Experiment No. 6

Title: NOSQL in MongoDB and PostgreSQL

Batch: A2 Roll No.: 16010421059 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 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 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
   i. MongoDB Compass
   ii. MongoDB Shell
6. Demonstrate pipeline in MongoDB with minimum three (03) stages.

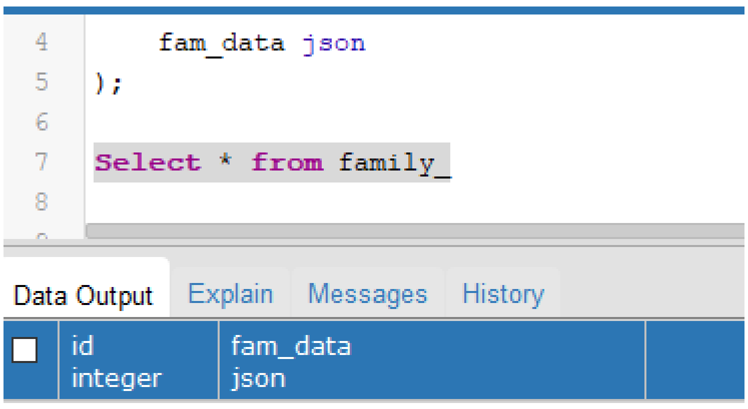
Results: *(Queries depicting the above said activity performed individually and snapshots of the results (if any))*

Query:

```
create table family_
(
        id serial PRIMARY KEY,
   fam_data json
);

Select * from family_
```

```
INSERT INTO family_ (fam_data)
VALUES
('{"Member": "Chinmay", "information": {"age": 19,"phone":
"1235246643","dob":"2/11/2003","gender":"M","email":"chinmay.mhatre@somaiya.edu","blo
od-group":"B+"}}'),
('{"Member": "Atharva", "information": {"age": 19,"phone":
"999999999","dob":"10/06/2003","gender":"M","email":"atharva@somaiya.edu","blood-
group":"A+"}}'),


('{"Member": "Pranav", "information": {"age": 22,"phone":
"1242141231","dob":"32/12/2003","gender":"M","email":"pranav@somaiya.edu","blood-
group":"O+"}}'),
('{"Member": "Manan", "information": {"age": 23,"phone":
"1243532154","dob":"18/03/2003","gender":"Pdf","email":"Manan@somaiya.edu","blood-
group":"B+"}}'),
('{"Member": "Arya", "information": {"age": 24,"phone":
"1245634221","dob":"20/10/2003","gender":"M","email":"Arya.N@somaiya.edu","blood-
group":"B+"}}'),
('{"Member": "vedant", "information": {"age": 25,"phone":
"7021875752","dob":"12/11/2003","gender":"M","email":"vedant@somaiya.edu","blood-
group":"B+"}}');

commit;

SELECT fam_data -> 'information' ->> 'age' AS age
FROM family_
order by age;

SELECT fam_data ->> 'Member' AS member_ from family_
where fam_data -> 'information' ->> 'age' = '19';


SELECT fam_data -> 'information' ->> 'gender' AS gender
FROM family_
order by gender;
COPY(SELECT * FROM family_) TO 'C:\harsh6\ad6.json';


SELECT fam_data -> 'Member' AS member_
FROM family_;
```

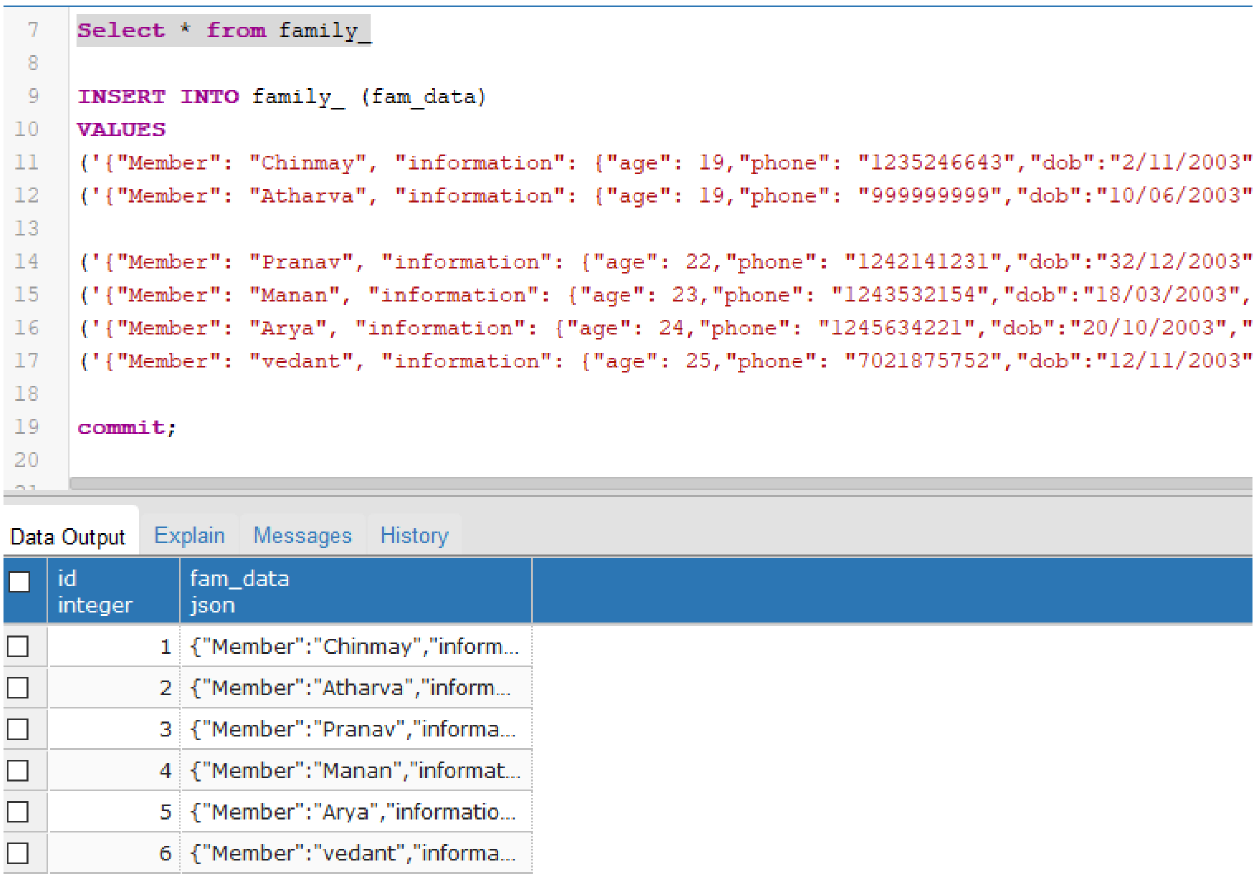
SQL Creation:



```
    fam_data json
);

Select * from family_
```

Data Output | Explain | Messages | History

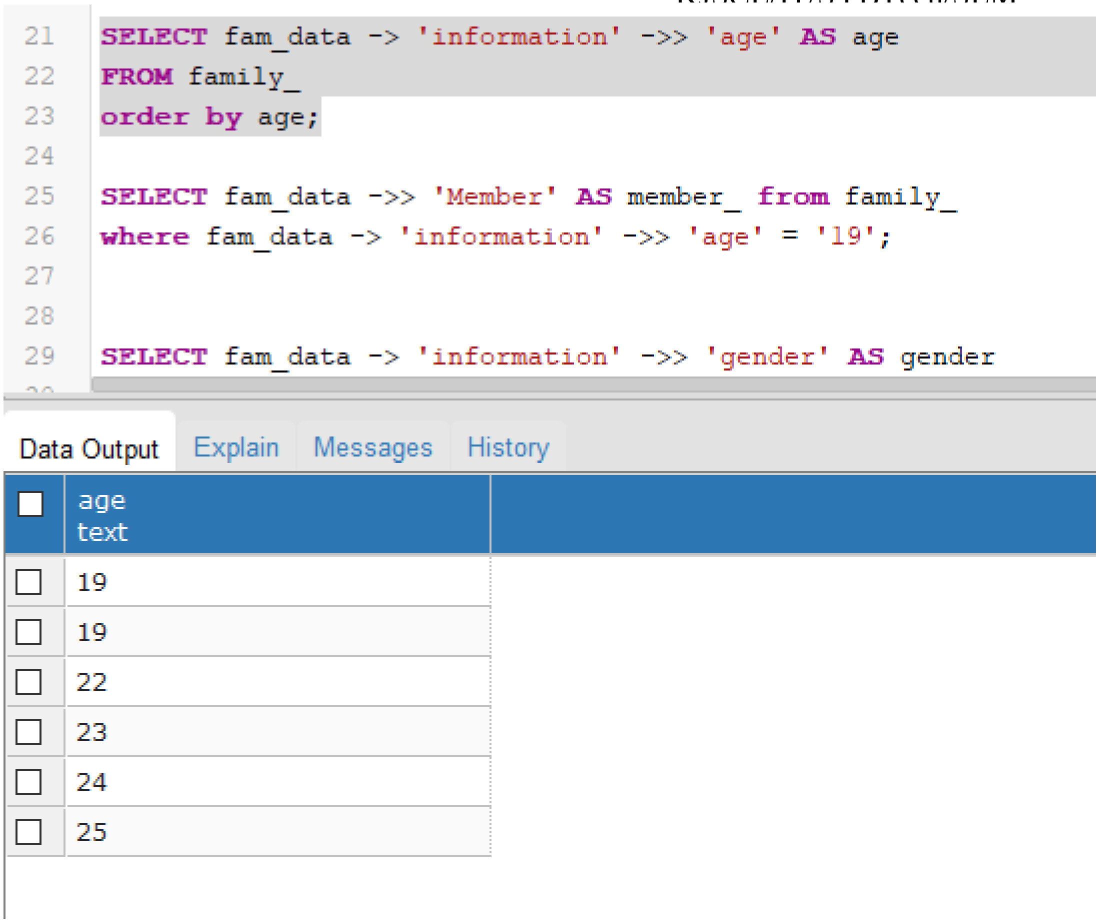
```
id        fam_data
integer   json
```



```
Select * from family_

INSERT INTO family_ (fam_data)
VALUES
('{"Member": "Chinmay", "information": {"age": 19,"phone": "1235246643","dob":"2/11/2003"
('{"Member": "Atharva", "information": {"age": 19,"phone": "999999999","dob":"10/06/2003"

('{"Member": "Pranav", "information": {"age": 22,"phone": "1242141231","dob":"32/12/2003"
('{"Member": "Manan", "information": {"age": 23,"phone": "1243532154","dob":"18/03/2003",
('{"Member": "Arya", "information": {"age": 24,"phone": "1245634221","dob":"20/10/2003","
('{"Member": "vedant", "information": {"age": 25,"phone": "7021875752","dob":"12/11/2003"

commit;
```

Data Output | Explain | Messages | History

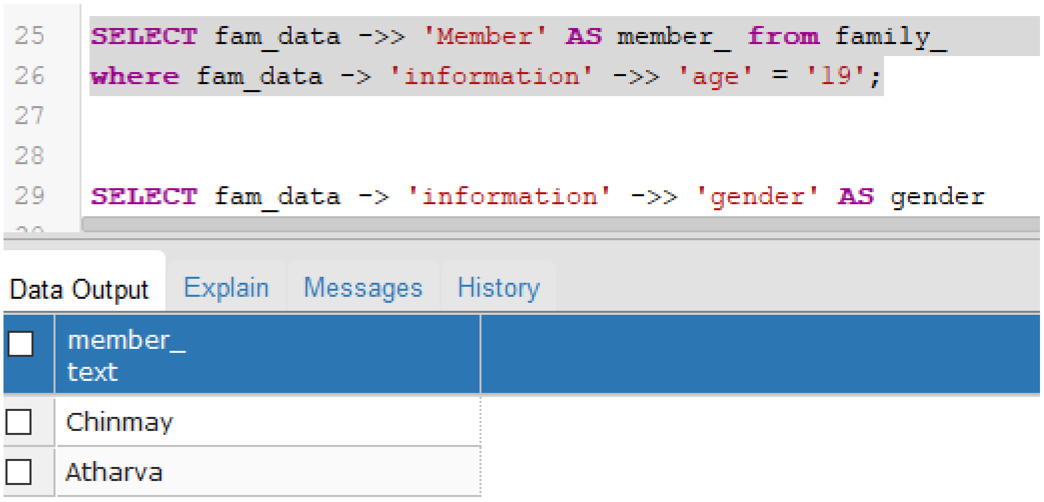
```
id        fam_data
integer   json
      1   {"Member":"Chinmay","inform...
      2   {"Member":"Atharva","inform...
      3   {"Member":"Pranav","informa...
      4   {"Member":"Manan","informat...
      5   {"Member":"Arya","informatio...
      6   {"Member":"vedant","informa...
```



```
SELECT fam_data -> 'information' ->> 'age' AS age
FROM family_
order by age;

SELECT fam_data ->> 'Member' AS member_ from family_
where fam_data -> 'information' ->> 'age' = '19';


SELECT fam_data -> 'information' ->> 'gender' AS gender
```

Data Output | Explain | Messages | History

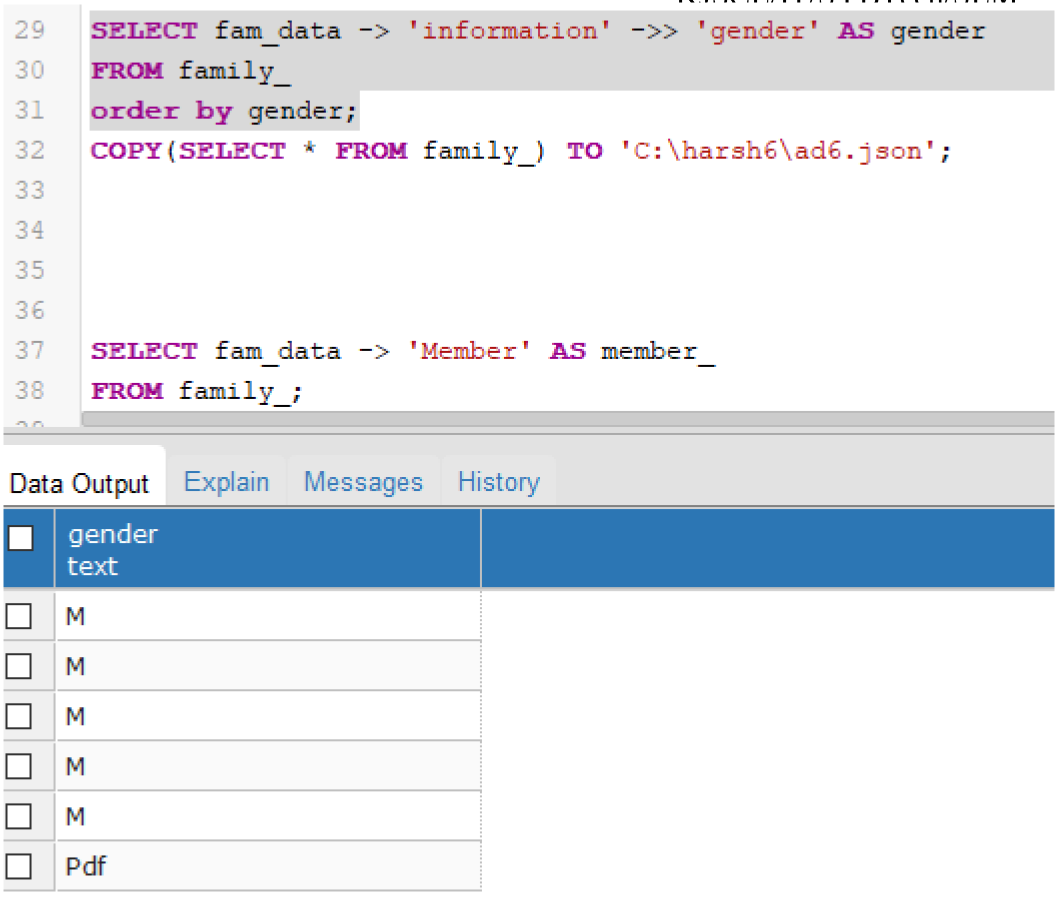
```
age
text
19
19
22
23
24
25
```



```
SELECT fam_data ->> 'Member' AS member_ from family_
where fam_data -> 'information' ->> 'age' = '19';


SELECT fam_data -> 'information' ->> 'gender' AS gender
```

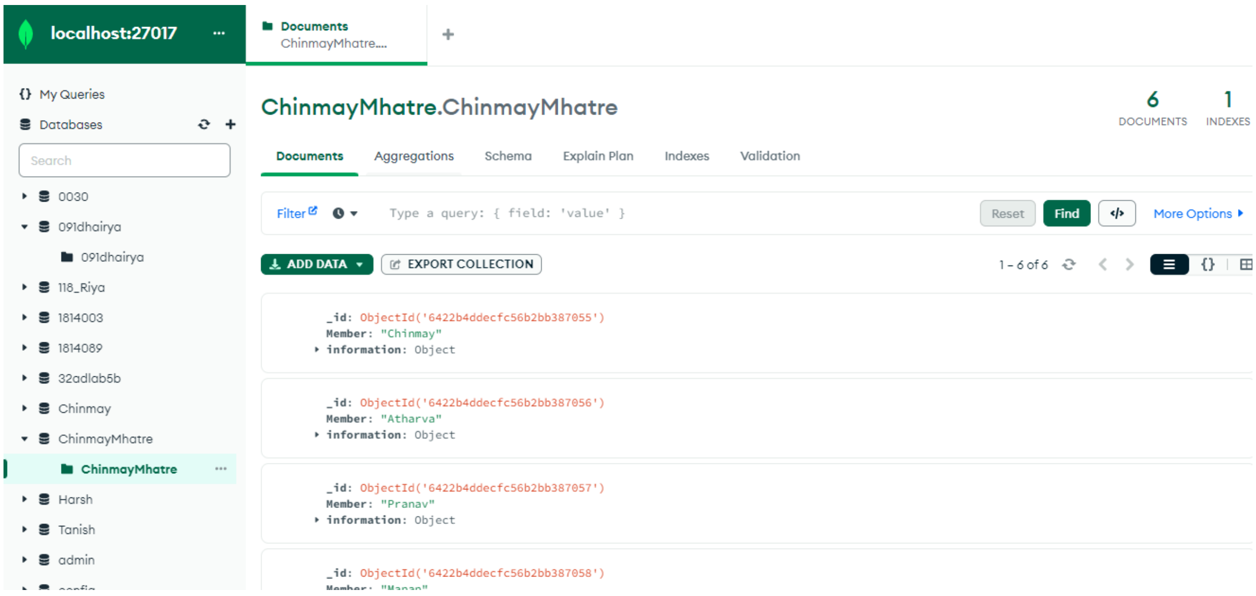
Data Output | Explain | Messages | History

```
member_
text
Chinmay
Atharva
```



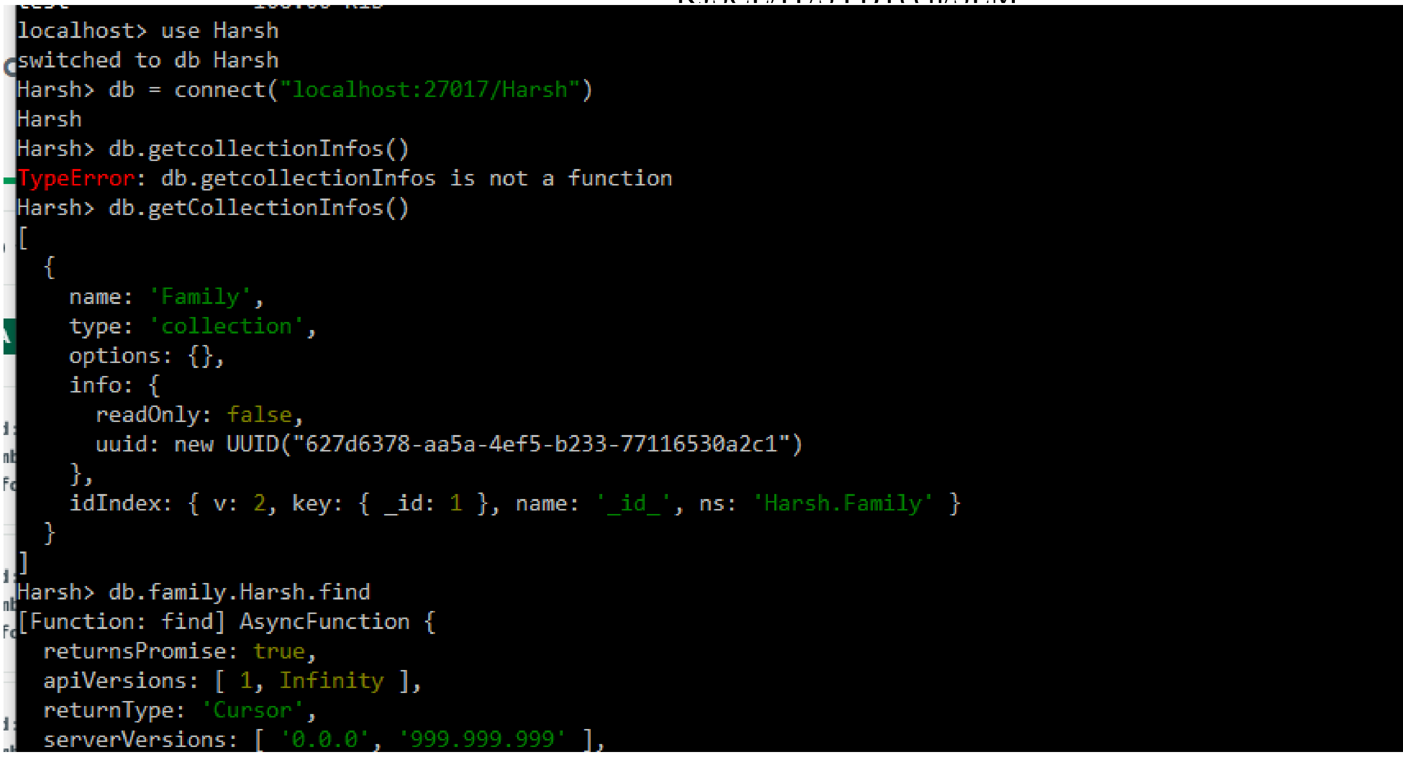
```
SELECT fam_data -> 'information' ->> 'gender' AS gender
FROM family_
order by gender;
COPY(SELECT * FROM family_) TO 'C:\harsh6\ad6.json';




SELECT fam_data -> 'Member' AS member_
FROM family_;
```

gender text
M
M
M
M
M
Pdf



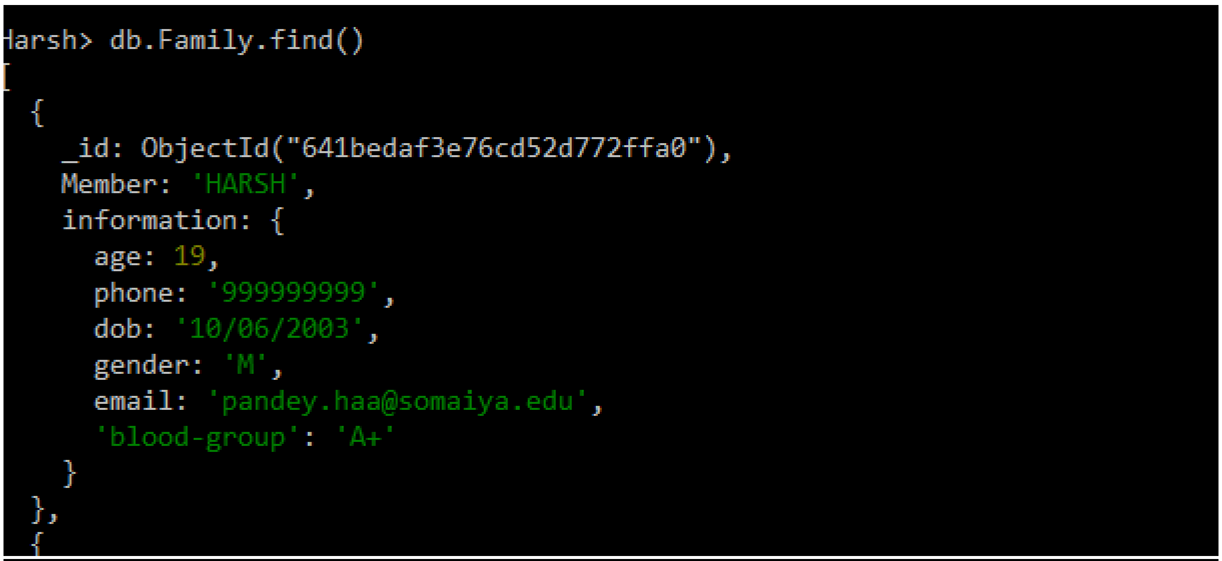
```
_id: ObjectId('6422b4ddecfc56b2bb387055')
Member: "Chinmay"
▸ information: Object

_id: ObjectId('6422b4ddecfc56b2bb387056')
Member: "Atharva"
▸ information: Object

_id: ObjectId('6422b4ddecfc56b2bb387057')
Member: "Pranav"
▸ information: Object

_id: ObjectId('6422b4ddecfc56b2bb387058')
```

<u>Connection to MongoDb:</u>



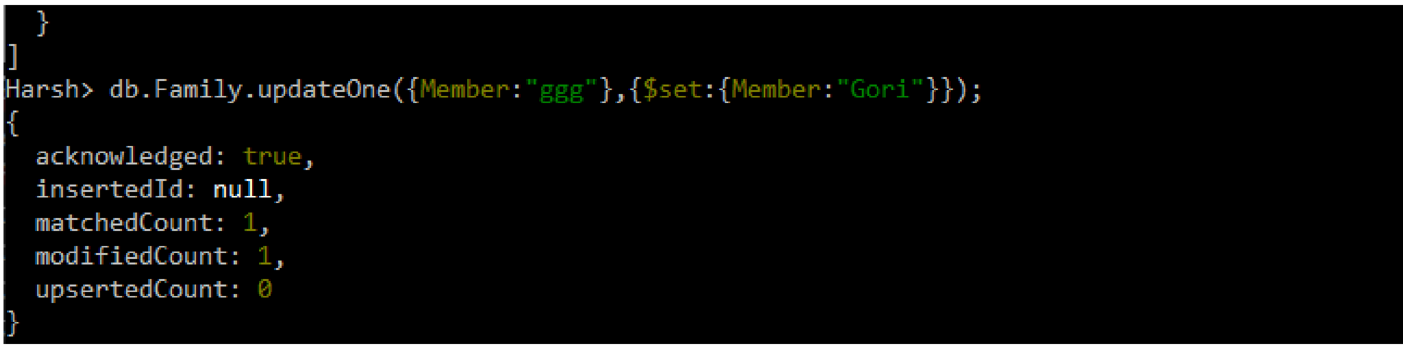
```
localhost> use Harsh
switched to db Harsh
Harsh> db = connect("localhost:27017/Harsh")
Harsh
Harsh> db.getcollectionInfos()
TypeError: db.getcollectionInfos is not a function
Harsh> db.getCollectionInfos()
[
  {
    name: 'Family',
    type: 'collection',
    options: {},
    info: {
      readOnly: false,
      uuid: new UUID("627d6378-aa5a-4ef5-b233-77116530a2c1")
    },
    idIndex: { v: 2, key: { _id: 1 }, name: '_id_', ns: 'Harsh.Family' }
  }
]
Harsh> db.family.Harsh.find
[Function: find] AsyncFunction {
  returnsPromise: true,
  apiVersions: [ 1, Infinity ],
  returnType: 'Cursor',
  serverVersions: [ '0.0.0', '999.999.999' ],
```

Find Cmnd in MongoDb:

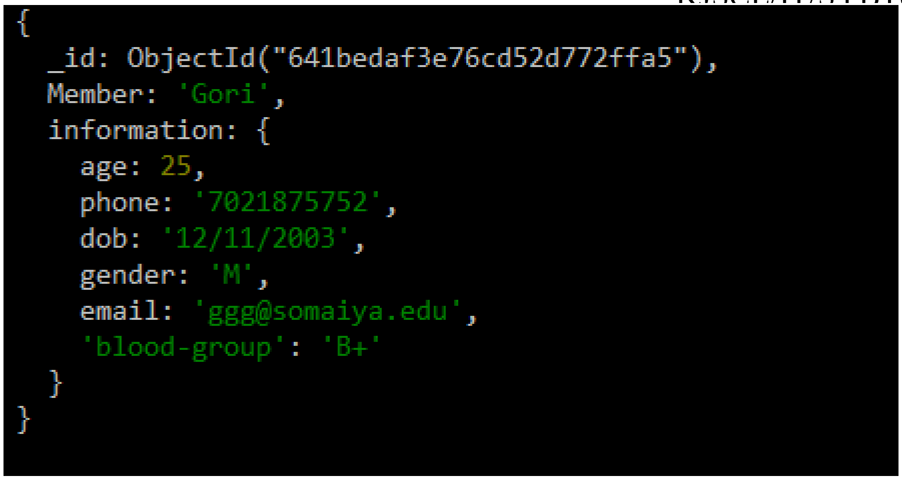


```
Harsh> db.Family.find()
[
  {
    _id: ObjectId("641bedaf3e76cd52d772ffa0"),
    Member: 'HARSH',
    information: {
      age: 19,
      phone: '999999999',
      dob: '10/06/2003',
      gender: 'M',
      email: 'pandey.haa@somaiya.edu',
      'blood-group': 'A+'
    }
  },
  {
```

Updation:

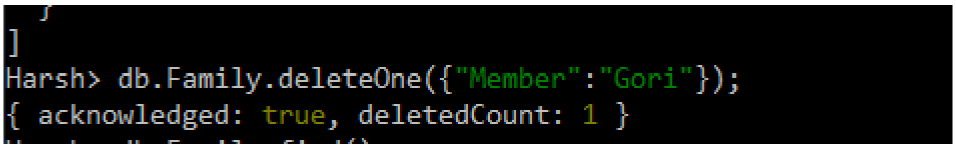


```
  }
]
Harsh> db.Family.updateOne({Member:"ggg"},{$set:{Member:"Gori"}});
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
```

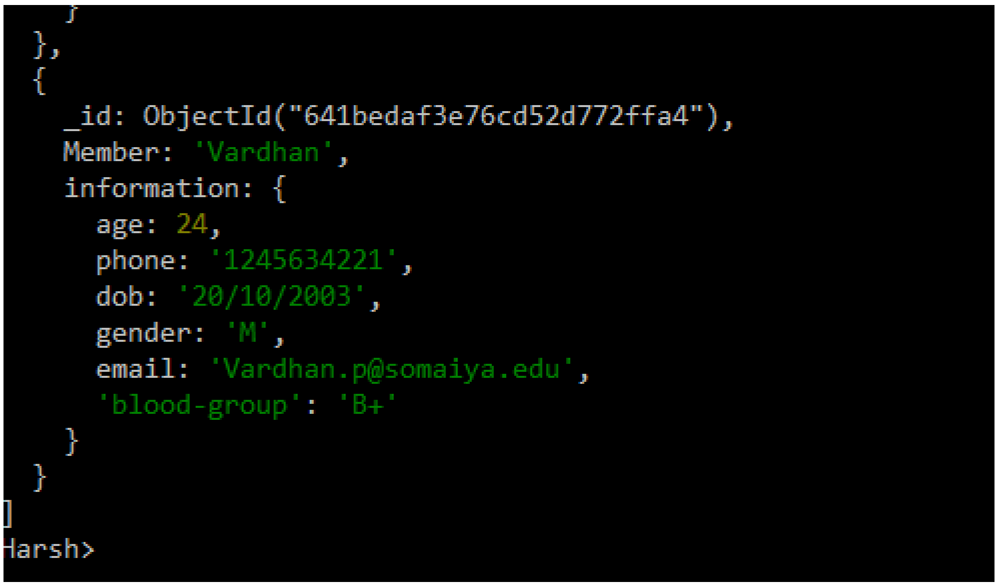


```
{
  _id: ObjectId("641bedaf3e76cd52d772ffa5"),
  Member: 'Gori',
  information: {
    age: 25,
    phone: '7021875752',
    dob: '12/11/2003',
    gender: 'M',
    email: 'ggg@somaiya.edu',
    'blood-group': 'B+'
  }
}
```

DELETION:



```
]
Harsh> db.Family.deleteOne({"Member":"Gori"});
{ acknowledged: true, deletedCount: 1 }
```



```
  },
  {
    _id: ObjectId("641bedaf3e76cd52d772ffa4"),
    Member: 'Vardhan',
    information: {
      age: 24,
      phone: '1245634221',
      dob: '20/10/2003',
      gender: 'M',
      email: 'Vardhan.p@somaiya.edu',
      'blood-group': 'B+'
    }
  }
]
Harsh>
```

Questions:

Explain with query implementation on relation created by you

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

Here the operators that are available for use with JSON data.

JSON Operators

Operator Description

Example
->
Get JSON array element

Operand type : int

'[1,2,3]'::json->2
->

Get JSON object field

Operand type : text

'{"a":1,"b":2}'::json->'b'
->>

Get JSON array element as text
Operand type : int

'[1,2,3]'::json->>2
->>

Get JSON object field as text
Operand type : text

'{"a":1,"b":2}'::json->>'b'
#>
Get JSON object at specified path

Operand type : array of text

'{"a":[1,2,3],"b":[4,5,6]}'::json#>'{a,2}'
#>>
Get JSON object at specified path as text

Operand type : array of text

'{"a":[1,2,3],"b":[4,5,6]}'::json#>>'{a,2}'

2. **Any five collection methods in MongoDB**
(Besides db.collection.insertOne, db.collection.deleteOne, db.collection.updateOne, db.collection.find)

   1. Compass
   2. Shell
   3. VS Code Plugin
   4. Atlas CLI
   5. Database Connectors
   6. Cluster-to-Cluster Sync
   7. Mongoose ODM Support
   8. Relational Migrator

Outcomes:

CO: 2 : - NOSQL Databases and JSON

Conclusion: (Conclusion to be based on outcomes achieved)

We can conclude that we have learnt about NOSQL Databases and JSON data

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