DENTAL CLINIC - MIGRATION FROM RELATIONAL TO NOSQL DATABASE

Professor: MSc. Benjamin Besimi

Presented by: Era Emurli, Jona Sela
Date: 16th June 2025

The Challenge

What's the problem?

Traditional relational databases are powerful, but they often struggle to adapt when systems grow larger or data becomes more interconnected. In our case, managing deeply linked patient data—like appointments, medical history, billing, and staff assignments—became complex and inefficient with multiple join operations. This complexity created performance challenges and made scaling difficult for real-world applications like dental clinic systems.

Our Solution

1. Relational Database Design in SQL Server

We designed and implemented a fully normalized relational database in SQL Server, including 8+ interconnected tables such as Patient, Appointment, Bill, and Employee. We applied constraints, relationships, and populated each table with 20+ meaningful records.

2. Python-Based Migration Script

We developed a custom Python script using pyodbc and pymongo to extract data from SQL Server, transform it, and load it into MongoDB—all with error handling and logging.

3. Document-Based Storage in MongoDB

Finally, we structured the data in MongoDB using collections like patients, employees, and dentists. Key relationships were embedded as subdocuments to optimize queries, scalability, and performance.

ER Diagram

This is the Entity Relationship (ER) diagram for our SQL database. Key entities include:

- **Patient**: core entity with appointments, bills, medical history
- **Appointment**: connected with patient, dentist, tech staff
- **Employee**: base entity for Dentist, Nurse, TechStaff

-We ensured 1:N and 1:1 mappings with appropriate foreign keys.

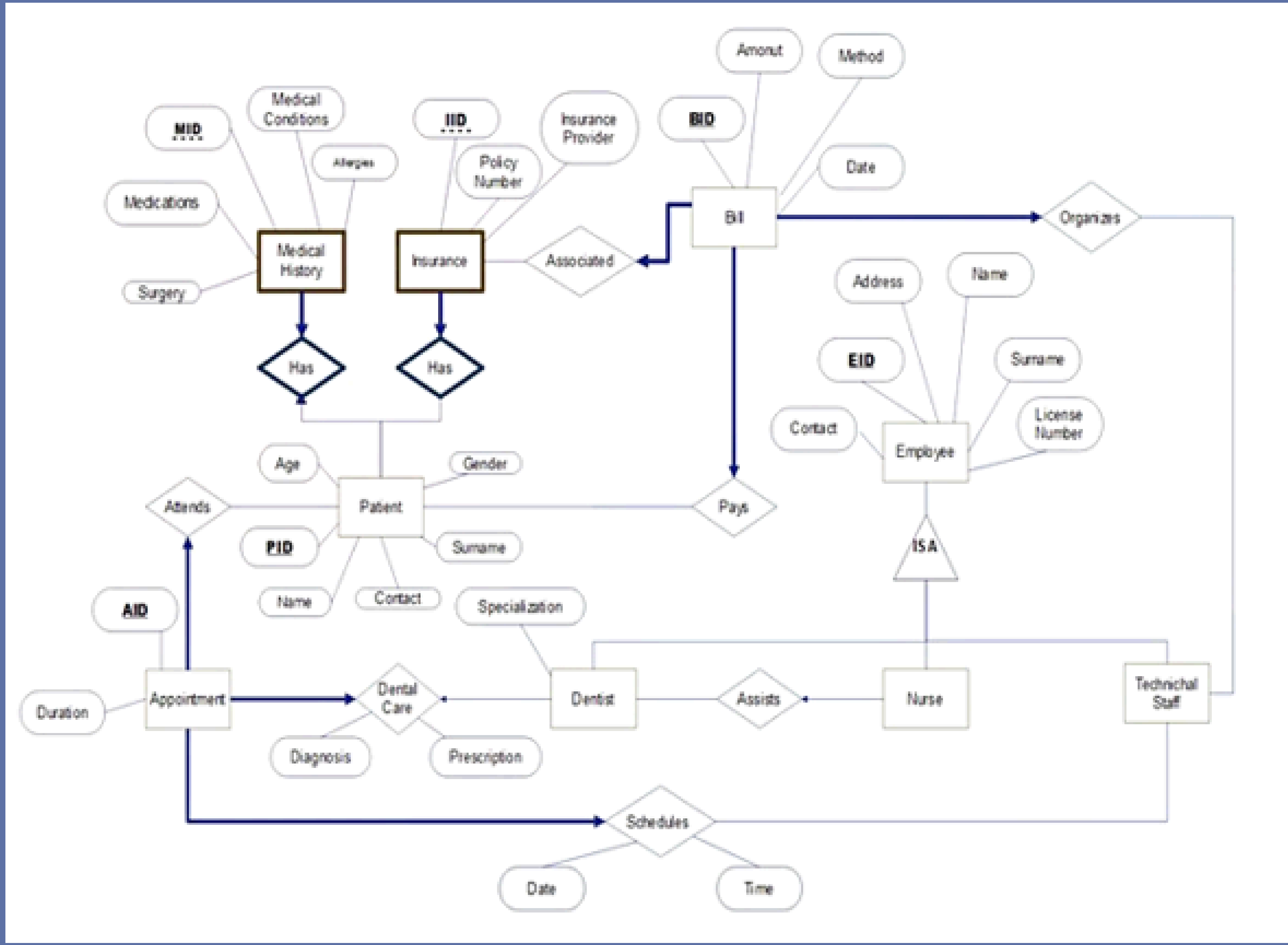
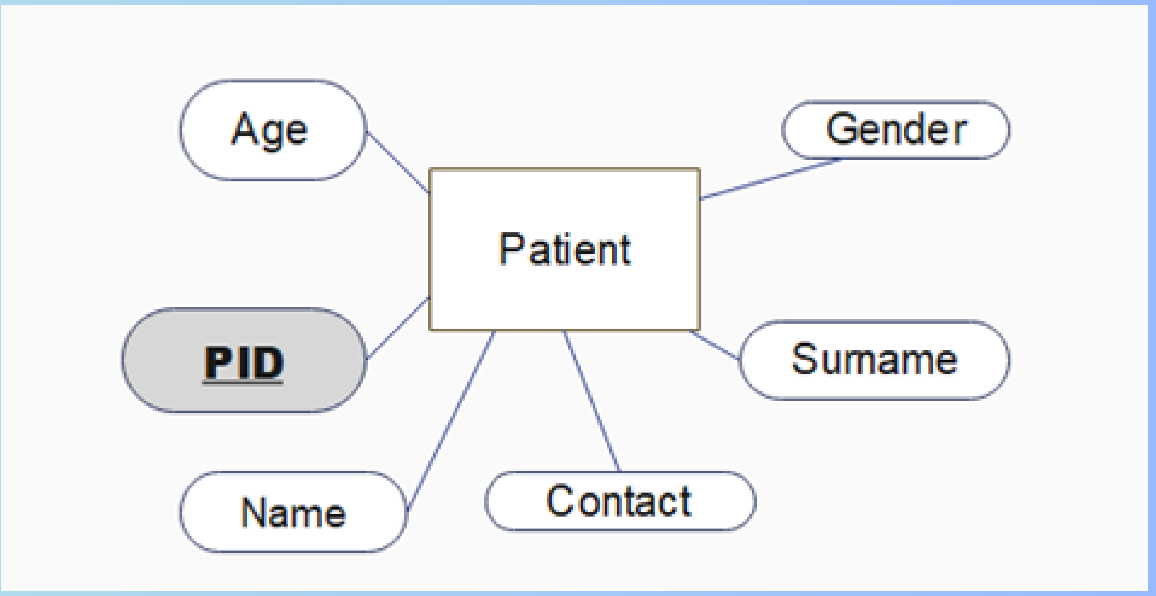


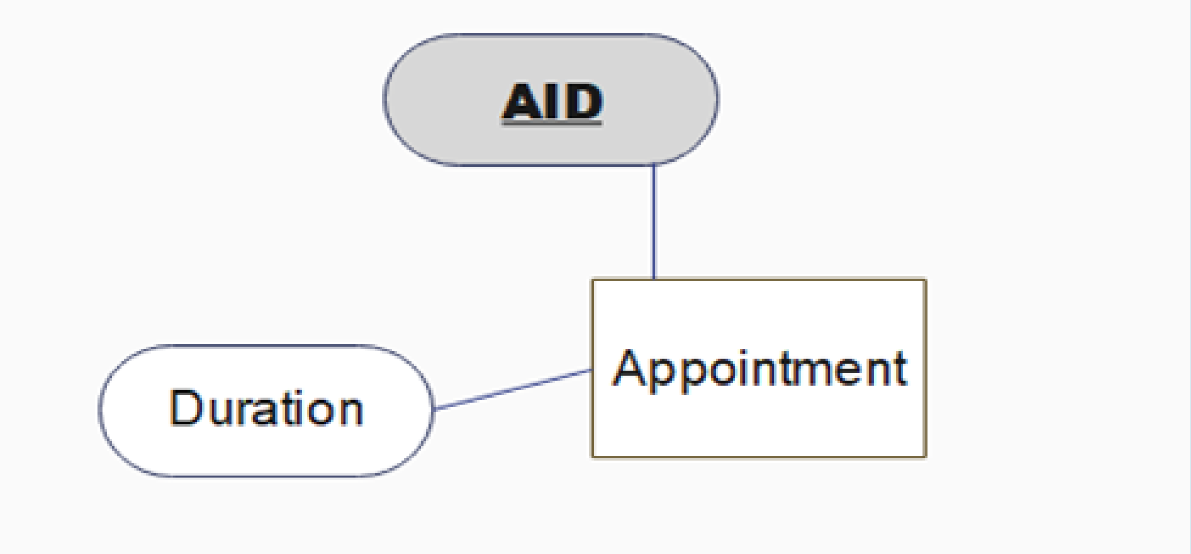
Table Structures

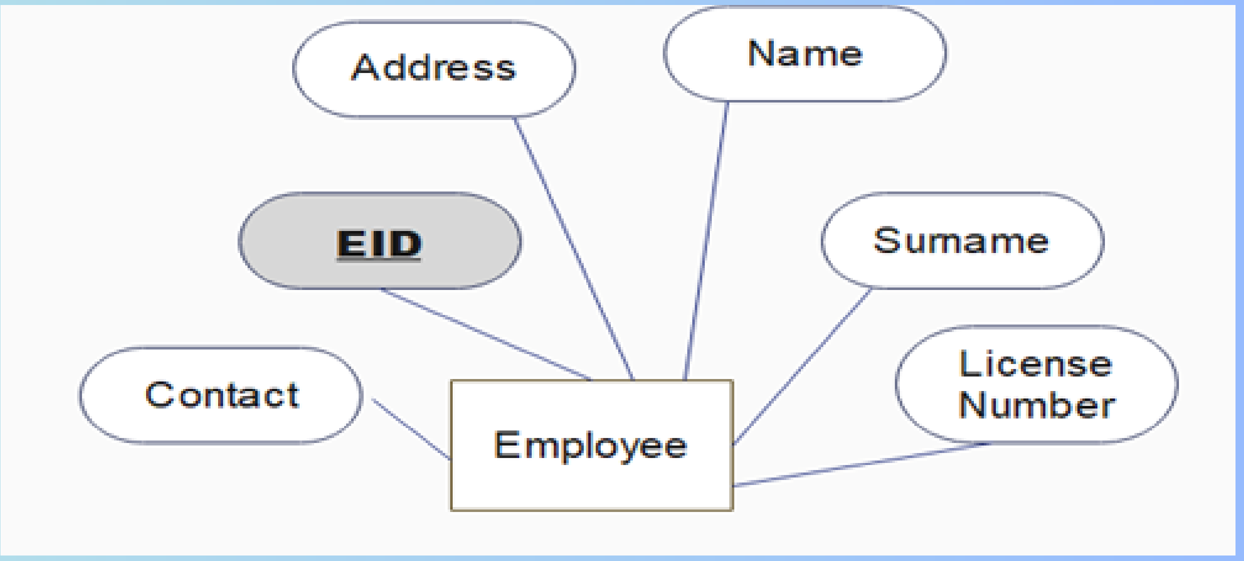
We created 10 relational tables. Here are a few examples:

-Patient (PID, Pname, Psurname, Ppage, Pgender, Pcontact)
-Appointment (AID, PID, TID, DID, Date, Time)
-Employee (EID, Ename, Esurname, ...)

Constraints were applied: primary keys, foreign keys, and check constraints for gender.







Data Population

Each table was populated with at least 20 rows. We used meaningful and realistic data to simulate a real clinic. Here you can see sample screenshots showing populated Patient and Appointment tables.

```sql
SELECT * FROM Patient;
SELECT * FROM Employee;
SELECT * FROM TechStaff;
SELECT * FROM Bill;
SELECT * FROM Dentist;
SELECT * FROM MedicalHistory;
SELECT * FROM Appointment;
SELECT * FROM Insurance;
SELECT * FROM Dentist;
```

68 %

Results | Messages

	PID	Pname	Psurname	Ppage	Pgender	Pcontact
1	0001	Jona	Sela	20	F	777-777-777
2	0002	Era	Sela	21	F	222-222-222
3	0003	Ajeta	Dauti	21	F	444-444-444
4	0004	Gjilizar	Zhuta	20	F	333-333-333
5	0005	Jon	Ziba	25	M	111-111-111
6	0006	Jeta	Sela	20	F	666-666-666
7	0007	Lin	Nushi	50	M	345-983-123
8	0008	Melisa	Kaba	23	F	999-999-999
9	0009	Gerti	Oda	60	M	504-503-403
10	0010	Gerta	Ismaili	28	F	700-893-010
11	0011	Jana	Sela	20	F	777-777-777
12	0012	Erra	Sela	21	F	222-222-222
13	0013	Aieta	Dauti	21	F	444-444-444
14	0014	Gilizar	Zhuta	20	F	333-333-333
15	0015	Ron	Ziba	25	M	111-111-111
16	0016	Leta	Sela	20	F	666-666-666
17	0017	Len	Nushi	50	M	345-983-123
18	0018	Merlisa	Kaba	23	F	999-999-999
19	0019	Gert	Oda	60	M	504-503-403
20	0020	Gerald	Ismaili	28	F	700-893-010

```sql
SELECT * FROM Patient;
SELECT * FROM Employee;
SELECT * FROM TechStaff;
SELECT * FROM Bill;
SELECT * FROM Dentist;
SELECT * FROM MedicalHistory;
SELECT * FROM Appointment;
SELECT * FROM Insurance;
SELECT * FROM Dentist;
```

68 %

Results | Messages

	EID	Ename	Esurname	Eaddress	Econtact
1	D0001	Lyra	Vita	Toronto, Canada, Red st	534-503-4
2	D0002	Kela	Zhuta	Marks Engels st	604-453-4
3	D0003	Albulena	Jonuzi	722 East St	004-503-8
4	D0004	Kaltrina	Bilali	111 Beka St	504-503-0
5	D0005	Ardian	Vrenezi	202 Elz St	500-233-4
6	D0006	Hanife	Vinca	303 Orbit St	474-503-4
7	D0007	Armend	Jakupi	404 Star St	564-903-4
8	D0008	Eva	Poposka	505 Moon St	504-503-4
9	D0009	Ajan	Zuta	606 New St	777-503-4
10	D0010	Leon	Lila	707 Stella St	564-666-4
11	D0011	Lira	Vita	Toronto, Canada, Red st	534-503-4
12	D0012	Keta	Zhuta	Marks Engels st	604-453-4
13	D0013	Arlbulena	Jonuzi	722 East St	004-503-8
14	D0014	Katalea	Bilali	111 Beka St	504-503-0
15	D0015	Adrian	Vrenezi	202 Elz St	500-233-4
16	D0016	Anife	Vinca	303 Orbit St	474-503-4
17	D0017	Admend	Jakupi	404 Star St	564-903-4
18	D0018	Eta	Poposka	505 Moon St	504-503-4
19	D0019	Ajani	Zuta	606 New St	777-503-4
20	D0020	Leoni	Lila	707 Stella St	564-666-4

```sql
SELECT * FROM Patient;
SELECT * FROM Employee;
SELECT * FROM TechStaff;
SELECT * FROM Bill;
SELECT * FROM Dentist;
SELECT * FROM MedicalHistory;
SELECT * FROM Appointment;
SELECT * FROM Insurance;
SELECT * FROM Dentist;
```

68 %

Results | Messages

	DID	Dspecialization
1	D0001	Orthodontics
2	D0002	Endodontics
3	D0003	Pediatric Dentistry
4	D0004	Periodontics
5	D0005	Oral Surgery
6	D0006	Prosthodontics
7	D0007	Oral Pathology
8	D0008	Public Health Dentistry
9	D0009	Oral Radiology
10	D0010	Implantology
11	D0011	Cosmetic Dentistry
12	D0012	Geriatric Dentistry
13	D0013	Maxillofacial Surgery
14	D0014	Laser Dentistry
15	D0015	Restorative Dentistry
16	D0016	Special Needs Dentistry
17	D0017	Temporomandibular Disorders
18	D0018	Oral Medicine
19	D0019	Forensic Odontology

Why MongoDB?

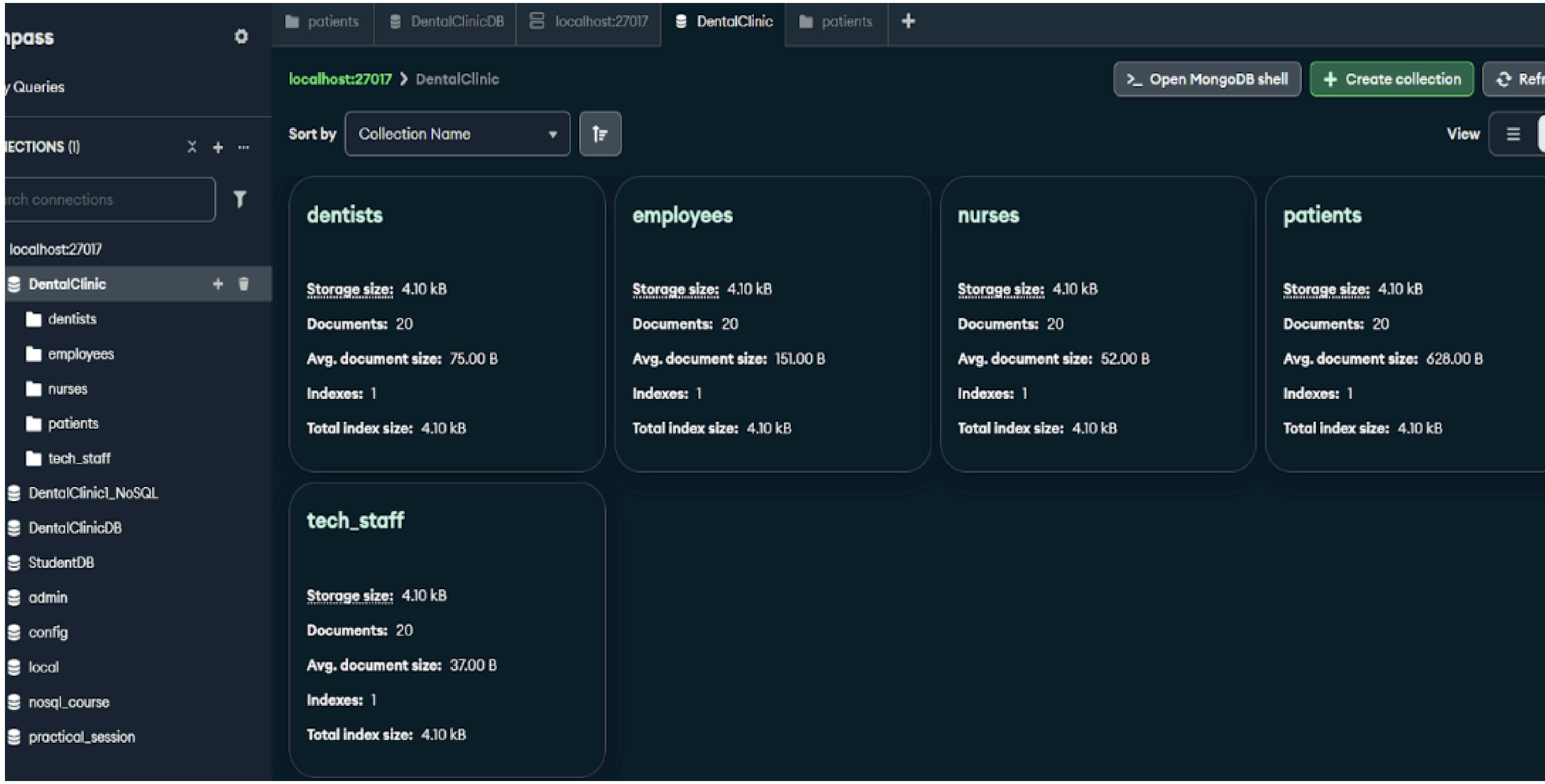
We chose *MongoDB* for these reasons:
- Document-based structure fits our data well (especially for embedding patient history)
- Scalable, flexible schema
- Good support for nested and dynamic documents

Compared to **Redis** (key-value only) and **Cassandra** (good for large-scale writes but column-family-based), MongoDB offered the best balance for our document-heavy case.

NoSQL Modeling

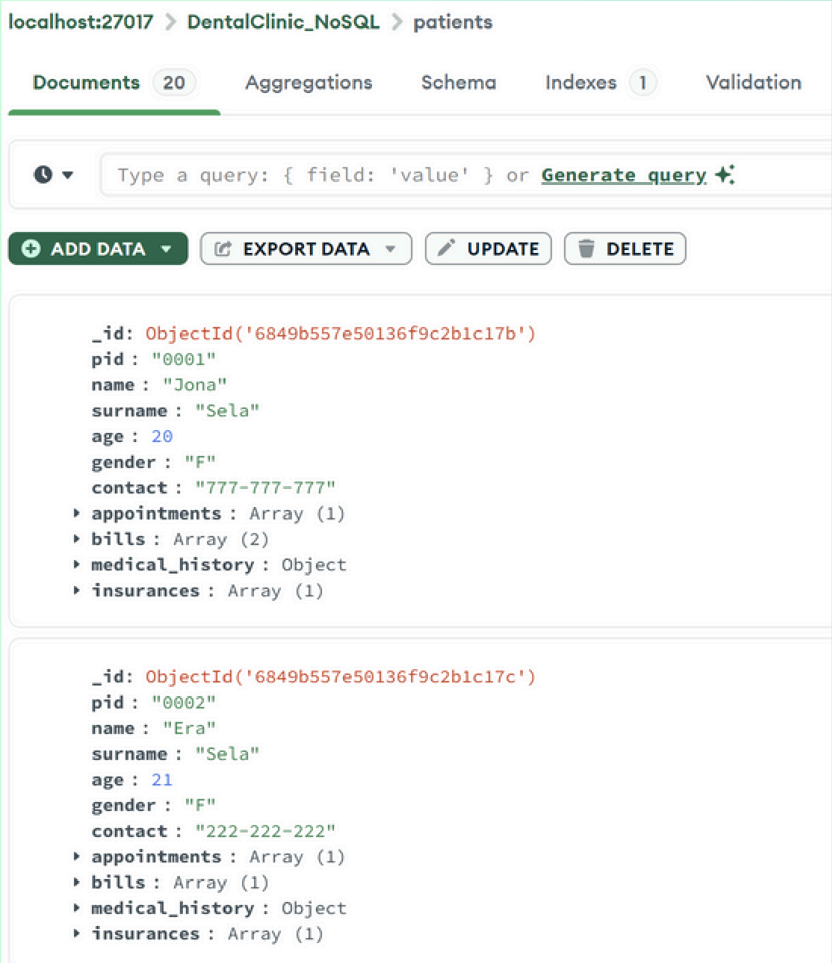
In MongoDB, we modeled each SQL table as a collection:
- Patient → embedded fields: appointments, bills, medical_history, insurance
- Other entities like Dentist, Nurse, TechStaff are separate collections.

This allowed us to reduce joins and improve read performance.

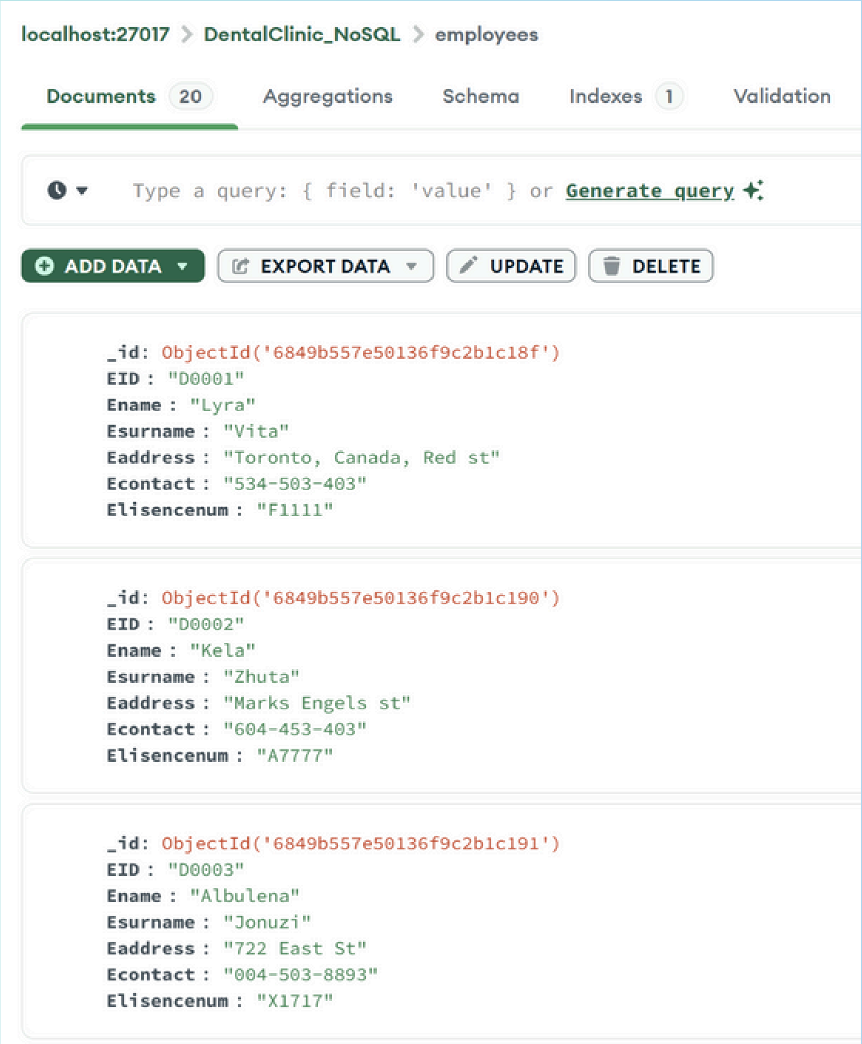


MongoDB Collections

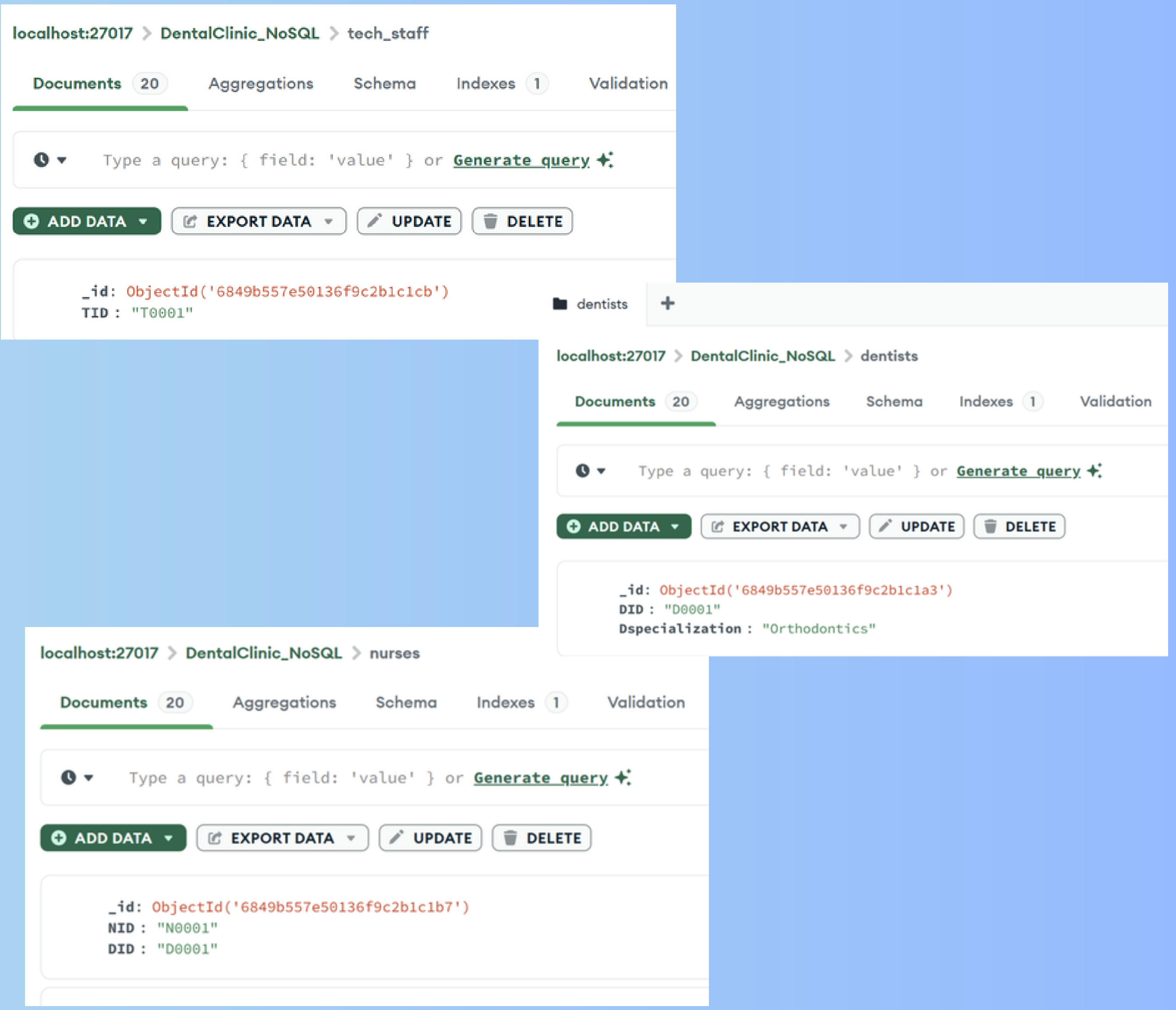
We created the following collections:



patients: includes embedded subdocuments for appointments, bills, medical history, and insurance



employees: generic collection for all staff



dentists, nurses, techstaff: specialized roles derived from Employee

Each document follows MongoDB's BSON format, optimized using a transformation function.

Data Migration Process

We used a Python script to:

C — Connect to SQL Server using

Q — Query each table and transform data types (like Decimal and Date)

I — Insert documents into MongoDB using pymongo

L&E — We included logging and error handling to track the process and ensure data consistency.

Challenges & Solutions

-**Authentication error (Login failed):** Fixed by changing database name and ensuring SQL permissions.
-**Date/Decimal incompatibility:** Solved with custom conversion function.
-**Document nesting:**
Carefully embedded arrays (bills, appointments) only when necessary.

Demonstration

Here's a demo of our script:

```python
# MIGRATE PATIENTS + Embedded Data
cursor.execute("SELECT * FROM Patient")
columns = [column[0] for column in cursor.description]
patients = [dict(zip(columns, row)) for row in cursor.fetchall()]
logging.info(f"Fetched {len(patients)} patients.")

for patient in patients:
    pid = patient["PID"]

    cursor.execute("SELECT * FROM Appointment WHERE PID = ?", pid)
    appointments = [dict(zip([col[0] for col in cursor.description], row)) for row in cursor.fetchall()]

    cursor.execute("SELECT * FROM Bill WHERE PID = ?", pid)
    bills = [dict(zip([col[0] for col in cursor.description], row)) for row in cursor.fetchall()]

    cursor.execute("SELECT * FROM MedicalHistory WHERE PID = ?", pid)
    row = cursor.fetchone()
    med_history = dict(zip([col[0] for col in cursor.description], row)) if row else None

    cursor.execute("SELECT * FROM Insurance WHERE PID = ?", pid)
    insurances = [dict(zip([col[0] for col in cursor.description], row)) for row in cursor.fetchall()]

    document = {
        "pid": pid,
        "name": patient["Pname"],
        "surname": patient["Psurname"],
        "age": patient["Ppage"],
        "gender": patient["Pgender"],
        "contact": patient["Pcontact"],
        "appointments": appointments,
        "bills": bills,
        "medical_history": med_history,
        "insurances": insurances
    }

    patients_col.insert_one(convert_for_mongo(document))
    print(f"Inserted patient {pid} into MongoDB.")
    logging.info(f"Inserted patient {pid} into MongoDB.")
```

Lessons Learned

- SQL modeling taught us normalization and constraints
- MongoDB showed us flexibility with embedded documents
- We learned how to convert schemas across paradigms
- We became more confident in data migration scripting

Conclusion & Final Reflection

We successfully:
- Built and populated a relational DB
- Modeled equivalent NoSQL structures
- Migrated data using Python scripts
- Validated the results in MongoDB

The project gave us full-cycle experience in modern data engineering.

We truly enjoyed working on this project. It deepened our interest in databases, especially MongoDB. We’re now considering learning more in the field of data engineering or backend development, and maybe even doing internships in these areas.

THANK YOU!