- **AI Visualizations**: Visuals like Key Influencers, Decomposition Tree, and Smart Narratives help with advanced data analysis.

These expanded notes provide a foundation for mastering Power BI, covering essential functions, best practices, and guidance for more advanced users. Let me know if you need further elaboration on any specific area!

same way give me mongo dbnotes

Here's a comprehensive, topic-wise breakdown of essential MongoDB notes for a fresher:

---

1. Introduction to MongoDB

- **What is MongoDB?**
  MongoDB is a NoSQL, document-oriented database. Unlike relational databases, it stores data in JSON-like documents, providing flexibility for unstructured data.
- **Key Characteristics**:
  - **Schema-less**: Allows dynamic fields, meaning documents in the same collection do not need to have the same set of fields.
  - **Horizontal Scalability**: Designed for distributed data and horizontal scaling.
  - **High Performance**: Optimized for read and write-heavy workloads.
- **Primary Components**:
  - **Database**: Logical grouping of collections.
  - **Collection**: Group of documents, similar to tables in RDBMS.
  - **Document**: Basic data unit, stored in BSON format (binary JSON).
- **Advantages**:
  - Flexible schema, fast development, scalability, and support for hierarchical data storage.

2. MongoDB Data Structure

- **Document Model**:
  - Documents are BSON (Binary JSON) format objects. Each document can have nested fields and arrays.
- **Collections and Documents**:
  - Collections are containers for documents, which store related data in a structured way.
  - Unlike rows in SQL, MongoDB documents can vary in structure within the same collection.
- **Datatypes**:
  - **String**: Textual data.
  - **Integer**: 32-bit or 64-bit integers.
  - **Boolean**: True or false values.
  - **Double**: Floating-point numbers.
  - **Arrays**: Lists of values.
  - **Objects/Embedded Documents**: Document inside another document.
  - **ObjectId**: 12-byte unique ID created automatically by MongoDB.

3. Basic MongoDB Operations

- **Connecting to MongoDB**:
  - Use the `mongo` shell or a MongoDB driver in languages like Python, JavaScript, or Java.
  - For cloud usage, connect using MongoDB Atlas.
- **CRUD Operations**:
  - **Create**: `insertOne()`, `insertMany()`
    - `db.collection.insertOne({})`: Insert a single document.
    - `db.collection.insertMany([{}])`: Insert multiple documents at once.
  - **Read**: `find()`, `findOne()`
    - `db.collection.find({})`: Retrieve all documents.

    - `db.collection.find({field: value})`: Retrieve documents with matching field value.
- **Update**: `updateOne()`, `updateMany()`, `replaceOne()`
    - `db.collection.updateOne({filter}, {update})`: Update a single document matching the filter.
    - `$set`, `$inc`, `$rename`, etc., are update operators to modify fields.
- **Delete**: `deleteOne()`, `deleteMany()`
    - `db.collection.deleteOne({})`: Deletes a single document.
    - `db.collection.deleteMany({})`: Deletes all documents matching the filter.

4. Indexing in MongoDB

- **Why Indexing?**
  Indexes improve the performance of queries by reducing the amount of data MongoDB scans.
- **Types of Indexes**:
    - **Single Field Index**: Index on a single field (e.g., `db.collection.createIndex({field: 1})`).
    - **Compound Index**: Index on multiple fields (e.g., `db.collection.createIndex({field1: 1, field2: -1})`).
    - **Text Index**: Supports text search in string fields.
    - **Geospatial Index**: Used for location-based queries.
- **Index Management**:
    - Use `db.collection.getIndexes()` to view all indexes.
    - Use `db.collection.dropIndex("index_name")` to delete an index.
- **Best Practices**:
    - Avoid over-indexing as it can impact performance.
    - Use indexing strategically based on frequent query fields.

5. Aggregation Framework

- **What is Aggregation?**
  Aggregation in MongoDB is a way to process data and return computed results.
- **Stages in Aggregation Pipeline**:
  - **$match**: Filters documents (similar to `find()`).
  - **$group**: Groups documents by a specified field and applies accumulators (e.g., `$sum`, `$avg`).
  - **$project**: Reshapes documents by including/excluding fields.
  - **$sort**: Sorts documents in ascending or descending order.
  - **$limit and $skip**: Limits the number of documents and skips specified number of documents.
  - **$unwind**: Deconstructs arrays into separate documents.
- **Example Pipeline**:

```javascript
db.collection.aggregate([
   { $match: { status: "active" } },
   { $group: { _id: "$category", total: { $sum: "$amount" } } },
   { $sort: { total: -1 } }
])
```

6. MongoDB Schema Design

- **Schema Design Strategies**:
  - **Embedded Data**: Store related information within the same document.
  - **Referenced Data**: Use references to other documents when data needs to be accessed independently.
  - **One-to-One Relationships**: Embed or reference based on size and retrieval requirements.
  - **One-to-Many Relationships**: Embed if the child documents are small and frequently retrieved with the parent.
  - **Many-to-Many Relationships**: Use arrays of references for scalability.
- **Best Practices**:

- Design schemas based on application usage patterns.
- Optimize for read-heavy or write-heavy applications.
- Limit document size to avoid performance degradation.

7. Transactions in MongoDB

- **What are Transactions?**
  Transactions allow atomic operations across multiple documents or collections, ensuring data consistency.
- **Multi-Document Transactions**:
  - Only supported on replica sets in MongoDB 4.0+.
  - Use with `session.startTransaction()`, `session.commitTransaction()`, and `session.abortTransaction()`.
- **Transaction Example**:

```javascript
const session = db.getMongo().startSession();
session.startTransaction();
try {
   db.collection1.insertOne({ ... }, { session });
   db.collection2.updateOne({ ... }, { $set: { ... } }, { session });
   session.commitTransaction();
} catch (error) {
   session.abortTransaction();
} finally {
   session.endSession();
}
```

8. Replication and High Availability

- **What is Replication?**
  Replication ensures data redundancy and availability by synchronizing data across multiple servers.
- **Replica Sets**:
  - A group of MongoDB servers where one server acts as the primary, and others as secondary.

    - The primary handles all write operations, while secondaries replicate data for failover.
- **High Availability**:
    - If the primary goes down, an election is held to select a new primary.
    - Provides fault tolerance and high availability in distributed systems.
- **Configuration**: Initiate replica sets with `rs.initiate()` and add members with `rs.add()`.

9. Sharding for Scalability

- **What is Sharding?**
  Sharding is a method for horizontal scaling, where data is partitioned across multiple servers (shards).
- **Components**:
    - **Shard**: Each shard holds a subset of data.
    - **Config Server**: Stores metadata about the data distribution.
    - **Query Router (mongos)**: Directs client requests to the appropriate shard.
- **Shard Keys**:
    - Used to partition data. Selecting an effective shard key is critical for balanced distribution.
- **Sharding Process**:
    - Enable sharding on a database (`sh.enableSharding("database")`).
    - Shard collections by specifying a shard key (`sh.shardCollection("database.collection", { key: 1 })`).

10. MongoDB Security

- **Authentication**:
    - Use username and password authentication to restrict database access.
    - Enable role-based access control (RBAC) to manage permissions.
- **Authorization**:
    - MongoDB offers predefined roles (e.g., `read`, `readWrite`, `dbAdmin`).

    - Custom roles can be created to fit specific needs.
- **Data Encryption**:
    - Enable TLS/SSL for secure data in transit.
    - Use MongoDB's built-in encryption for data at rest.
- **Network Security**:
    - Limit network access to the MongoDB instance and use firewalls to control traffic.

11. MongoDB Administration and Monitoring

- **Backup and Restore**:
    - Use `mongodump` and `mongorestore` for data backup and restoration.
    - For production, consider incremental backups and backup replicas.
- **Performance Monitoring**:
    - Use `mongostat` and `mongotop` for real-time statistics.
    - Leverage the MongoDB Monitoring Service (MMS) for advanced metrics.
- **Scaling**:
    - For scaling, consider horizontal scaling (sharding) or vertical scaling (adding resources to existing instances).
    - Optimize indexes, queries, and schema design to improve performance.

12. MongoDB Atlas

- **What is MongoDB Atlas?**
  MongoDB's fully managed cloud database service, offering automated backups, scaling, and security

Give pandas notes for freshers interview in detail

Here's a detailed set of notes on **Pandas** that covers fundamental concepts for a fresher's interview:

---

1. Introduction to Pandas

Give Mongo Dbnotes for freshers interview in detail in more deatils with code

Here's a comprehensive MongoDB guide for freshers covering important concepts, commands, and example queries. MongoDB, a NoSQL database, is frequently used for its flexible schema and scalability. This guide includes key topics often discussed in interviews.

---

1. Introduction to MongoDB

- **MongoDB** is a NoSQL, document-oriented database where data is stored in JSON-like format (Binary JSON or BSON).
- **Collections**: Similar to tables in relational databases.
- **Documents**: Similar to rows in relational databases, stored as JSON objects.
- **MongoDB Database**: Collection of collections.

---

2. Basic MongoDB Commands

a. Connecting to MongoDB

- **MongoDB Shell**:

```bash
mongo
```

- **Using a Database**:

```js
use database_name
```

b. Show Databases and Collections

```js
```

```
show dbs;        // Lists all databases
show collections; // Lists all collections in the current database
```

c. Creating and Dropping Databases

- **Create** a database by switching to it (MongoDB creates it when data is added).

```js
use new_database;
```

- **Drop** a database:

```js
db.dropDatabase();
```

---

3. Data Model in MongoDB

- MongoDB data is stored in **documents** with **key-value pairs**.
- Documents can have different structures in the same collection (schema flexibility).

Example Document

```js
{
   "_id": ObjectId("507f191e810c19729de860ea"),
   "name": "Alice",
   "age": 25,
   "department": "HR",
   "skills": ["Communication", "Recruiting"],
   "address": {
       "city": "New York",
       "zip": "10001"
   }
}
```

4. CRUD Operations

a. Create Documents

- **Insert One Document:**

```js
db.collection_name.insertOne({
    "name": "John",
    "age": 30,
    "department": "Finance"
});
```

- **Insert Multiple Documents:**

```js
db.collection_name.insertMany([
    {"name": "Alice", "age": 25, "department": "HR"},
    {"name": "Bob", "age": 28, "department": "Engineering"}
]);
```

b. Read Documents

- **Find All Documents:**

```js
db.collection_name.find();
```

- **Find with Conditions:**

```js
db.collection_name.find({"age": {$gt: 25}});
```

- **Find Specific Fields:**

```js
db.collection_name.find({"age": {$gt: 25}}, {"name": 1, "age": 1});
```

- **Find One Document:**

```js
db.collection_name.findOne({"name": "Alice"});
```

c. Update Documents

- **Update One Document:**

```js
db.collection_name.updateOne(
    {"name": "John"},
    { $set: {"age": 31} }
);
```

- **Update Multiple Documents:**

```js
db.collection_name.updateMany(
    {"department": "Finance"},
    { $set: {"department": "Accounting"} }
);
```

- **Replace Document:**

```js
db.collection_name.replaceOne(
    {"name": "Alice"},
    {"name": "Alice", "age": 26, "department": "HR"}
);
```

d. Delete Documents

- **Delete One Document:**

```js
db.collection_name.deleteOne({"name": "John"});
```

- **Delete Many Documents:**

```js
db.collection_name.deleteMany({"age": {$lt: 25}});
```

---

5. Query Operators

- **Comparison Operators**:
  - `$gt` : Greater than
  - `$lt` : Less than
  - `$gte` : Greater than or equal
  - `$lte` : Less than or equal
  - `$ne` : Not equal
  - **Example**:

```js
db.collection_name.find({"age": {$gte: 25, $lte: 30}});
```

- **Logical Operators**:
  - `$and` : Both conditions must be true
  - `$or` : Either condition can be true
  - `$not` : Negates the condition
  - `$nor` : Neither condition is true
  - **Example**:

```js
db.collection_name.find({$or: [{"department": "HR"}, {"department": "Finance"}]});
```

- **Element Operators**:
  - `$exists` : Checks if a field exists
  - `$type` : Checks data type of field

- **Example:**

```js
db.collection_name.find({"skills": {$exists: true}});
```

---

6. Aggregation in MongoDB

- **Aggregation** processes data records and returns computed results.
- Aggregation pipelines use multiple stages, such as `$match`, `$group`, `$sort`.

a. Basic Aggregation Pipeline Example

```js
db.collection_name.aggregate([
    { $match: {"department": "HR"} },
    { $group: { _id: "$department", avgAge: { $avg: "$age" } } }
]);
```

b. Pipeline Stages

- `$match`: Filters documents by conditions.
- `$group`: Groups documents by a specific field.
- `$sort`: Sorts the documents by a specified field.
- `$project`: Specifies fields to include or exclude.
- `$limit`: Limits the number of documents returned.
- **Example:**

```js
db.collection_name.aggregate([
    { $match: {"age": {$gte: 25}} },
    { $group: { _id: "$department", totalEmployees: { $sum: 1 } } },
    { $sort: { totalEmployees: -1 } },
    { $limit: 5 }
]);
```

7. Indexes

- **Indexes** improve search performance in MongoDB.
- **Creating an Index**:

```js
db.collection_name.createIndex({"name": 1});  // 1 for ascending, -1 for descending
```

- **Compound Index** (Multiple fields):

```js
db.collection_name.createIndex({"department": 1, "age": -1});
```

- **Drop Index**:

```js
db.collection_name.dropIndex("index_name");
```

- **List All Indexes**:

```js
db.collection_name.getIndexes();
```

---

8. Schema Design Principles

- **Embedding**: Store related data in a single document (recommended when data is accessed together).
  - **Example**:

```js

```

```
{
  "_id": ObjectId("507f191e810c19729de860ea"),
  "name": "Alice",
  "address": {
      "city": "New York",
      "zip": "10001"
  }
}
```

- **Referencing**: Store related data in separate documents with references (recommended for large, frequently modified data).
  - **Example**:

```js
// Employees collection
{ "_id": ObjectId("507f1f77bcf86cd799439011"), "name": "Alice",
"address_id": ObjectId("507f191e810c19729de860ea") }

// Addresses collection
{ "_id": ObjectId("507f191e810c19729de860ea"), "city": "New York", "zip":
"10001" }
```

---

9. Transactions in MongoDB

- **MongoDB Transactions** allow atomic operations on multiple documents (available in replica sets and sharded clusters).
- **Start a Transaction**:

```js
const session = db.getMongo().startSession();
session.startTransaction();

try {
    const employeesCollection = session.getDatabase("company").employees;
    const departmentsCollection = session.getDatabase("company").departments;
```

```
    employeesCollection.updateOne(
        {"name": "Alice"},
        { $set: {"salary": 50000} },
        { session }
    );

    departmentsCollection.updateOne(
        {"name": "HR"},
        { $inc: {"budget": -50000} },
        { session }
    );

    session.commitTransaction();
} catch (error) {
    session.abortTransaction();
} finally {
    session.endSession();
}
```

---

10. Common Interview Questions

1. **What is MongoDB and how is it different from relational databases?**
2. **Explain collections and documents in MongoDB.**
3. **What are indexes, and why are they used?**
4. **Describe the aggregation pipeline in MongoDB.**
5. **What are the advantages of NoSQL databases like MongoDB?**
6. **What are the differences between `find` and `aggregate`?**
7. **How do you handle transactions in MongoDB?**

---

11. Useful MongoDB Commands Cheat Sheet

- **Show all databases:**

```js
show dbs;
```

- **Create/Use a database:**

```js
use database_name;
```

- **Show collections:**

```js
show collections;
```

- **Drop a database:**

```js
db.dropDatabase();
```

- **Insert a document:**

```js
db.collection_name.insertOne({...});
```

- **Find a document:**

```js
db.collection_name.find({...});
```

- **Update a document:**

```js
db.collection_name.updateOne({...}, {$set: {...}});
```

- **Delete a document:**

```js
```

```
db.collection_name.deleteOne({...});
```

These MongoDB notes provide foundational knowledge for freshers, helping them prepare for typical interview questions and scenarios.

Give spark notes for freshers interview in detail in more deatils with code

Apache Spark is a popular distributed data processing framework used for big data applications. Here's a comprehensive Spark guide for freshers, covering important concepts, terminology, and example code in PySpark (Python API for Spark), which is commonly used in interviews.

---

1. Introduction to Apache Spark

- **Apache Spark** is an open-source, distributed computing system that processes large datasets across multiple nodes.
- It supports various programming languages, such as Python (PySpark), Scala, Java, and R.
- **Components** of Spark:
  - **Spark Core**: The foundational engine for processing large datasets.
  - **Spark SQL**: Module for structured data processing using SQL queries.
  - **Spark Streaming**: Real-time data stream processing.
  - **MLlib**: Machine learning library for scalable ML algorithms.
  - **GraphX**: Library for graph and graph-parallel computations.

---

2. Spark Architecture