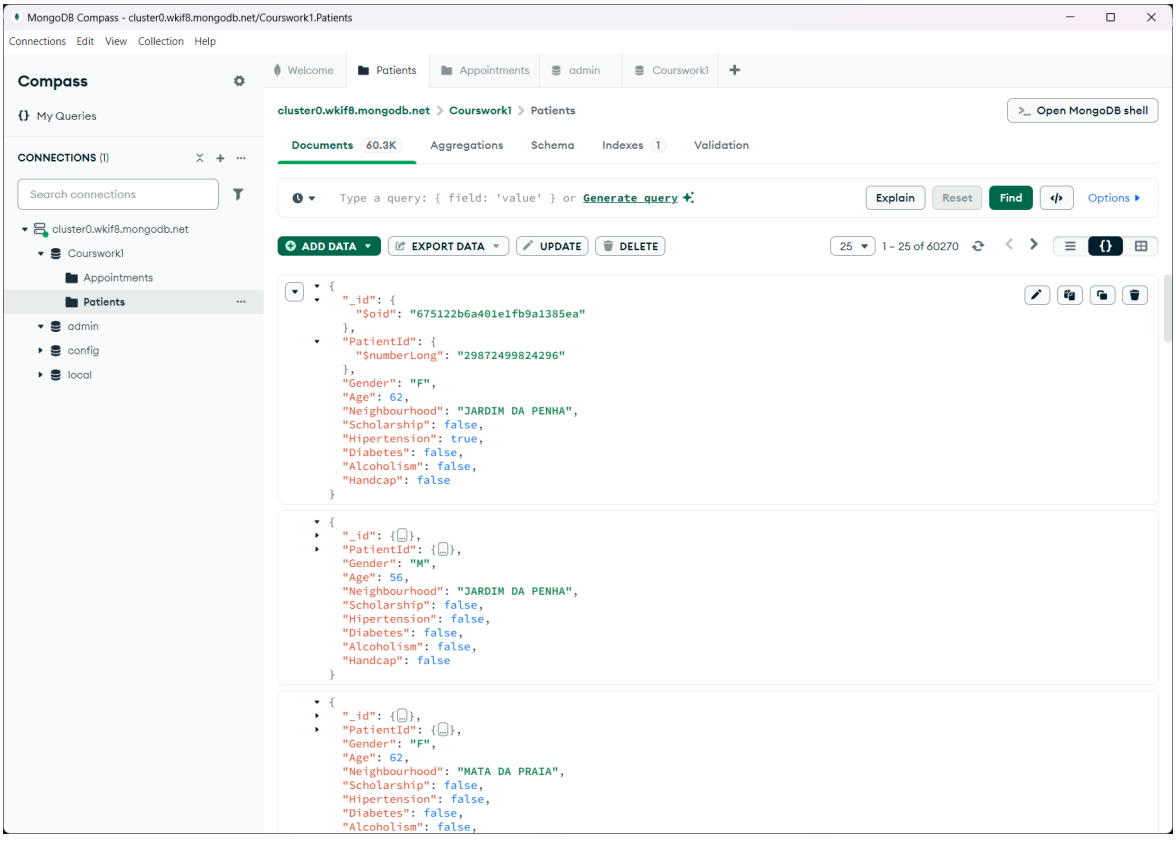


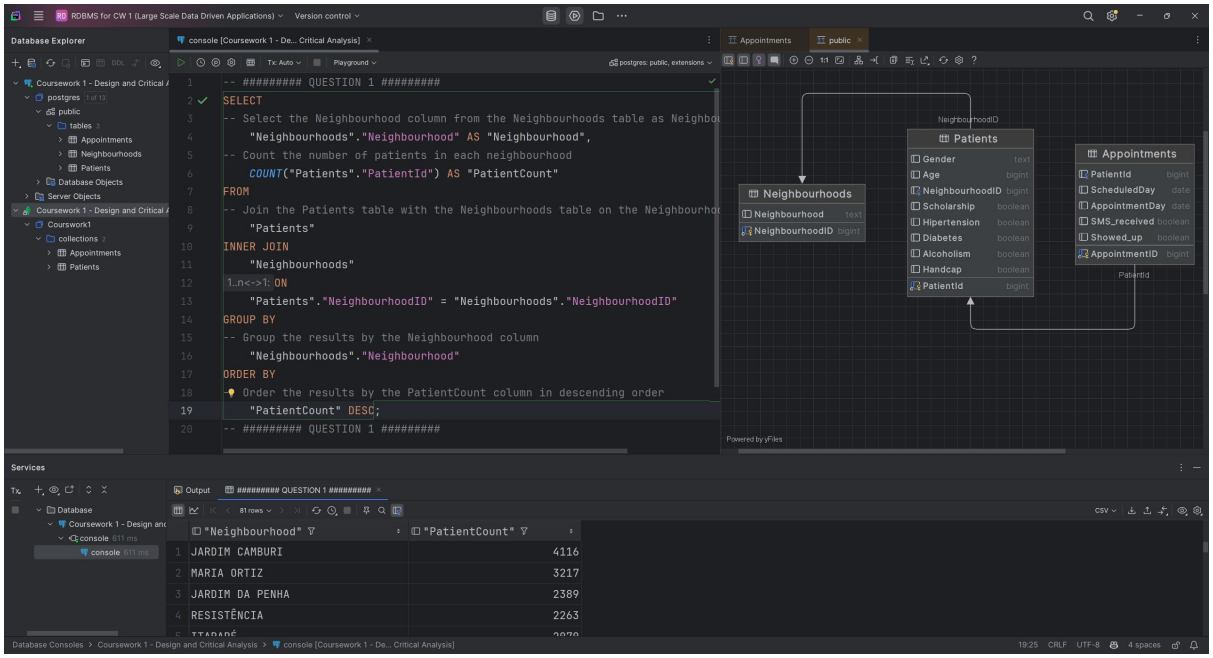
Figure 11 - MongoDB Patients Collection

Figure 12 - MongoDB Scheme Visualiser

Analysis of Data and Reflection

“How many patients are registered in each neighbourhood?”

SQL Database



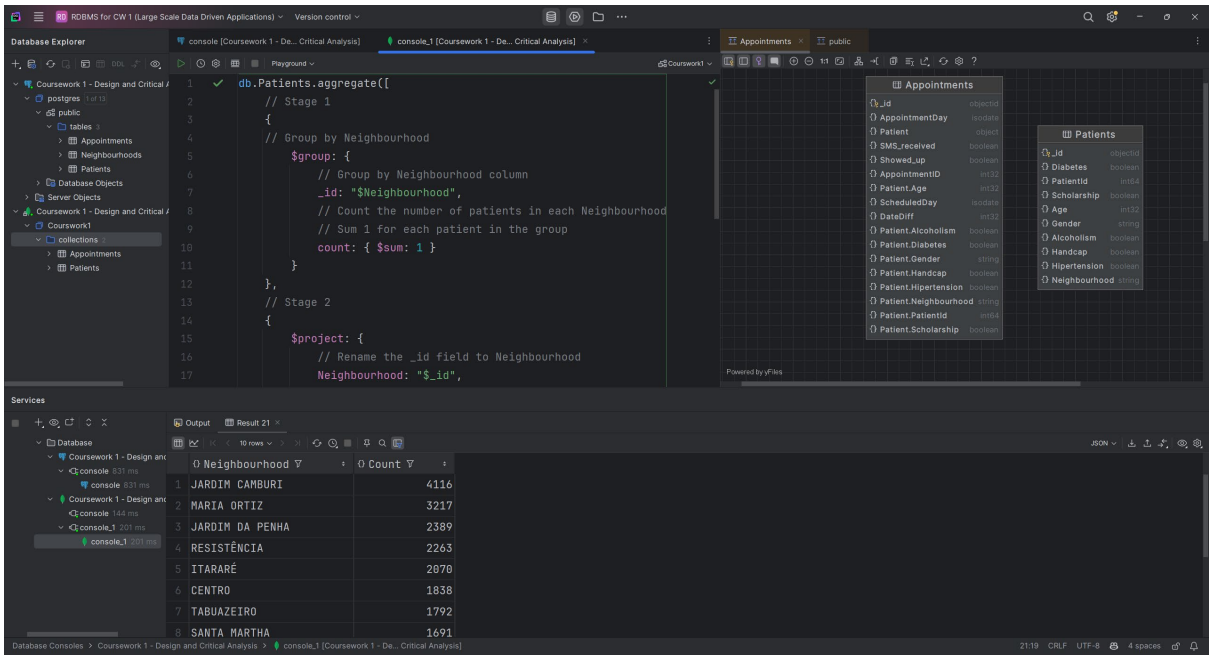
SQL Code

```
-- ######### QUESTION 1 #########
SELECT
-- Select the Neighbourhood column from the Neighbourhoods table as Neighbour-
hood
    "Neighbourhoods"."Neighbourhood" AS "Neighbourhood",
-- Count the number of patients in each neighbourhood
    COUNT("Patients"."PatientId") AS "PatientCount"
FROM
-- Join the Patients table with the Neighbourhoods table on the Neighbour-
hoodID column
    "Patients"
INNER JOIN
    "Neighbourhoods"
ON
    "Patients"."NeighbourhoodID" = "Neighbourhoods"."NeighbourhoodID"
GROUP BY
-- Group the results by the Neighbourhood column
    "Neighbourhoods"."Neighbourhood"
ORDER BY
-- Order the results by the PatientCount column in descending order
    "PatientCount" DESC;
-- ######### QUESTION 1 #########
```

SQL Result (LIMIT 10)

Neighbourhood	PatientCount
JARDIM CAMBURI	4116
MARIA ORTIZ	3217
JARDIM DA PENHA	2389
RESISTÊNCIA	2263
ITARARÉ	2070
CENTRO	1838
TABUAZEIRO	1792
SANTA MARTHA	1691
SANTO ANTÔNIO	1606
BONFIM	1565

MongoDB Database



MongoDB Code

```
// ######### QUESTION 1 #########
db.Patients.aggregate([
    // Stage 1
    {
    // Group by Neighbourhood
        $group: {
            // Group by Neighbourhood column
            _id: "$Neighbourhood",
            // Count the number of patients in each Neighbourhood
            // Sum 1 for each patient in the group
            count: { $sum: 1 }
        }
    },
    // Stage 2
    {
        $project: {
            // Rename the _id field to Neighbourhood
            Neighbourhood: "$_id",
            // Include the count field
            Count: "$count",
            // Exclude the _id field
            _id: 0
        }
    },
    // Stage 3
    {
        // Sort by count in descending order
        $sort: { Count: -1 }
    }
])
// ######### QUESTION 1 #########
```

MongoDB Result (Limit 9)

Count	Neighbourhood
4116	JARDIM CAMBURI
3217	MARIA ORTIZ
2389	JARDIM DA PENHA
2263	RESISTÊNCIA
2070	ITARARÉ
1838	CENTRO
1792	TABUAZEIRO
1691	SANTA MARTHA
1606	SANTO ANTÔNIO

Reflection

The results for both databases are the same. The greatest number of patients are from “JARDIM CAMBURI” while the least are from “PARQUE INDUSTRIAL” with only 1 patient.

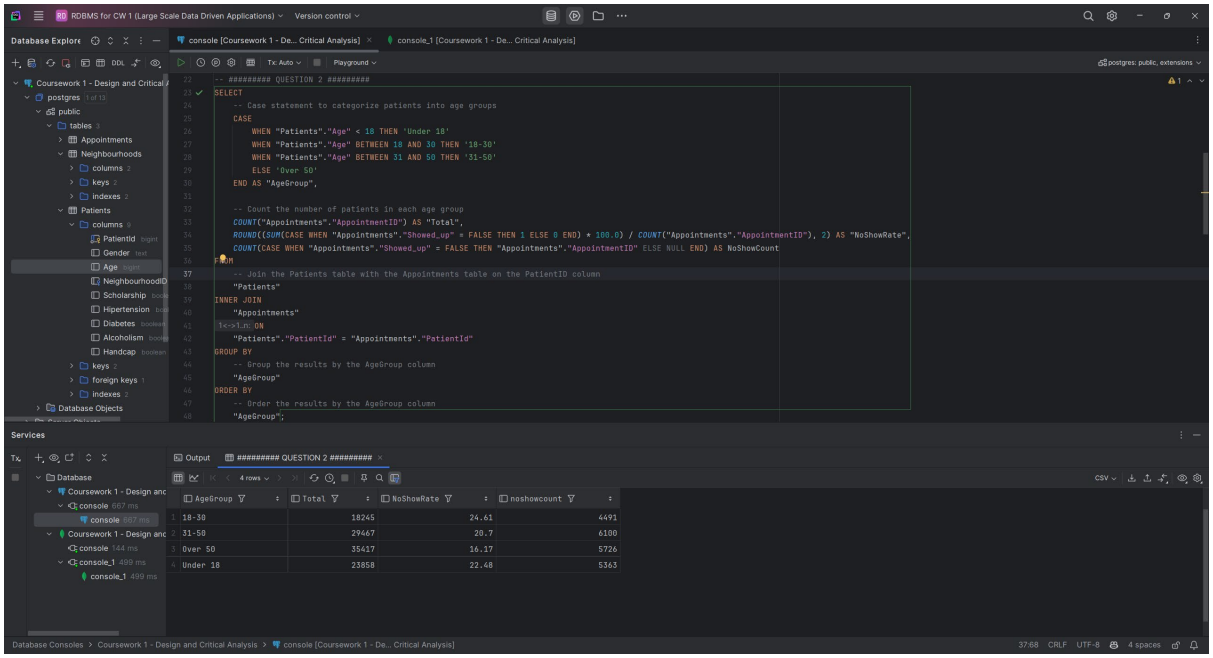
The location of the clinic is unknown but this could indicate that the locations listed below are far away from the clinic or there could be a lack of transportation in those areas making it harder to access medical services.

Count	Neighbourhood
1	PARQUE INDUSTRIAL
2	ILHAS OCEÂNICAS DE TRINDADE
5	ILHA DO FRADE
7	AEROPORTO
22	ILHA DO BOI

The other areas with a higher count may be closer to the clinic and have better transportation access in the area to reach the clinic. One way to mitigate this problem may be to implement better transport in the area or allow for a way to have remote clinic “visits”

“What are the no-show rates categorised by age groups (e.g., under 18, 18-30, 31-50, over 50)?”

SQL Database



SQL Code

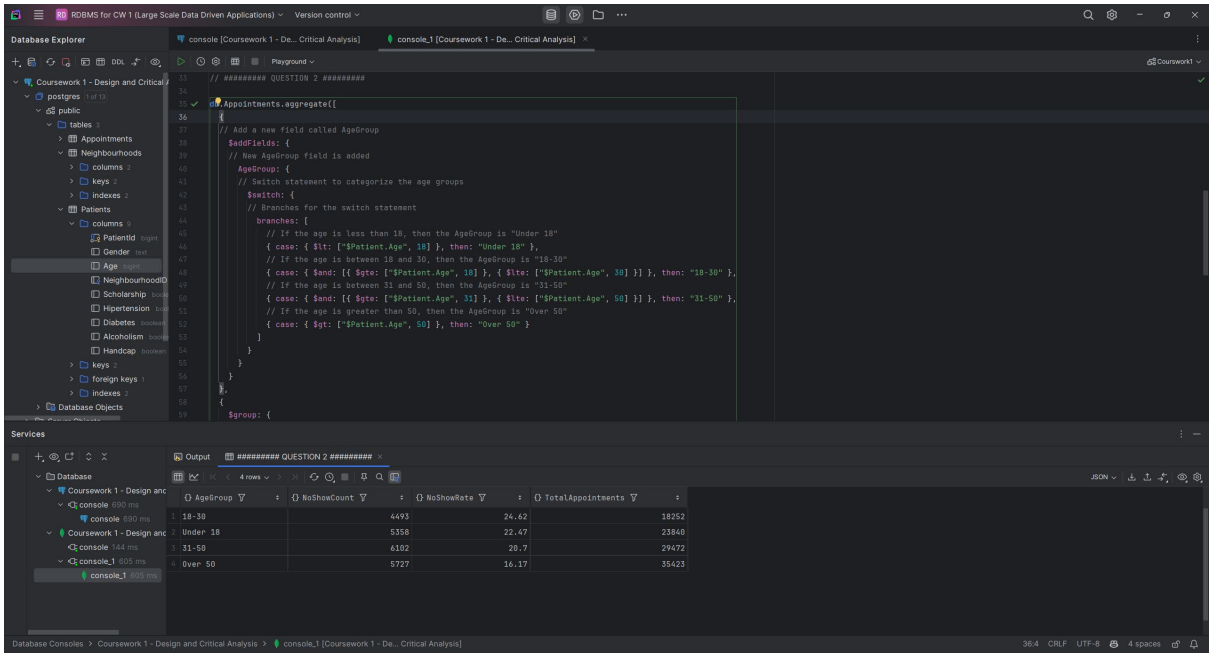
```
-- ######### QUESTION 2 #########
SELECT
    -- Case statement to categorize patients into age groups
    CASE
        WHEN "Patients"."Age" < 18 THEN 'Under 18'
        WHEN "Patients"."Age" BETWEEN 18 AND 30 THEN '18-30'
        WHEN "Patients"."Age" BETWEEN 31 AND 50 THEN '31-50'
        ELSE 'Over 50'
    END AS "AgeGroup",

    -- Count the number of patients in each age group
    COUNT("Appointments"."AppointmentID") AS "Total",
    ROUND((SUM(CASE WHEN "Appointments"."Showed_up" = FALSE THEN 1 ELSE 0 END)
* 100.0) / COUNT("Appointments"."AppointmentID"), 2) AS "NoShowRate",
    COUNT(CASE WHEN "Appointments"."Showed_up" = FALSE THEN
"Appointments"."AppointmentID" ELSE NULL END) AS “NoShowCount”
FROM
    -- Join the Patients table with the Appointments table on the PatientID
column
    "Patients"
INNER JOIN
    "Appointments"
ON
    "Patients"."PatientId" = "Appointments"."PatientId"
GROUP BY
    -- Group the results by the AgeGroup column
    "AgeGroup"
ORDER BY
    -- Order the results by the AgeGroup column
    "AgeGroup";
-- ######### QUESTION 2 #########
```

SQL Result

AgeGroup	Total	NoShowRate	NoShowCount
18-30	18245	24.61	4491
31-50	29467	20.7	6100
Over 50	35417	16.17	5726
Under 18	23858	22.48	5363

MongoDB Database



MongoDB Code

```
// ########## QUESTION 2 ##########
db.Appointments.aggregate([
    {
    // Add a new field called AgeGroup
      $addFields: {
      // New AgeGroup field is added
        AgeGroup: {
        // Switch statement to categorize the age groups
          $switch: {
          // Branches for the switch statement
            branches: [
              // If the age is less than 18, then the AgeGroup is "Under 18"
              { case: { $lt: ["$Patient.Age", 18] }, then: "Under 18" },
              // If the age is between 18 and 30, then the AgeGroup is "18-30"
              { case: { $and: [{ $gte: ["$Patient.Age", 18] }, { $lte: ["$Patient.Age", 30] }] }, then: "18-30" },
              // If the age is between 31 and 50, then the AgeGroup is "31-50"
              { case: { $and: [{ $gte: ["$Patient.Age", 31] }, { $lte: ["$Patient.Age", 50] }] }, then: "31-50" },
              // If the age is greater than 50, then the AgeGroup is "Over 50"
              { case: { $gt: ["$Patient.Age", 50] }, then: "Over 50" }
            ]
          }
        }
      }
    },
```

```
 {
      $group: {
        // Group by the AgeGroup field
        _id: "$AgeGroup",
        // Count the total number of appointments in each AgeGroup
        TotalAppointments: { $sum: 1 },
        // Count the number of no-show appointments in each AgeGroup
        // NoShowCount will be incremented by 1 if Showed_up is false
        // NoShowCount will be incremented by 0 if Showed_up is true
        NoShowCount: { $sum: { $cond: [{ $eq: ["$Showed_up", false] }, 1, 0] }
}
      }
    },
    {
      // Add another field for the NoShowRate
      $addFields: {
        // Calculate the NoShowRate by dividing the NoShowCount by the
TotalAppointments
        NoShowRate: {
          // Round the NoShowRate to 2 decimal places
          $round: [
            { $multiply: [{ $divide: ["$NoShowCount", "$TotalAppointments"] },
100] },
            2
          ]
        }
      }
    },
    {
      // Project the fields to include in the output
      $project: {
        // Exclude the _id field
        _id: 0,
        // Include the AgeGroup field
        AgeGroup: "$_id",
        // Include the rest of the relevant fields
        TotalAppointments: 1,
        NoShowCount: 1,
        NoShowRate: 1
      }
    }
  ]);
  // ######### QUESTION 2 #########
```

MongoDB Result

AgeGroup	NoShowCount	NoShowRate	TotalAppointments
Over 50	5727	16.17	35423
31-50	6102	20.7	29472
Under 18	5358	22.47	23840
18-30	4493	24.62	18252

Reflection

In this instance the same results were not obtained from both databases. The correct answer was obtained from MongoDB but the result from the SQL database is wrong even if the values are very similar. The database for the SQL contains the correct number of rows for patient data when queried (106987 results) but when the conditional statement is added the result returned is wrong.

I tested this with the patient count for those under 18 in the appointments table and it would return 18 more values. This could be because I designed the database wrong or the query is incorrect.

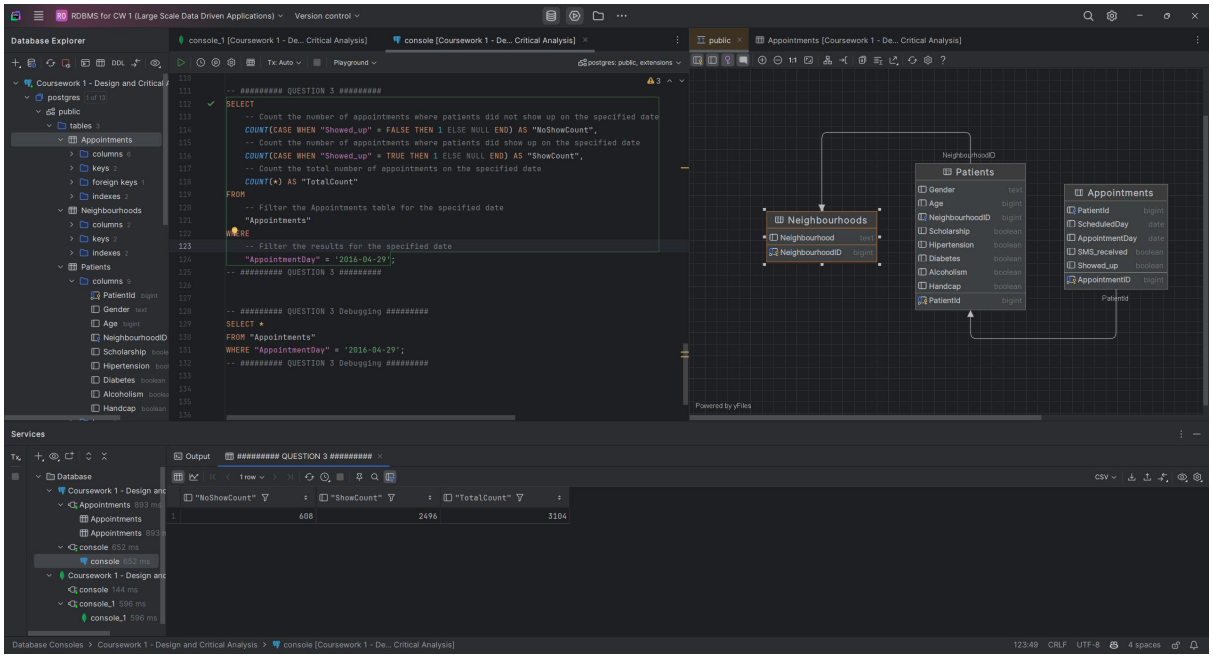
The data obtained from the MongoDB query indicates the highest no-show rate is 24.62% from the 18-30 age group. The lowest is for those over the age of 50 at 16.17%. The group with the highest no-show rate has the lowest number of total appointments. This could be because of many reasons such as work or school commitments.

The group over the age of 50 may have health conditions that have a greater effect on them which is why they are more likely to show up for their appointments at the clinic whereas the younger group may not feel the need to go as they are young.

A potential fix for this problem could be to book appointments at specific times for certain age groups. For example, the group 18-30 could be busy at work or with their studies at a certain time and a system could be implemented to avoid these hours.

What is the total number of no-shows on a particular date?

SQL Database



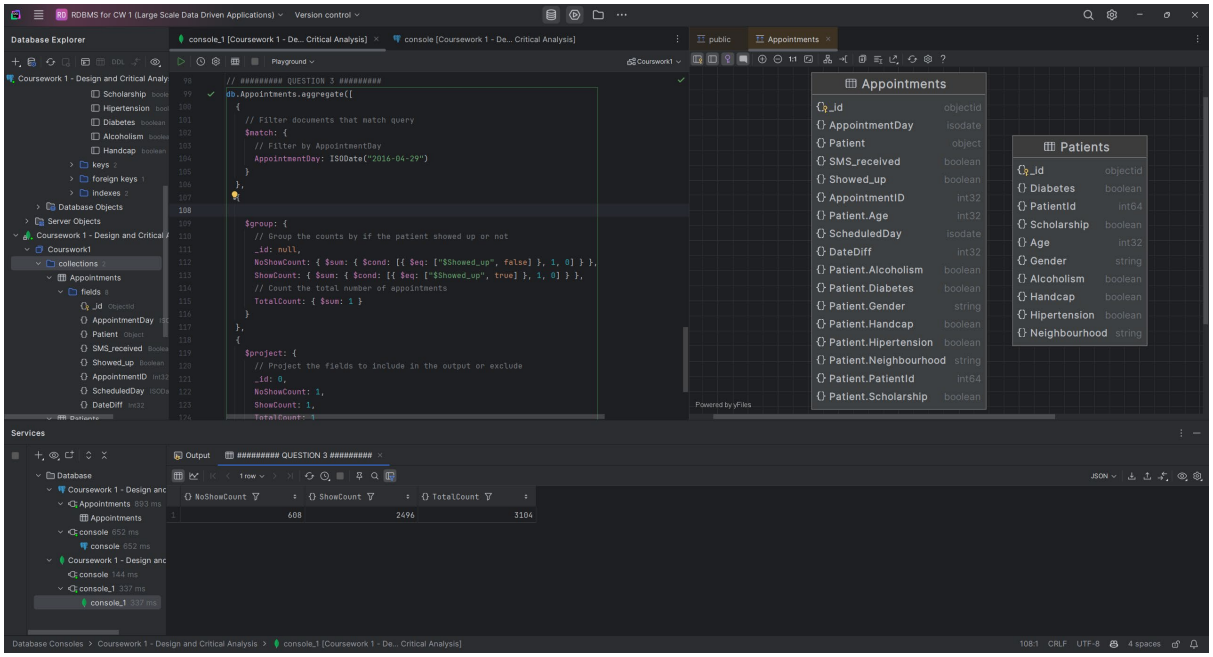
SQL Code

```
-- ######### QUESTION 3 #########
SELECT
    -- Count the number of appointments where patients did not show up on the
specified date
    COUNT(CASE WHEN "Showed_up" = FALSE THEN 1 ELSE NULL END) AS
"NoShowCount",
    -- Count the number of appointments where patients did show up on the
specified date
    COUNT(CASE WHEN "Showed_up" = TRUE THEN 1 ELSE NULL END) AS "ShowCount",
    -- Count the total number of appointments on the specified date
    COUNT(*) AS "TotalCount"
FROM
    -- Filter the Appointments table for the specified date
    "Appointments"
WHERE
    -- Filter the results for the specified date
    "AppointmentDay" = '2016-04-29';
-- ######### QUESTION 3 #########
```

SQL Result

NoShowCount	ShowCount	TotalCount
608	2496	3104

MongoDB Database



MongoDB Code

```
// ######### QUESTION 3 #########
db.Appointments.aggregate([
    {
      // Filter documents that match query
      $match: {
        // Filter by AppointmentDay
        AppointmentDay: ISODate("2016-04-29")
      }
    },
    {

      $group: {
        // Group the counts by if the patient showed up or not
        _id: null,
        NoShowCount: { $sum: { $cond: [{ $eq: ["$Showed_up", false] }, 1, 0] }
},
        ShowCount: { $sum: { $cond: [{ $eq: ["$Showed_up", true] }, 1, 0] } },
        // Count the total number of appointments
        TotalCount: { $sum: 1 }
      }
    },
    {
      $project: {
        // Project the fields to include in the output or exclude
        _id: 0,
        NoShowCount: 1,
        ShowCount: 1,
        TotalCount: 1
      }
    }
  ]);
  // ######### QUESTION 3 #########
```

MongoDB Result

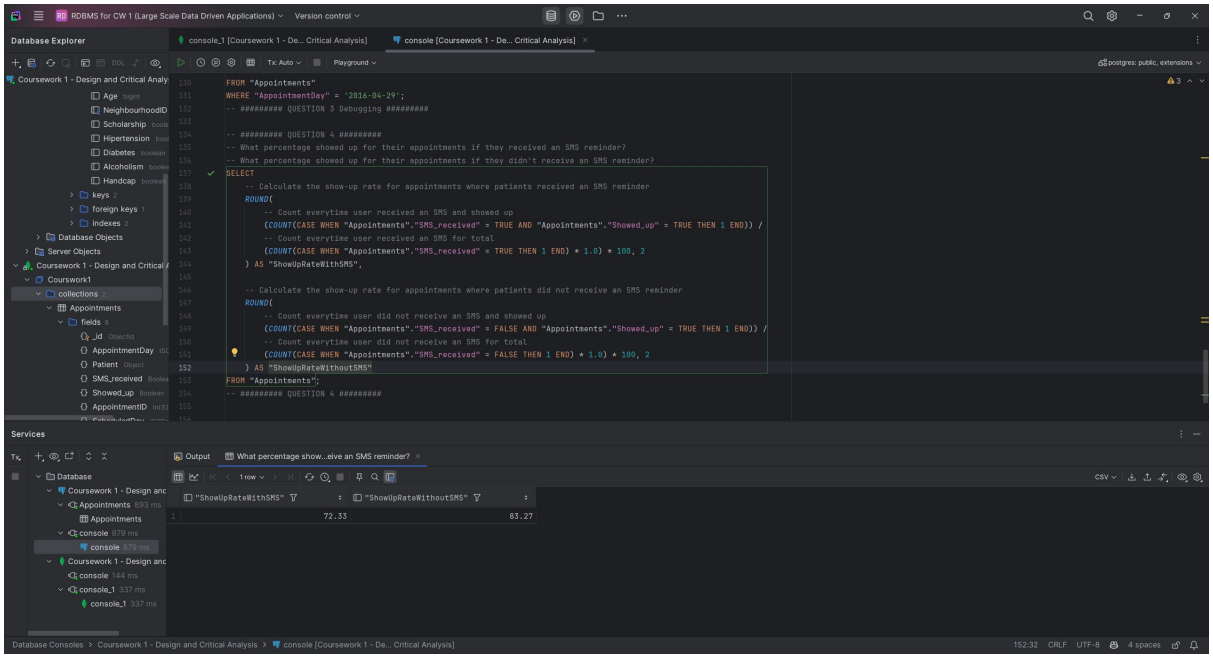
```
+-----------+---------+----------+
|NoShowCount|ShowCount|TotalCount|
+-----------+---------+----------+
|608        |2496     |3104      |
+-----------+---------+----------+
```

Reflection

The results for both of the databases are the same for the date “29-04-2016”. From the data that was obtained from this particular day we can see that there was a total of 3104 appointments and 608 did not show up. 19.58% did not show up to their appointment on this day. This result is similar to the previous question. There would also be a variety of other reasons as to why this particular day had 19.58% of people not show up to their appointment such as bad weather.

What percentage of patients who received SMS reminders showed up for their appointments compared to those who did not receive reminders?

SQL Database



SQL Code

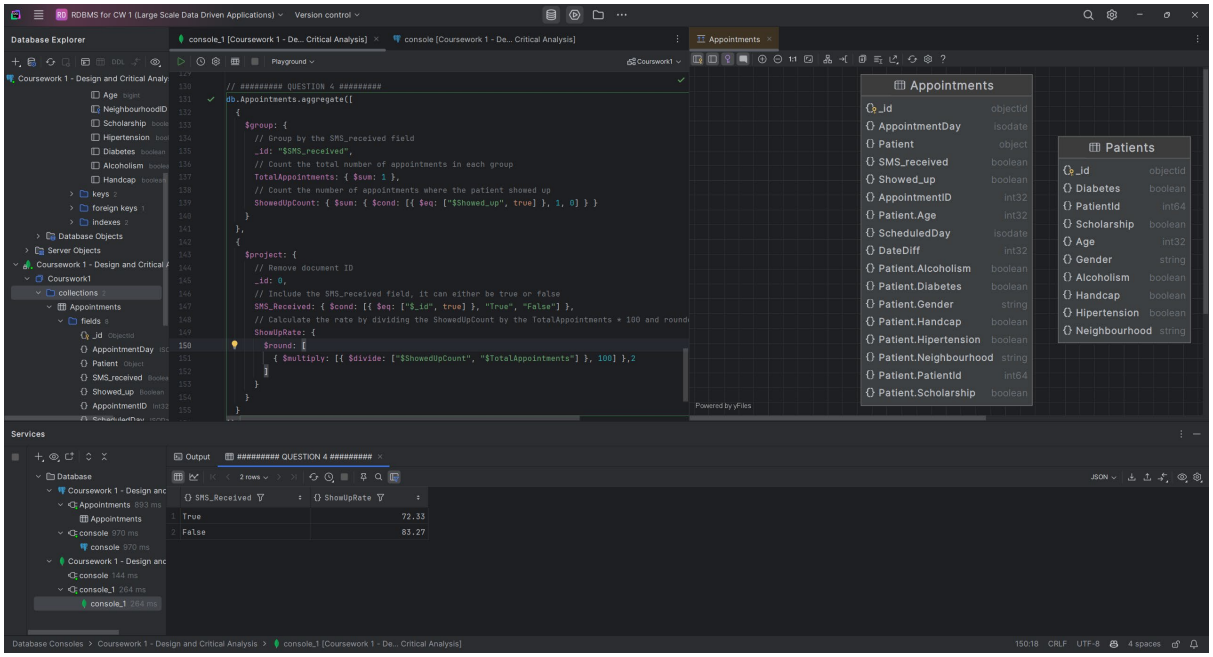
```sql
-- ######### QUESTION 4 #########
-- What percentage showed up for their appointments if they received an SMS
reminder?
-- What percentage showed up for their appointments if they didn't receive an
SMS reminder?
SELECT
    -- Calculate the show-up rate for appointments where patients received an
SMS reminder
    ROUND(
        -- Count everytime user received an SMS and showed up
        (COUNT(CASE WHEN "Appointments"."SMS_received" = TRUE AND
"Appointments"."Showed_up" = TRUE THEN 1 END)) /
        -- Count everytime user received an SMS for total
        (COUNT(CASE WHEN "Appointments"."SMS_received" = TRUE THEN 1 END) *
1.0) * 100, 2
    ) AS "ShowUpRateWithSMS",

    -- Calculate the show-up rate for appointments where patients did not
receive an SMS reminder
    ROUND(
        -- Count everytime user did not receive an SMS and showed up
        (COUNT(CASE WHEN "Appointments"."SMS_received" = FALSE AND
"Appointments"."Showed_up" = TRUE THEN 1 END)) /
        -- Count everytime user did not receive an SMS for total
        (COUNT(CASE WHEN "Appointments"."SMS_received" = FALSE THEN 1 END) *
1.0) * 100, 2
    ) AS "ShowUpRateWithoutSMS"
FROM "Appointments";
-- ######### QUESTION 4 #########
```

SQL Result

ShowUpRateWithSMS	ShowUpRateWithoutSMS
72.33	83.27

MongoDB Database



MongoDB Code

```
// ######### QUESTION 4 #########
db.Appointments.aggregate([
  {
    $group: {
      // Group by the SMS_received field
      _id: "$SMS_received",
      // Count the total number of appointments in each group
      TotalAppointments: { $sum: 1 },
      // Count the number of appointments where the patient showed up
      ShowedUpCount: { $sum: { $cond: [{ $eq: ["$Showed_up", true] }, 1, 0]
} }
    }
  },
  {
    $project: {
      // Remove document ID
      _id: 0,
      // Include the SMS_received field, it can either be true or false
      SMS_Received: { $cond: [{ $eq: ["$_id", true] }, "True", "False"] },
      // Calculate the rate by dividing the ShowedUpCount by the
TotalAppointments * 100 and rounded to 2 decimal places
      ShowUpRate: {
        $round: [
          { $multiply: [{ $divide: ["$ShowedUpCount", "$TotalAppointments"]
}, 100] },2
        ]
      }
    }
  }
]);
// ######### QUESTION 4 #########
```

MongoDB Result

SMS_Received	ShowUpRate
True	72.33
False	83.27

Reflection

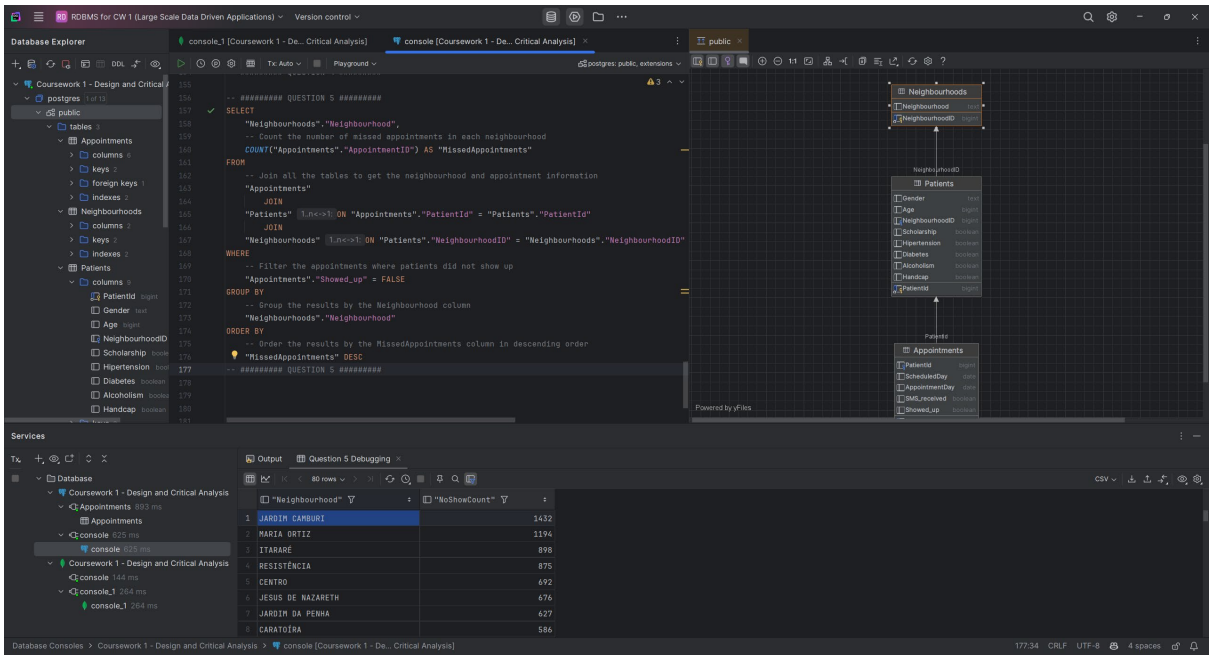
The results surprisingly show that users that did not receive SMS had a higher show up rate for their appointments. This could be for a few different reasons such as demographic. More younger people are more likely to receive SMS as they may be more technologically advanced than the older generation and that demographic has the highest no show-up rate.

There could also be technical issues such as old phone numbers so the messages are being undelivered or messages being sent to spam. Other alternatives could be used for scenarios like this such as email notifications.

Users may also be more desensitised to messages because most cooperate messages can be non-personal and lack attention to detail. A more personalised message may be more welcoming and get more patients into the clinic if they have booked appointments. For cases like this AI may be a great tool that can be used to personalise SMS messages more for patients and it can also remind them of why it is important for them to come to their appointment. Frequency of messages can also be important, if a user is busy and the frequency of messages is low, they may be more likely to forget about their appointment. One solution to this problem would be to increase the messaging frequency.

Which neighbourhoods have the highest number of no-shows?

SQL Database



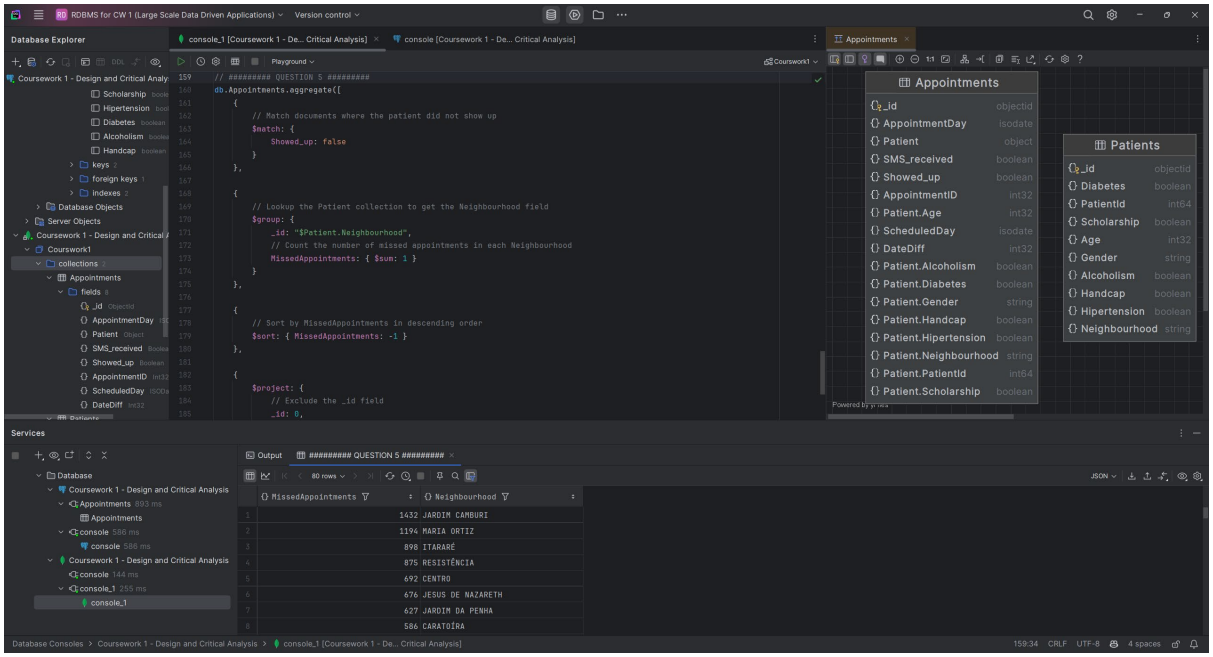
SQL Code

```
-- ######### QUESTION 5 #########
SELECT
    "Neighbourhoods"."Neighbourhood",
    -- Count the number of missed appointments in each neighbourhood
    COUNT("Appointments"."AppointmentID") AS "MissedAppointments"
FROM
    -- Join all the tables to get the neighbourhood and appointment
information
    "Appointments"
        JOIN
    "Patients" ON "Appointments"."PatientId" = "Patients"."PatientId"
        JOIN
    "Neighbourhoods" ON "Patients"."NeighbourhoodID" =
"Neighbourhoods"."NeighbourhoodID"
WHERE
    -- Filter the appointments where patients did not show up
    "Appointments"."Showed_up" = FALSE
GROUP BY
    -- Group the results by the Neighbourhood column
    "Neighbourhoods"."Neighbourhood"
ORDER BY
    -- Order the results by the MissedAppointments column in descending order
    "MissedAppointments" DESC
-- ######### QUESTION 5 #########
```

SQL Result (Limit 16)

Neighbourhood	MissedAppointments
JARDIM CAMBURI	1432
MARIA ORTIZ	1194
ITARARÉ	898
RESISTÊNCIA	875
CENTRO	692
JESUS DE NAZARETH	676
JARDIM DA PENHA	627
CARATOÍRA	586
TABUAZEIRO	552
BONFIM	537
ILHA DO PRÍNCIPE	525
ANDORINHAS	509
SÃO PEDRO	497
SANTA MARTHA	489
SANTO ANDRÉ	486
SANTO ANTÔNIO	475

MongoDB Database



MongoDB Code

```
// ######### QUESTION 5 #########
db.Appointments.aggregate([
    {
        // Match documents where the patient did not show up
        $match: {
            Showed_up: false
        }
    },
    {
        // Lookup the Patient collection to get the Neighbourhood field
        $group: {
            _id: "$Patient.Neighbourhood",
            // Count the number of missed appointments in each
Neighbourhood
            MissedAppointments: { $sum: 1 }
        }
    },
    {
        // Sort by MissedAppointments in descending order
        $sort: { MissedAppointments: -1 }
    },
    {
        $project: {
            // Exclude the _id field
            _id: 0,
            // List the Neighbourhood field
            Neighbourhood: "$_id",
            // List the MissedAppointments
            MissedAppointments: 1
        }
    }
])
// ######### QUESTION 5 #########
```

MongoDB Result

MissedAppointments	Neighbourhood
1432	JARDIM CAMBURI
1194	MARIA ORTIZ
898	ITARARÉ
875	RESISTÊNCIA
692	CENTRO
676	JESUS DE NAZARETH
627	JARDIM DA PENHA
586	CARATOÍRA
552	TABUAZEIRO
537	BONFIM
525	ILHA DO PRÍNCIPE

Reflection

The highest number of no-shows are from “JARDIM CAMBURI”. “MARIA ORTIZ” is also another neighbourhood with missed appointments over 1000. This could be because of the geographical locations of these areas which makes it harder for people here to access the clinic.

The area could also potentially have financial issues which makes it harder to go to the clinic. Whether that is paying for the services of the clinic or transport costs.

To mitigate these problems transport in these areas could be improved or more reminders could be sent to these areas specifically.

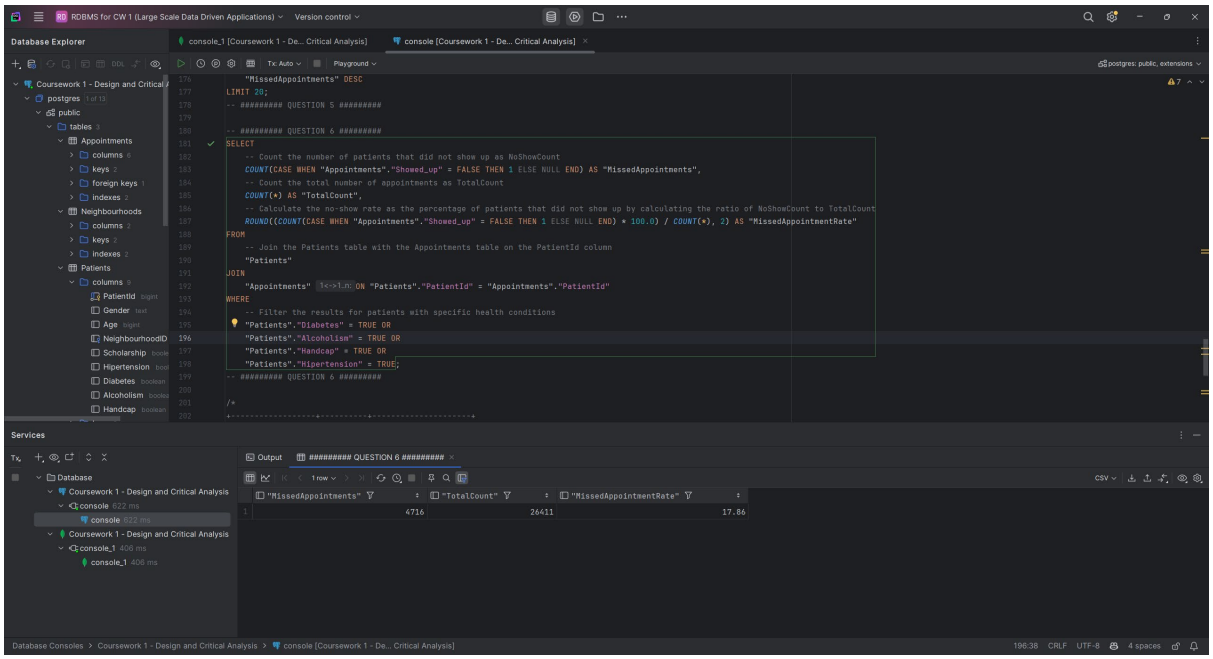
The areas with the lowest number of no-shows are below:

MissedAppointments	Neighbourhood
1	AEROPORTO
2	ILHA DO FRADE
2	ILHAS OCEÂNICAS DE TRINDADE
3	ILHA DO BOI
12	PONTAL DE CAMBURI
15	MORADA DE CAMBURI
26	SEGURANÇA DO LAR
27	NAZARETH
32	UNIVERSITÁRIO

Areas like “AEROPORTO” only had 1 appointment booked in the year. These areas may have access to another clinic in the area which may explain why there are so little appointments made at this specific clinic. It may be better to check what neighbourhood people are from so suggestions can be made to them to re-direct them to another clinic.

What is the no-show rate among patients with hypertension, diabetes, or other chronic conditions?

SQL Database



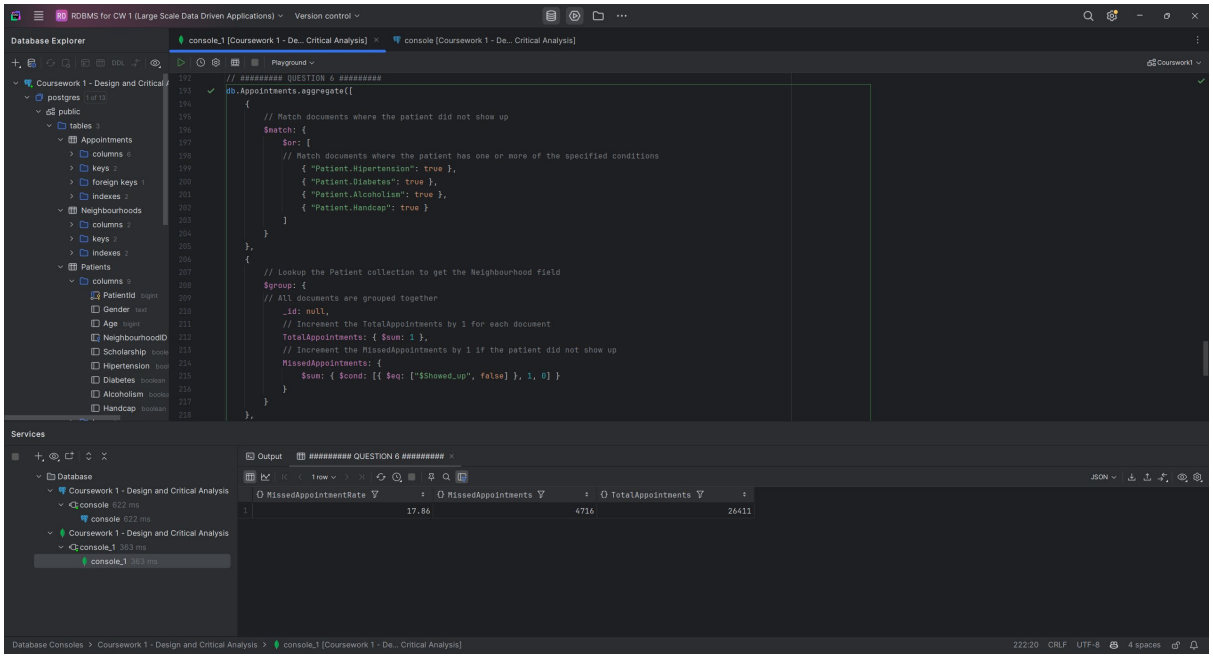
SQL Code

```
-- ######### QUESTION 6 #########
SELECT
    -- Count the number of patients that did not show up as NoShowCount
    COUNT(CASE WHEN "Appointments"."Showed_up" = FALSE THEN 1 ELSE NULL
END) AS "MissedAppointments",
    -- Count the total number of appointments as TotalCount
    COUNT(*) AS "TotalCount",
    -- Calculate the no-show rate as the percentage of patients that did
not show up by calculating the ratio of NoShowCount to TotalCount
    ROUND((COUNT(CASE WHEN "Appointments"."Showed_up" = FALSE THEN 1 ELSE
NULL END) * 100.0) / COUNT(*), 2) AS "MissedAppointmentRate"
FROM
    -- Join the Patients table with the Appointments table on the PatientId
column
    "Patients"
JOIN
    "Appointments" ON "Patients"."PatientId" = "Appointments"."PatientId"
WHERE
    -- Filter the results for patients with specific health conditions
    "Patients"."Diabetes" = TRUE OR
    "Patients"."Alcoholism" = TRUE OR
    "Patients"."Handcap" = TRUE OR
    "Patients"."Hipertension" = TRUE;
-- ######### QUESTION 6 #########
```

SQL Result

```
+------------------+----------+---------------------+
|MissedAppointments|TotalCount|MissedAppointmentRate|
+------------------+----------+---------------------+
|4716              |26411     |17.86                |
+------------------+----------+---------------------+
```

MongoDB Database



MongoDB Code

```
// ######### QUESTION 6 #########
db.Appointments.aggregate([
    {
        // Match documents where the patient did not show up
        $match: {
            $or: [
            // Match documents where the patient has one or more of the
specified conditions
                { "Patient.Hipertension": true },
                { "Patient.Diabetes": true },
                { "Patient.Alcoholism": true },
                { "Patient.Handcap": true }
            ]
        }
    },
    {
        // Lookup the Patient collection to get the Neighbourhood field
        $group: {
        // All documents are grouped together
            _id: null,
            // Increment the TotalAppointments by 1 for each document
            TotalAppointments: { $sum: 1 },
            // Increment the MissedAppointments by 1 if the patient did not
show up
            MissedAppointments: {
                $sum: { $cond: [{ $eq: ["$Showed_up", false] }, 1, 0] }
            }
        }
    },
    {
        $project: {
            // Exclude the _id field
            _id: 0,
            // List the TotalAppointments
            TotalAppointments: 1,
            // List the MissedAppointments
            MissedAppointments: 1,
            // Calculate the NoShowRate by dividing the MissedAppointments
by the TotalAppointments * 100 and rounded to 2 decimal places
            MissedAppointmentRate: {
                $round: [
                    { $multiply: [{ $divide: ["$MissedAppointments", "$To-
talAppointments"] }, 100] },
                    2
                ]
            }
        }
    }
])
// ######### QUESTION 6 #########
```

MongoDB Result

```
+------------------+----------+-----------------+
|MissedAppointments|NoShowRate|TotalAppointments|
+------------------+----------+-----------------+
|4716              |17.86     |26411            |
+------------------+----------+-----------------+
```

Reflection

Both databases provided the same output. The missed appointment rate amongst those with chronic conditions is high and the clinic must find a way to reduce this number drastically. There are many reasons as to why these patients may be missing their scheduled appointments.

One of the reasons is transport. Those that live in areas with underdeveloped transport may not be able to come to the clinic. Considering the fact that they also have chronic conditions, they may find it difficult to visit without assistance and have to rely on family and friends who may not be available at the scheduled date. To circumvent this issue transport infrastructure could be improved. The other option would be to offer remote consultations for those that cannot attend. This group is at a higher risk because if they miss their appointment, their already terrible condition can get worse.

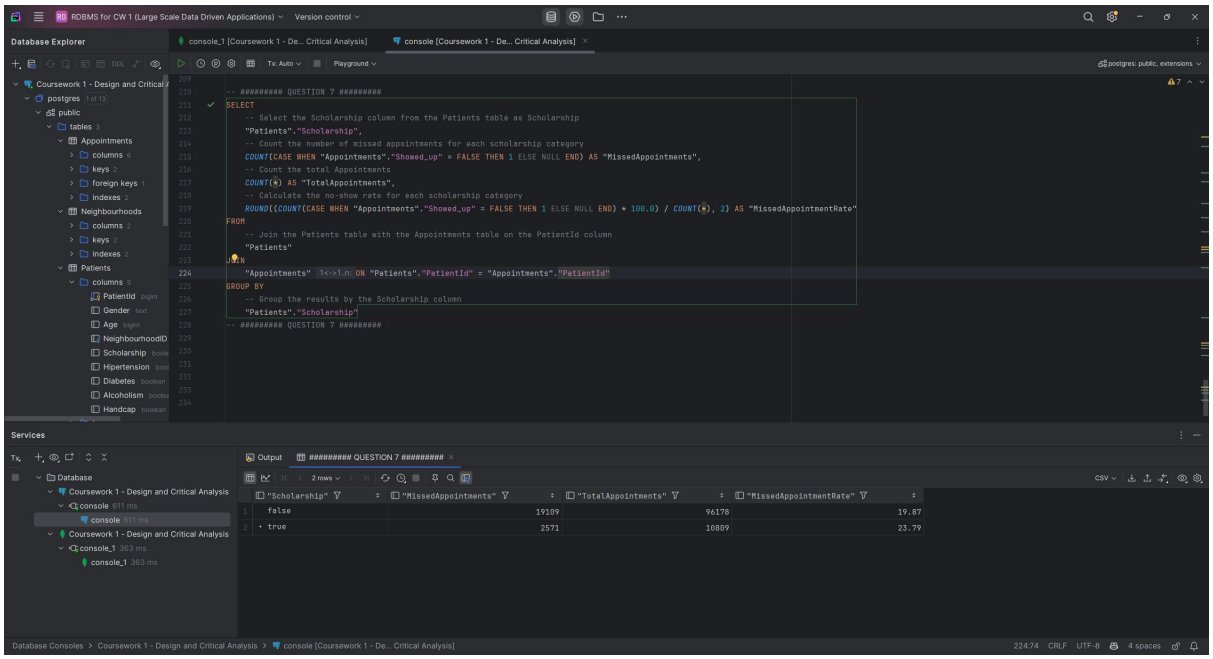
Another reason why these patients could be missing their appointments is that they are not receiving enough reminders. As stated previously, the frequency of reminders could be updated to remind patients to attend. A more personalised message describing why it is important to attend for their specific condition may decrease the no-show rate.

Most people in modern times do own phones, but that won't be the case for very deprived areas of the world. Some groups of people may not have access to communication devices frequently. This could be fixed by using postal services for reminders for patients with chronic conditions.

Another way to decrease this would be to offer a more flexible scheduling system to those with serious conditions. This group of patients requires extra attention and having a system to allow people to cancel their appointments with ease may offer a way for the clinic to offer the missed slot to someone that needs it more urgently. A system could be utilised for cancellations were if an appointment is cancelled, a computer or server could find several patients within a close proximity to offer them the slot instead via a notification system.

Is there a difference in no-show rates between patients with and without scholarships?

SQL Database



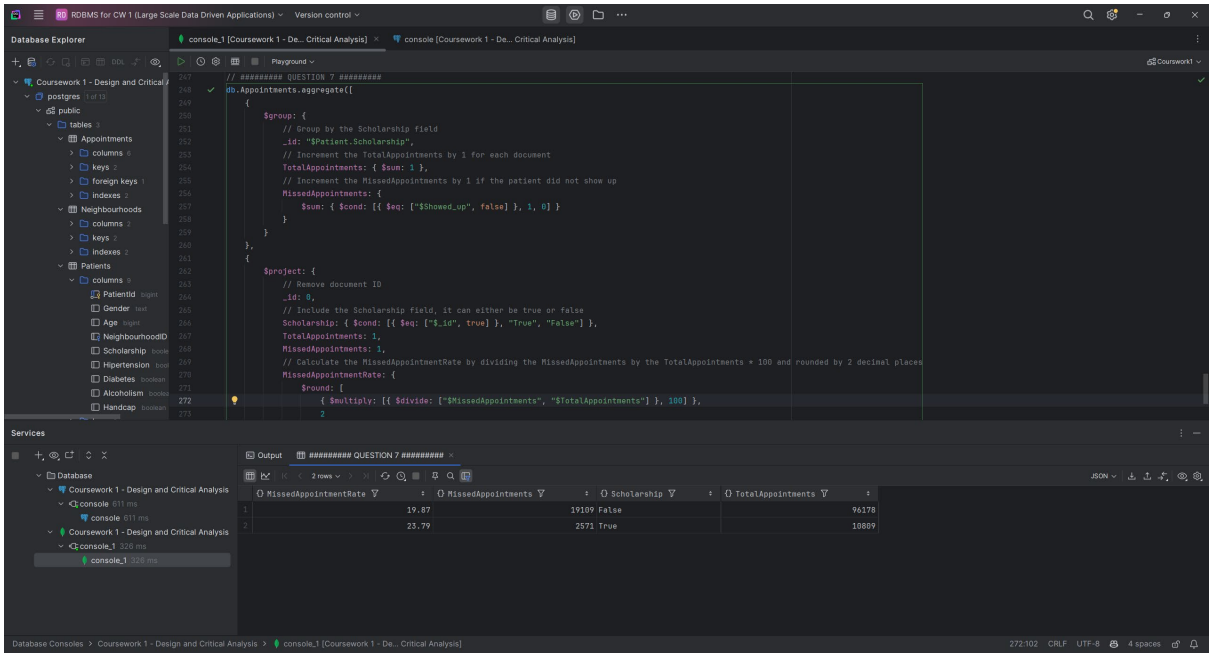
SQL Code

```
-- ######### QUESTION 7 #########
SELECT
    -- Select the Scholarship column from the Patients table as Scholarship
    "Patients"."Scholarship",
    -- Count the number of missed appointments for each scholarship
category
    COUNT(CASE WHEN "Appointments"."Showed_up" = FALSE THEN 1 ELSE NULL
END) AS "MissedAppointments",
    -- Count the total Appointments
    COUNT(*) AS "TotalAppointments",
    -- Calculate the no-show rate for each scholarship category
    ROUND((COUNT(CASE WHEN "Appointments"."Showed_up" = FALSE THEN 1 ELSE
NULL END) * 100.0) / COUNT(*), 2) AS "MissedAppointmentRate"
FROM
    -- Join the Patients table with the Appointments table on the PatientId
column
    "Patients"
JOIN
    "Appointments" ON "Patients"."PatientId" = "Appointments"."PatientId"
GROUP BY
    -- Group the results by the Scholarship column
    "Patients"."Scholarship"
-- ######### QUESTION 7 #########
```

SQL Results

Scholarship	MissedAppointments	TotalAppointments	MissedAppointmentRate
false	19109	96178	19.87
true	2571	10809	23.79

MongoDB Database



MongoDB Code

```
// ######### QUESTION 7 #########
db.Appointments.aggregate([
    {
        $group: {
            // Group by the Scholarship field
            _id: "$Patient.Scholarship",
            // Increment the TotalAppointments by 1 for each document
            TotalAppointments: { $sum: 1 },
            // Increment the MissedAppointments by 1 if the patient did not
show up
            MissedAppointments: {
                $sum: { $cond: [{ $eq: ["$Showed_up", false] }, 1, 0] }}}},
    {
        $project: {
            // Remove document ID
            _id: 0,
            // Include the Scholarship field, it can either be true or
false
            Scholarship: { $cond: [{ $eq: ["$_id", true] }, "True",
"False"] },
            TotalAppointments: 1,
            MissedAppointments: 1,
            MissedAppointmentRate: {
                $round: [
                    { $multiply: [{ $divide: ["$MissedAppointments",
"$TotalAppointments"] }, 100] },
                    2] }}}])
// ######### QUESTION 7 #########
```

MongoDB Result

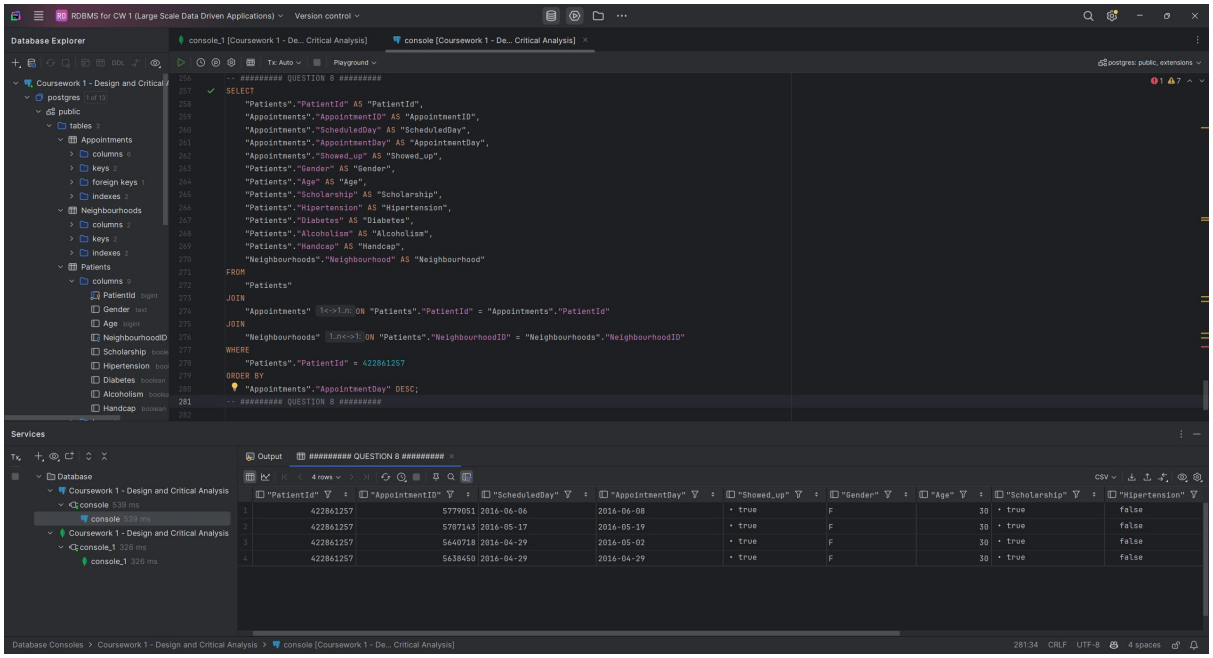
```
+---------------------+------------------+-----------+-----------------+
|MissedAppointmentRate|MissedAppointments|Scholarship|TotalAppointments|
+---------------------+------------------+-----------+-----------------+
|19.87                |19109             |False      |96178            |
|23.79                |2571              |True       |10809            |
+---------------------+------------------+-----------+-----------------+
```

Reflection

Those with scholarships have a very high rate of missed appointments compared to the total appointments. This could be because those with scholarships are busy and often forget about their appointments or don't have the time to go to them. This could be fixed by allowing these patients to re-schedule their appointments or by sending more notifications to this user group. Another way to increase the show-up rate would be to educate people on why it is important to not miss their appointments.

What is the attendance history of a specific patient? You may select any random patient from your database.

SQL Database



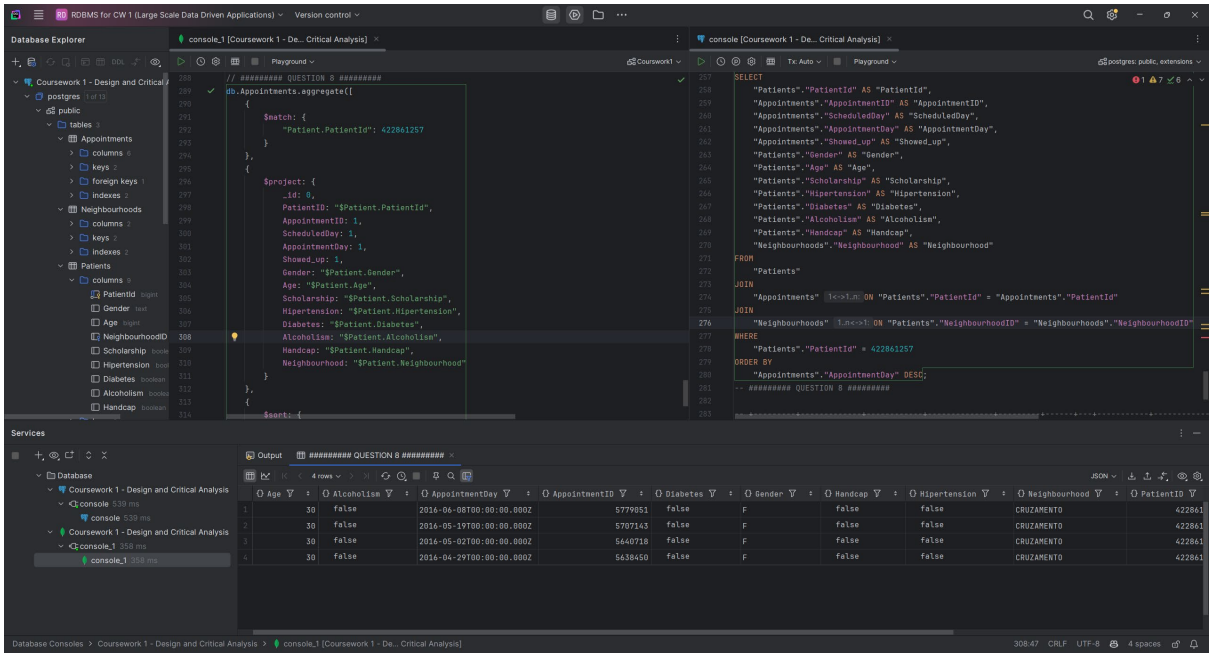
SQL Code

```
-- ######### QUESTION 8 #########
SELECT
    "Patients"."PatientId" AS "PatientId",
    "Appointments"."AppointmentID" AS "AppointmentID",
    "Appointments"."ScheduledDay" AS "ScheduledDay",
    "Appointments"."AppointmentDay" AS "AppointmentDay",
    "Appointments"."Showed_up" AS "Showed_up",
    "Patients"."Gender" AS "Gender",
    "Patients"."Age" AS "Age",
    "Patients"."Scholarship" AS "Scholarship",
    "Patients"."Hipertension" AS "Hipertension",
    "Patients"."Diabetes" AS "Diabetes",
    "Patients"."Alcoholism" AS "Alcoholism",
    "Patients"."Handcap" AS "Handcap",
    "Neighbourhoods"."Neighbourhood" AS "Neighbourhood"
FROM
    "Patients"
JOIN
    "Appointments" ON "Patients"."PatientId" = "Appointments"."PatientId"
JOIN
    "Neighbourhoods" ON "Patients"."NeighbourhoodID" =
"Neighbourhoods"."NeighbourhoodID"
WHERE
    "Patients"."PatientId" = 422861257
ORDER BY
    "Appointments"."AppointmentDay" DESC;
-- ######### QUESTION 8 #########
```

SQL Result

```
+---------+------------+-----------+-------------+---------+------+---+-
---------+-----------+--------+----------+-------+-------------+
|PatientId|AppointmentID|ScheduledDay|AppointmentDay|Showed_up|Gender|Age|S
cholarship|Hipertension|Diabetes|Alcoholism|Handcap|Neighbourhood|
+---------+------------+-----------+-------------+---------+------+---+-
---------+-----------+--------+----------+-------+-------------+
|422861257|5779051     |2016-06-06  |2016-06-08    |true     |F     |30
|true       |false       |false   |false     |false  |CRUZAMENTO   |
|422861257|5707143     |2016-05-17  |2016-05-19    |true     |F     |30
|true       |false       |false   |false     |false  |CRUZAMENTO   |
|422861257|5640718     |2016-04-29  |2016-05-02    |true     |F     |30
|true       |false       |false   |false     |false  |CRUZAMENTO   |
|422861257|5638450     |2016-04-29  |2016-04-29    |true     |F     |30
|true       |false       |false   |false     |false  |CRUZAMENTO   |
+---------+------------+-----------+-------------+---------+------+---+-
---------+-----------+--------+----------+-------+-------------+
```

MongoDB Database



MongoDB Code

```
// ######### QUESTION 8 #########
db.Appointments.aggregate([
    {
        $match: {
            "Patient.PatientId": 422861257
        }
    },
    {
        $project: {
            _id: 0,
            PatientID: "$Patient.PatientId",
            AppointmentID: 1,
            ScheduledDay: 1,
            AppointmentDay: 1,
            Showed_up: 1,
            Gender: "$Patient.Gender",
            Age: "$Patient.Age",
            Scholarship: "$Patient.Scholarship",
            Hipertension: "$Patient.Hipertension",
            Diabetes: "$Patient.Diabetes",
            Alcoholism: "$Patient.Alcoholism",
            Handcap: "$Patient.Handcap",
            Neighbourhood: "$Patient.Neighbourhood"
        }
    },
    {
        $sort: {
            AppointmentDay: -1
        }
    }

])
// ######### QUESTION 8 #########
```

MongoDB Result

```
+---+----------+-----------------------+------------+--------+------+----
---+------------+-------------+---------+-----------------------+---------
--+----------+
|Age|Alcoholism|AppointmentDay
|AppointmentID|Diabetes|Gender|Handcap|Hipertension|Neighbourhood|PatientID
|ScheduledDay            |Scholarship|Showed_up|
+---+----------+-----------------------+------------+--------+------+----
---+------------+-------------+---------+-----------------------+---------
--+----------+
|30 |false     |2016-06-08T00:00:00.000Z|5779051      |false   |F
|false  |false       |CRUZAMENTO   |422861257|2016-06-06T00:00:00.000Z|true
|true     |
|30 |false     |2016-05-19T00:00:00.000Z|5707143      |false   |F
|false  |false       |CRUZAMENTO   |422861257|2016-05-17T00:00:00.000Z|true
|true     |
|30 |false     |2016-05-02T00:00:00.000Z|5640718      |false   |F
|false  |false       |CRUZAMENTO   |422861257|2016-04-29T00:00:00.000Z|true
|true     |
|30 |false     |2016-04-29T00:00:00.000Z|5638450      |false   |F
|false  |false       |CRUZAMENTO   |422861257|2016-04-29T00:00:00.000Z|true
|true     |
+---+----------+-----------------------+------------+--------+------+----
---+------------+-------------+---------+-----------------------+---------
--+----------+
```

Reflection

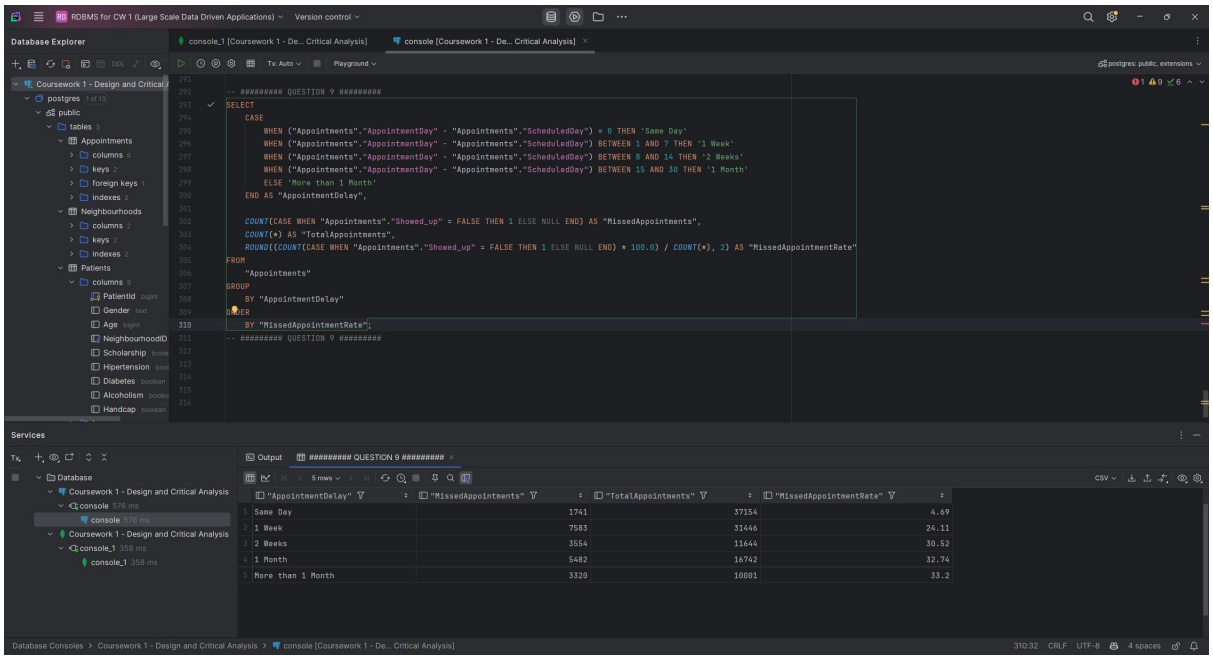
By having the ability to search for individuals by using their patient ID we can increase the efficiency of the workers at the clinic as they will easily have access to information about the person and make the clinic workflow more efficient.

A server would also search for specific individuals and calculate their show-up rate and those with a very low show-up rate could have more notifications sent to them as they are missing too many appointments. This would be a better use of resources instead of spamming everyone at the clinic with notifications at the same frequency.

If patients are booking appointments frequently and missing all of them, they are causing problems for not only the staff but for other patients at the clinic. For these individuals, you could have a system where if they miss too many appointments, they could be given a warning and then fined if they still continue to miss appointments unless they provide a valid reason.

Does the length of time between scheduling and the actual appointment affect whether a patient shows up?

SQL Database



SQL Code

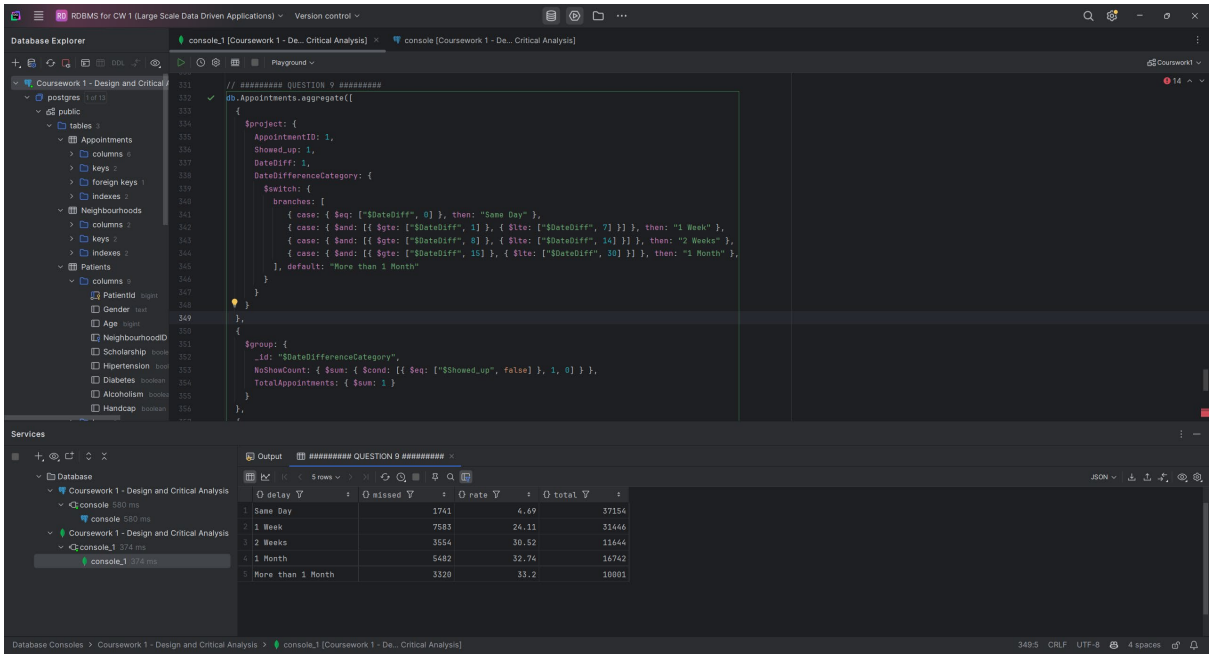
```
-- ######### QUESTION 9 #########
SELECT
    CASE
        WHEN ("Appointments"."AppointmentDay" -
"Appointments"."ScheduledDay") = 0 THEN 'Same Day'
        WHEN ("Appointments"."AppointmentDay" -
"Appointments"."ScheduledDay") BETWEEN 1 AND 7 THEN '1 Week'
        WHEN ("Appointments"."AppointmentDay" -
"Appointments"."ScheduledDay") BETWEEN 8 AND 14 THEN '2 Weeks'
        WHEN ("Appointments"."AppointmentDay" -
"Appointments"."ScheduledDay") BETWEEN 15 AND 30 THEN '1 Month'
        ELSE 'More than 1 Month'
    END AS "AppointmentDelay",

    COUNT(CASE WHEN "Appointments"."Showed_up" = FALSE THEN 1 ELSE NULL
END) AS "MissedAppointments",
    COUNT(*) AS "TotalAppointments",
    ROUND((COUNT(CASE WHEN "Appointments"."Showed_up" = FALSE THEN 1 ELSE
NULL END) * 100.0) / COUNT(*), 2) AS "MissedAppointmentRate"
FROM
    "Appointments"
GROUP
    BY "AppointmentDelay"
ORDER
    BY "MissedAppointmentRate";
-- ######### QUESTION 9 #########
```

SQL Result

```
+------------------+------+-----+-----+
|delay             |missed|total|rate |
+------------------+------+-----+-----+
|Same Day          |1741  |37154|4.69 |
|1 Week            |7583  |31446|24.11|
|2 Weeks           |3554  |11644|30.52|
|1 Month           |5482  |16742|32.74|
|More than 1 Month|3320  |10001|33.2 |
+------------------+------+-----+-----+
```

MongoDB Database



MongoDB Code

```
// ######### QUESTION 9 #########
db.Appointments.aggregate([
  {
    $project: {
      AppointmentID: 1,
      Showed_up: 1,
      DateDiff: 1,
      DateDifferenceCategory: {
        $switch: {
          branches: [
              { case: { $eq: ["$DateDiff", 0] }, then: "Same Day" },
              { case: { $and: [{ $gte: ["$DateDiff", 1] }, { $lte:
["$DateDiff", 7] }] }, then: "1 Week" },
              { case: { $and: [{ $gte: ["$DateDiff", 8] }, { $lte:
["$DateDiff", 14] }] }, then: "2 Weeks" },
              { case: { $and: [{ $gte: ["$DateDiff", 15] }, { $lte:
["$DateDiff", 30] }] }, then: "1 Month" },
          ], default: "More than 1 Month"
        }
      }
    }
  },
  {
    $group: {
      _id: "$DateDifferenceCategory",
      NoShowCount: { $sum: { $cond: [{ $eq: ["$Showed_up", false] }, 1, 0]
} },
      TotalAppointments: { $sum: 1 }
    }
  },
  {
    $project: {
      _id: 0,
      delay: "$_id",
      missed: "$NoShowCount",
      total: "$TotalAppointments",
      rate: {
        $round: [
          { $multiply: [{ $divide: ["$NoShowCount", "$TotalAppointments"]
}, 100] },
          2
        ]
      }
    }
  },
  {
    $sort: { rate: 1 }
  }
]);
// ######### QUESTION 9 #########
```

MongoDB Result

```
+-----------------+------+-----+-----+
|delay            |missed|rate |total|
+-----------------+------+-----+-----+
|Same Day         |1741  |4.69 |37154|
|1 Week           |7583  |24.11|31446|
|2 Weeks          |3554  |30.52|11644|
|1 Month          |5482  |32.74|16742|
|More than 1 Month|3320  |33.2 |10001|
+-----------------+------+-----+-----+
```

Reflection

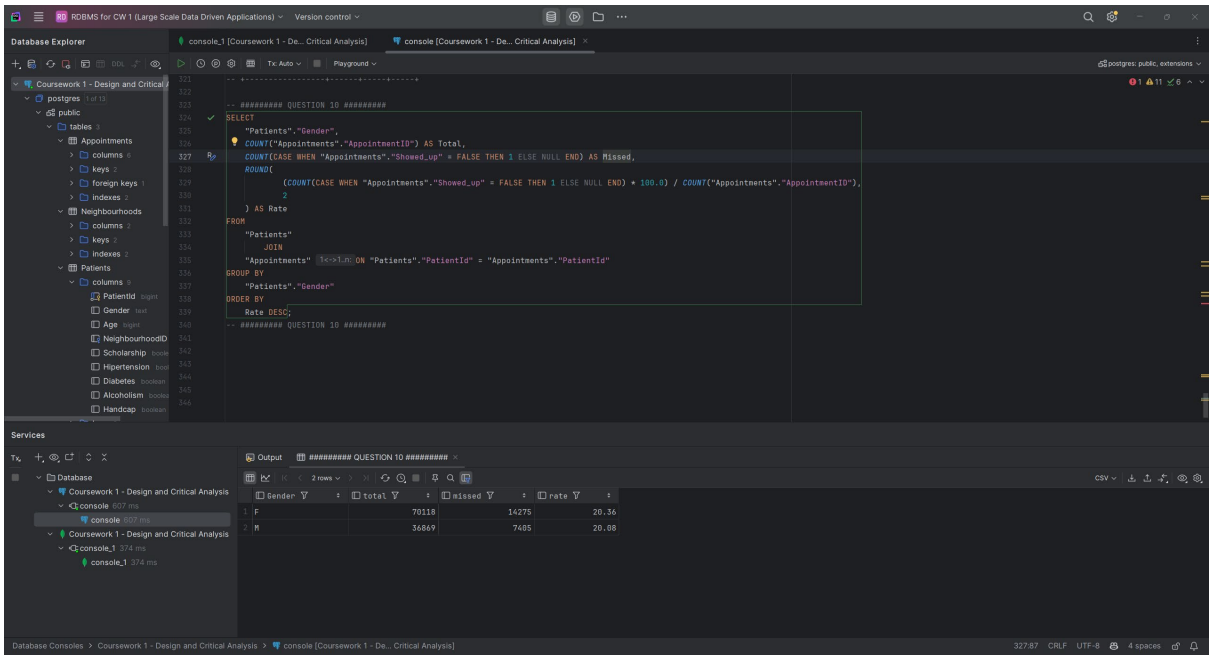
Those that booked appointments for the same day had the lowest number of missed appointments with only 4.69% of appointments missed. These patients don't need many reminders but this statistic could be improved by reminder them a few hours before their appointment.

Patients that booked the appointment within 1 week have a no-show rate of 24.11% which needs to be improved. The total amount of people that book is at 31000 and with so many people missing their appointment, it could disrupt the flow of the clinic. One way to eliminate this problem would be to make it easier for people to re-schedule appointments and to send them frequent reminders.

Users that have scheduled appointments later than 8 days have a higher rate of missing their appointments. This could be due to the fact that their situation can change between the time that they called and the day of their appointment as it is a long period of time. One way to fix this would be to send more reminders to those patients when it is getting closer to their due date. Another aspect could be that they only had a minor condition that has already been solved so they don't feel the need to go in. For this situation the clinic could make it easier to cancel appointments.

Are there any notable differences in no-show rates between male and female patients?

SQL Database



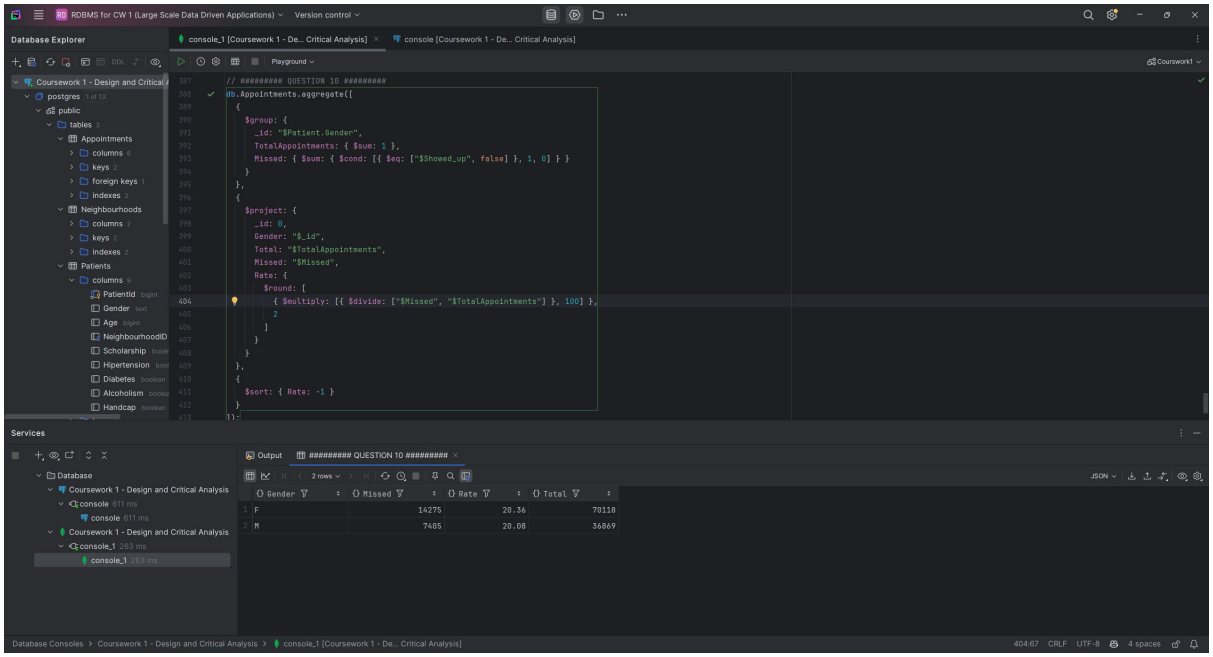
SQL Code

```
-- #########  QUESTION 10  #########
SELECT
    "Patients"."Gender",
    COUNT("Appointments"."AppointmentID") AS Total,
    COUNT(CASE WHEN "Appointments"."Showed_up" = FALSE THEN 1 ELSE NULL
END) AS Missed,
    ROUND(
            (COUNT(CASE WHEN "Appointments"."Showed_up" = FALSE THEN 1 ELSE
NULL END) * 100.0) / COUNT("Appointments"."AppointmentID"),
            2
    ) AS Rate
FROM
    "Patients"
        JOIN
    "Appointments" ON "Patients"."PatientId" = "Appointments"."PatientId"
GROUP BY
    "Patients"."Gender"
ORDER BY
    Rate DESC;
-- #########  QUESTION 10  #########
```

SQL Result

```
+------+-----+------+-----+
|Gender|total|missed|rate |
+------+-----+------+-----+
|F     |70118|14275 |20.36|
|M     |36869|7405  |20.08|
+------+-----+------+-----+
```

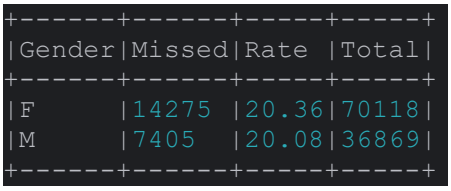
MongoDB Database



MongoDB Code

```
// ######### QUESTION 10 #########
db.Appointments.aggregate([
  {
    $group: {
      _id: "$Patient.Gender",
      TotalAppointments: { $sum: 1 },
      Missed: { $sum: { $cond: [{ $eq: ["$Showed_up", false] }, 1, 0] } }
    }
  },
  {
    $project: {
      _id: 0,
      Gender: "$_id",
      Total: "$TotalAppointments",
      Missed: "$Missed",
      Rate: {
        $round: [
          { $multiply: [{ $divide: ["$Missed", "$TotalAppointments"] },
100] },
          2
        ]
      }
    }
  },
  {
    $sort: { Rate: -1 }
  }
]);
// ######### QUESTION 10 #########
```

MongoDB Result



```
+------+------+-----+-----+
|Gender|Missed|Rate |Total|
+------+------+-----+-----+
|F     |14275 |20.36|70118|
|M     |7405  |20.08|36869|
+------+------+-----+-----+
```

Reflection

There is not a massive difference between the no-show rate between men and women but less men did book appointments at the clinic (33249 less). This could be due to the fact that men are more likely to not seek help at clinics when their condition could be treated a lot earlier (Baker, 2021). To fix this issue the clinic could work on a campaign that educates men on the importance of early intervention when it comes to medical conditions.

The rate similarity could also mean that gender isn't a big reason as to why people miss their appointments so less resources could be invested into this and into other aspects to decrease the no-show rate.

Reflection on the Choice of Database Technologies

For this project I will be developing a web based interactive dashboard. The system will help analyse key factors such as patient data and statistics about appointments to uncover information about patterns behind no-shows. The system should give health care providers to real-time insights on no-show rates by different categories and provide information on how effective reminders are.

Both databases have differences that enable them to be each have their own unique qualities and instances where they are the most useful. The advantages and disadvantages of these databases need to be analysed to see which of these databases will be better suited for our use case which is developing an interactive dashboard.

Relational databases store information in tables whereas NoSQL stores them in a document-oriented database (MongoDB, 2024).

Advantages of Relational Database

The data structure for both databases are different. SQL uses relational databases with keys that show relations. This reduces duplicate information which helps with storage and shows relations between data. The process of normalisation of databases also means that the storage is lower, which is good for cost reductions. It is also an easy language to learn which makes it a choice for data analysts when they are dealing with many rows of information that applications like Excel cannot handle. There are also many tools that allow developers to interact with their SQL databases making it a choice for many data analysts (MongoDB, 2024).

Disadvantages of Relational Database

There are however disadvantages to using SQL databases such as scalability. Most relational databases can only run on one system which means that if the system is not well equipped to deal with a large number of queries, the system will need to be improved in

terms of hardware (MongoDB, 2024). If multiple patients are trying to access information and are using the system we will develop then the system may not be well equipped enough to handle all the queries, resulting in very slow response times which is not viable in an environment where users need instant access to information. The same logic can also be applied to staff who are required to work in a fast pace environment and are required to have quick access to information to ensure no time is wasted and that all appointments for the given day are being fulfilled.

Relational databases also have a rigid schema which can make it difficult to change the database in the future if the clinic would like to add more information to the database to analyse the reason behind no-shows. Relational databases require the initial design of the database to be structured, if a change is implemented to the database, it will require a lot more work which can cause downtime for the database which is not a good scenario for the clinic because they are required to be operational at all times as they provide a critical front line service (MongoDB, 2024). If there is downtime of the system, in the worst-case scenarios it can critically effect patients or even kill them.

If a relational database is complex in nature with many tables and many relations, performing queries on the database may take up valuable time (MongoDB, 2024).

Advantages of NoSQL

The primary reason as to why NoSQL databases are that they are scalable. NoSQL databases have the unique feature of easily scaling out. Relational databases often require scaling up (getting a better machine) which can be a costly venture and 1 machine can have a physical limit where a machine is too expensive to purchase. NoSQL can be scaled out because of its document-oriented nature which allows the data to be split up across multiple servers. It can also balance data and load across a cluster which allows for faster queries. If more data needs to be added, a new machine can be added to the cluster easily (Chodorow & Dirolf, 2010).

MongoDB also supports indexing which can improve the query speed on the database. Aggregation tools can also enable developers to operations to multiple data which can give valuable insight into data (Chodorow & Dirolf, 2010).

NoSQL databases are also designed to be fast. They have many features that allow for this to happen such as using different protocols and using a dynamic query optimizer. NoSQL databases can also offload process and logic to the client machine by using drivers or the user's application code (Chodorow & Dirolf, 2010). All of these features allow these databases to achieve high performance.

If a master server also goes down for any reason, NoSQL databases can fall back to a backup slave and promote that slave to a master which means downtime is small. Database programs like MongoDB simplify administration like this by letting them administrate themselves, when possible, which makes it highly reliable and prevents downtime (Chodorow & Dirolf, 2010).

Disadvantages of NoSQL

Because NoSQL are document-oriented they have low complexity meaning that complex operations to get more insightful information on data can be difficult because NoSQL does not use joins.

Normalisation of databases in relation databases can also reduce the storage requirements for database. There is also fewer duplicate data in a well-designed relational database as there are multiple tables with different keys.

Conclusion

As the clinic is a critical service that requires constant uptime it would be best to use a NoSQL database like MongoDB. The scalability of the database would be high whether that is scaling up or out. As the system could be used by many employees the scalability is important so that the performance of queries is fast and data is easily accessible. This also means that if more patients join the clinic, they can be added to the database as more storage space can be added easily. Features like indexing can also make queries faster and tools like aggregation can be used to gain insight into data. Speed is important in this scenario so MongoDB would be suited for this application.

Uptime of the database is also important as downtime can affect patients and staff. Master servers can easily be replaced by slaves in case of failure. The database is also flexible so that data can be added without having high down times.

Non-relational databases are optimised to get real-time data for operational and analytical purposes because we are dealing with big data (MongoDB, 2024).

Bibliography

Baker, P. (2021, May 5). *Men and Primary Care: Removing the Barriers*. Retrieved from Harvard Medical School: https://info.primarycare.hms.harvard.edu/perspectives/articles/men-primary-care

Chodorow, K., & Dirolf, M. (2010). *MongoDB: The Definitive Guide.* O'Reilly .

Decomplexify. (2021, November 21). *Learn Database Normalization - 1NF, 2NF, 3NF, 4NF, 5NF* . Retrieved from YouTube: https://www.youtube.com/watch?v=GFQaEYEc8_8

Microsoft. (2024, June 6). *Description of the database normalization basics*. Retrieved from Learn: https://learn.microsoft.com/en-us/office/troubleshoot/access/database-normalization-description

MongoDB. (2024, December 08). *Relational vs. Non-Relational Databases*. Retrieved from MongoDB Resources: https://www.mongodb.com/resources/compare/relational-vs-non-relational-databases