Experiment No.5

Title: NOSQL in MongoDB and PostgreSQL

Batch:B1 Roll No.:16010420133 Experiment No.:5

Aim: To implement NOSQL database using MongoDB and PostgreSQL.

Resources needed: MongoDB, PostgreSQL

Theory:

MongoDB:

MongoDB is a general-purpose document database 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
  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.

- **Collections:** Grouping Documents
  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
  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
  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 expressions, enabling you to process, transform, and analyze data of any structure at scale. One recent addition is the Union stage, which flexibly aggregate results from multiple collections.

Besides this MongoDB provides,

- variety of indexing strategies for speeding up the queries along with the Performance Advisor, 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, Perl, PHP, 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

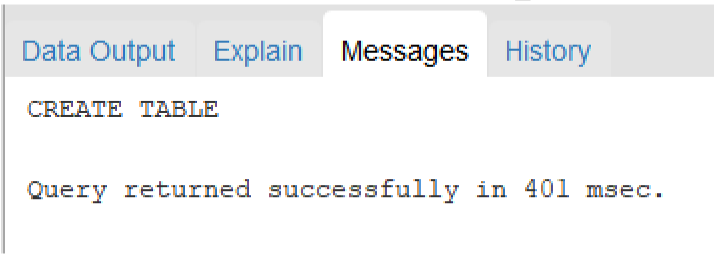
The **`json`** and **`jsonb`** data types accept *almost* identical sets of values as input. The major practical difference is one of efficiency. The **`json`** data type stores an exact copy of the input text, which processing functions must reparse on each execution; while **`jsonb`** data is stored 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
   i. MongoDB Compass
   ii. MongoDB Shell
6. Demonstrate pipeline in MongoDB with minimum three (03) stages.

Results:
POSTGRES JSONB

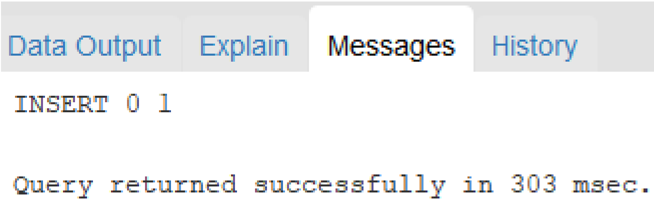
```
create table family1(id SERIAL,my_info jsonb NOT NULL);
```




```
INSERT INTO family(id,my_info)
    VALUES(1,
           '{ "name" :"SOUMEN SAMANTA",
              "age": "18",
              "gender": "M",
```

```
          "blood grp":"B+",
           "D.O.B":"24/07/2002",
          "Relationship":"Single",
          "PWD":"NO"}'

      );
```




```
INSERT INTO family(id,my_info)
  VALUES(1,
        '{ "name" :"VIRAT kohli",
          "age": "28",
          "gender": "M",
          "blood grp":"B+",
           "D.O.B":"22/08/2000",
          "Relationship":"married",
          "PWD":"NO"}'

  );

   INSERT INTO family(id,my_info)
  VALUES(1,
        '{ "name" :"XYZ",
          "age": "30",
          "gender": "M",
          "blood grp":"A+",
           "D.O.B":"25/09/1900",
          "Relationship":"married",
          "PWD":"NO"}'

  );

   INSERT INTO family(id,my_info)
  VALUES(1,
        '{ "name" :"ABC",
          "age": "48",
          "gender": "M",
          "blood grp":"B+",
           "D.O.B":"22/08/1945",
          "Relationship":"married",
          "PWD":"NO"}'

  );

   INSERT INTO family(id,my_info)
  VALUES(1,
        '{ "name" :"SAM",
          "age": "12",
          "gender": "F",
          "blood grp":"B+",
           "D.O.B":"22/08/2009",
          "Relationship":"single",
          "PWD":"YES"}'

  );
```

```
 INSERT INTO family(id,my_info)
VALUES(1,
       '{ "name" :"Tanmay",
          "age": "42",
          "gender": "M",
          "blood grp":"O-",
           "D.O.B":"22/08/2000",
          "Relationship":"married",
          "PWD":"NO"}'
     );
```

Data Output | Explain | Messages | History

```
INSERT 0 1

Query returned successfully in 311 msec.
```



```
Select * from family
```

Data Output | Explain | Messages | History

id integer	my_info jsonb
1	{"name":"SOUMEN SAMANTA","Relationship":"Single","gender":"M","age":"18","PWD":"NO","D.O.B":"24/07/2002","blood ...
2	{"name":"VIRAT kohli","Relationship":"married","gender":"M","age":"28","PWD":"NO","D.O.B":"22/08/2000","blood grp":"...
3	{"name":"XYZ","Relationship":"married","gender":"M","age":"30","PWD":"NO","D.O.B":"25/09/1900","blood grp":"A+"}
4	{"name":"ABC","Relationship":"married","gender":"M","age":"48","PWD":"NO","D.O.B":"22/08/1945","blood grp":"B+"}
5	{"name":"SAM","Relationship":"single","gender":"F","age":"12","PWD":"YES","D.O.B":"22/08/2009","blood grp":"B+"}
6	{"name":"Tanmay","Relationship":"married","gender":"M","age":"42","PWD":"NO","D.O.B":"22/08/2000","blood grp":"O-"}

```
select
* from family1;
```

```
alter table family1
add constraint family_is_object
check (jsonb_typeof(fam_info)='object');
```

Data Output | Explain | Messages | History

```
ALTER TABLE

Query returned successfully in 336 msec.
```

```
alter table family1 add constraint family_format check(
(fam_info->'Name') IS NOT NULL AND
    (fam_info->'Age') IS NOT NULL AND
```

```
    (fam_info->'Gender') IS NOT NULL AND
    (fam_info->'Blood Group') IS NOT NULL AND
    (fam_info->'Email Id') IS NOT NULL AND
    (fam_info->'Relationship') IS NOT NULL AND
    (fam_info->'DOB') IS NOT NULL
);
```

Data Output | Explain | Messages | History

```
ALTER TABLE

Query returned successfully in 551 msec.
```

```
CREATE UNIQUE INDEX unique_family on
family1((fam_info->>'Name'),(fam_info->>'Email Id'))
```

Data Output | Explain | Messages | History

```
CREATE INDEX

Query returned successfully in 373 msec.
```

MONGODB

INSERT

ADD DATA | VIEW | Displaying documents 1 - 6 of 6 | REFRESH

```
_id: ObjectId("624175ff2bb59816bc0e8668")
PWD: "NO"
age: "18"
name: "SOUMEN SAMANTA"
gender: "M"
blood grp: "B+"
Relationship: "Single"
```

```
_id: ObjectId("624175ff2bb59816bc0e8669")
PWD: "NO"
age: "28"
name: "VIRAT kohli"
gender: "M"
blood grp: "B+"
Relationship: "married"
```

```
_id: ObjectId("624175ff2bb59816bc0e866a")
PWD: "NO"
age: "30"
name: "XYZ"
gender: "M"
blood grp: "A+"
Relationship: "married"
```

```
_id: ObjectId("624175ff2bb59816bc0e866b")
PWD: "NO"
age: "48"
name: "ABC"
gender: "M"
blood grp: "B+"
Relationship: "married"
```

```
_id: ObjectId("624175ff2bb59816bc0e866c")
PWD: "YES"
age: "12"
name: "SAM"
gender: "F"
blood grp: "B+"
Relationship: "single"
```

```
_id: ObjectId("624175ff2bb59816bc0e866d")
PWD: "NO"
age: "42"
name: "Tanmay"
gender: "M"
blood grp: "O-"
Relationship: "married"
```

Search: {gender:"M"}

FILTER {gender:"M"} | OPTIONS | FIND | RESET

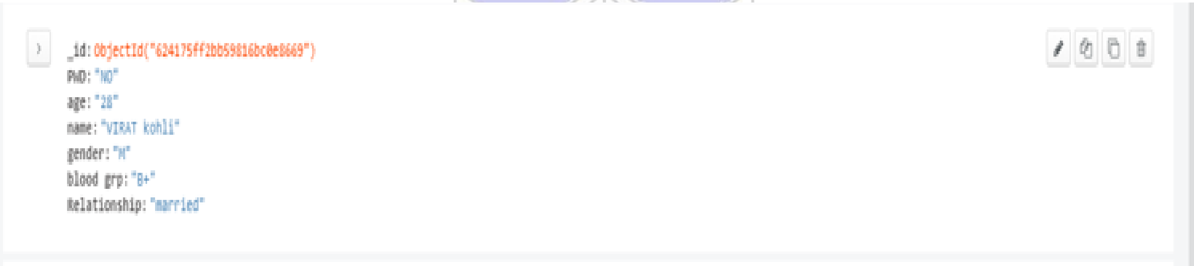
ADD DATA | VIEW | Displaying documents 1 - 5 of 5 | REFRESH

133

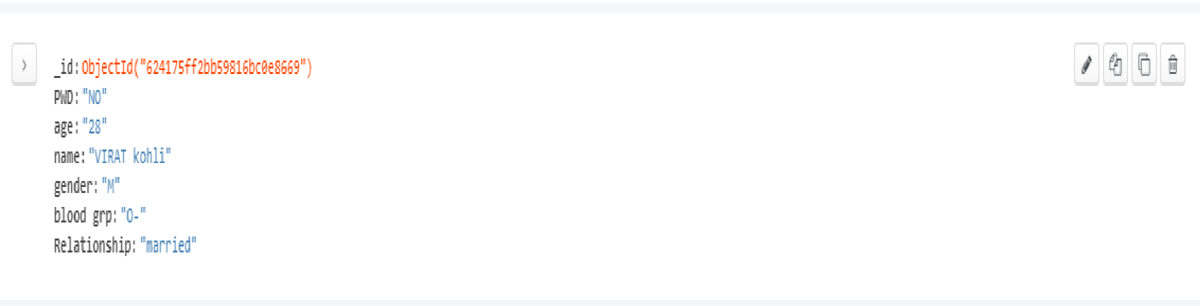
	_id ObjectId	PWD String	age String	name String	gender String	blood grp String
1	624175ff2bb59816bc0e8668	"NO"	"18"	"SOUMEN SAMANTA"	"M"	"B+"
2	624175ff2bb59816bc0e8669	"NO"	"28"	"VIRAT kohli"	"M"	"B+"
3	624175ff2bb59816bc0e866a	"NO"	"30"	"XYZ"	"M"	"A+"
4	624175ff2bb59816bc0e866b	"NO"	"48"	"ABC"	"M"	"B+"
5	624175ff2bb59816bc0e866d	"NO"	"42"	"Tanmay"	"M"	"O-"

UPDATE:

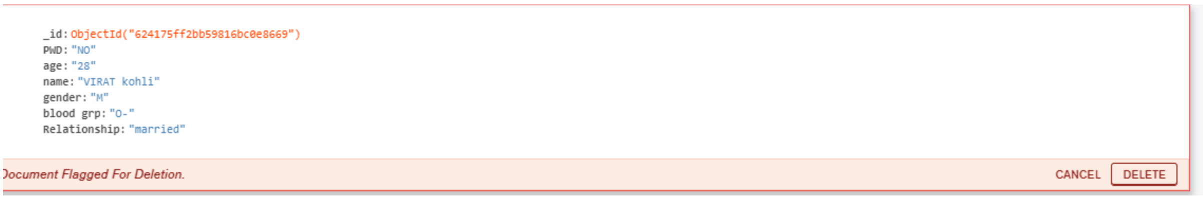
Before:

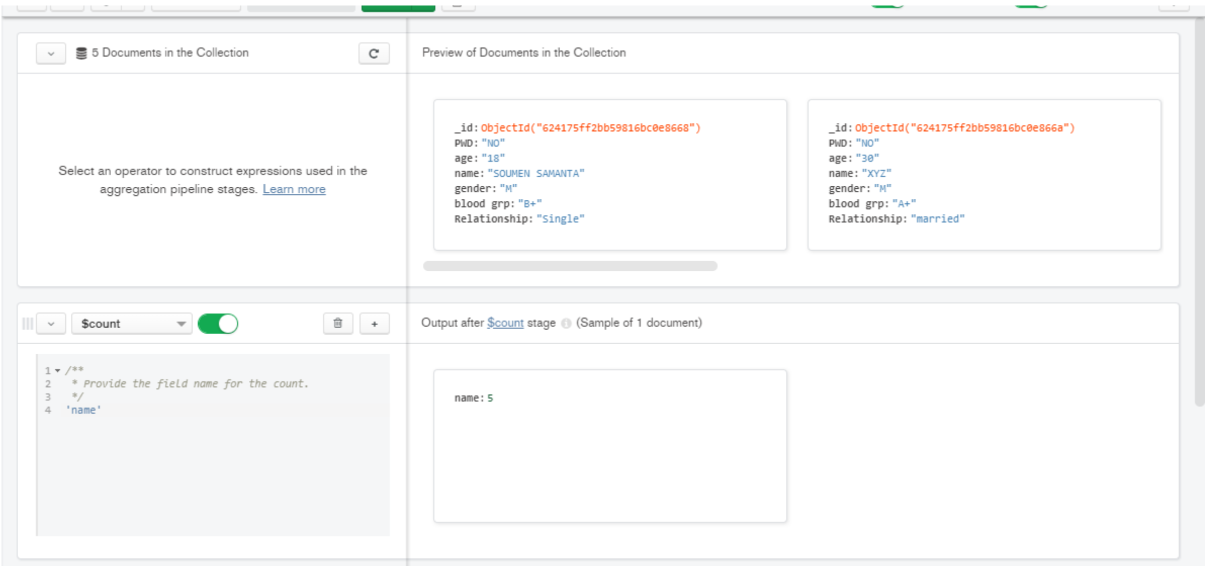


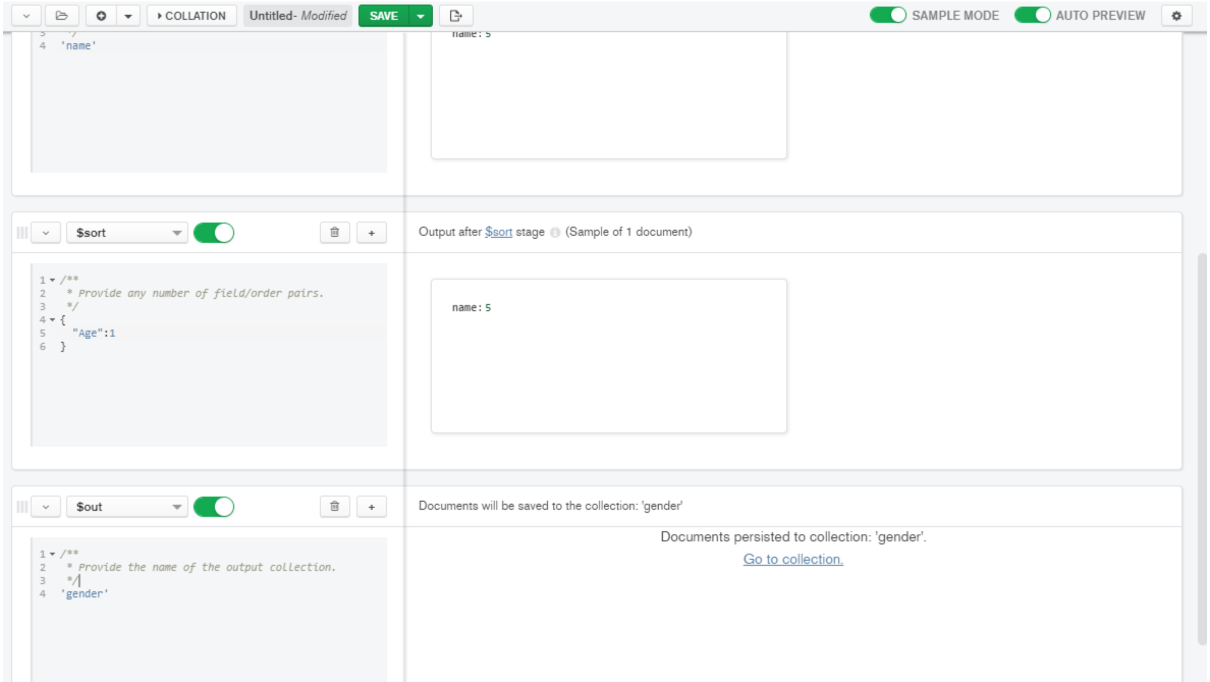
After : //blood grp is modified//



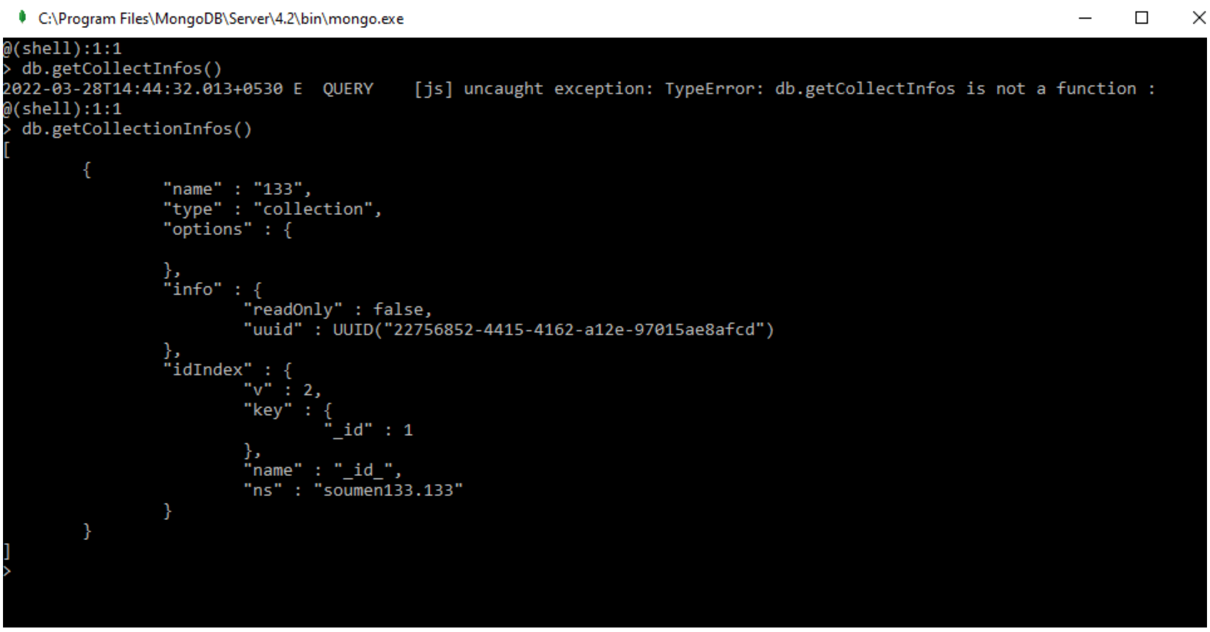
Delete







Command prompt



```
> db.getMongo()
connection to 127.0.0.1:27017
> db=connect("localhost:27017/soumen133")
connecting to: mongodb://localhost:27017/soumen133
Implicit session: session { "id" : UUID("efea4ca1-2d34-4e24-935b-9f2bba2ea5b5") }
MongoDB server version: 4.2.3
soumen133
```

```
> db.getCollectionInfos()
[
        {
                "name" : "133",
                "type" : "collection",
                "options" : {

                },
                "info" : {
                        "readOnly" : false,
                        "uuid" : UUID("22756852-4415-4162-a12e-97015ae8afcd")
                },
                "idIndex" : {
                        "v" : 2,
                        "key" : {
                                "_id" : 1
                        },
                        "name" : "_id_",
                        "ns" : "soumen133.133"
                }
        },
        {
                "name" : "exp5",
                "type" : "collection",
                "options" : {

                },
                "info" : {
                        "readOnly" : false,
                        "uuid" : UUID("c8f6b1c1-cacd-48b1-935a-23208d708987")
                },
                "idIndex" : {
                        "v" : 2,
                        "key" : {
                                "_id" : 1
                        },
                        "name" : "_id_",
                        "ns" : "soumen133.exp5"
                }
        },
        {
                "name" : "try",
                "type" : "collection",
                "options" : {

                },
                "info" : {
                        "readOnly" : false,
                        "uuid" : UUID("b260dc23-b81c-40ef-8d02-d75f925bf35e")
                },
                "idIndex" : {
```

```
                }
        },
        {
                "name" : "try",
                "type" : "collection",
                "options" : {

                },
                "info" : {
                        "readOnly" : false,
                        "uuid" : UUID("b260dc23-b81c-40ef-8d02-d75f925bf35e")
                },
                "idIndex" : {
                        "v" : 2,
                        "key" : {
                                "_id" : 1
                        },
                        "name" : "_id_",
                        "ns" : "soumen133.try"
                }
        }
]
> db.getCollectionNames()
[ "133", "exp5", "try" ]
```

Insert

```
> db.try.insertOne({"PWD":"No","Age":"19","name":"Scdj","gender":"M","blood grp":"b+","Relationship:":"Single"})
{
        "acknowledged" : true,
        "insertedId" : ObjectId("6241879f640138dfe0158ad9")
}
```

```
> db.try.insertOne({"PWD":"No","Age":"19","name":"sdfj","gender":"M","blood grp":"b+","Relationship:":"Single"})
{
        "acknowledged" : true,
        "insertedId" : ObjectId("6241885f640138dfe0158ada")
}
> db.try.insertOne({"PWD":"No","Age":"19","name":"fdrgt","gender":"M","blood grp":"b+","Relationship:":"Single"})
{
        "acknowledged" : true,
        "insertedId" : ObjectId("6241886b640138dfe0158adb")
}
```

```
> db.try.find();
{ "_id" : ObjectId("62417ff32bb59816bc0e866e"), "PWD" : "NO", "age" : "18", "name" : "SOUMEN SAMANTA", "gender" : "M", "blood grp" : "B+", "Relationship" : "Single" }
{ "_id" : ObjectId("6241879f640138dfe0158ad9"), "PWD" : "No", "Age" : "19", "name" : "Scdj", "gender" : "M", "blood grp" : "b+", "Relationship:" : "Single" }
{ "_id" : ObjectId("6241885f640138dfe0158ada"), "PWD" : "No", "Age" : "19", "name" : "sdfj", "gender" : "M", "blood grp" : "b+", "Relationship:" : "Single" }
{ "_id" : ObjectId("6241886b640138dfe0158adb"), "PWD" : "No", "Age" : "19", "name" : "fdrgt", "gender" : "M", "blood grp" : "b+", "Relationship:" : "Single" }
>
```

Delete

```
> db.try.deleteOne({"name":"Soumen Samanta"})
{ "acknowledged" : true, "deletedCount" : 0 }
```

Update:

```
> db.try.updateOne({name:"sdfj"},{$set:{name:"SDHJFJ"}})
{ "acknowledged" : true, "matchedCount" : 1, "modifiedCount" : 1 }
>
```

Questions:

Explain with query implementation on relation created by you

1. **Any five `jsonb` specific operators in PostgreSQL**

Operator	Right Operand Type	Description	Example	Example Result
->	int	Get JSON array element (indexed from zero, negative integers count from the end)	'[{"a":"foo"},{"b":"bar"},{"c":"baz"}]'::json->2	{"c":"baz"}
->	text	Get JSON object field by key	'{"a": {"b":"foo"}}'::json->'a'	{"b":"foo"}
->>	int	Get JSON array element as text	'[1,2,3]'::json->>2	3
->>	text	Get JSON object field as text	'{"a":1,"b":2}'::json->>'b'	2
#>	text[]	Get JSON object at specified path	'{"a": {"b":{"c": "foo"}}}'::json#>'{a,b}'	{"c": "foo"}
#>>	text[]	Get JSON object at specified path as text	'{"a":[1,2,3],"b":[4,5,6]}'::json#>>'{a,2}'	3

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

db.collection.bulkWrite()
The bulkWrite() method performs multiple write operations with the order of execution control. Array of write operations are executed by this operation. Operations are executed in a specific order by default.

db.collection.count(query, option)
The count() method returns the number of documents that would match a find method query for the collection or view.

Db.collection.countDocuments(query, options)
The countDocument() method returns the number of documents that match the query for a collection or view. it does not use the metadata to return the count.

db.collection.dataSize()
The data size method has a cover around the output of the collStats (i.e. db.collection.stats()) command.

db.collection.aggregate(pipeline, option)
The aggregate method calculates mass values for the data in a collection/table or in a view.
Pipeline: It is an array of mass data operations or stages. It can accept the pipeline as a separate argument, not as an element in an array. If the pipeline is not specified as an array, then the second parameter will not be specified.
Option: A document that passes the aggregate command. It will be available only when you specify the pipeline as an array.

Outcomes:

CO2: Design advanced database systems using Object Relational, Spatial and NOSQL Databases and its implementation.

Conclusion: (Conclusion to be based on outcomes achieved)

Installed MongoDB and designed a NoSQL database using it and implemented different queries to insert, update, delete and retrieve data. Also explored different methods in MongoDB.

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

Signature of faculty in-charge with date

References:

1. https://www.mongodb.com/basics
2. https://www.postgresql.org/about/
3. https://www.postgresql.org/docs/13/datatype-json.html