KJSCE/IT/SYBTech/SEM IV/AD/2022-23



Page No.:

**Batch:A2 Roll No.:16010421073 Experiment No.:6**

**Aim: To implement NOSQL database using MongoDB and PostgreSQL.**



# Resources needed: MongoDB, PostgreSQL



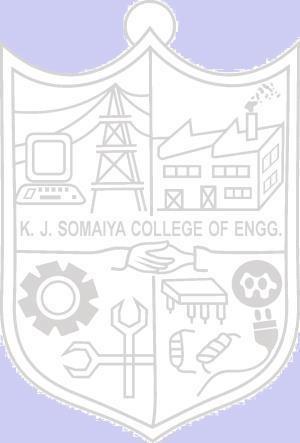
**Theory:**

# MongoDB:

MongoDB is a general-purpose [document database](https://www.mongodb.com/document-databases) designed for modern application development and for the cloud. Its scale-out architecture allows you to meet the increasing demand for your system by adding more nodes to share the load

MongoDB is having following key concepts,

* [**Documents:** The Records in a Document Database](https://www.mongodb.com/basics#documents)

MongoDB stores data as JSON documents. The document data model maps naturally to objects in application code, making it simple for developers to learn and use. The fields in a JSON document can vary from document to document. Documents can be nested to express hierarchical relationships and to store structures such as arrays. The document model provides flexibility to work with complex, fast-changing, messy data from numerous sources. It enables developers to quickly deliver new application functionality. For faster access internally and to support more data types, MongoDB converts documents into a format called Binary JSON or BSON. But from a developer perspective, MongoDB is a [JSON database](https://www.mongodb.com/json-and-bson).

* [**Collections:** Grouping Documents](https://www.mongodb.com/basics#collections)

In MongoDB, a collection is a group of documents. Collection can be seen as tables, but collections in MongoDB are far more flexible. Collections do not enforce a schema, and documents in the same collection can have different fields. Each collection is associated with one MongoDB database

* [**Replica Sets**: For High Availability](https://www.mongodb.com/basics#replica-sets)

In MongoDB, high availability is built right into the design. When a database is created in MongoDB, the system automatically creates at least two more copies of the data, referred to as a replica set. A replica set is a group of at least three MongoDB instances that continuously replicate data between them, offering redundancy and protection against downtime in the face of a system failure or planned maintenance.

* [**Sharding:** For Scalability to Handle Massive Data Growth](https://www.mongodb.com/basics#sharding)

A modern data platform needs to be able to handle very fast queries and massive datasets using ever bigger clusters of small machines. Sharding is the term for distributing data intelligently across multiple machines. MongoDB shards data at the collection level, distributing documents in a collection across the shards in a cluster. The result is a scale-out architecture that supports even the largest applications.

* **Aggregation Pipelines:** For Fast Data Flows

MongoDB offers a flexible framework for creating data processing pipelines called aggregation pipelines. It features [dozens of stages and over 150 operators and](https://www.mongodb.com/basics/aggregation-pipeline) [expressions](https://www.mongodb.com/basics/aggregation-pipeline), enabling you to process, transform, and analyze data of any structure at

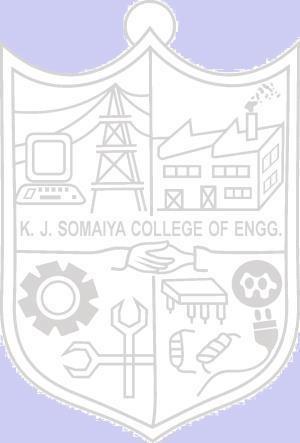
scale. One recent addition is the [Union stage,](https://docs.mongodb.com/master/reference/operator/aggregation/unionWith/#pipe._S_unionWith) which flexibly aggregate results from multiple collections.

Besides this MongoDB provides,

* variety of indexing strategies for speeding up the queries along with the [Performance](https://docs.atlas.mongodb.com/performance-advisor/?_ga=2.142056561.217296718.1646912899-350686066.1644301195) [Advisor,](https://docs.atlas.mongodb.com/performance-advisor/?_ga=2.142056561.217296718.1646912899-350686066.1644301195) which analyses queries and suggests indexes that would improve query performance
* Support for different programming languages which includes Node.js, C, C++, C#, Go, [Java](https://www.mongodb.com/java), Perl, PHP, [Python,](https://www.mongodb.com/python) Ruby, Rust, Scala, and Swift with actively maintained library updated with newly added features.
* Various tools and utilities for monitoring MongoDB.
* Cloud services

# PostgreSQL:

PostgreSQL is a powerful, open source object-relational database system that uses and extends the SQL language combined with many features that safely store and scale the most complicated data workloads. The origins of PostgreSQL date back to 1986 as part of the POSTGRES project at the University of California at Berkeley and has more than 30 years of active development on the core platform.

PostgreSQL comes with many features aimed to help developers build applications, administrators to protect data integrity and build fault-tolerant environments, and help you manage your data no matter how big or small the dataset. In addition to being free and open source, PostgreSQL is highly extensible. For example, you can define your own data types, build out custom functions, and even write code from different programming languages without recompiling your database.

Some of the features of PostgreSQL are as follows,

# Data Types

* + Primitives: Integer, Numeric, String, Boolean
  + Structured: Date/Time, Array, Range / Multirange, UUID
  + Document: JSON/JSONB, XML, Key-value (Hstore)
  + Geometry: Point, Line, Circle, Polygon
  + Customizations: Composite, Custom Types

# Data Integrity

* + UNIQUE, NOT NULL
  + Primary Keys
  + Foreign Keys
  + Exclusion Constraints
  + Explicit Locks, Advisory Locks

# Concurrency, Performance

* + Indexing: B-tree, Multicolumn, Expressions, Partial
  + Advanced Indexing: GiST, SP-Gist, KNN Gist, GIN, BRIN, Covering indexes,

Bloom filters

* + Sophisticated query planner / optimizer, index-only scans, multicolumn statistics
  + Transactions, Nested Transactions (via savepoints)
  + Multi-Version concurrency Control (MVCC)
  + Parallelization of read queries and building B-tree indexes
  + Table partitioning
  + All transaction isolation levels defined in the SQL standard, including Serializable
  + Just-in-time (JIT) compilation of expressions

# Reliability, Disaster Recovery

* + Write-ahead Logging (WAL)
  + Replication: Asynchronous, Synchronous, Logical
  + Point-in-time-recovery (PITR), active standbys
  + Tablespaces

# Security

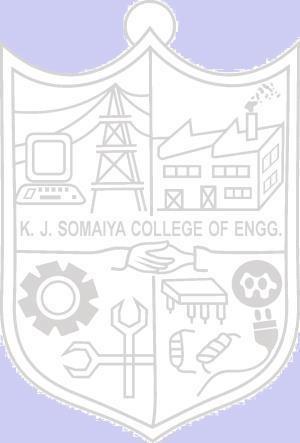
* **Extensibility**

# Internationalisation, Text Search

**PostgreSQL types for NOSQL:**

JSON data types are for storing JSON (JavaScript Object Notation) data. Such data can also be stored as text, but the JSON data types have the advantage of enforcing that each stored value is valid according to the JSON rules. There are also assorted JSON-specific functions and operators available for data stored in these data types.

PostgreSQL offers two types for storing JSON data: **json** and **jsonb**. To implement efficient query mechanisms for these data types PostgreSQL also provides the **jsonpath** data type

The **json** and j**sonb** data types accept *almost* identical sets of values as input. The major practical difference is one of efficiency. The **json** data type stores an exact copy of the input text, which processing functions must reparse on each execution; while **jsonb** data is stored in a decomposed binary format that makes it slightly slower to input due to added conversion overhead, but significantly faster to process, since no reparsing is needed. **jsonb** also supports indexing, which can be a significant advantage.



# Procedure:

1. Create a repository of documents containing six family member of yours(including yourself), with minimum seven attributes each, in POSTGRES
2. Perform selection and projection queries with different criterias on the created relation
3. Export the relation to json document
4. Import the document to MongoDB
5. Perform Insert, Search, Update, and Delete operations on the collection using
   1. MongoDB Compass
   2. MongoDB Shell
6. Demonstrate pipeline in MongoDB with minimum three (03) stages.



**Results:** *(Queries depicting the above said activity performed individually and snapshots of the results (if any))*

**USE COURIER NEW FONT WITH SIZE = 10 FOR QUERY STATEMENTS**

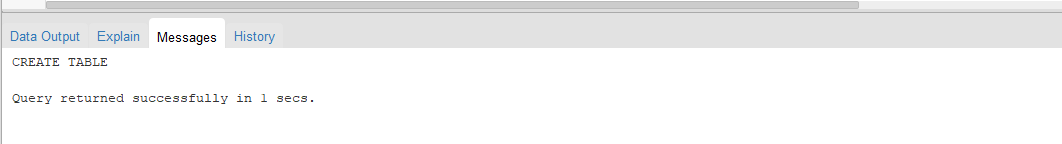
**CREATE TABLE family**

**(**

**id serial NOT NULL PRIMARY KEY,**

**info json NOT NULL**

**);**



**INSERT INTO family (info)**

**VALUES('{ "Member": "Grandfather", "attribute": {"Name": "TJ","Age": 80,"Gender":"Male","DOB":"05-1-1943","Phone\_Number":1111,"Email\_ID":"tj\_2207@yahoo.com","Blood\_Group":"O+ve"}}'),**

**('{ "Member": "Grandmother", "attribute": {"Name": "UL","Age": 73,"Gender":"Female","DOB":"06-11-1951","Phone\_Number":1112,"Email\_ID":"ul\_0709@hotmail.com","Blood\_Group":"B+ve"}}'),**

**('{ "Member": "Father", "attribute": {"Name": "ST","Age": 49,"Gender":"Male","DOB":"11-11-1974","Phone\_Number":1113,"Email\_ID":"st\_1805@gmail.com","Blood\_Group":"O+ve"}}'),**

**('{ "Member": "Mother", "attribute": {"Name": "NT","Age": 51,"Gender":"Female","DOB":"16-2-1972","Phone\_Number":1114,"Email\_ID":"nt\_1602@apple.com","Blood\_Group":"O-ve"}}'),**

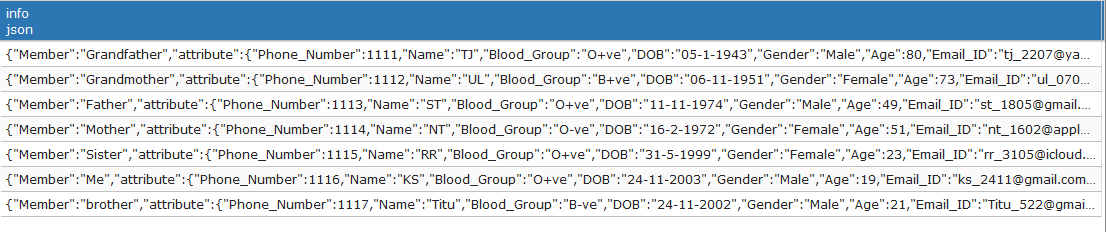
**('{ "Member": "Sister", "attribute": {"Name": "RR","Age": 23,"Gender":"Female","DOB":"31-5-1999","Phone\_Number":1115,"Email\_ID":"rr\_3105@icloud.com","Blood\_Group":"O+ve"}}'),**

**('{ "Member": "Me", "attribute": {"Name": "KS","Age": 19,"Gender":"Male","DOB":"24-11-2003","Phone\_Number":1116,"Email\_ID":"ks\_2411@gmail.com","Blood\_Group":"O+ve"}}'),**

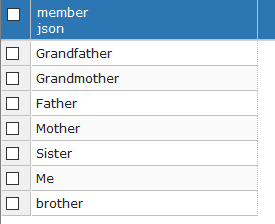
**('{ "Member": "brother", "attribute": {"Name": "Titu","Age": 21,"Gender":"Male","DOB":"24-11-2002","Phone\_Number":1117,"Email\_ID":"Titu\_522@gmail.com","Blood\_Group":"B-ve"}}');**



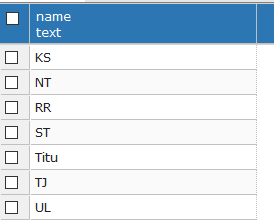
**SELECT info FROM family;**



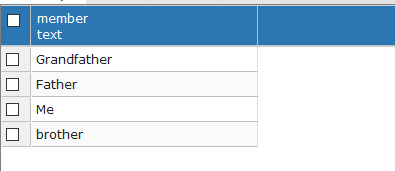
**SELECT info -> 'Member' AS Member FROM family;**



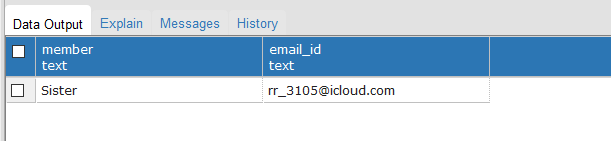
**SELECT info -> 'attribute' ->> 'Name' as Name FROM family ORDER BY Name;**



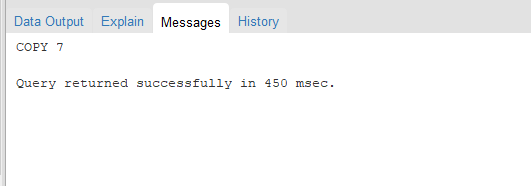
**SELECT info ->> 'Member' AS Member FROM family WHERE info -> 'attribute' ->> 'Gender' = 'Male';**



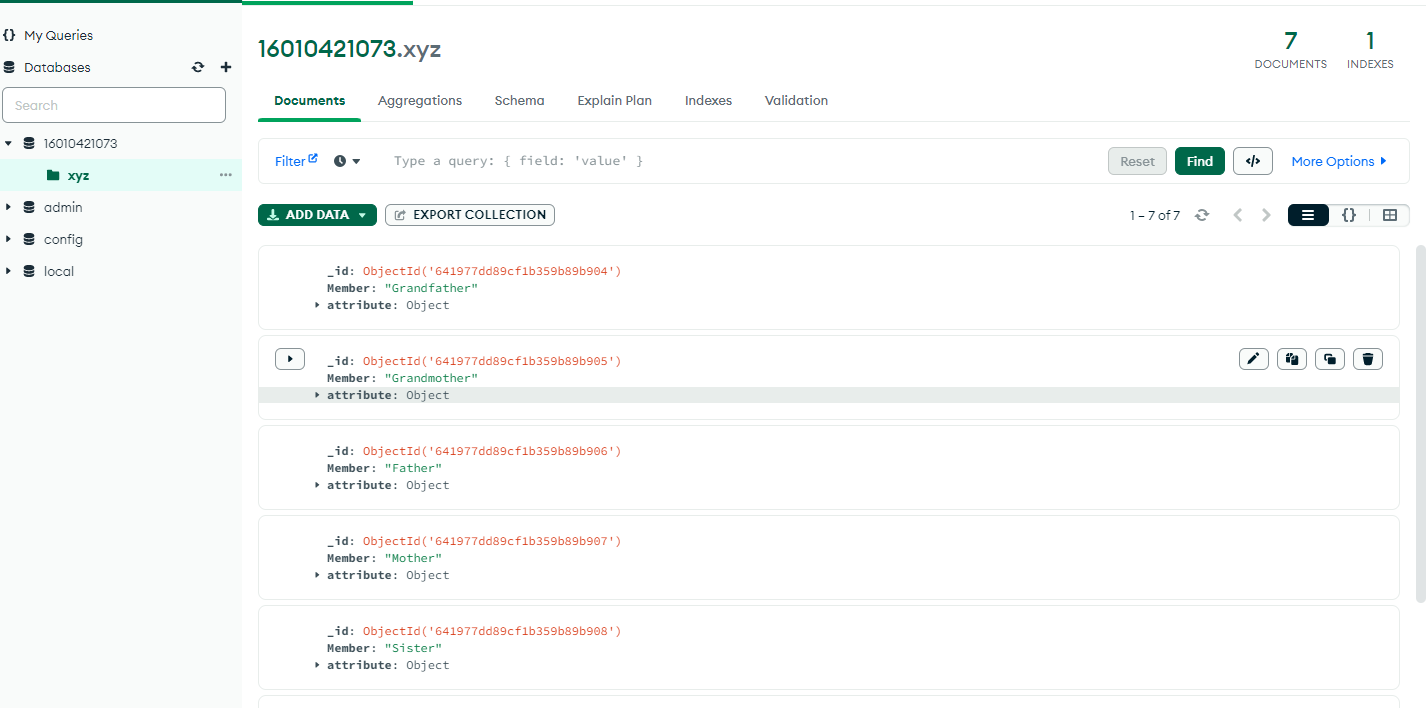
**SELECT info ->> 'Member' AS Member, info -> 'attribute' ->> 'Email\_ID' AS Email\_ID FROM Family WHERE CAST ( info -> 'attribute' ->> 'Age' AS INTEGER) = 23;**

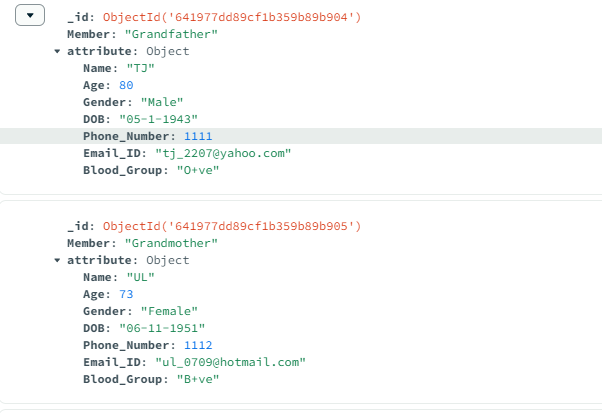


**COPY(SELECT \* FROM Family) TO 'E:\KP73\EXP6.json';**

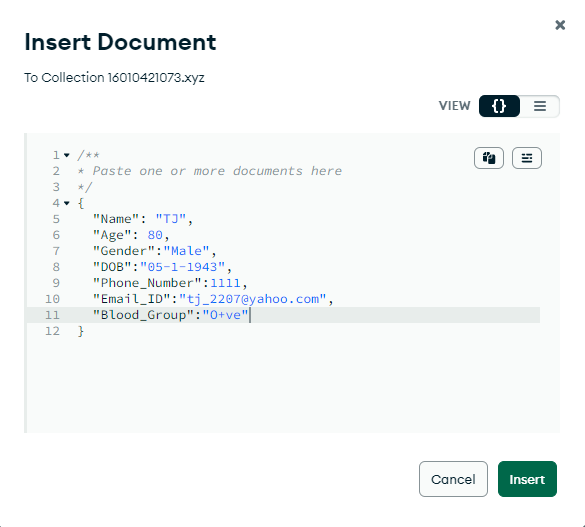


**MONGODB**





**Insert Operation(MongoDB Compass)**





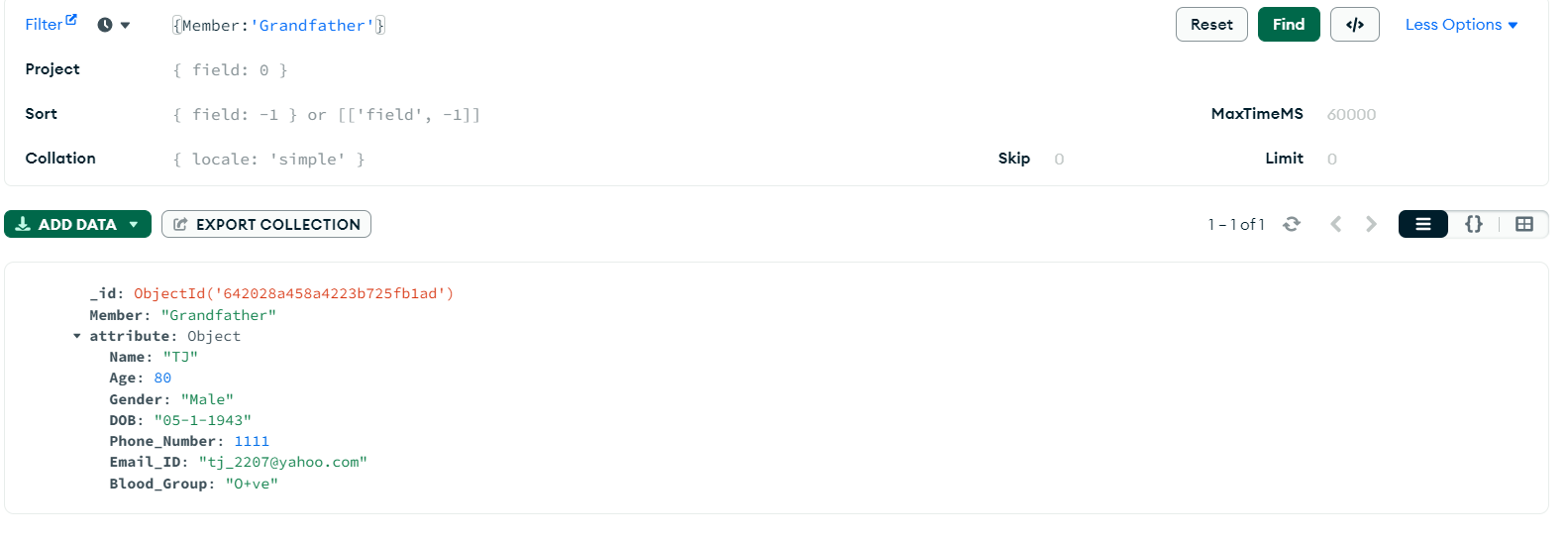
**Delete Operation(MongoDB Compass)**



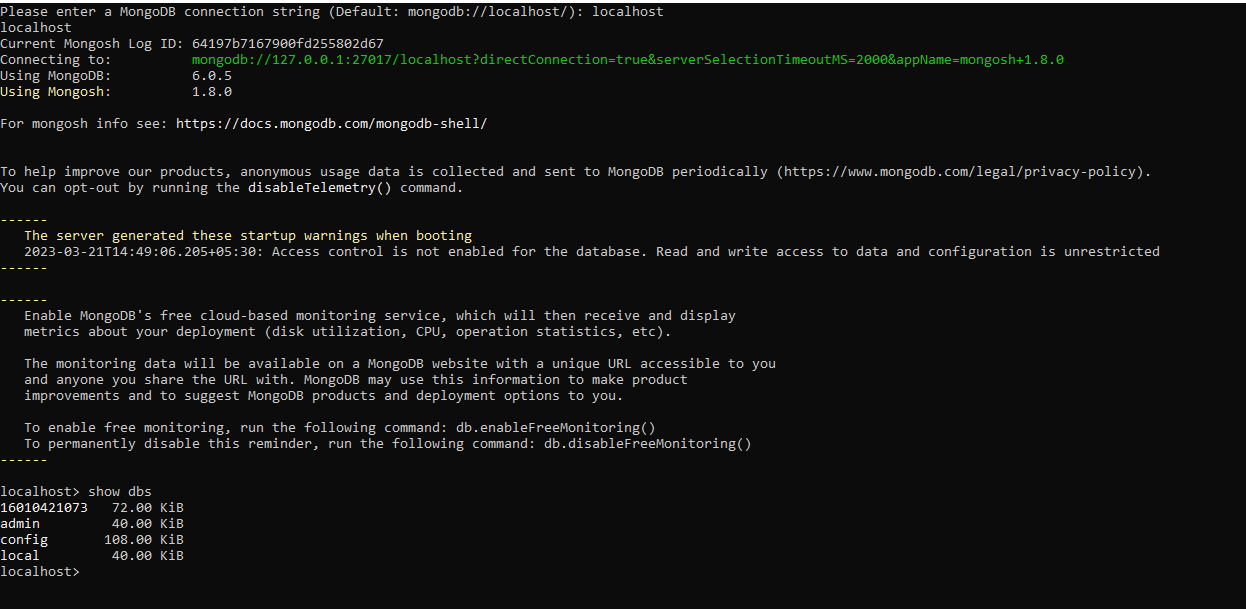
**Update Operation(MongoDB Compass)**



**Search Operation(MongoDB Compass)**

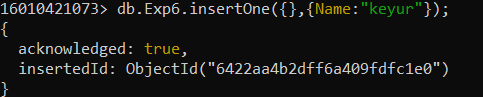


**MONGODB SHELL**

****

****

**Insert Operation(MongoDB Shell)**

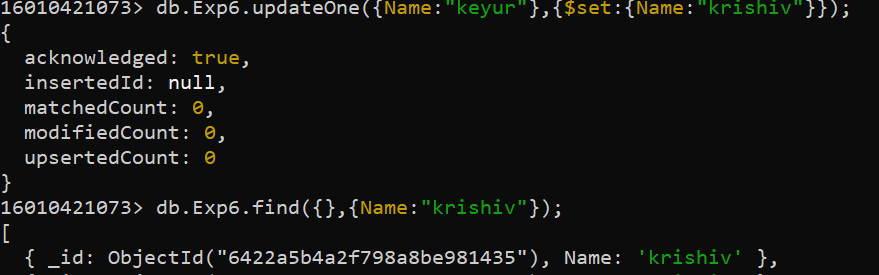
****

****

**Delete Operation(MongoDB Shell)**

****

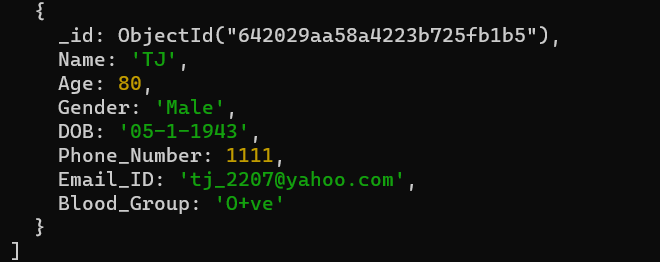
**Update Operation(MongoDB Shell)**

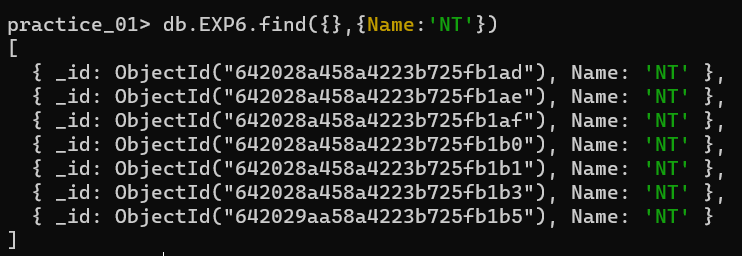
****

**Search Operation(MongoDB Shell)**

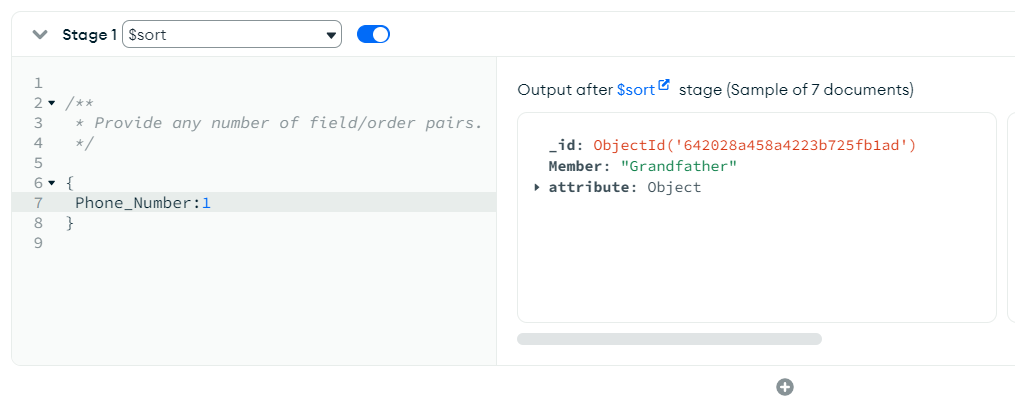
****

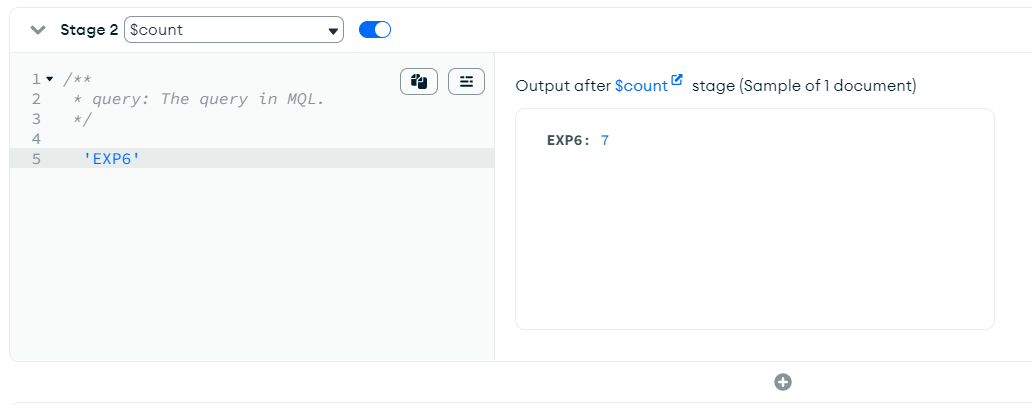
****

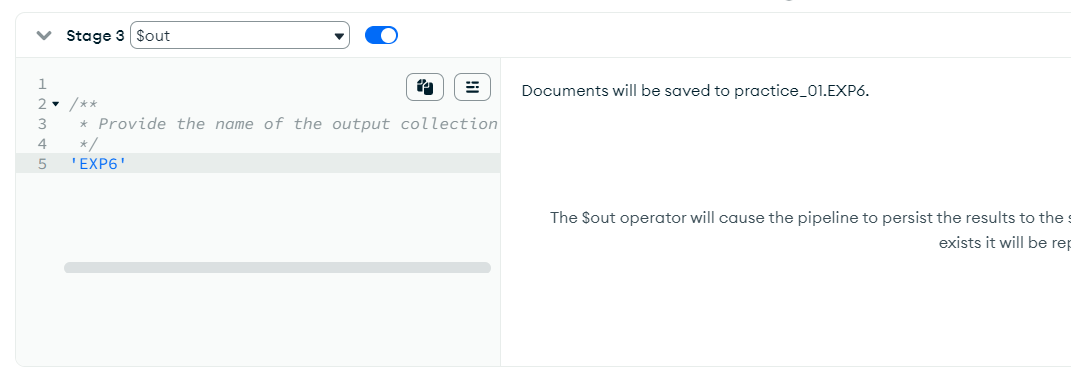
****

****

**PipeLine in MongoDb Compass(3stages)**

****

****

****



# Questions:

**Explain with query implementation on relation created by you**

# Any five jsonb specific operators in PostgreSQL

**Ans:**

* **-> Get JSON object field by key:**

This operator returns the value of the specified key in a JSON object. For example, SELECT '{"name": "John", "age": 30}'::jsonb -> 'name'; would return the value "John".

* **->> - Get JSON object field as text:**

This operator returns the value of the specified key in a JSON object as text. For example, SELECT '{"name": "John", "age": 30}'::jsonb ->> 'name'; would return the text "John".

* **#> - Get JSON object at specified path:**

This operator returns the value at the specified path in a JSON object. For example, SELECT '{"person": {"name": "John", "age": 30}}'::jsonb #> '{person, name}'; would return the value "John".

* **#>> - Get JSON object at specified path as text:**

This operator returns the value at the specified path in a JSON object as text. For example, SELECT '{"person": {"name": "John", "age": 30}}'::jsonb #>> '{person, name}'; would return the text "John".

* **- - Delete key/value pair or array element:**

This operator deletes the specified key/value pair or array element from a JSON object or array. For example, SELECT '{"name": "John", "age": 30}'::jsonb - 'age'; would return the JSON object {"name": "John"}.

1. **Any five collection methods in MongoDB**

# (Besides db.collection.insertOne, db.collection.deleteOne, db.collection.updateOne, db.collection.find)

**Ans:**

* **db.collection.findOneAndUpdate():**
  + - * + This method finds a document in a collection and updates it atomically.
        + It returns the updated document.
* **db.collection.aggregate():**

This method performs aggregation operations on a collection.

It can perform complex data transformations such as grouping, filtering, sorting, and computing new fields based on existing fields.

* **db.collection.distinct():**

This method returns an array of unique values for a specified field in a collection.

It is similar to the SQL DISTINCT keyword.

* **db.collection.countDocuments():**

This method returns the number of documents that match a specified filter in a collection.

* **db.collection.createIndex():**

This method creates an index on one or more fields in a collection. Indexes can improve query performance by allowing MongoDB to locate data more quickly.



**Outcomes:**

**CO2: Design advanced database systems using Object Relational, Spatial and NOSQL**

**Databases and its implementation**



# Conclusion: (Conclusion to be based on outcomes achieved)

# Hence we successfully implemented NOSQL Database using mongodb and postgresql.



**Grade: AA / AB / BB / BC / CC / CD /DD**

# Signature of faculty in-charge with date



**References:**

1. https:/[/www.mongodb.c](http://www.mongodb.com/basics)o[m/basics](http://www.mongodb.com/basics)
2. https:/[/www.post](http://www.postgresql.org/about/)g[resql.org/about/](http://www.postgresql.org/about/)
3. https:/[/www.post](http://www.postgresql.org/docs/13/datatype-json.html)g[resql.org/docs/13/datatype-json.html](http://www.postgresql.org/docs/13/datatype-json.html)