Experiment No. 6

Title: NOSQL in MongoDB and PostgreSQL

Batch: B-4 **Roll No.: 16010422234** **Name: Chandana Ramesh Galgali**

Experiment No.:6

Aim: To implement a 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

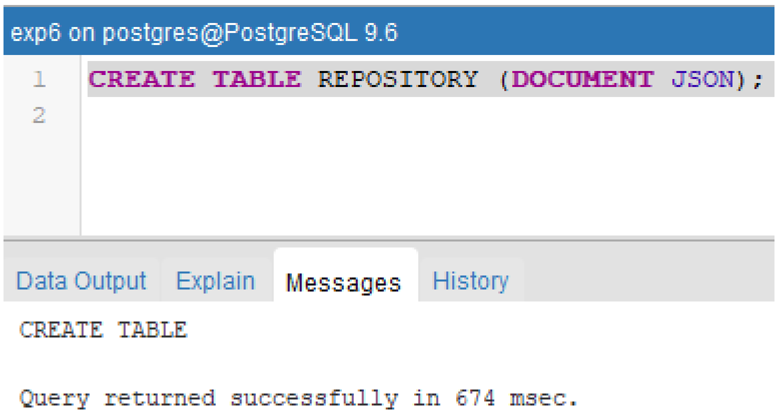
The `json` and `jsonb` data types accept *almost* identical sets of values as input. The major practical difference is one of efficiency. The `json` data type stores an exact copy of the input text, which processing functions must reparse on each execution; while `jsonb` data is stored in a decomposed 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.
   The video resources link is
   https://drive.google.com/drive/folders/1z1KeZZH5LS_CHRYOKJlmB8o9_a4HFod9?usp=sharing

---

Results: *(Queries depicting the above said activity performed individually and snapshots of the results (if any))*
PostGRESQL:
```
CREATE TABLE REPOSITORY (DOCUMENT JSON);
INSERT INTO REPOSITORY VALUES ('{"id":1, "name":"Chandana Galgali",
"age":19, "gender":"Female", "relationship":"Self",
"occupation":"Engineer", "city":"New York",
"email":"chandana.g@example.com"}');
INSERT INTO REPOSITORY VALUES ('{"id":2, "name":"Nayu Galgali", "age":13,
"gender":"Female", "relationship":"Sister", "occupation":"Student",
"city":"Mumbai", "email":"nayu.g@example.com"}');
```

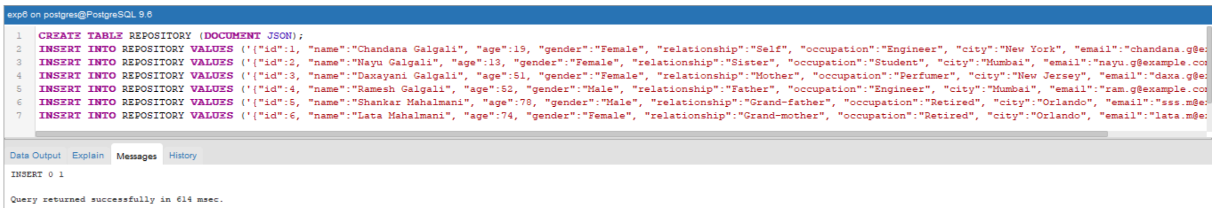
```
INSERT INTO REPOSITORY VALUES ('{"id":3, "name":"Daxayani Galgali",
"age":51, "gender":"Female", "relationship":"Mother",
"occupation":"Perfumer", "city":"New Jersey",
"email":"daxa.g@example.com"}');
INSERT INTO REPOSITORY VALUES ('{"id":4, "name":"Ramesh Galgali", "age":52,
"gender":"Male", "relationship":"Father", "occupation":"Engineer",
"city":"Mumbai", "email":"ram.g@example.com"}');
INSERT INTO REPOSITORY VALUES ('{"id":5, "name":"Shankar Mahalmani",
"age":78, "gender":"Male", "relationship":"Grand-father",
"occupation":"Retired", "city":"Orlando", "email":"sss.m@example.com"}');
INSERT INTO REPOSITORY VALUES ('{"id":6, "name":"Lata Mahalmani", "age":74,
"gender":"Female", "relationship":"Grand-mother", "occupation":"Retired",
"city":"Orlando", "email":"lata.m@example.com"}');
SELECT * FROM REPOSITORY;
SELECT DOCUMENT->'name', DOCUMENT->'age' AS NAME_AGE FROM REPOSITORY;
SELECT DOCUMENT->'name', DOCUMENT->'occupation' AS PARENT_NAME_OCC FROM
REPOSITORY WHERE DOCUMENT->>'relationship' = 'Mother' or
DOCUMENT->>'relationship' = 'Father';
SELECT COUNT(*) AS RETIRED_MEMBERS FROM REPOSITORY WHERE
DOCUMENT->>'occupation' = 'Retired';
SELECT DOCUMENT->>'name' AS NAME, DOCUMENT->>'age' AS AGE FROM REPOSITORY
WHERE DOCUMENT->>'gender' = 'Female' ORDER BY DOCUMENT->>'age';
SELECT (DOCUMENT->'name')::jsonb || (DOCUMENT->'relationship')::jsonb AS
FAMILY_MEMBERS FROM REPOSITORY;
SELECT r.DOCUMENT->>'occupation' AS OCCUPATION, COUNT(*) AS COUNT FROM
REPOSITORY AS r GROUP BY r.DOCUMENT->>'occupation';
COPY(SELECT * FROM REPOSITORY)TO 'C:\Users\Public\rep.json';
CREATE TABLE DOCUMENTS
(
      id SERIAL,
      document jsonb NOT NULL
);
ALTER TABLE DOCUMENTS
ADD CONSTRAINT DOCUMENTS_IS_OBJECT
CHECK (jsonb_typeof(document)='object');
ALTER TABLE DOCUMENTS
ADD CONSTRAINT DOCUMENTS_FORMAT
CHECK (
      (document->'name') IS NOT NULL AND
      (document->'age') IS NOT NULL AND
      (document->'gender') IS NOT NULL AND
      (document->'relationship') IS NOT NULL AND
      (document->'occupation') IS NOT NULL AND
      (document->'city') IS NOT NULL AND
      (document->'email') IS NOT NULL AND
      ((document->'email')::TEXT) LIKE '_%@%.%'
);
INSERT INTO DOCUMENTS(id, document)
VALUES (1,
            '{ "name":"Chandana Galgali",
            "age":19,
            "gender":"Female",
            "relationship":"Self",
            "occupation":"Engineer",
            "city":"New York",
            "email":"chandana.g@example.com"
            }'
)
INSERT INTO DOCUMENTS(id, document)
VALUES (DEFAULT,
            '{ "name":"Nayu Galgali",
            "age":13,
            "gender":"Female",
            "relationship":"Sister",
            "occupation":"Student",
            "city":"Mumbai",
            "email":"nayu.g@example.com"
            }'
)
INSERT INTO DOCUMENTS(id, document)
VALUES (DEFAULT,
            '{ "name":"Daxayani Galgali",
            "age":51,
            "gender":"Female",
            "relationship":"Mother",
            "occupation":"Perfumer",
            "city":"New Jersey",
            "email":"daxa.g@example.com"
            }'
)
INSERT INTO DOCUMENTS(id, document)
VALUES (DEFAULT,
            '{ "name":"Ramesh Galgali",
            "age":52,
            "gender":"Male",
            "relationship":"Father",
            "occupation":"Engineer",
            "city":"Mumbai",
            "email":"ram.g@example.com"
            }'
)
INSERT INTO DOCUMENTS(id, document)
VALUES (DEFAULT,
            '{ "name":"Shankar Mahalmani",
            "age":78,
            "gender":"Male",
            "relationship":"Grand-father",
            "occupation":"Retired",
            "city":"Orlando",
            "email":"sss.m@example.com"
            }'
)
INSERT INTO DOCUMENTS(id, document)
VALUES (DEFAULT,
            '{ "name":"Lata Mahalmani",
            "age":74,
            "gender":"Female",
            "relationship":"Grand-mother",
            "occupation":"Retired",
            "city":"Orlando",
            "email":"lata.m@example.com"
            }'
)
SELECT jsonb(document) FROM DOCUMENTS;
SELECT document->'name', document->'age' AS NAME_AGE FROM DOCUMENTS;
SELECT document->'name' AS NAME, document->'occupation' AS OCCUPATION FROM
DOCUMENTS WHERE document->>'relationship' = 'Mother' or
document->>'relationship' = 'Father';
SELECT COUNT(*) AS RETIRED_MEMBERS FROM DOCUMENTS WHERE
document->>'occupation' = 'Retired';
SELECT document->>'name' AS NAME, document->>'age' AS AGE FROM DOCUMENTS
WHERE document->>'gender' = 'Female' AND (document->>'age')::INT>35 ORDER
BY DOCUMENT->>'age';
```


exp6 on postgres@PostgreSQL 9.6

```sql
CREATE TABLE REPOSITORY (DOCUMENT JSON);
```

Data Output | Explain | Messages | History

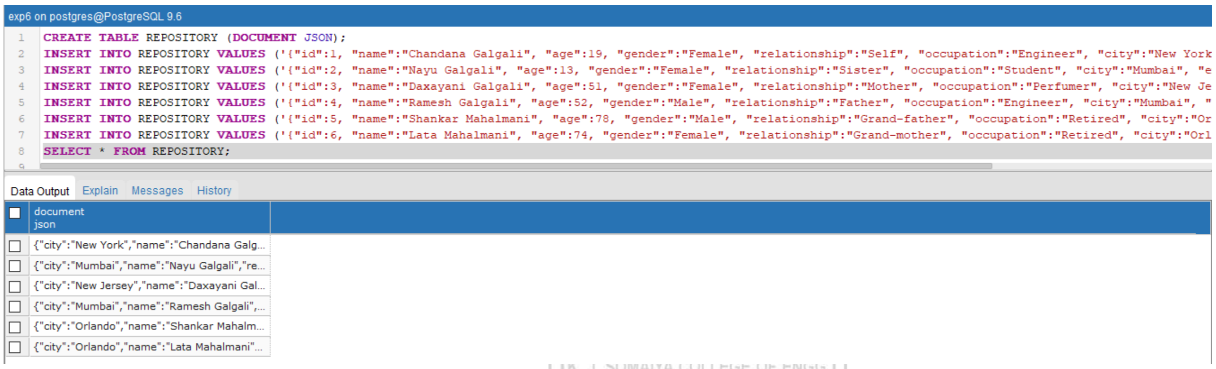
```
CREATE TABLE

Query returned successfully in 674 msec.
```


exp6 on postgres@PostgreSQL 9.6

```sql
CREATE TABLE REPOSITORY (DOCUMENT JSON);
INSERT INTO REPOSITORY VALUES ('{"id":1, "name":"Chandana Galgali", "age":19, "gender":"Female", "relationship":"Self", "occupation":"Engineer", "city":"New York", "email":"chandana.g@e
INSERT INTO REPOSITORY VALUES ('{"id":2, "name":"Nayu Galgali", "age":13, "gender":"Female", "relationship":"Sister", "occupation":"Student", "city":"Mumbai", "email":"nayu.g@example.co
INSERT INTO REPOSITORY VALUES ('{"id":3, "name":"Daxayani Galgali", "age":51, "gender":"Female", "relationship":"Mother", "occupation":"Perfumer", "city":"New Jersey", "email":"daxa.g@e
INSERT INTO REPOSITORY VALUES ('{"id":4, "name":"Ramesh Galgali", "age":52, "gender":"Male", "relationship":"Father", "occupation":"Engineer", "city":"Mumbai", "email":"ram.g@example.co
INSERT INTO REPOSITORY VALUES ('{"id":5, "name":"Shankar Mahalmani", "age":78, "gender":"Male", "relationship":"Grand-father", "occupation":"Retired", "city":"Orlando", "email":"sss.m@e
INSERT INTO REPOSITORY VALUES ('{"id":6, "name":"Lata Mahalmani", "age":74, "gender":"Female", "relationship":"Grand-mother", "occupation":"Retired", "city":"Orlando", "email":"lata.m@e
```

Data Output | Explain | Messages | History

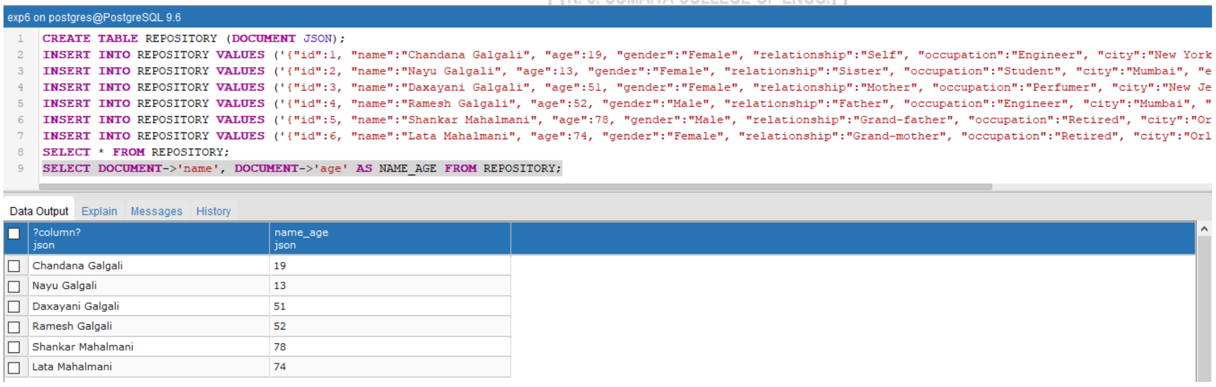
```
INSERT 0 1

Query returned successfully in 614 msec.
```


exp6 on postgres@PostgreSQL 9.6

```sql
CREATE TABLE REPOSITORY (DOCUMENT JSON);
INSERT INTO REPOSITORY VALUES ('{"id":1, "name":"Chandana Galgali", "age":19, "gender":"Female", "relationship":"Self", "occupation":"Engineer", "city":"New York
INSERT INTO REPOSITORY VALUES ('{"id":2, "name":"Nayu Galgali", "age":13, "gender":"Female", "relationship":"Sister", "occupation":"Student", "city":"Mumbai", "e
INSERT INTO REPOSITORY VALUES ('{"id":3, "name":"Daxayani Galgali", "age":51, "gender":"Female", "relationship":"Mother", "occupation":"Perfumer", "city":"New Je
INSERT INTO REPOSITORY VALUES ('{"id":4, "name":"Ramesh Galgali", "age":52, "gender":"Male", "relationship":"Father", "occupation":"Engineer", "city":"Mumbai", "
INSERT INTO REPOSITORY VALUES ('{"id":5, "name":"Shankar Mahalmani", "age":78, "gender":"Male", "relationship":"Grand-father", "occupation":"Retired", "city":"Or
INSERT INTO REPOSITORY VALUES ('{"id":6, "name":"Lata Mahalmani", "age":74, "gender":"Female", "relationship":"Grand-mother", "occupation":"Retired", "city":"Orl
SELECT * FROM REPOSITORY;
```

Data Output | Explain | Messages | History

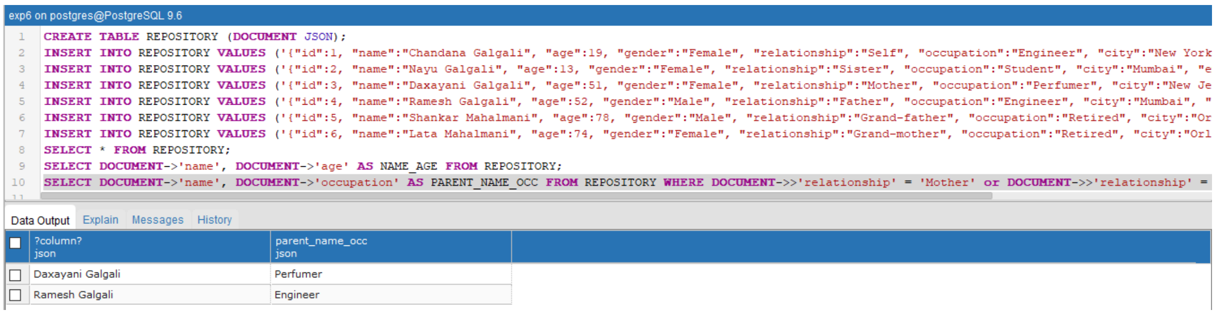
document json
{"city":"New York","name":"Chandana Galg...
{"city":"Mumbai","name":"Nayu Galgali","re...
{"city":"New Jersey","name":"Daxayani Gal...
{"city":"Mumbai","name":"Ramesh Galgali",...
{"city":"Orlando","name":"Shankar Mahalm...
{"city":"Orlando","name":"Lata Mahalmani"...


exp6 on postgres@PostgreSQL 9.6

```sql
CREATE TABLE REPOSITORY (DOCUMENT JSON);
INSERT INTO REPOSITORY VALUES ('{"id":1, "name":"Chandana Galgali", "age":19, "gender":"Female", "relationship":"Self", "occupation":"Engineer", "city":"New York
INSERT INTO REPOSITORY VALUES ('{"id":2, "name":"Nayu Galgali", "age":13, "gender":"Female", "relationship":"Sister", "occupation":"Student", "city":"Mumbai", "e
INSERT INTO REPOSITORY VALUES ('{"id":3, "name":"Daxayani Galgali", "age":51, "gender":"Female", "relationship":"Mother", "occupation":"Perfumer", "city":"New Je
INSERT INTO REPOSITORY VALUES ('{"id":4, "name":"Ramesh Galgali", "age":52, "gender":"Male", "relationship":"Father", "occupation":"Engineer", "city":"Mumbai", "
INSERT INTO REPOSITORY VALUES ('{"id":5, "name":"Shankar Mahalmani", "age":78, "gender":"Male", "relationship":"Grand-father", "occupation":"Retired", "city":"Or
INSERT INTO REPOSITORY VALUES ('{"id":6, "name":"Lata Mahalmani", "age":74, "gender":"Female", "relationship":"Grand-mother", "occupation":"Retired", "city":"Orl
SELECT * FROM REPOSITORY;
SELECT DOCUMENT->'name', DOCUMENT->'age' AS NAME_AGE FROM REPOSITORY;
```

Data Output | Explain | Messages | History

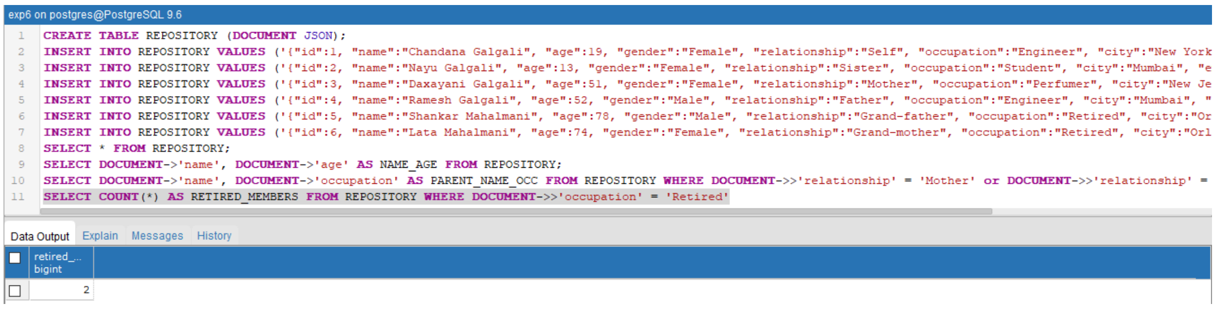
?column? json	name_age json
Chandana Galgali	19
Nayu Galgali	13
Daxayani Galgali	51
Ramesh Galgali	52
Shankar Mahalmani	78
Lata Mahalmani	74


exp6 on postgres@PostgreSQL 9.6

```sql
CREATE TABLE REPOSITORY (DOCUMENT JSON);
INSERT INTO REPOSITORY VALUES ('{"id":1, "name":"Chandana Galgali", "age":19, "gender":"Female", "relationship":"Self", "occupation":"Engineer", "city":"New York
INSERT INTO REPOSITORY VALUES ('{"id":2, "name":"Nayu Galgali", "age":13, "gender":"Female", "relationship":"Sister", "occupation":"Student", "city":"Mumbai", "e
INSERT INTO REPOSITORY VALUES ('{"id":3, "name":"Daxayani Galgali", "age":51, "gender":"Female", "relationship":"Mother", "occupation":"Perfumer", "city":"New Je
INSERT INTO REPOSITORY VALUES ('{"id":4, "name":"Ramesh Galgali", "age":52, "gender":"Male", "relationship":"Father", "occupation":"Engineer", "city":"Mumbai", "
INSERT INTO REPOSITORY VALUES ('{"id":5, "name":"Shankar Mahalmani", "age":78, "gender":"Male", "relationship":"Grand-father", "occupation":"Retired", "city":"Or
INSERT INTO REPOSITORY VALUES ('{"id":6, "name":"Lata Mahalmani", "age":74, "gender":"Female", "relationship":"Grand-mother", "occupation":"Retired", "city":"Orl
SELECT * FROM REPOSITORY;
SELECT DOCUMENT->'name', DOCUMENT->'age' AS NAME_AGE FROM REPOSITORY;
SELECT DOCUMENT->'name', DOCUMENT->'occupation' AS PARENT_NAME_OCC FROM REPOSITORY WHERE DOCUMENT->>'relationship' = 'Mother' or DOCUMENT->>'relationship' =
```

Data Output | Explain | Messages | History

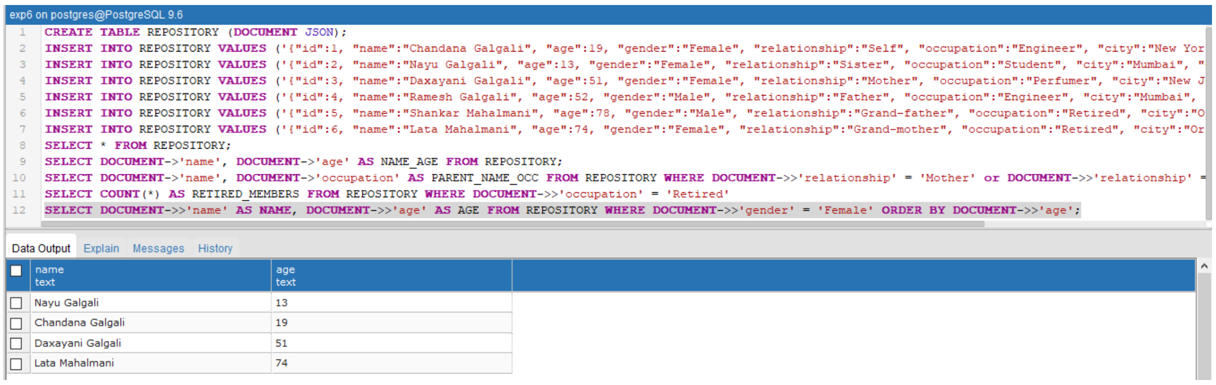
?column? json	parent_name_occ json
Daxayani Galgali	Perfumer
Ramesh Galgali	Engineer


exp6 on postgres@PostgreSQL 9.6

```sql
CREATE TABLE REPOSITORY (DOCUMENT JSON);
INSERT INTO REPOSITORY VALUES ('{"id":1, "name":"Chandana Galgali", "age":19, "gender":"Female", "relationship":"Self", "occupation":"Engineer", "city":"New York
INSERT INTO REPOSITORY VALUES ('{"id":2, "name":"Nayu Galgali", "age":13, "gender":"Female", "relationship":"Sister", "occupation":"Student", "city":"Mumbai", "e
INSERT INTO REPOSITORY VALUES ('{"id":3, "name":"Daxayani Galgali", "age":51, "gender":"Female", "relationship":"Mother", "occupation":"Perfumer", "city":"New Je
INSERT INTO REPOSITORY VALUES ('{"id":4, "name":"Ramesh Galgali", "age":52, "gender":"Male", "relationship":"Father", "occupation":"Engineer", "city":"Mumbai", "
INSERT INTO REPOSITORY VALUES ('{"id":5, "name":"Shankar Mahalmani", "age":78, "gender":"Male", "relationship":"Grand-father", "occupation":"Retired", "city":"Or
INSERT INTO REPOSITORY VALUES ('{"id":6, "name":"Lata Mahalmani", "age":74, "gender":"Female", "relationship":"Grand-mother", "occupation":"Retired", "city":"Orl
SELECT * FROM REPOSITORY;
SELECT DOCUMENT->'name', DOCUMENT->'age' AS NAME_AGE FROM REPOSITORY;
SELECT DOCUMENT->'name', DOCUMENT->'occupation' AS PARENT_NAME_OCC FROM REPOSITORY WHERE DOCUMENT->>'relationship' = 'Mother' or DOCUMENT->>'relationship' =
SELECT COUNT(*) AS RETIRED_MEMBERS FROM REPOSITORY WHERE DOCUMENT->>'occupation' = 'Retired'
```

Data Output | Explain | Messages | History

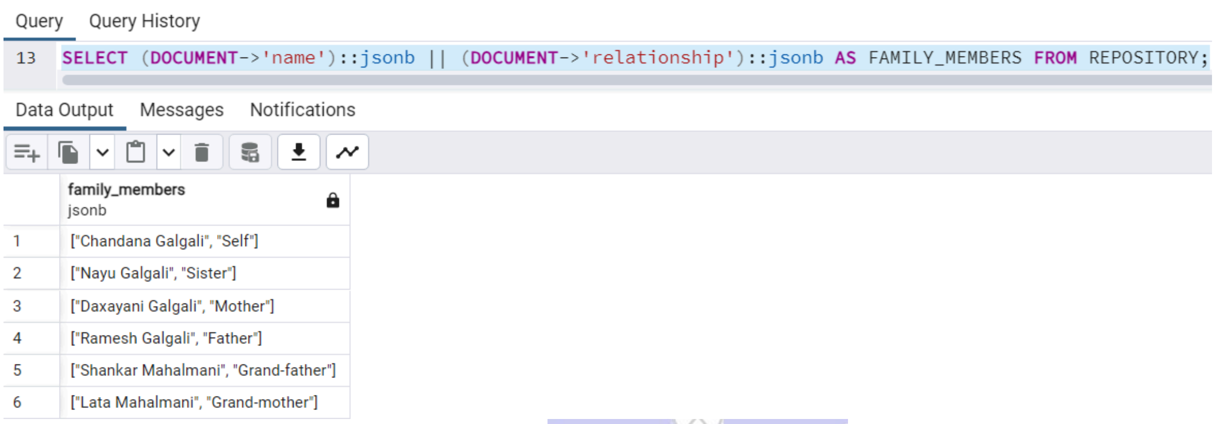
retired_... bigint
2



exp6 on postgres@PostgreSQL 9.6

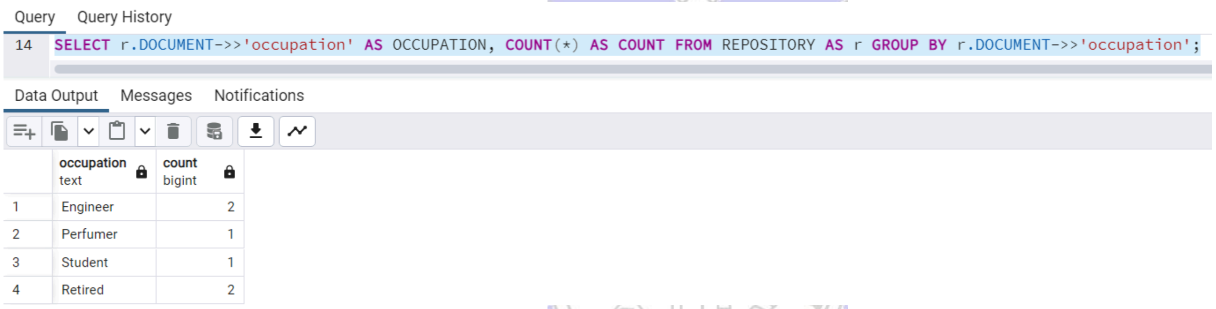
```sql
CREATE TABLE REPOSITORY (DOCUMENT JSON);
INSERT INTO REPOSITORY VALUES ('{"id":1, "name":"Chandana Galgali", "age":19, "gender":"Female", "relationship":"Self", "occupation":"Engineer", "city":"New Yor
INSERT INTO REPOSITORY VALUES ('{"id":2, "name":"Nayu Galgali", "age":13, "gender":"Female", "relationship":"Sister", "occupation":"Student", "city":"Mumbai", "
INSERT INTO REPOSITORY VALUES ('{"id":3, "name":"Daxayani Galgali", "age":51, "gender":"Female", "relationship":"Mother", "occupation":"Perfumer", "city":"New J
INSERT INTO REPOSITORY VALUES ('{"id":4, "name":"Ramesh Galgali", "age":52, "gender":"Male", "relationship":"Father", "occupation":"Engineer", "city":"Mumbai",
INSERT INTO REPOSITORY VALUES ('{"id":5, "name":"Shankar Mahalmani", "age":78, "gender":"Male", "relationship":"Grand-father", "occupation":"Retired", "city":"O
INSERT INTO REPOSITORY VALUES ('{"id":6, "name":"Lata Mahalmani", "age":74, "gender":"Female", "relationship":"Grand-mother", "occupation":"Retired", "city":"Or
SELECT * FROM REPOSITORY;
SELECT DOCUMENT->'name', DOCUMENT->'age' AS NAME_AGE FROM REPOSITORY;
SELECT DOCUMENT->'name', DOCUMENT->'occupation' AS PARENT_NAME_OCC FROM REPOSITORY WHERE DOCUMENT->>'relationship' = 'Mother' or DOCUMENT->>'relationship' =
SELECT COUNT(*) AS RETIRED_MEMBERS FROM REPOSITORY WHERE DOCUMENT->>'occupation' = 'Retired'
SELECT DOCUMENT->>'name' AS NAME, DOCUMENT->>'age' AS AGE FROM REPOSITORY WHERE DOCUMENT->>'gender' = 'Female' ORDER BY DOCUMENT->>'age';
```

name text	age text
Nayu Galgali	13
Chandana Galgali	19
Daxayani Galgali	51
Lata Mahalmani	74



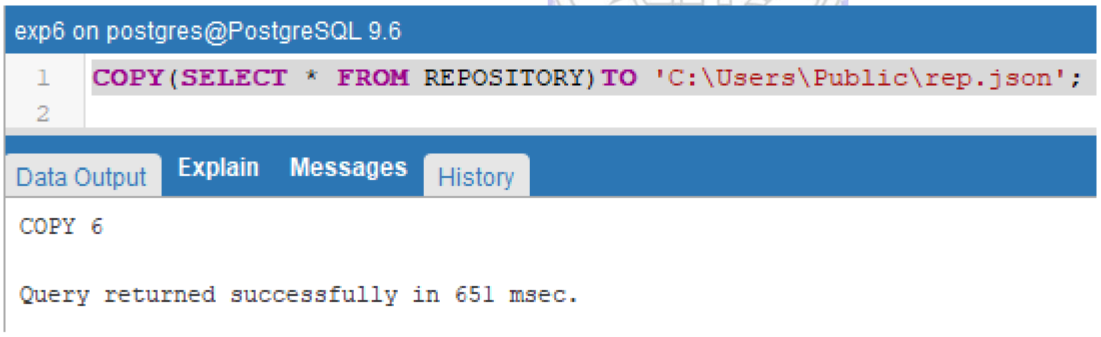
```sql
SELECT (DOCUMENT->'name')::jsonb || (DOCUMENT->'relationship')::jsonb AS FAMILY_MEMBERS FROM REPOSITORY;
```

	family_members jsonb
1	["Chandana Galgali", "Self"]
2	["Nayu Galgali", "Sister"]
3	["Daxayani Galgali", "Mother"]
4	["Ramesh Galgali", "Father"]
5	["Shankar Mahalmani", "Grand-father"]
6	["Lata Mahalmani", "Grand-mother"]



```sql
SELECT r.DOCUMENT->>'occupation' AS OCCUPATION, COUNT(*) AS COUNT FROM REPOSITORY AS r GROUP BY r.DOCUMENT->>'occupation';
```

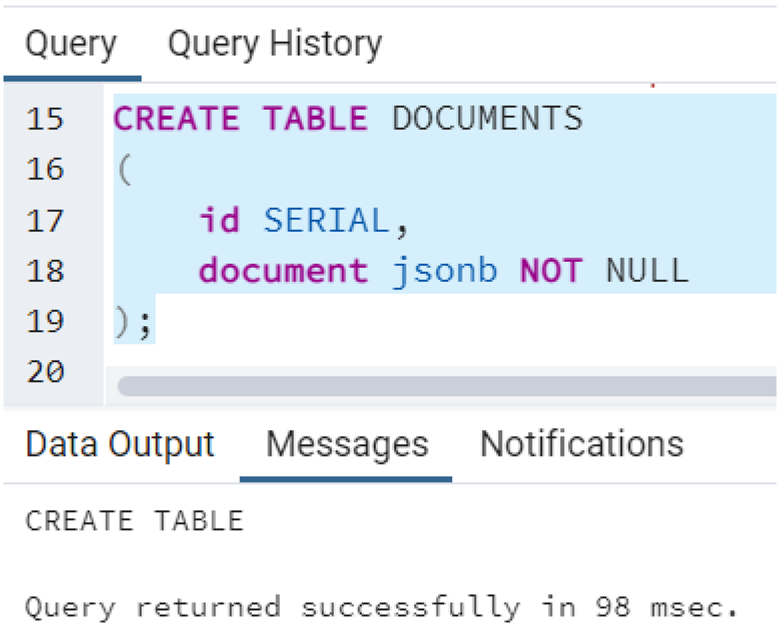
	occupation text	count bigint
1	Engineer	2
2	Perfumer	1
3	Student	1
4	Retired	2



exp6 on postgres@PostgreSQL 9.6

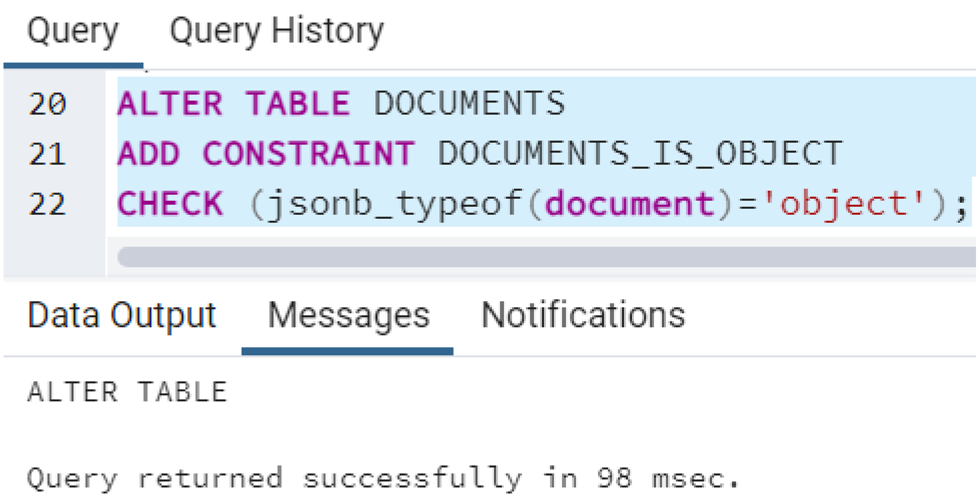
```sql
COPY(SELECT * FROM REPOSITORY)TO 'C:\Users\Public\rep.json';
```

```
COPY 6

Query returned successfully in 651 msec.
```



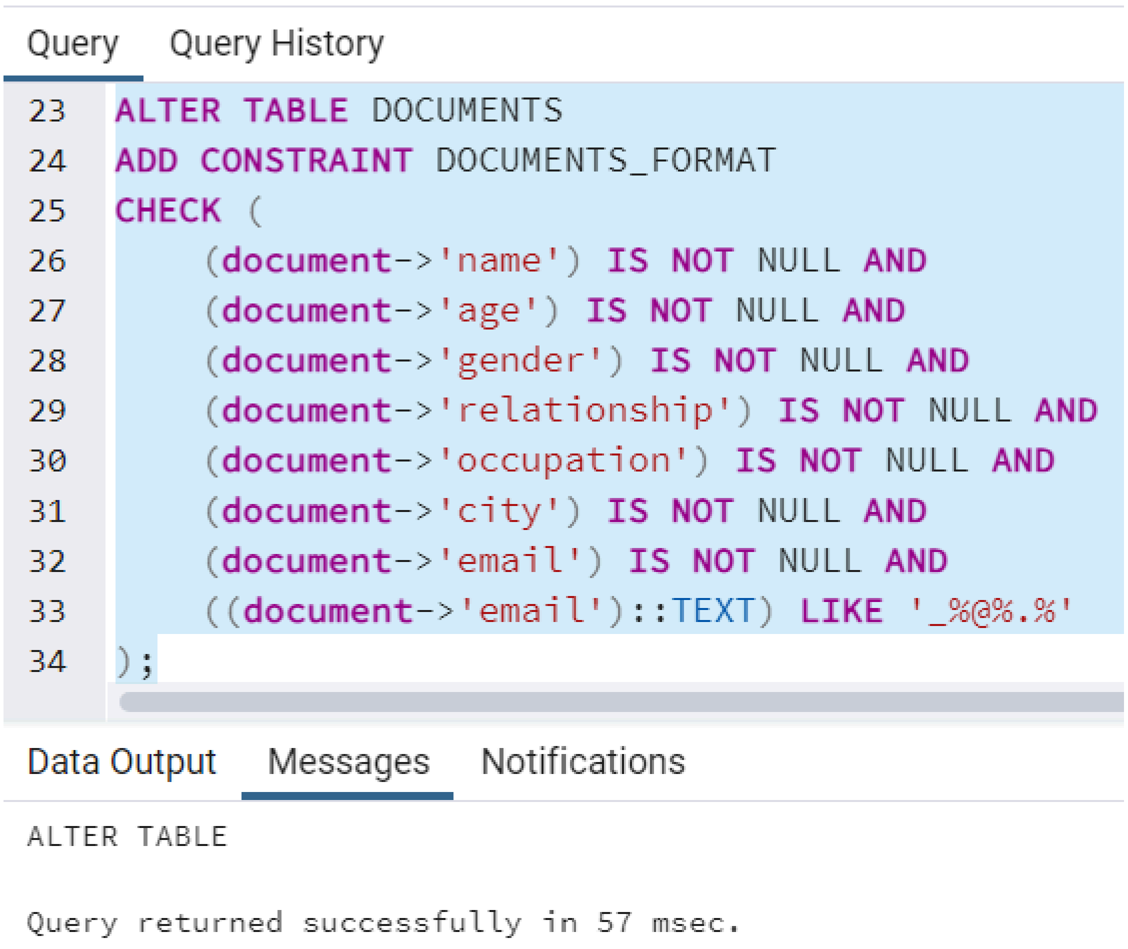
```sql
CREATE TABLE DOCUMENTS
(
    id SERIAL,
    document jsonb NOT NULL
);
```

```
CREATE TABLE

Query returned successfully in 98 msec.
```



```sql
ALTER TABLE DOCUMENTS
ADD CONSTRAINT DOCUMENTS_IS_OBJECT
CHECK (jsonb_typeof(document)='object');
```

```
ALTER TABLE

Query returned successfully in 98 msec.
```

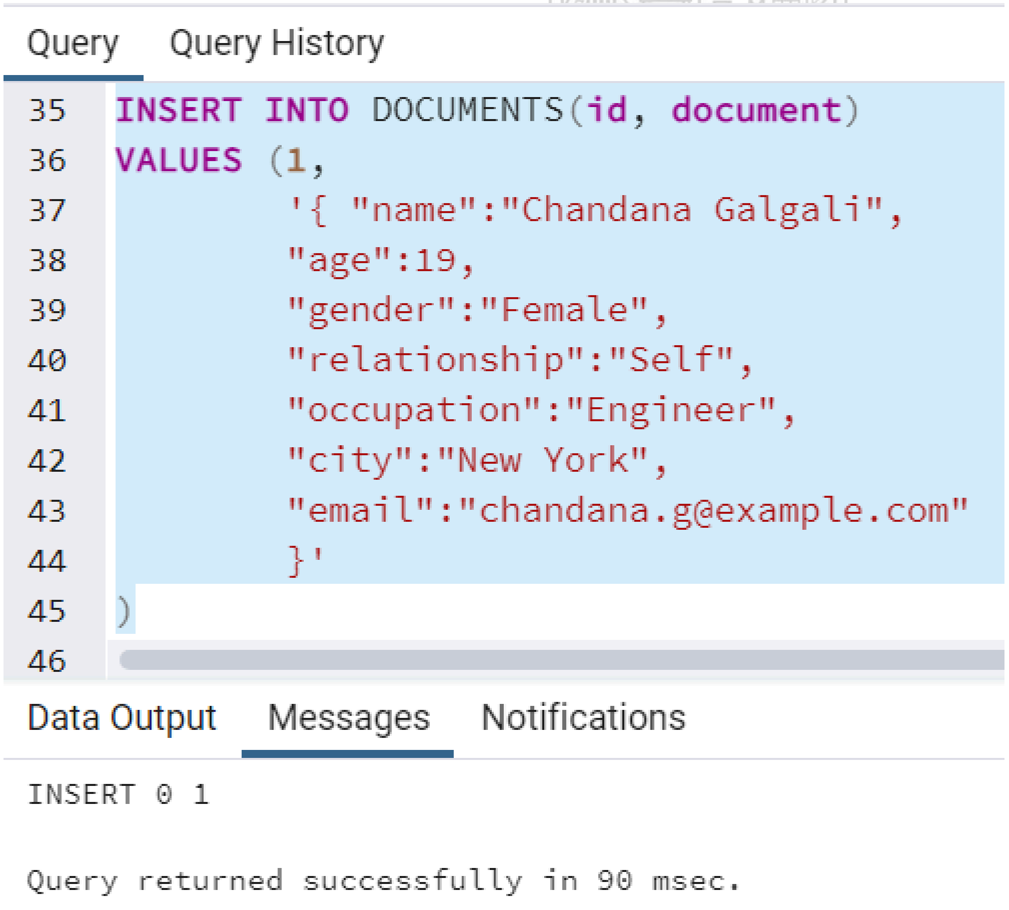


Query   Query History

```
ALTER TABLE DOCUMENTS
ADD CONSTRAINT DOCUMENTS_FORMAT
CHECK (
      (document->'name') IS NOT NULL AND
      (document->'age') IS NOT NULL AND
      (document->'gender') IS NOT NULL AND
      (document->'relationship') IS NOT NULL AND
      (document->'occupation') IS NOT NULL AND
      (document->'city') IS NOT NULL AND
      (document->'email') IS NOT NULL AND
      ((document->'email')::TEXT) LIKE '_%@%.%'
);
```

Data Output   Messages   Notifications

```
ALTER TABLE

Query returned successfully in 57 msec.
```

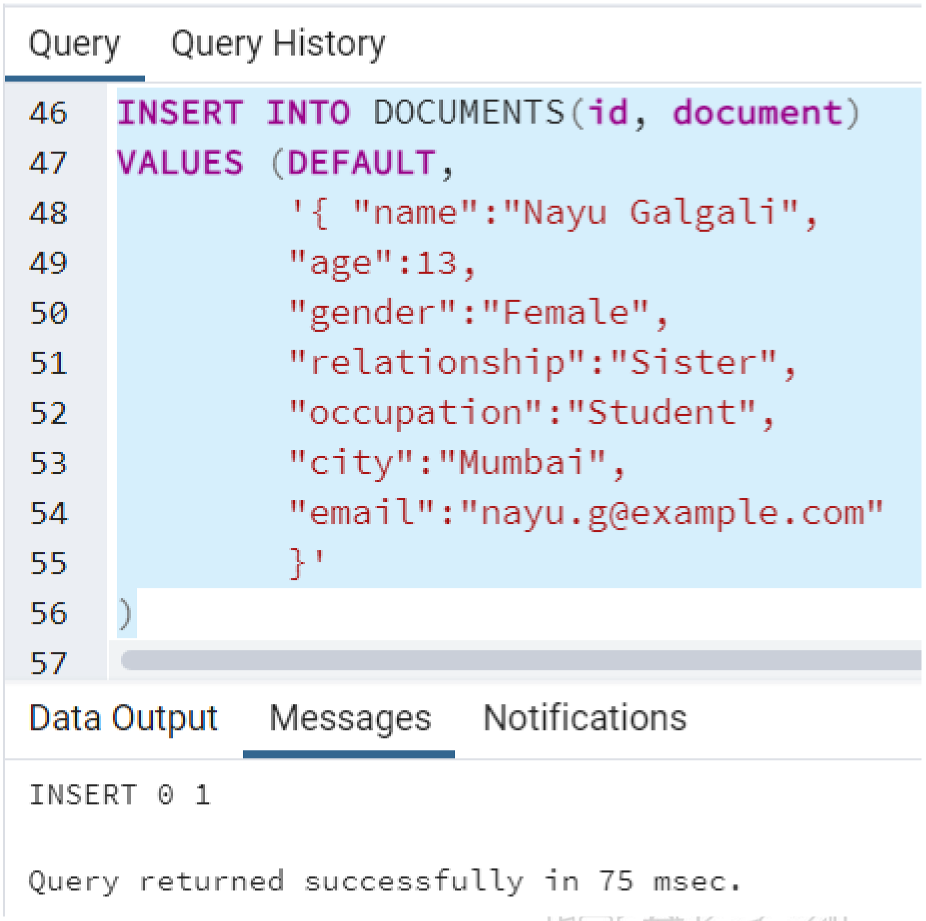


Query   Query History

```
INSERT INTO DOCUMENTS(id, document)
VALUES (1,
        '{ "name":"Chandana Galgali",
        "age":19,
        "gender":"Female",
        "relationship":"Self",
        "occupation":"Engineer",
        "city":"New York",
        "email":"chandana.g@example.com"
        }'
)

```

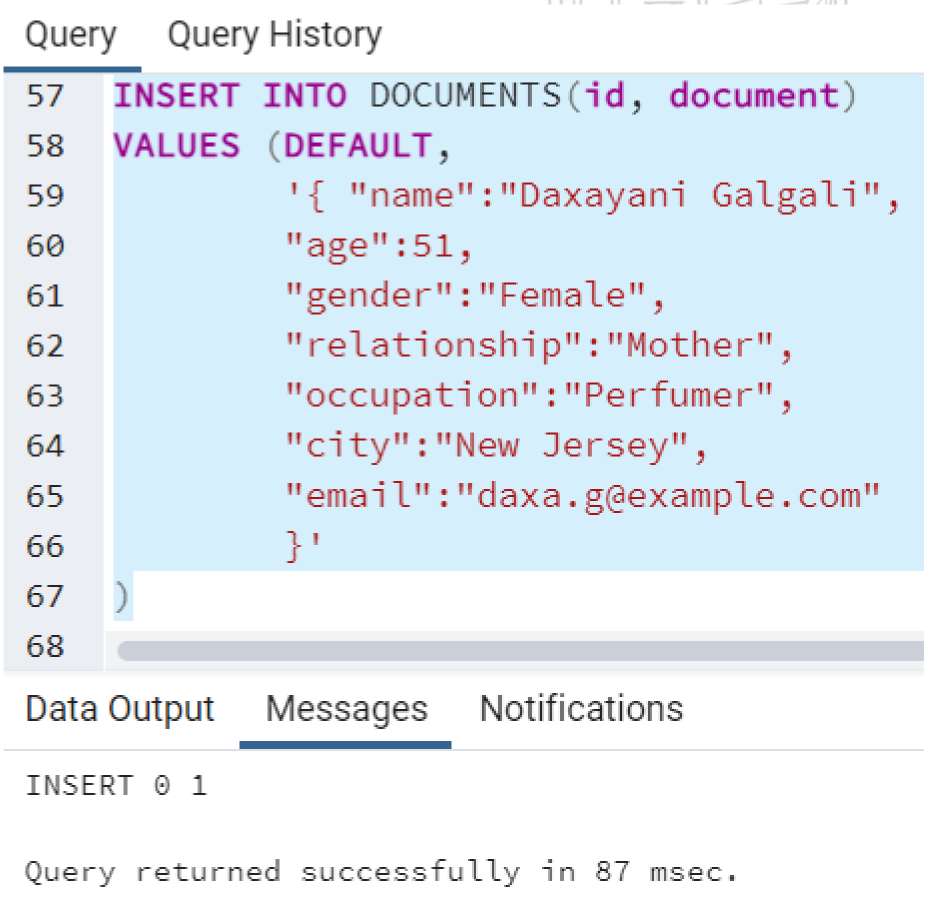
Data Output   Messages   Notifications

```
INSERT 0 1

Query returned successfully in 90 msec.
```



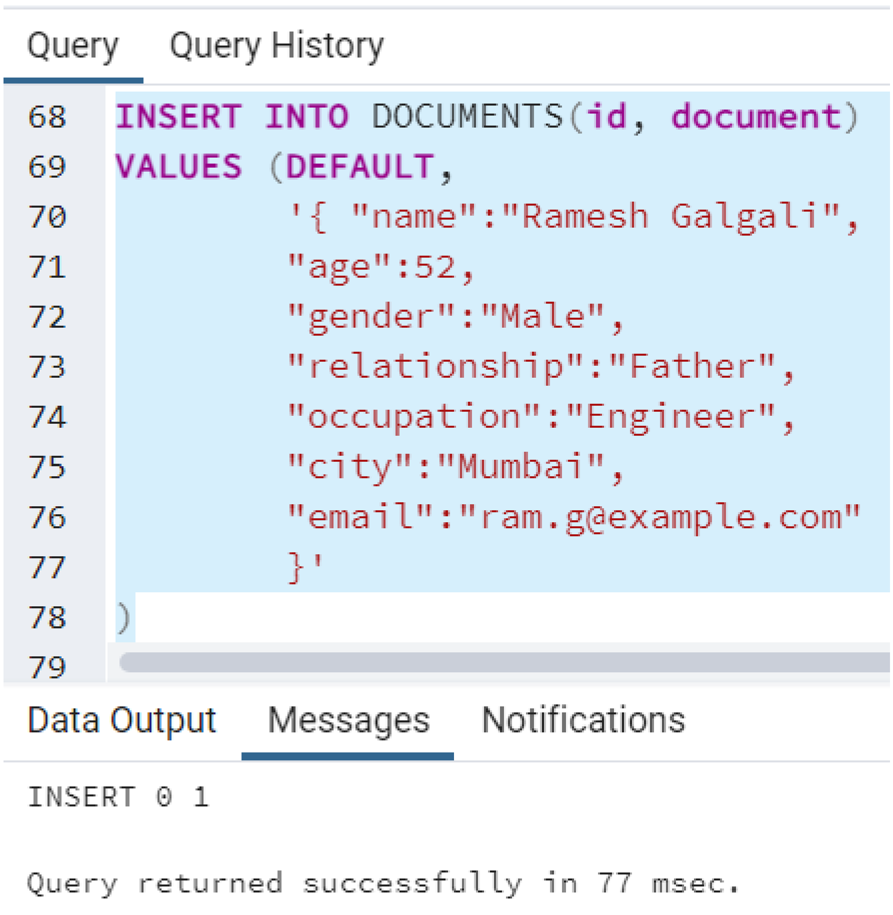
```sql
INSERT INTO DOCUMENTS(id, document)
VALUES (DEFAULT,
        '{ "name":"Nayu Galgali",
        "age":13,
        "gender":"Female",
        "relationship":"Sister",
        "occupation":"Student",
        "city":"Mumbai",
        "email":"nayu.g@example.com"
        }'
)
```

```
INSERT 0 1

Query returned successfully in 75 msec.
```



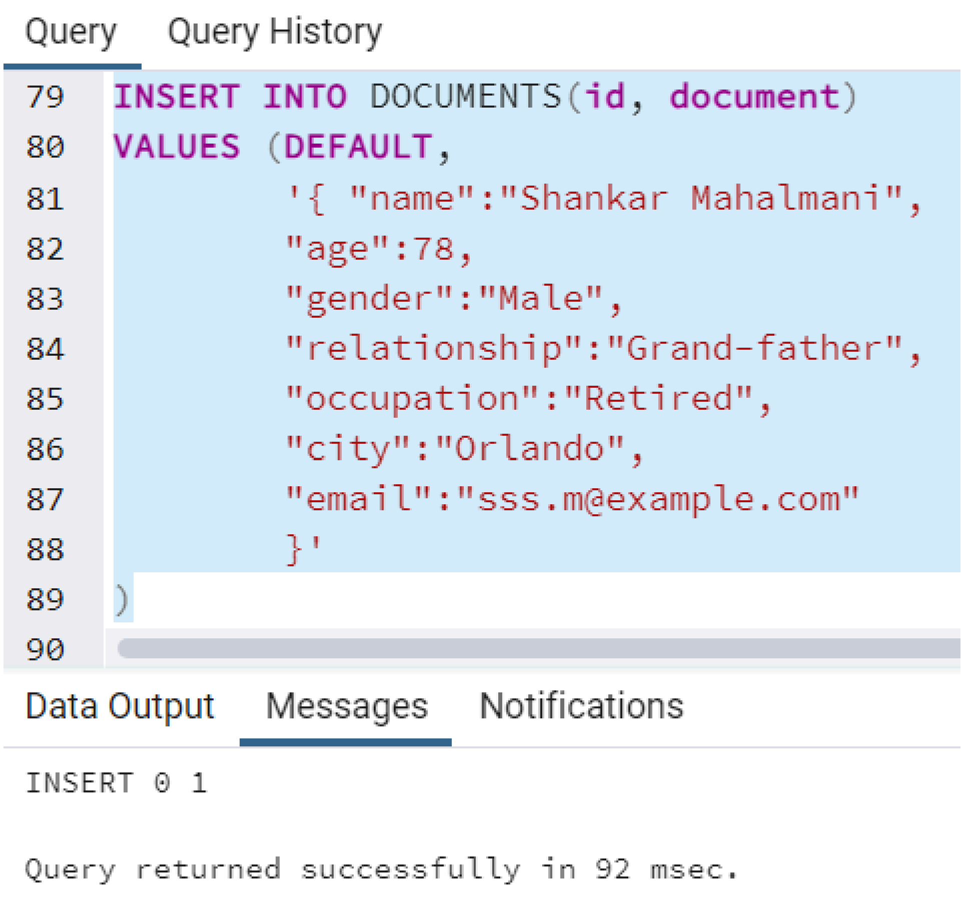
```sql
INSERT INTO DOCUMENTS(id, document)
VALUES (DEFAULT,
        '{ "name":"Daxayani Galgali",
        "age":51,
        "gender":"Female",
        "relationship":"Mother",
        "occupation":"Perfumer",
        "city":"New Jersey",
        "email":"daxa.g@example.com"
        }'
)
```

```
INSERT 0 1

Query returned successfully in 87 msec.
```



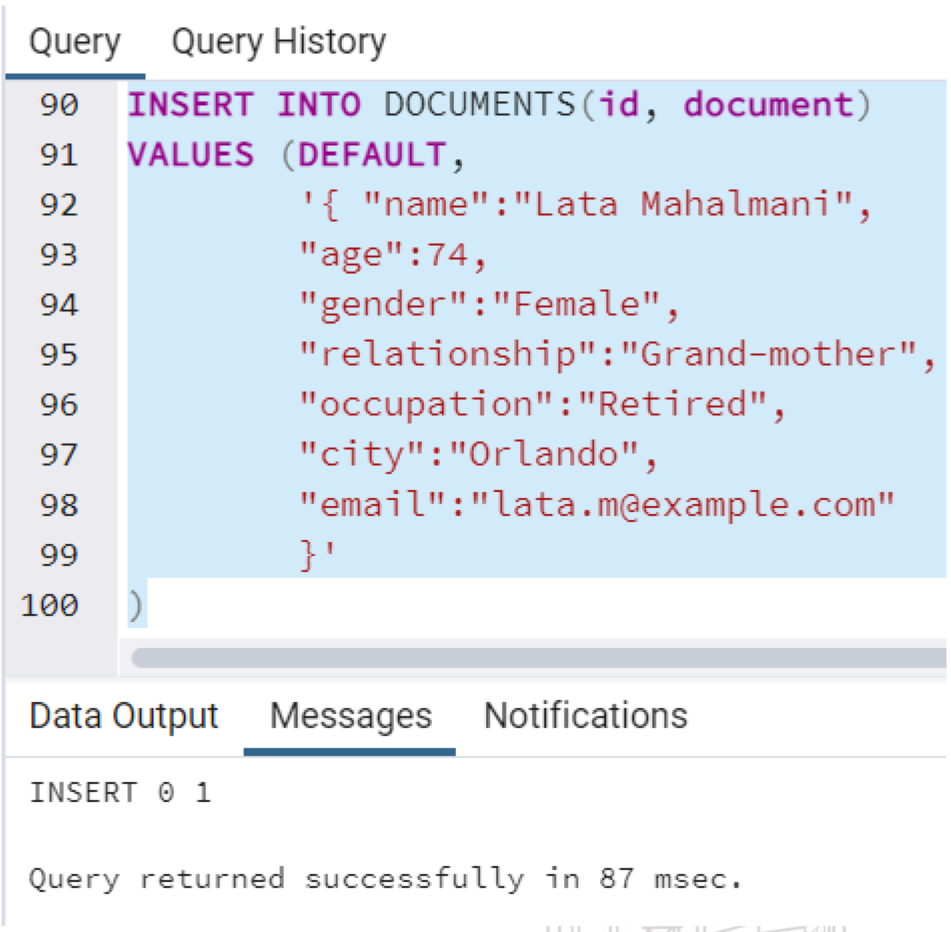
```
INSERT INTO DOCUMENTS(id, document)
VALUES (DEFAULT,
        '{ "name":"Ramesh Galgali",
        "age":52,
        "gender":"Male",
        "relationship":"Father",
        "occupation":"Engineer",
        "city":"Mumbai",
        "email":"ram.g@example.com"
        }'
)
```

```
INSERT 0 1

Query returned successfully in 77 msec.
```



```
INSERT INTO DOCUMENTS(id, document)
VALUES (DEFAULT,
        '{ "name":"Shankar Mahalmani",
        "age":78,
        "gender":"Male",
        "relationship":"Grand-father",
        "occupation":"Retired",
        "city":"Orlando",
        "email":"sss.m@example.com"
        }'
)
```

```
INSERT 0 1

Query returned successfully in 92 msec.
```

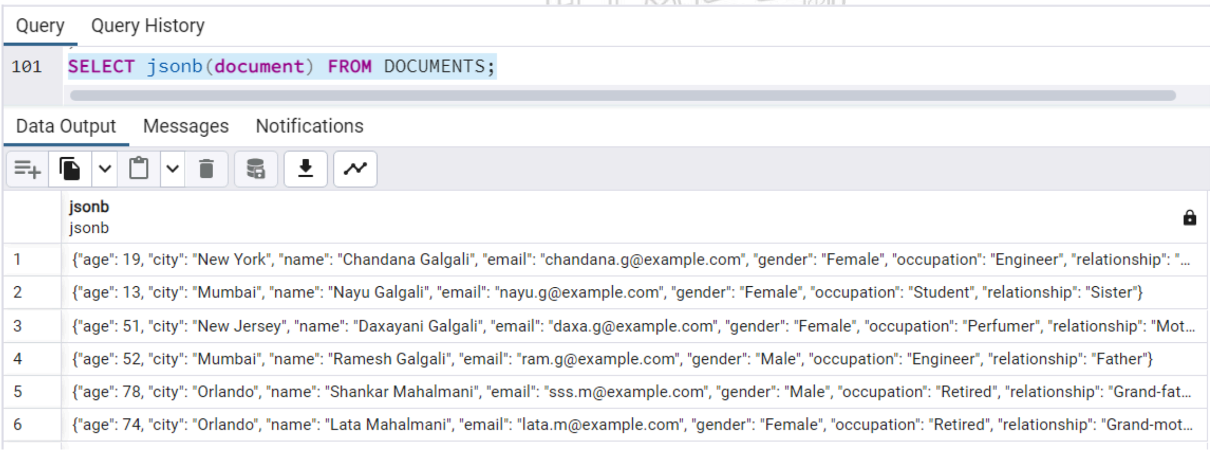


Query Query History

```sql
INSERT INTO DOCUMENTS(id, document)
VALUES (DEFAULT,
        '{ "name":"Lata Mahalmani",
        "age":74,
        "gender":"Female",
        "relationship":"Grand-mother",
        "occupation":"Retired",
        "city":"Orlando",
        "email":"lata.m@example.com"
        }'
)
```

Data Output Messages Notifications

```
INSERT 0 1

Query returned successfully in 87 msec.
```

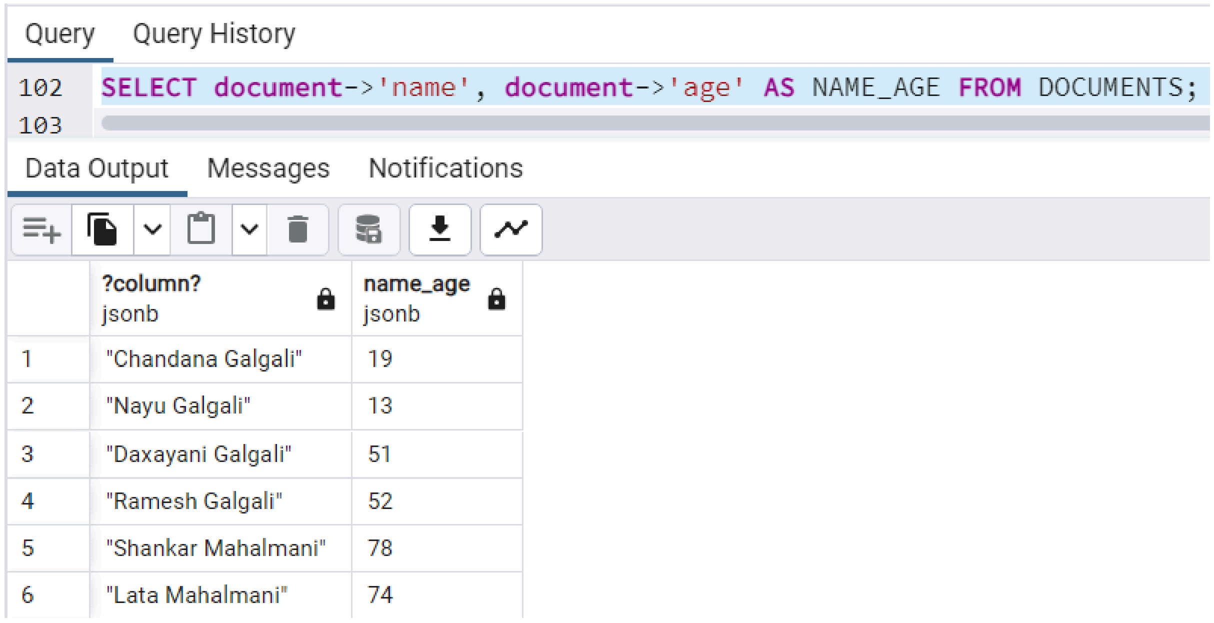


Query Query History

```sql
SELECT jsonb(document) FROM DOCUMENTS;
```

Data Output Messages Notifications

```
  jsonb
  jsonb
1 {"age": 19, "city": "New York", "name": "Chandana Galgali", "email": "chandana.g@example.com", "gender": "Female", "occupation": "Engineer", "relationship": "...
2 {"age": 13, "city": "Mumbai", "name": "Nayu Galgali", "email": "nayu.g@example.com", "gender": "Female", "occupation": "Student", "relationship": "Sister"}
3 {"age": 51, "city": "New Jersey", "name": "Daxayani Galgali", "email": "daxa.g@example.com", "gender": "Female", "occupation": "Perfumer", "relationship": "Mot...
4 {"age": 52, "city": "Mumbai", "name": "Ramesh Galgali", "email": "ram.g@example.com", "gender": "Male", "occupation": "Engineer", "relationship": "Father"}
5 {"age": 78, "city": "Orlando", "name": "Shankar Mahalmani", "email": "sss.m@example.com", "gender": "Male", "occupation": "Retired", "relationship": "Grand-fat...
6 {"age": 74, "city": "Orlando", "name": "Lata Mahalmani", "email": "lata.m@example.com", "gender": "Female", "occupation": "Retired", "relationship": "Grand-mot...
```

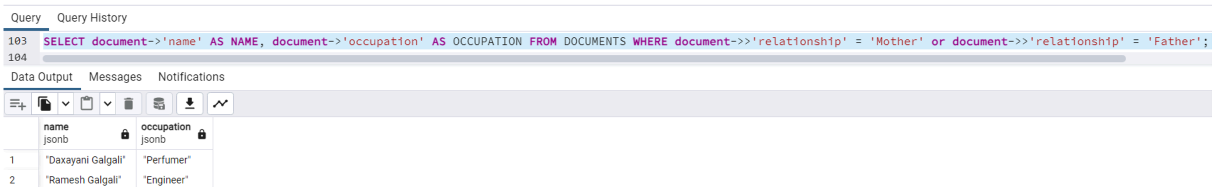


Query Query History

```sql
SELECT document->'name', document->'age' AS NAME_AGE FROM DOCUMENTS;
```

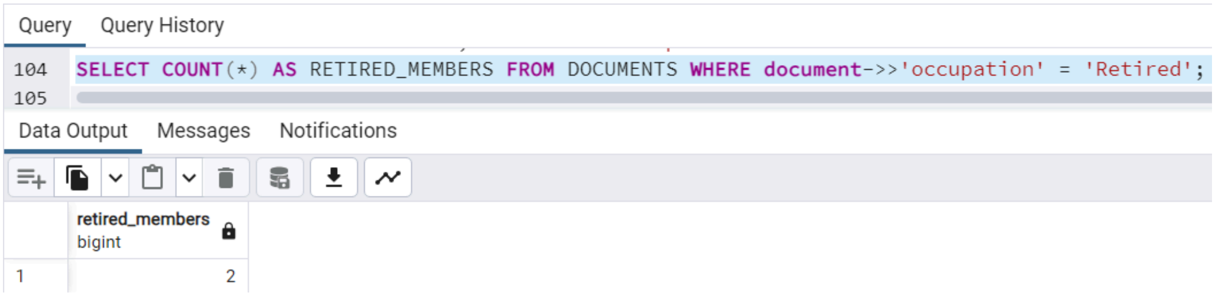
Data Output Messages Notifications

	?column? jsonb	name_age jsonb
1	"Chandana Galgali"	19
2	"Nayu Galgali"	13
3	"Daxayani Galgali"	51
4	"Ramesh Galgali"	52
5	"Shankar Mahalmani"	78
6	"Lata Mahalmani"	74



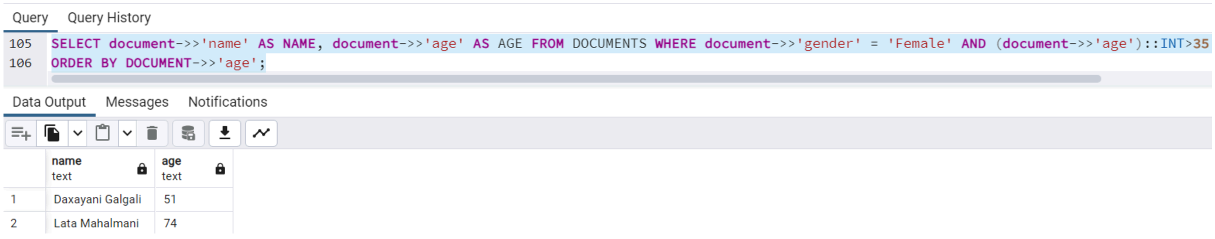
```sql
SELECT document->'name' AS NAME, document->'occupation' AS OCCUPATION FROM DOCUMENTS WHERE document->>'relationship' = 'Mother' or document->>'relationship' = 'Father';
```

	name jsonb	occupation jsonb
1	"Daxayani Galgali"	"Perfumer"
2	"Ramesh Galgali"	"Engineer"



```sql
SELECT COUNT(*) AS RETIRED_MEMBERS FROM DOCUMENTS WHERE document->>'occupation' = 'Retired';
```

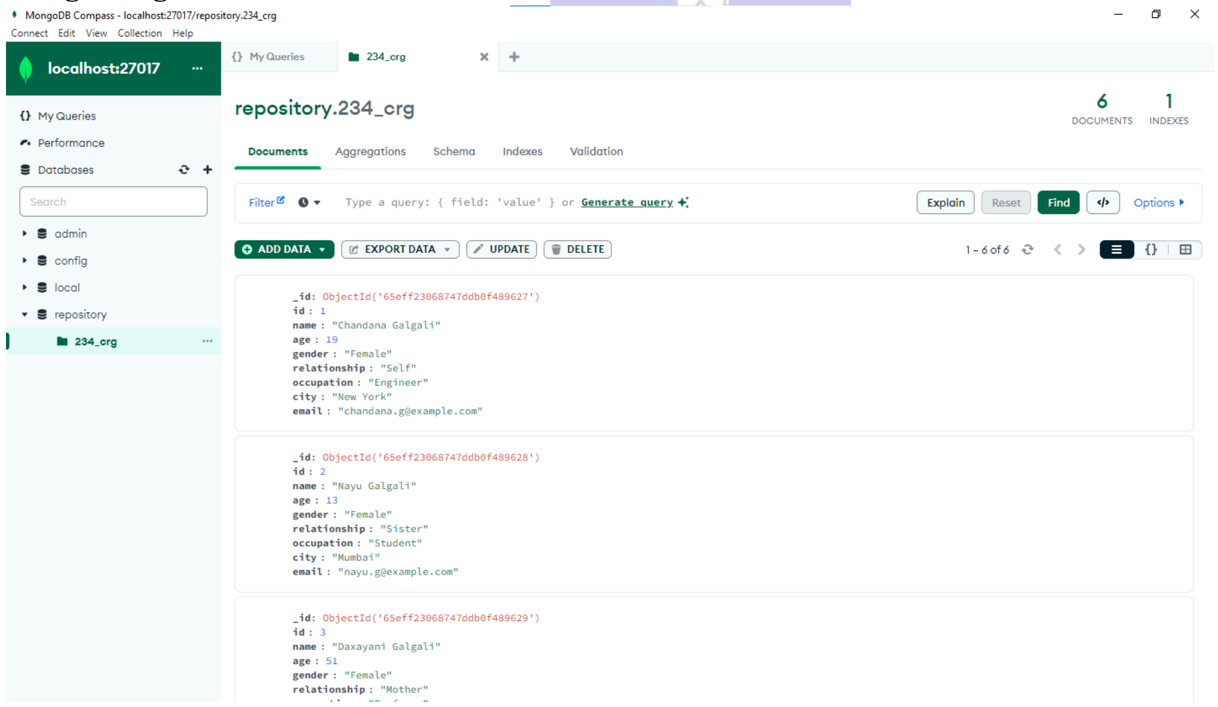
	retired_members bigint
1	2



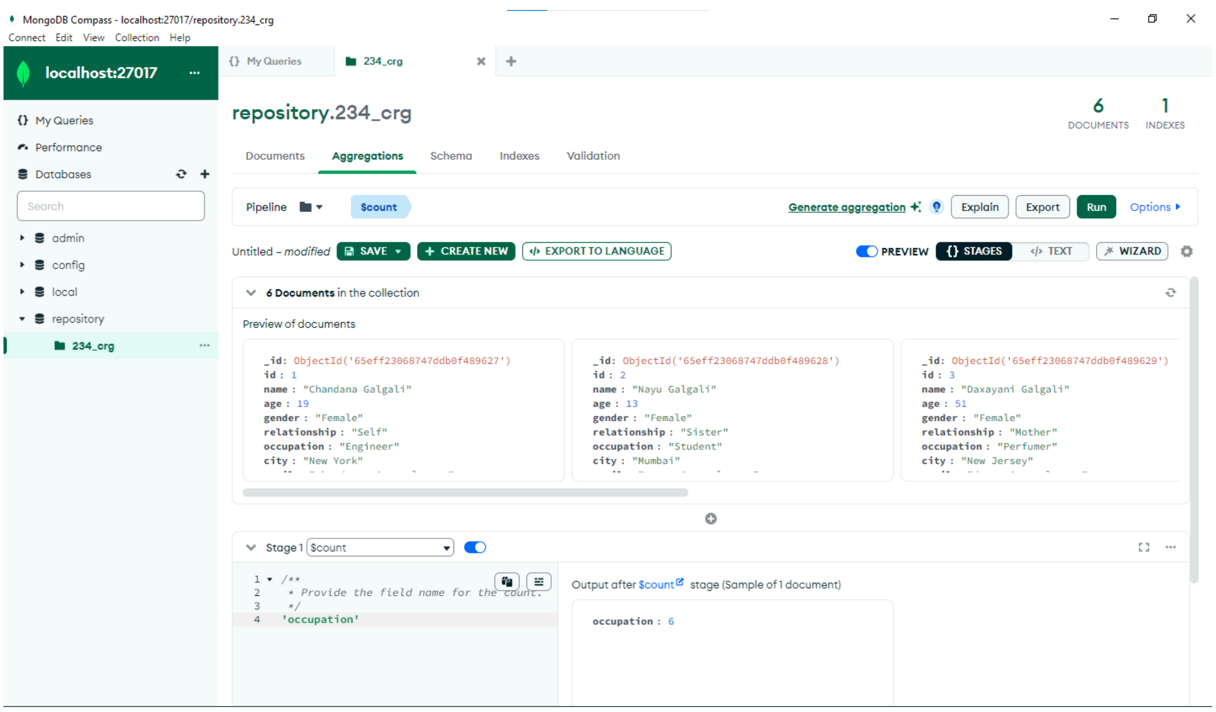
```sql
SELECT document->>'name' AS NAME, document->>'age' AS AGE FROM DOCUMENTS WHERE document->>'gender' = 'Female' AND (document->>'age')::INT>35
ORDER BY DOCUMENT->>'age';
```

	name text	age text
1	Daxayani Galgali	51
2	Lata Mahalmani	74

Using MongoDB:



```
_id: ObjectId('65eff23068747ddb0f489627')
id : 1
name : "Chandana Galgali"
age : 19
gender : "Female"
relationship : "Self"
occupation : "Engineer"
city : "New York"
email : "chandana.g@example.com"

_id: ObjectId('65eff23068747ddb0f489628')
id : 2
name : "Nayu Galgali"
age : 13
gender : "Female"
relationship : "Sister"
occupation : "Student"
city : "Mumbai"
email : "nayu.g@example.com"

_id: ObjectId('65eff23068747ddb0f489629')
id : 3
name : "Daxayani Galgali"
age : 51
gender : "Female"
relationship : "Mother"
```



repository.234_crg

6 Documents in the collection

Preview of documents

```
_id: ObjectId('65eff23068747ddb0f489627')
id : 1
name : "Chandana Galgali"
age : 19
gender : "Female"
relationship : "Self"
occupation : "Engineer"
city : "New York"
```

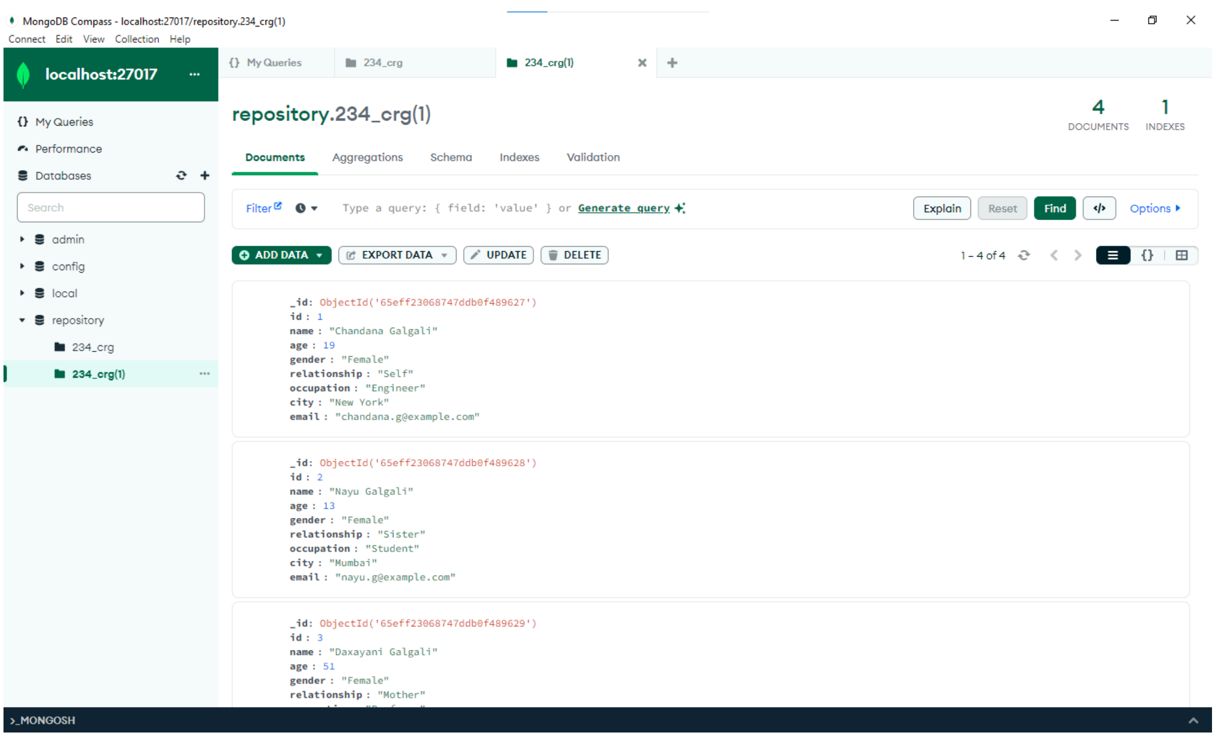
```
_id: ObjectId('65eff23068747ddb0f489628')
id : 2
name : "Nayu Galgali"
age : 13
gender : "Female"
relationship : "Sister"
occupation : "Student"
city : "Mumbai"
```

```
_id: ObjectId('65eff23068747ddb0f489629')
id : 3
name : "Daxayani Galgali"
age : 51
gender : "Female"
relationship : "Mother"
occupation : "Perfumer"
city : "New Jersey"
```

```
/**
 * Provide the field name for the count.
 */
'occupation'
```

Output after $count stage (Sample of 1 document)

```
occupation : 6
```

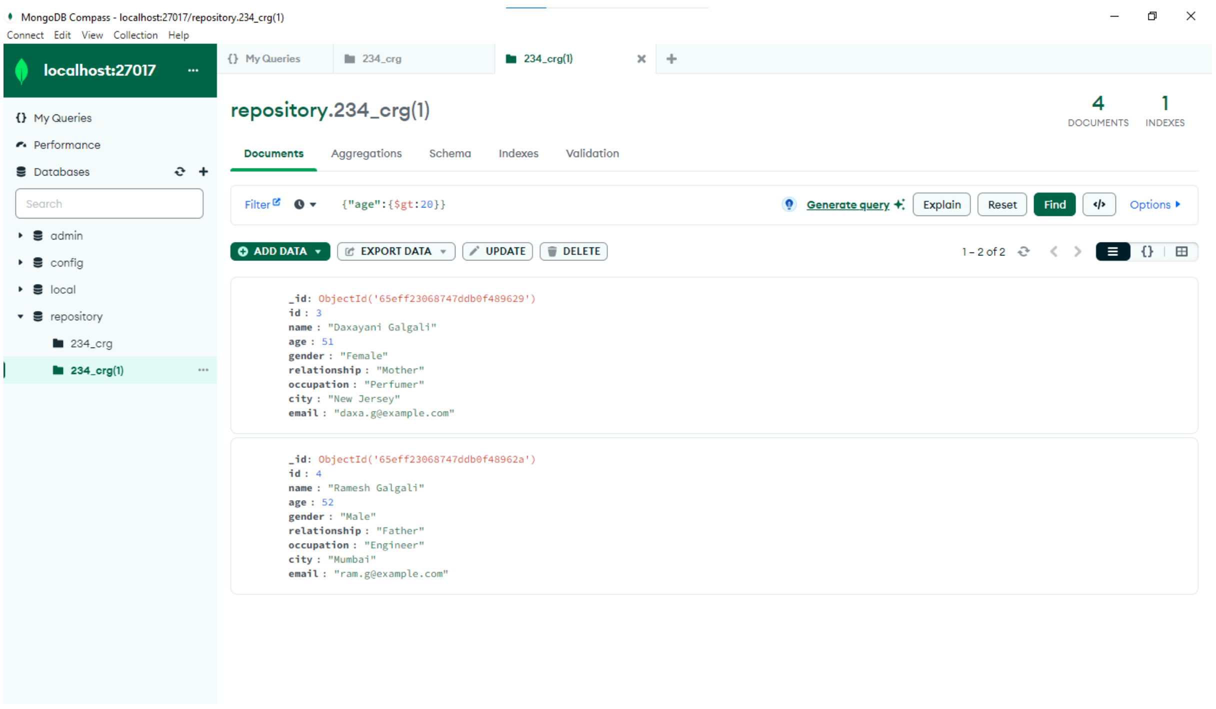


repository.234_crg(1)

```
_id: ObjectId('65eff23068747ddb0f489627')
id : 1
name : "Chandana Galgali"
age : 19
gender : "Female"
relationship : "Self"
occupation : "Engineer"
city : "New York"
email : "chandana.g@example.com"
```

```
_id: ObjectId('65eff23068747ddb0f489628')
id : 2
name : "Nayu Galgali"
age : 13
gender : "Female"
relationship : "Sister"
occupation : "Student"
city : "Mumbai"
email : "nayu.g@example.com"
```

```
_id: ObjectId('65eff23068747ddb0f489629')
id : 3
name : "Daxayani Galgali"
age : 51
gender : "Female"
relationship : "Mother"
```

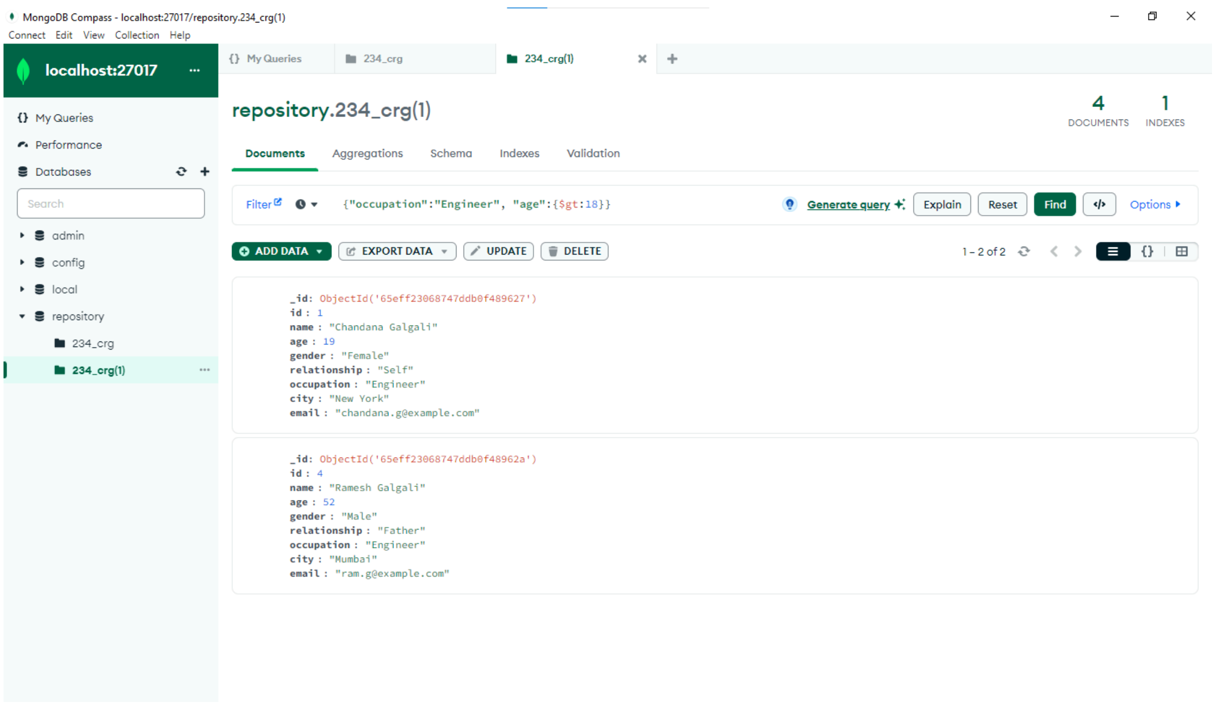


repository.234_crg(1)

```
{"age":{$gt:20}}
```

```
_id: ObjectId('65eff23068747ddb0f489629')
id : 3
name : "Daxayani Galgali"
age : 51
gender : "Female"
relationship : "Mother"
occupation : "Perfumer"
city : "New Jersey"
email : "daxa.g@example.com"
```

```
_id: ObjectId('65eff23068747ddb0f48962a')
id : 4
name : "Ramesh Galgali"
age : 52
gender : "Male"
relationship : "Father"
occupation : "Engineer"
city : "Mumbai"
email : "ram.g@example.com"
```

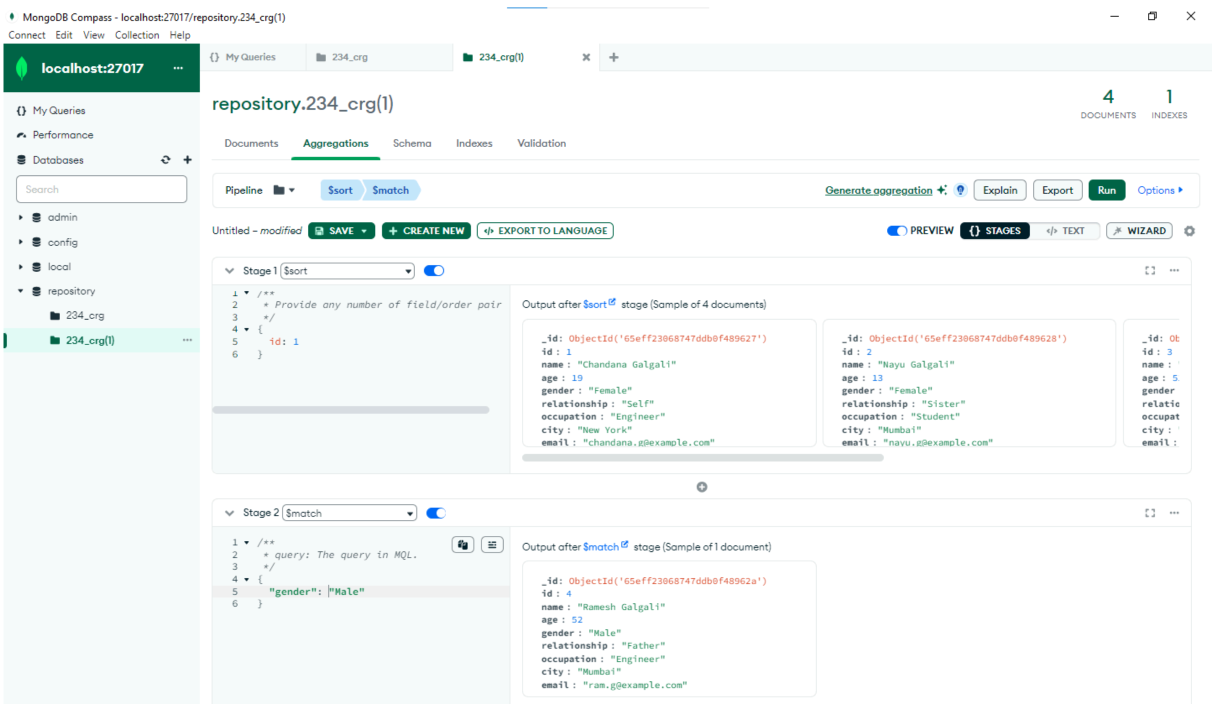


repository.234_crg(1)

```
{"occupation":"Engineer", "age":{$gt:18}}
```

```
_id: ObjectId('65eff23068747ddb0f489627')
id : 1
name : "Chandana Galgali"
age : 19
gender : "Female"
relationship : "Self"
occupation : "Engineer"
city : "New York"
email : "chandana.g@example.com"
```

```
_id: ObjectId('65eff23068747ddb0f48962a')
id : 4
name : "Ramesh Galgali"
age : 52
gender : "Male"
relationship : "Father"
occupation : "Engineer"
city : "Mumbai"
email : "ram.g@example.com"
```



MongoDB Compass - localhost:27017/repository.234_crg(1)

repository.234_crg(1) — 4 DOCUMENTS, 1 INDEXES

Pipeline: $sort → $match

Stage 1 $sort

```
/**
 * Provide any number of field/order pair
 */
{
  id: 1
}
```

Output after $sort stage (Sample of 4 documents)

```
_id: ObjectId('65eff23068747ddb0f489627')
id : 1
name : "Chandana Galgali"
age : 19
gender : "Female"
relationship : "Self"
occupation : "Engineer"
city : "New York"
email : "chandana.g@example.com"
```

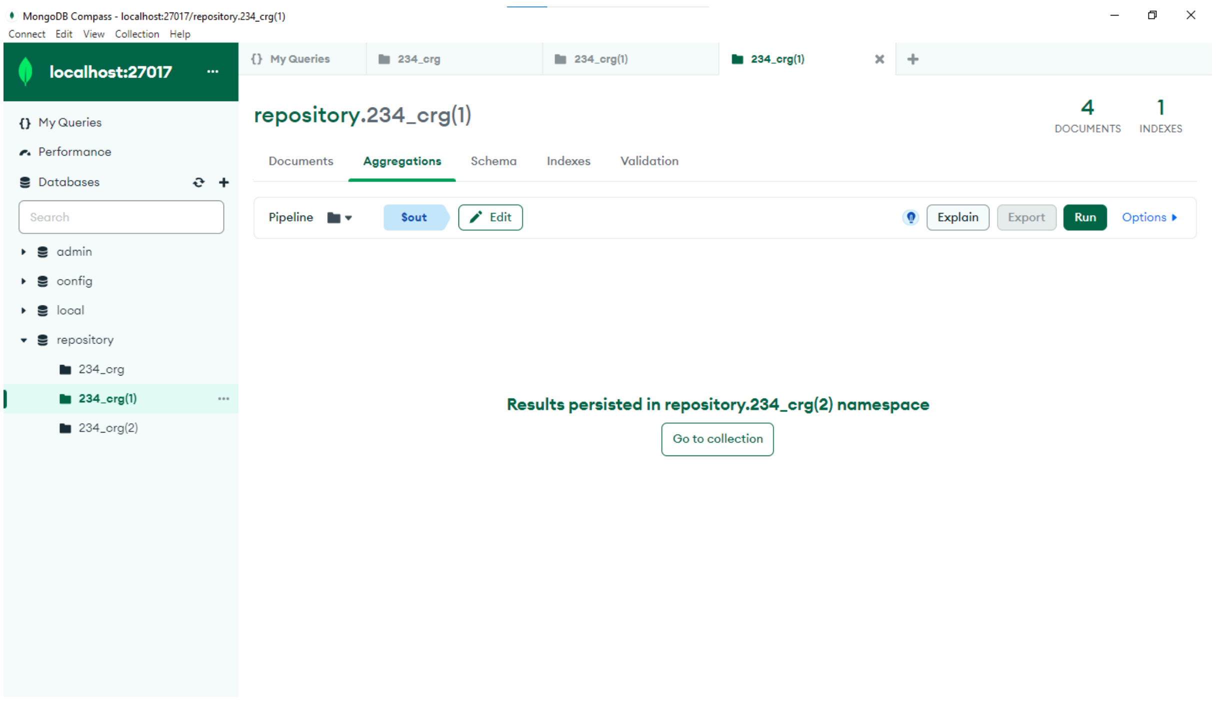
```
_id: ObjectId('65eff23068747ddb0f489628')
id : 2
name : "Nayu Galgali"
age : 13
gender : "Female"
relationship : "Sister"
occupation : "Student"
city : "Mumbai"
email : "nayu.g@example.com"
```

Stage 2 $match

```
/**
 * query: The query in MQL.
 */
{
  "gender": "Male"
}
```

Output after $match stage (Sample of 1 document)

```
_id: ObjectId('65eff23068747ddb0f48962a')
id : 4
name : "Ramesh Galgali"
age : 52
gender : "Male"
relationship : "Father"
occupation : "Engineer"
city : "Mumbai"
email : "ram.g@example.com"
```



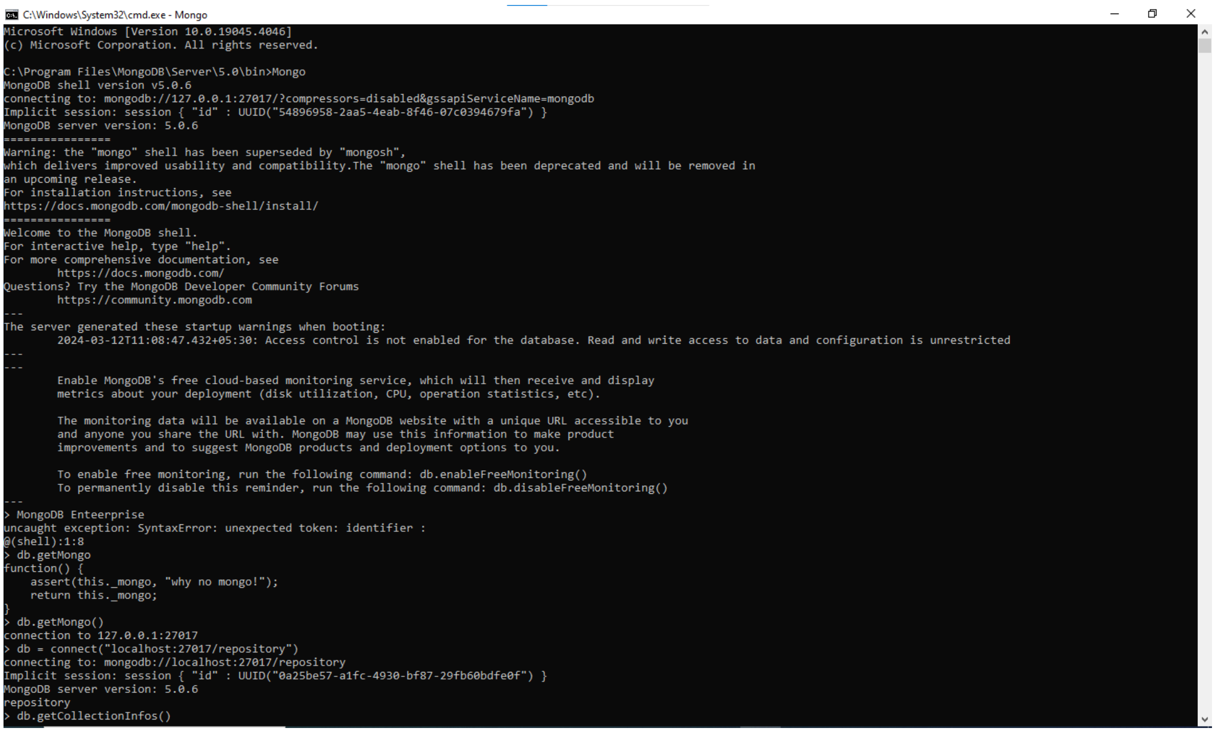
MongoDB Compass - localhost:27017/repository.234_crg(1)

repository.234_crg(1) — 4 DOCUMENTS, 1 INDEXES

Pipeline: $out

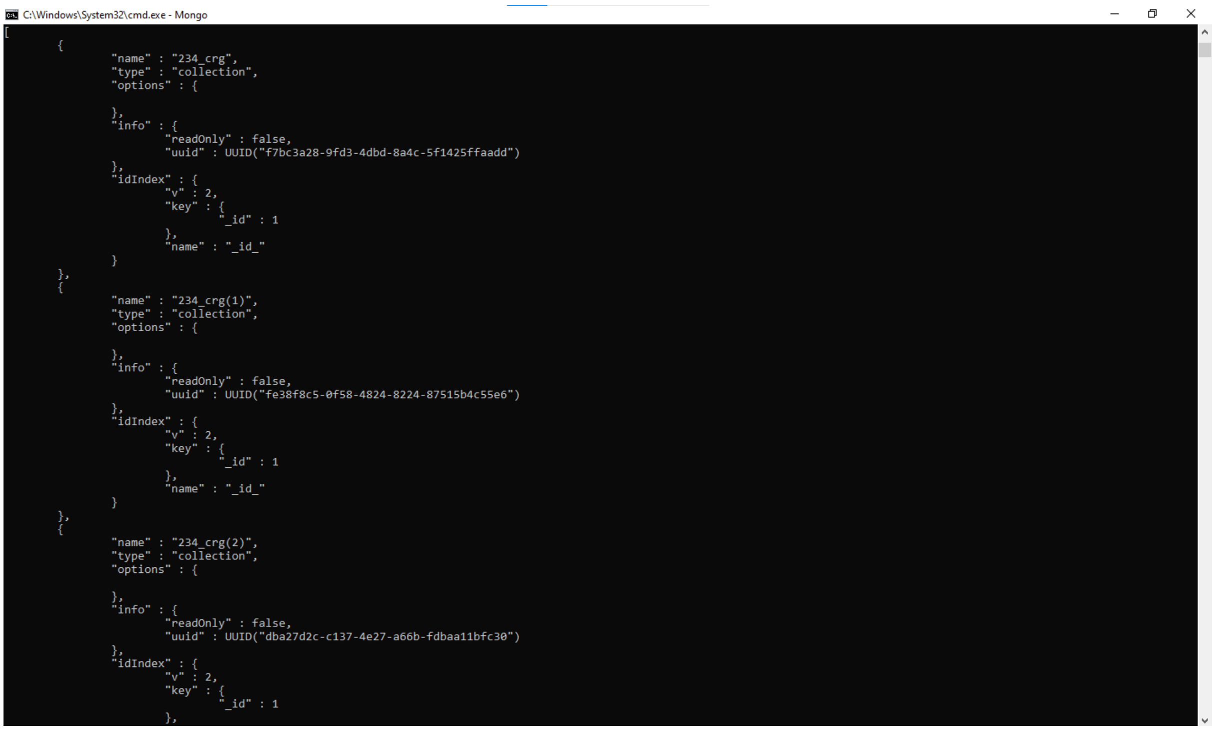
Results persisted in repository.234_crg(2) namespace

Go to collection



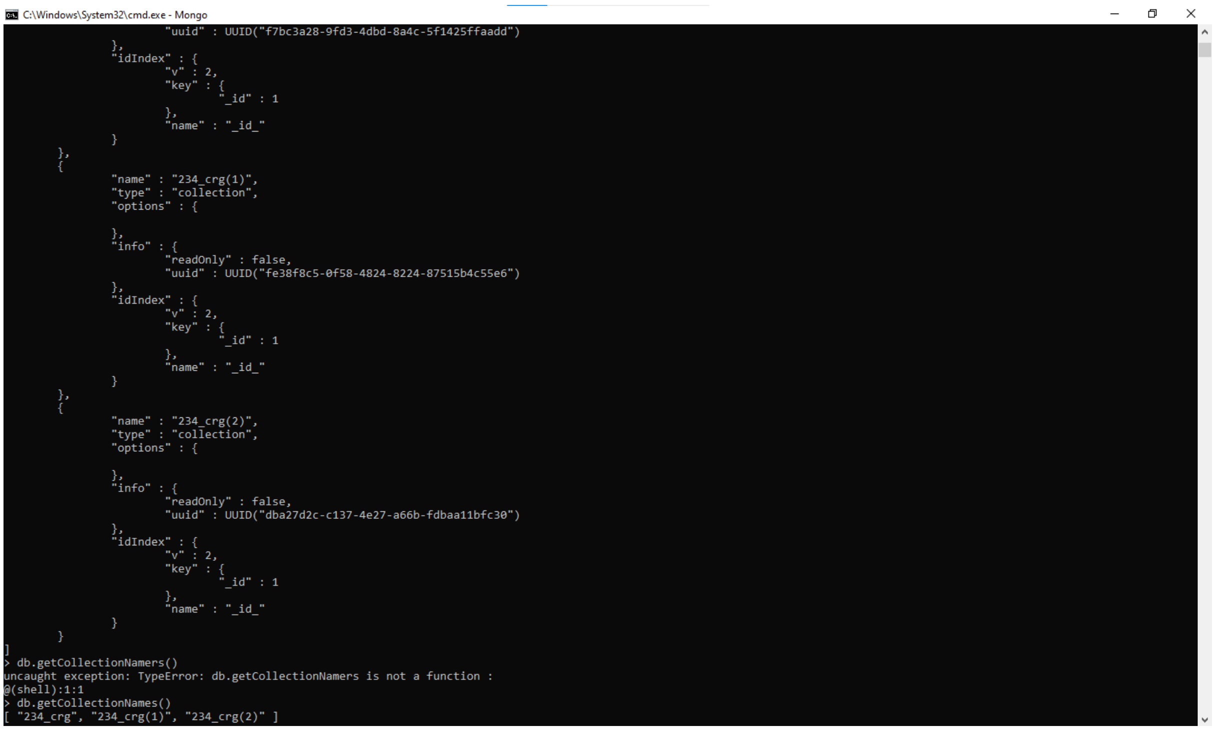
C:\Windows\System32\cmd.exe - Mongo

```
Microsoft Windows [Version 10.0.19045.4046]
(c) Microsoft Corporation. All rights reserved.

C:\Program Files\MongoDB\Server\5.0\bin>Mongo
MongoDB shell version v5.0.6
connecting to: mongodb://127.0.0.1:27017/?compressors=disabled&gssapiServiceName=mongodb
Implicit session: session { "id" : UUID("54896958-2aa5-4eab-8f46-07c0394679fa") }
MongoDB server version: 5.0.6
================
Warning: the "mongo" shell has been superseded by "mongosh",
which delivers improved usability and compatibility.The "mongo" shell has been deprecated and will be removed in
an upcoming release.
For installation instructions, see
https://docs.mongodb.com/mongodb-shell/install/
================
Welcome to the MongoDB shell.
For interactive help, type "help".
For more comprehensive documentation, see
        https://docs.mongodb.com/
Questions? Try the MongoDB Developer Community Forums
        https://community.mongodb.com
---
The server generated these startup warnings when booting:
        2024-03-12T11:08:47.432+05:30: Access control is not enabled for the database. Read and write access to data and configuration is unrestricted
---
---
        Enable MongoDB's free cloud-based monitoring service, which will then receive and display
        metrics about your deployment (disk utilization, CPU, operation statistics, etc).

        The monitoring data will be available on a MongoDB website with a unique URL accessible to you
        and anyone you share the URL with. MongoDB may use this information to make product
        improvements and to suggest MongoDB products and deployment options to you.

        To enable free monitoring, run the following command: db.enableFreeMonitoring()
        To permanently disable this reminder, run the following command: db.disableFreeMonitoring()
---
> MongoDB Enteerprise
uncaught exception: SyntaxError: unexpected token: identifier :
@(shell):1:8
> db.getMongo
function() {
    assert(this._mongo, "why no mongo!");
    return this._mongo;
}
> db.getMongo()
connection to 127.0.0.1:27017
> db = connect("localhost:27017/repository")
connecting to: mongodb://localhost:27017/repository
Implicit session: session { "id" : UUID("0a25be57-a1fc-4930-bf87-29fb60bdfe0f") }
MongoDB server version: 5.0.6
repository
> db.getCollectionInfos()
```



C:\Windows\System32\cmd.exe - Mongo

```
[
        {
                "name" : "234_crg",
                "type" : "collection",
                "options" : {

                },
                "info" : {
                        "readOnly" : false,
                        "uuid" : UUID("f7bc3a28-9fd3-4dbd-8a4c-5f1425ffaadd")
                },
                "idIndex" : {
                        "v" : 2,
                        "key" : {
                                "_id" : 1
                        },
                        "name" : "_id_"
                }
        },
        {
                "name" : "234_crg(1)",
                "type" : "collection",
                "options" : {

                },
                "info" : {
                        "readOnly" : false,
                        "uuid" : UUID("fe38f8c5-0f58-4824-8224-87515b4c55e6")
                },
                "idIndex" : {
                        "v" : 2,
                        "key" : {
                                "_id" : 1
                        },
                        "name" : "_id_"
                }
        },
        {
                "name" : "234_crg(2)",
                "type" : "collection",
                "options" : {

                },
                "info" : {
                        "readOnly" : false,
                        "uuid" : UUID("dba27d2c-c137-4e27-a66b-fdbaa11bfc30")
                },
                "idIndex" : {
                        "v" : 2,
                        "key" : {
                                "_id" : 1
                        },
```



C:\Windows\System32\cmd.exe - Mongo

```
                        "uuid" : UUID("f7bc3a28-9fd3-4dbd-8a4c-5f1425ffaadd")
                },
                "idIndex" : {
                        "v" : 2,
                        "key" : {
                                "_id" : 1
                        },
                        "name" : "_id_"
                }
        },
        {
                "name" : "234_crg(1)",
                "type" : "collection",
                "options" : {

                },
                "info" : {
                        "readOnly" : false,
                        "uuid" : UUID("fe38f8c5-0f58-4824-8224-87515b4c55e6")
                },
                "idIndex" : {
                        "v" : 2,
                        "key" : {
                                "_id" : 1
                        },
                        "name" : "_id_"
                }
        },
        {
                "name" : "234_crg(2)",
                "type" : "collection",
                "options" : {

                },
                "info" : {
                        "readOnly" : false,
                        "uuid" : UUID("dba27d2c-c137-4e27-a66b-fdbaa11bfc30")
                },
                "idIndex" : {
                        "v" : 2,
                        "key" : {
                                "_id" : 1
                        },
                        "name" : "_id_"
                }
        }
]
> db.getCollectionNamers()
uncaught exception: TypeError: db.getCollectionNamers is not a function :
@(shell):1:1
> db.getCollectionNames()
[ "234_crg", "234_crg(1)", "234_crg(2)" ]
```

Questions:
Explain with query implementation on relation created by you

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

Ans: JSONB operators in PostgreSQL are powerful tools for querying and manipulating JSONB data. Below are five JSONB-specific operators, with explanations and query implementations based on the relation you've created above:

1. -> Operator: Get JSONB Object Field by Key
This operator fetches the value of a specified key in a JSONB object as a JSONB value.
Example Query:
```
SELECT document->'name' AS name FROM DOCUMENTS WHERE id = 1;
```
This query retrieves the name field from the document JSONB column for the record with id = 1.

2. ->> Operator: Get JSONB Object Field as Text
This operator fetches the value of a specified key in a JSONB object as text.
Example Query:
```
SELECT document->>'name' AS name FROM DOCUMENTS WHERE id = 1;
```
This query retrieves the name field as text from the document JSONB column for the record with id = 1.

3. #> Operator: Get JSONB Object at Specified Path
This operator fetches the JSONB object at the specified path, allowing access to nested objects.
Example Query:
```
-- Assuming there's a deeper structure, for illustration purposes
SELECT document#>'{parent, occupation}' AS occupation FROM DOCUMENTS WHERE
id = 1;
```
This hypothetical query would retrieve the occupation of the parent if the JSON structure had nested objects under parent and occupation.

4. #>> Operator: Get JSONB Object at Specified Path as Text
This operator fetches the JSONB object at the specified path as text, useful for accessing values within nested objects directly as text.
Example Query:
```
-- Assuming a nested structure for illustration
SELECT document#>>'{parent, occupation}' AS occupation FROM DOCUMENTS WHERE
id = 1;
```

Like the previous hypothetical example, this would retrieve the occupation of the parent as text.
5. @> Operator: Contains
This operator checks if the left JSONB value contains the right JSONB value.
Example Query:
```
SELECT * FROM DOCUMENTS WHERE document @> '{"gender": "Female"}';
```
This query selects all records from DOCUMENTS where the document column contains a JSONB object with "gender": "Female".

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

Ans: i) insertMany(): This method is used to insert multiple documents into a collection at once. It takes an array of documents as input. This method is efficient for bulk insert operations.
```
db.collection.insertMany([
  { item: "card", qty: 15 },
  { item: "envelope", qty: 20 },
  { item: "stamps", qty: 30 }
]);
```
ii) deleteMany(): Similar to deleteOne(), but instead of deleting a single document, it deletes all documents that match the specified filter. This is useful for removing multiple documents at once based on certain criteria.
```
db.collection.deleteMany({ item: "envelope" });
```
iii) updateMany(): This method updates all documents that match the given filter. Like updateOne(), it can modify existing fields, add new fields, or use MongoDB's update operators. This is useful for bulk updates based on specific criteria.
```
db.collection.updateMany(
  { item: "stamps" },
  { $set: { qty: 100 } }
);
```
iv) findOne(): Retrieves a single document from the collection that matches the specified query criteria. If multiple documents match the query, only the first one found is returned. This method is useful when you only need one document that matches your criteria or to check the existence of a document.
```
db.collection.findOne({ item: "card" });
```
v) aggregate(): This method performs a multi-stage pipeline that processes and aggregates documents into grouped results. It is one of the most powerful methods in MongoDB, enabling complex data transformations and analysis.
```
db.collection.aggregate([
  { $match: { status: "A" } },
  { $group: { _id: "$cust_id", total: { $sum: "$amount" } } }
]);
```

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

---

Conclusion: (Conclusion to be based on outcomes achieved)
The experiment highlighted the complementary strengths of both databases. MongoDB offers scalability, a flexible schema design, and is well-suited for applications that need to evolve quickly or handle varied data types. PostgreSQL offers transactional integrity, complex query capabilities, and is ideal for applications requiring strict data integrity and relational data models.

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