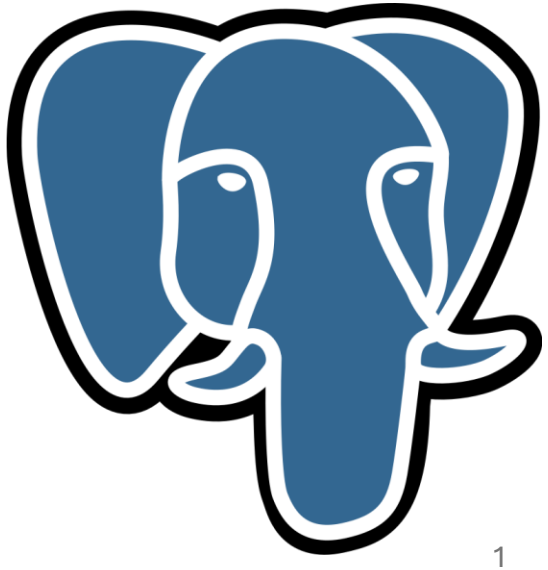
Data Management

Sahar Khanlari - 2107563

Marco Natale – 1929854

A.A. 2024/2025



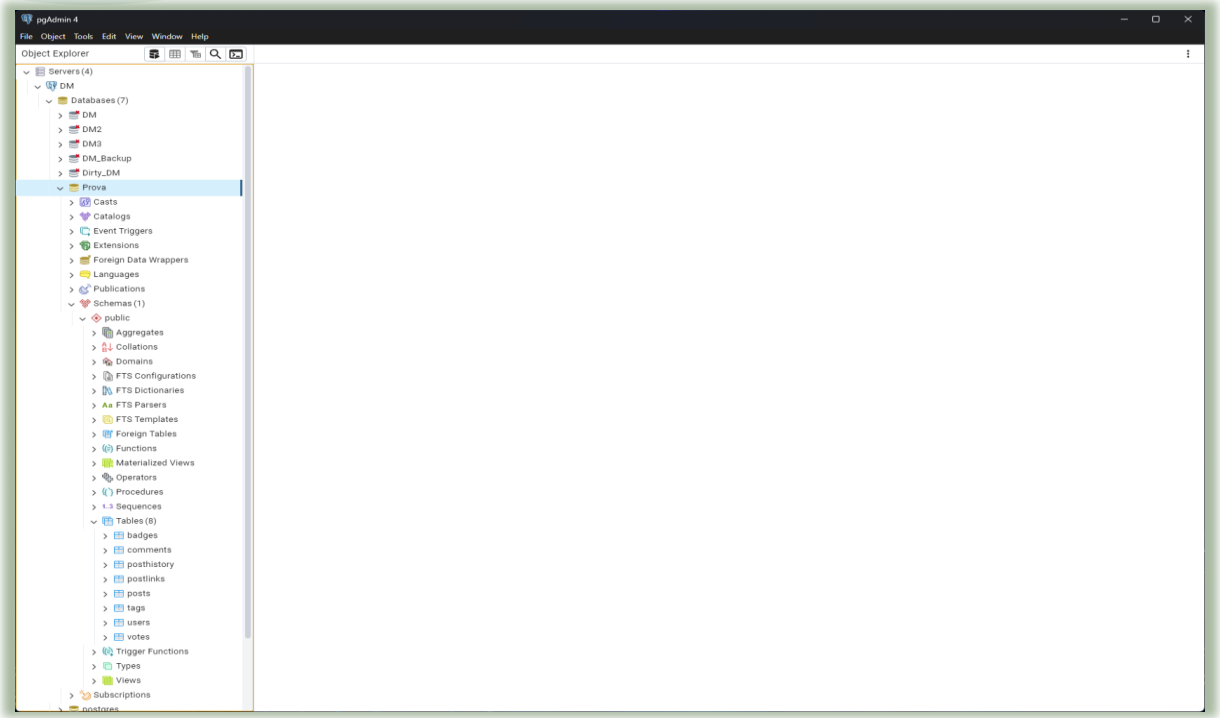
Introduction

- SQL vs NoSQL (PostgreSQL vs MongoDB)
- Goals:
  - Interface Usability
  - Query Language Comparison
  - Flexibility
  - Scalability
  - Performance

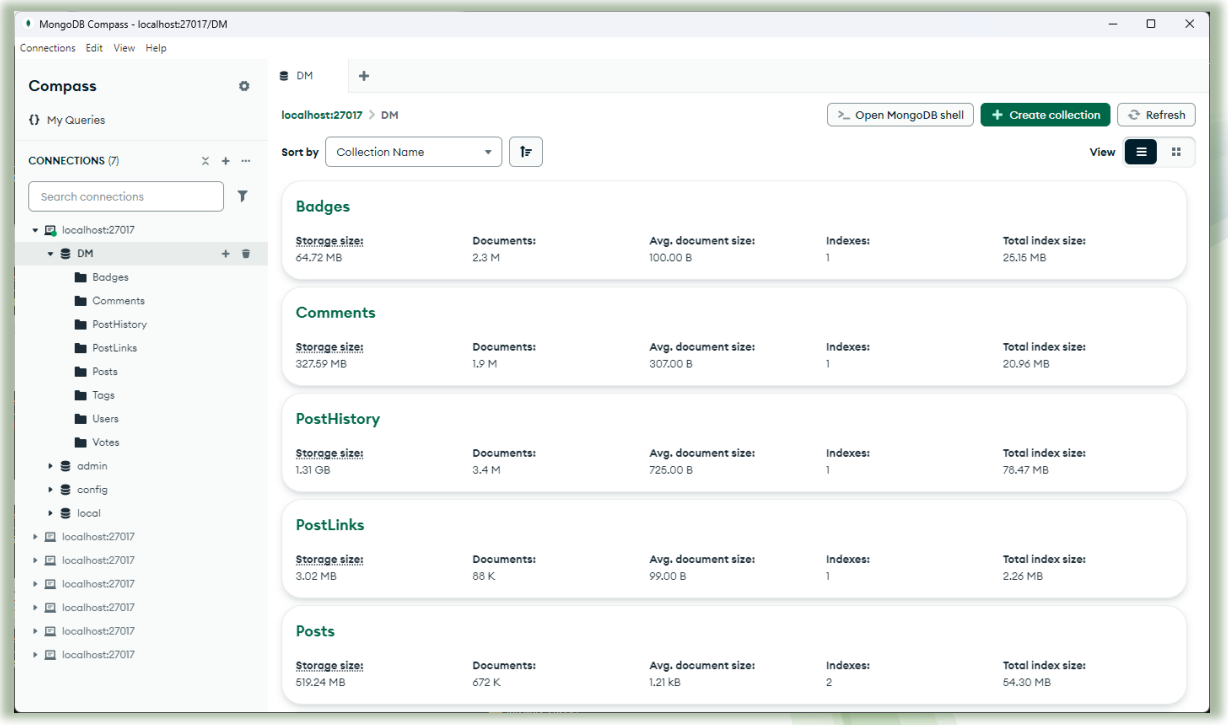
Interface Usability

Layout

pgAdmin



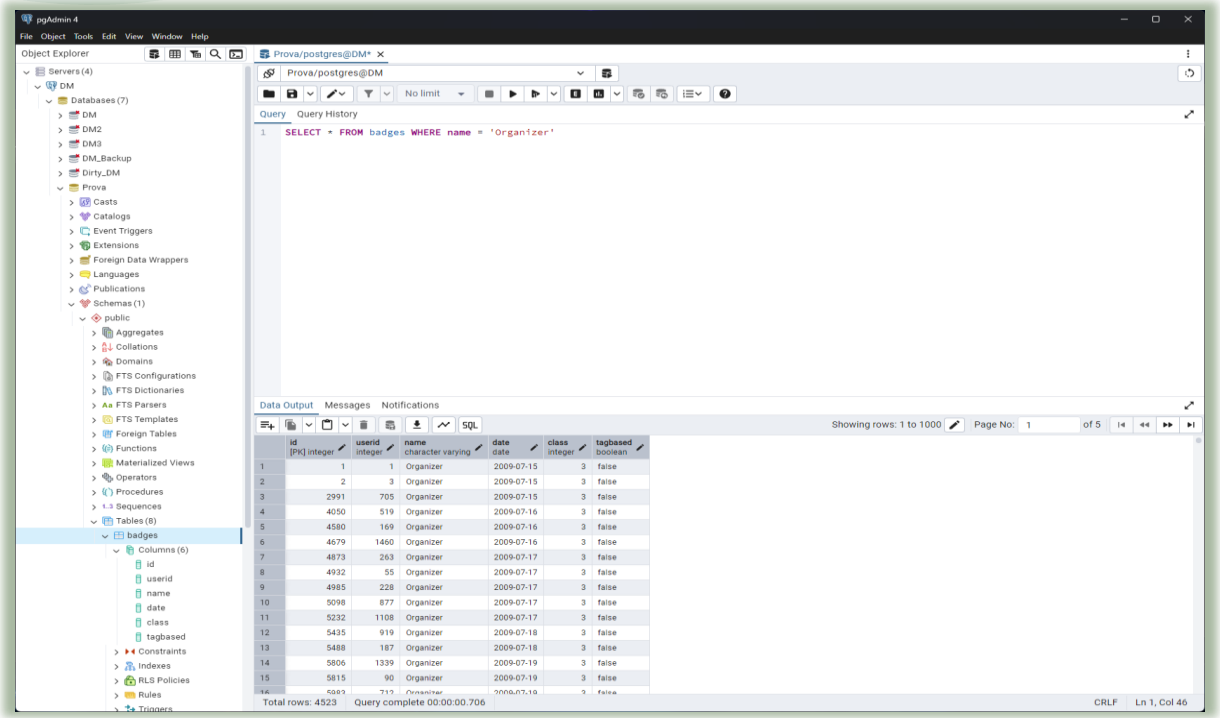
MongoDB Compass



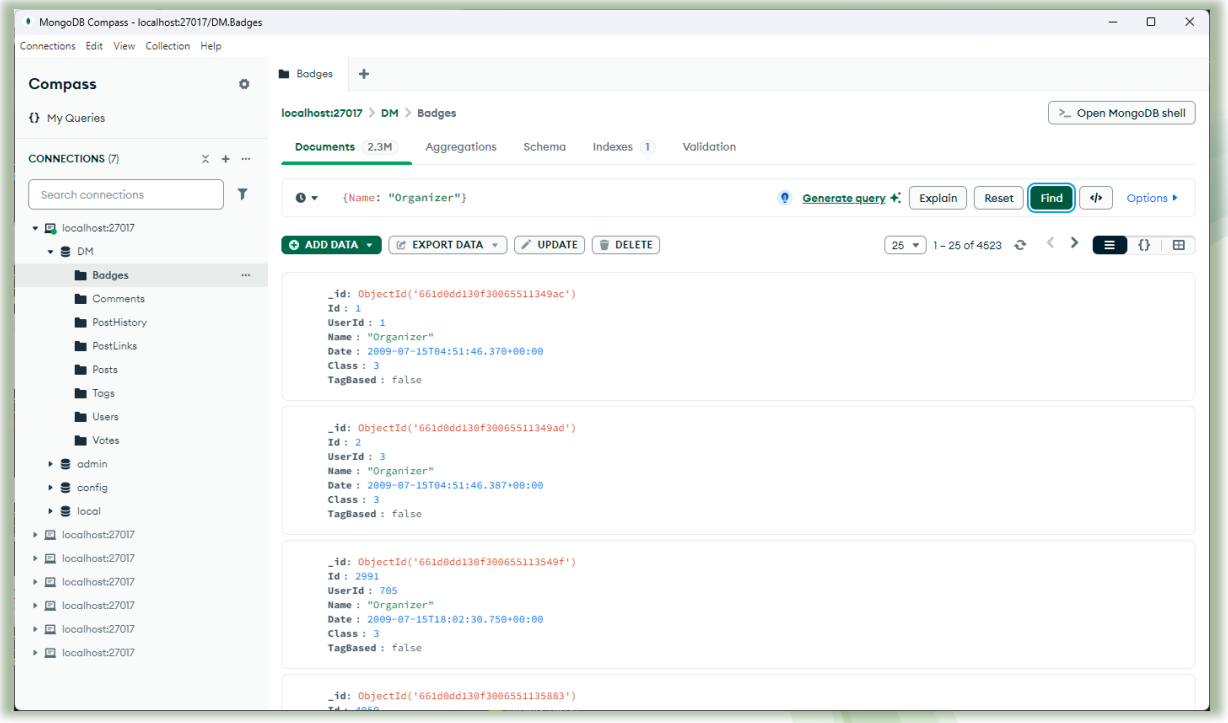
Interface Usability

Navigation & Usability

pgAdmin



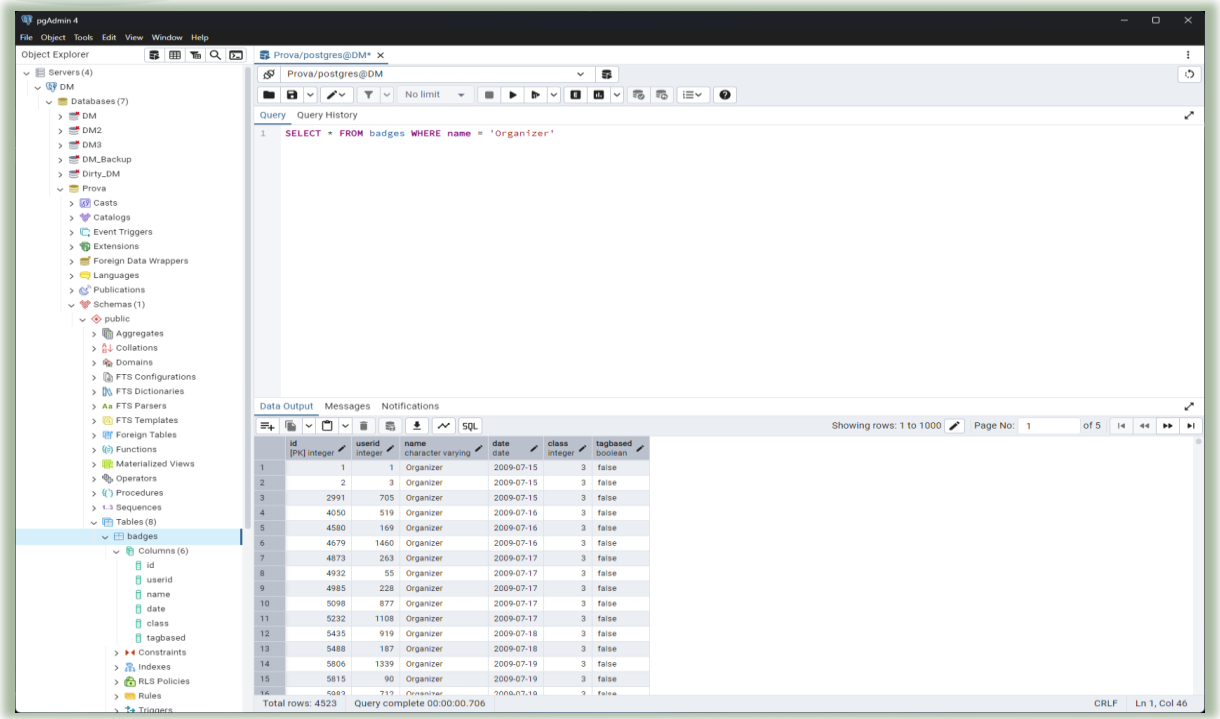
MongoDB Compass



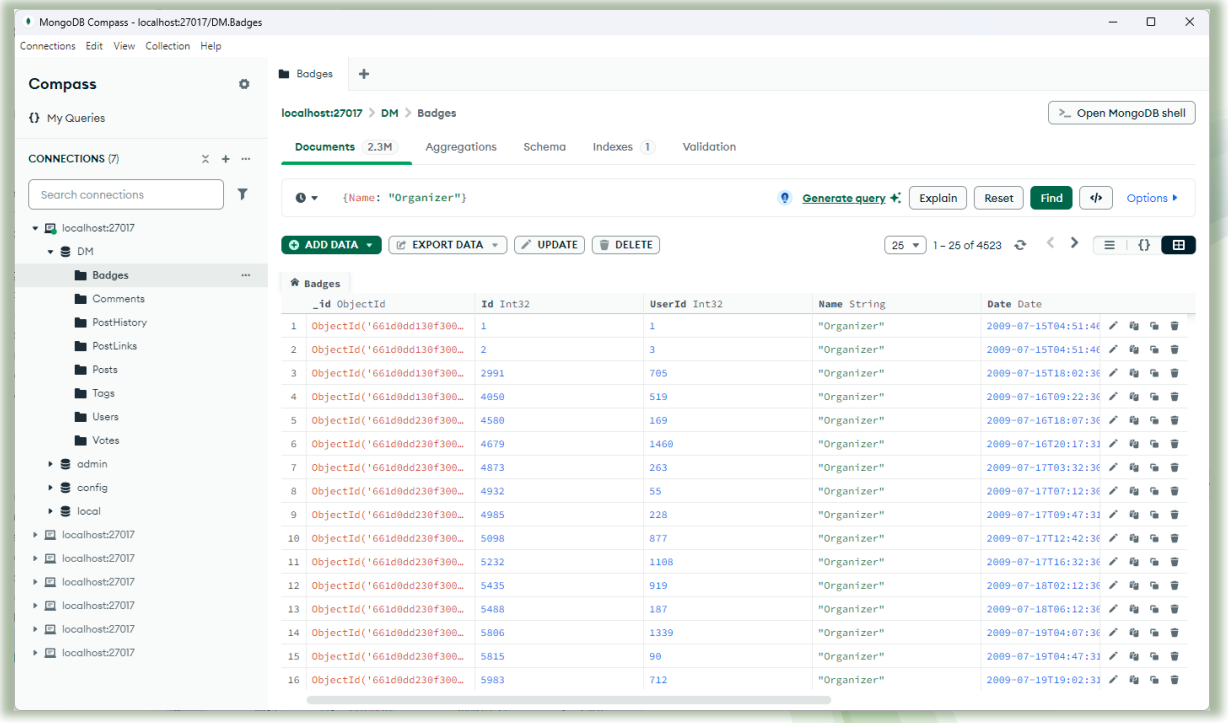
Interface Usability

Navigation & Usability

pgAdmin



MongoDB Compass



Interface Usability

Functionality

Differences

Query Interface:

- pgAdmin: Uses a simple text editor for writing SQL queries.
- MongoDB Compass: Provides a visual aggregation pipeline builder for NoSQL queries.

Predefined Queries:

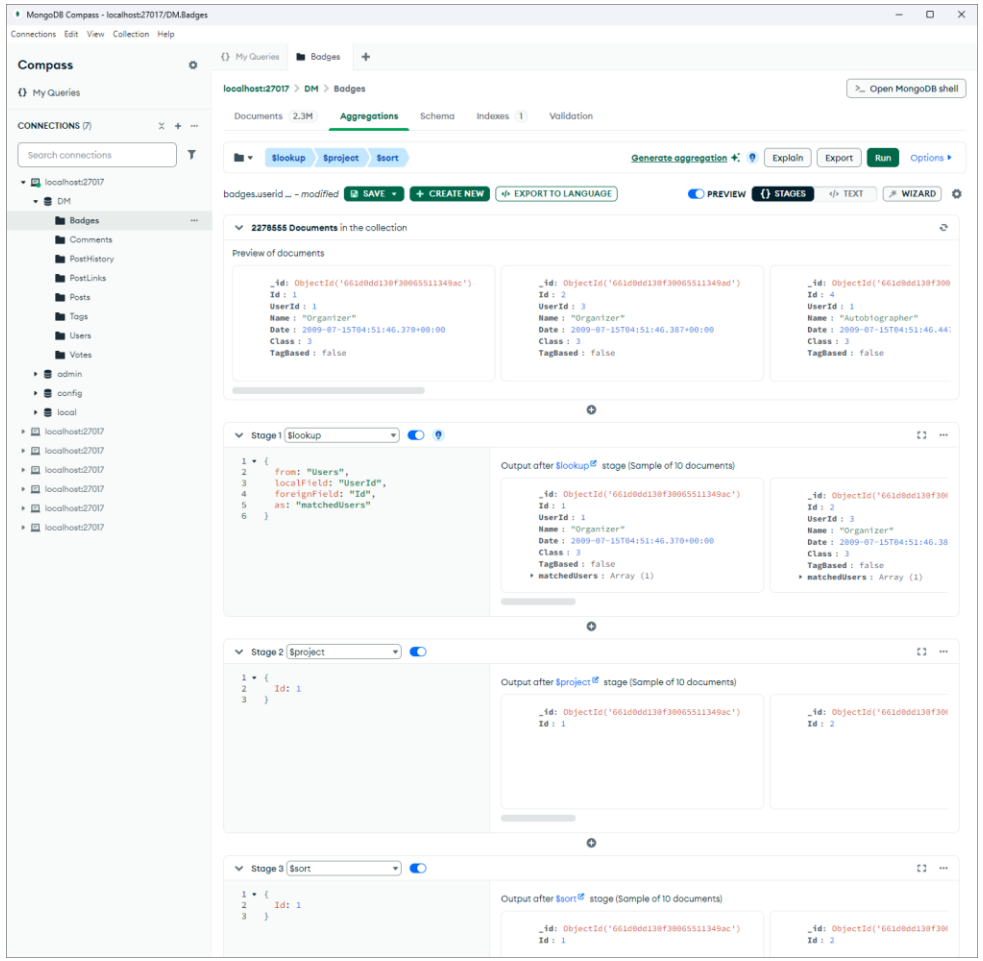
- pgAdmin: Offers predefined queries (e.g., view data, CRUD scripts) via right-click.
- MongoDB Compass: No predefined SQL-like queries, relies on document-based query building.

Similarities

- Both provide a GUI interface for database management.
- Both allow import/export of data.
- Both include a CLI window for command-line interaction.

Interface Usability

Functionality



Query Language Comparison

MongoDB

- **Document-Based Syntax**: Uses JSON/BSON for queries.

  Example: db["Users"].find({ age: { $gt: 30 } })

- **Operators & Expressions**: Supports $eq, $gt, $in, $and, $or for flexible filtering.

- **Aggregation Framework**: Replaces SQL GROUP BY with a multi-stage pipeline ($match, $group, $project, $sort).

- **Joins via $lookup**: Supports cross-collection joins within aggregation pipelines.

- **Flexibility**: Schema-less design allows dynamic document structure changes.

Query Language Comparison

PostgreSQL

- **Declarative, ANSI-Compliant Syntax**: Uses standardized SQL.

  Example: SELECT * FROM users WHERE age > 30;

- **Structured & Schema-Driven**: Enforces strong data integrity with predefined tables and relationships.

- **Powerful Joins & Subqueries**: Supports INNER, LEFT, RIGHT, FULL joins and complex queries across multiple tables.

- **Advanced Aggregation & Window Functions**: Uses GROUP BY, window functions, and WITH queries for analytics.

- **Extensibility**: Supports user-defined functions, stored procedures, and procedural languages for business logic.

Query Language Comparison

Syntax and Structure

- **MongoDB**:
  - Uses JSON-like documents to specify query criteria.
  - Queries are written as object literals, which can be more natural for developers working in JavaScript or other object-oriented languages.

```
db[“Orders”].find({ status: "shipped", total: { $gte: 50 } })
```

- **PostgreSQL:**
  - Uses a declarative, text-based language with clear keywords.
  - The language is standardized, making it easier to port skills between different SQL databases.

```
SELECT * FROM orders WHERE status = 'shipped' AND total >= 50;
```

Query Language Comparison

Data Model

- **MongoDB**:
  - Designed for hierarchical, nested data. Its query language naturally expresses conditions on embedded documents and arrays.
  - Aggregation pipelines can perform multi-stage transformations that align with the flexible document model.

- **PostgreSQL:**
  - Optimized for flat, tabular data. SQL queries express relationships and joins between tables through well-defined keys.
  - Complex aggregations, subqueries, and window functions enable robust data analysis on structured datasets.

Query Language Comparison

Joins and Relationships

MongoDB:

- Lacks a native, SQL-like join operation because the data is often stored in denormalized documents.
- Uses the $lookup stage in aggregation pipelines for joining documents from different collections, but with more limited capabilities compared to SQL joins.

PostgreSQL:

- Joins are first-class citizens. You can easily join tables with various join types, which is essential for normalized data models.
- The declarative nature of SQL joins makes it straightforward to enforce relationships and retrieve combined datasets.

Query Language Comparison

Aggregations

- **MongoDB**:
  - The aggregation pipeline provides a procedural approach where each stage transforms the data gradually.
  - This can be very powerful for operations such as grouping, filtering, and reshaping data within a document-centric model.

- **PostgreSQL:**
  - Uses aggregate functions with GROUP BY clauses and advanced window functions to compute summaries.
  - The set-based nature of SQL makes it efficient for operations that work across rows of data in a table.

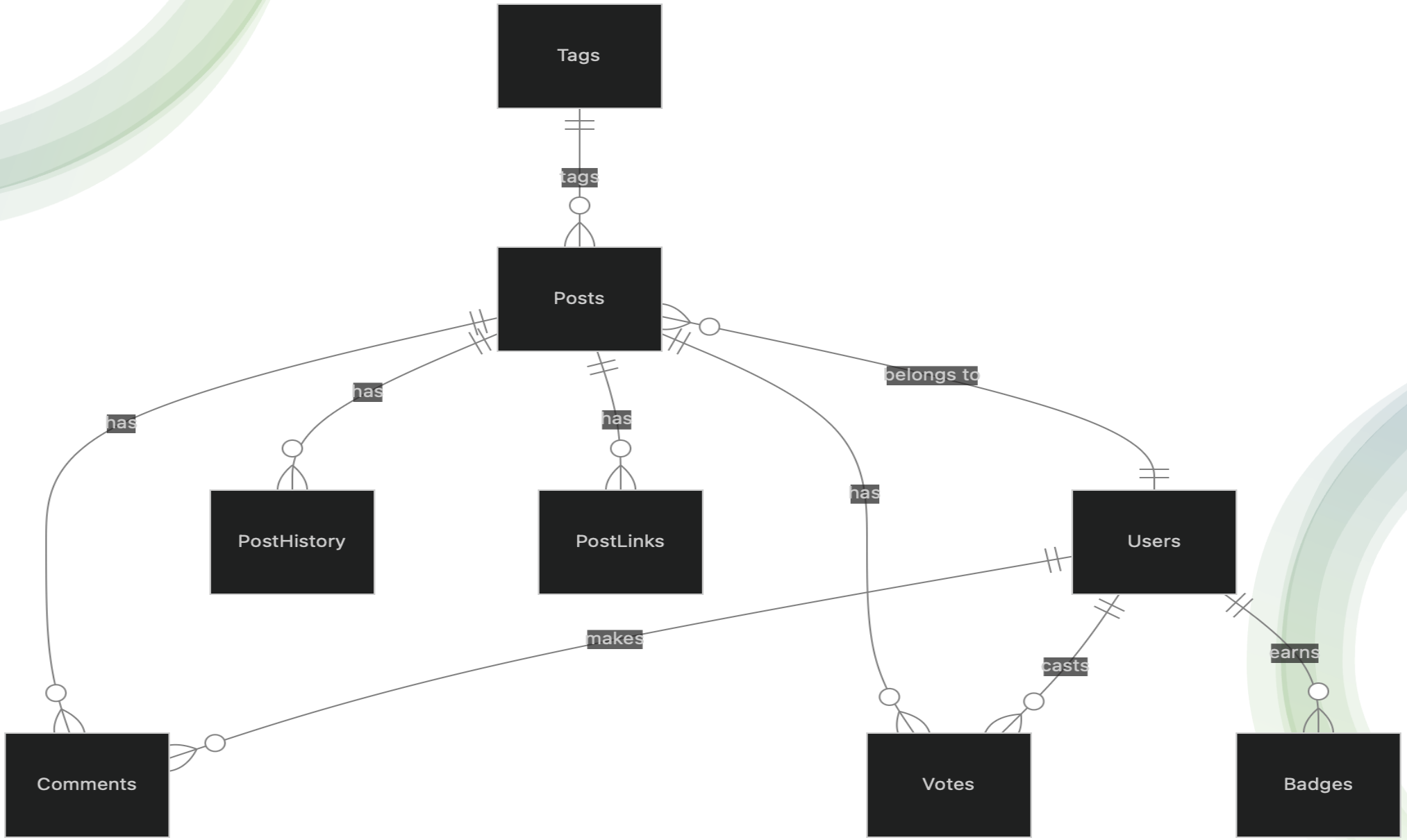
Flexibility

Feature	SQL (Relational Databases)	MongoDB (NoSQL)
Schema	Strict, predefined schema	Schema-less (Flexible documents)
Structure	Tables, rows, and columns	Documents (JSON/BSON format)
Data Relationships	Normalized, uses joins to relate tables	Denormalized, embeds related data
Schema Changes	Requires migrations, downtime	Dynamic, fields can be added/removed
Use Case	Best for structured & transactional data	Best for unstructured, evolving data

Scalability

Feature	SQL (Relational Databases)	MongoDB (NoSQL)
Scaling Strategy	Vertical Scaling (Scale-Up)	Horizontal Scaling (Scale-Out)
Data Distribution	Difficult to shard, manual partitioning	Built-in sharding for automatic distribution
Read/Write Performance	Efficient for transactions, joins slow large queries	Faster reads, distributed writes
Replication	Master-slave or cluster-based replication	Replica sets with automatic failover
High Availability	Requires complex setup	Native support for replication & redundancy
Global Distribution	Requires additional tools	Geo-distributed databases supported

The Dataset


Tags
tags
Posts
has
has
has
has
belongs to
PostHistory
PostLinks
Users
makes
casts
earns
Comments
Votes
Badges

Importing the tables

- **PostgreSQL**:
  - Badges: 12,78 sec
  - Comments: 16,87 sec
  - PostHistory: 62 sec
  - PostLinks: 0,72 sec
  - Posts: 27,38 sec
  - Tags: 0,11 sec
  - Users: 14,72 sec
  - Votes: 23,34 sec

- **MongoDB** doesn't measure time

Performance

Integrity Checks

Data Cleaning

Foreign Keys (only for SQL)

Different Query Types

Key Observation

Checking if PostLinks references Posts

- **Inefficient query**:

```
SELECT pl.id
FROM postlinks pl
WHERE pl.postid NOT IN (SELECT p.id FROM posts p);
```

Execution Time: 2 hours 37 minutes

- **Efficient query**:

```
SELECT pl.id
FROM postlinks pl
EXCEPT
SELECT pl.id
FROM postlinks pl JOIN posts p ON p.id = pl.postid
ORDER BY id;
```

Execution Time: 258 milliseconds

Integrity Checks

Among 8 queries	PostgreSQL	MongoDB
BEST **PostLinks → Posts**	258 ms	1.825 sec
AVERAGE	4.212 sec	33.466 sec (~7x)
WORST **Votes → Posts**	7.914 sec	76.631 sec

Data Cleaning

Among 8 queries	PostgreSQL	MongoDB
BEST **PostLinks → Posts**	379 ms	1.460 sec
AVERAGE	9.644 sec	34.108 sec (~2.5x)
WORST	29.958 sec PostHistory → Posts	87.51 sec Votes → Posts

- Votes table is **too large** for MongoDB's $lookup.
- PostgreSQL' s join is optimized on large tables.

Foreign Keys

- **Posts** references:
  - Users(Id) → OwnerUserId
  - Posts(Id) → ParentId & AcceptedAnswerId
- **Comments** references:
  - Posts(Id) → PostId
- **Votes** references:
  - Users(Id) → UserId
  - Posts(Id) → PostId
- **Badges** references:
  - Users(Id) → UserId
- **PostHistory** references:
  - Users(Id) → UserId
  - Posts(Id) → PostId
- **PostLinks** references:
  - Posts(Id) → PostId & RelatedPostId

Different Queries

	Read	Write	Joins	Aggregation	Text Search	Nested Updates	Complex Joins	Pagination	Deletion	Count
SQL	0.779	0.453	14.673	2.236	1.286	3.259	8.998	0.583	>8h	0.884
MongoDB	6.867	0.003	88.112	2.518	2.3296	1.7278	>6h	2.470	10.869	1.144

Thank you

Sahar Khanlari - 2107563

Marco Natale – 1929854

A.A. 2024/2025