Mongo DB

Mini-Project 3

CIS8045

Unstructured Data Management

Master's in Information Systems Fall 2018

Georgia State University

Rahul Sharma

Shruthi Jabshetty

Shrikant Patel

Deepali Pareek

Anubhav Pradhan

Purpose 1:

a- Overall total number of products per category

Query:

```
db.review_data.aggregate([
{$group: {_id:{categories: "$isCategory",product: "$asin"}}},
{$group: {_id:{category: "$_id.categories"}, count :{$sum :1}}}
])
```

Output:

```
> db.review_data.aggregate([
... {$group: {_id:{categories: "$isCategory",product: "$asin"}}},
... {$group: {_id:{category: "$_id.categories"}, count :{$sum :1}}}
... ])
{ "_id" : { "category" : "Digital Music" }, "count" : 649 }
{ "_id" : { "category" : "Musical Instruments" }, "count" : 900 }
{ "_id" : { "category" : "Automotive" }, "count" : 268 }
{ "_id" : { "category" : "Instant Video" }, "count" : 854 }
{ "_id" : { "category" : "Office Products" }, "count" : 489 }
>
```

Summary of the Results of the Query:

There are no explicit match criterions mentioned in the query as we want to showcase the complete output. The first group, groups the documents, basis categories, and the product. This will list down all the products per category and to find the number of product per category we have used the second group by. The second group takes the output of the first as its input. Then groups it further based on categories, to fetch the count of products that belong to each category.

b- Overall total number of reviews per category

Query:

```
db.review_data.aggregate([
$group: {_id:{categories: "$isCategory"}, reviewCount:{$sum: 1}}}
])
```

Output:

```
> db.review_data.aggregate([
... {$group: {_id:{categories: "$isCategory"}, reviewCount:{$sum: 1}}}
... ])
{ "_id" : { "categories" : "Office Products" }, "reviewCount" : 8454 }
{ "_id" : { "categories" : "Musical Instruments" }, "reviewCount" : 10261 }
{ "_id" : { "categories" : "Digital Music" }, "reviewCount" : 12037 }
{ "_id" : { "categories" : "Automotive" }, "reviewCount" : 3198 }
{ "_id" : { "categories" : "Instant Video" }, "reviewCount" : 23502 }
>
```

Summary of the Results of the Query:

As we want to find the number of reviews per category, the $group operator groups the documents, basis categories and $sum returns number of reviews per category

c- Overall total number of reviewers per category

Query:

```
db.review_data.aggregate([
{$group: {_id:{categories: "$isCategory", reveiwer : "$reviewerID"}, count:{$sum: 1}}},
{$group: {_id : {categories : "$_id.categories"}, count:{$sum :1}}}
])
```

Output:

```
> db.review_data.aggregate([
... {$group: {_id:{categories: "$isCategory", reveiwer : "$reviewerID"}, count:{$sum: 1}}},
... {$group: {_id : {categories : "$_id.categories"}, count:{$sum :1}}}
... ])
{ "_id" : { "categories" : "Digital Music" }, "count" : 3237 }
{ "_id" : { "categories" : "Musical Instruments" }, "count" : 1429 }
{ "_id" : { "categories" : "Automotive" }, "count" : 1773 }
{ "_id" : { "categories" : "Instant Video" }, "count" : 22275 }
{ "_id" : { "categories" : "Office Products" }, "count" : 3549 }
>
```

Summary of the Results of the Query:

The first group operator, groups the documents, basis categories and reviewer. This will list down all the reviewers per category, and to find the number of reviewers per category we have used the second group operator. The second group operator, takes the output of the first as its input. Then groups it further based on categories, to fetch the total number of reviewers in each category.

d- Date of the oldest review per category

Query:

```
db.review_data.aggregate([
{$sort:  {unixReviewTime : 1}},
{$group:  {_id:{categories: "$isCategory"},firstReviewDate: {$first : "$unixReviewTime"},
date: {$first: "$reviewTime"}}},
{$project: {"_id.categories":1,"date":1}}
])
```

Output:

```
> db.review_data.aggregate([
... {$sort:   {unixReviewTime : 1}},
... {$group:  {_id:{categories: "$isCategory"},firstReviewDate: {$first : "$unixReviewTime"},
... date: {$first: "$reviewTime"}}},
... {$project: {"_id.categories":1,"date":1}}
... ])
{ "_id" : { "categories" : "Office Products" }, "date" : "09 29, 2000" }
{ "_id" : { "categories" : "Automotive" }, "date" : "07 15, 2005" }
{ "_id" : { "categories" : "Instant Video" }, "date" : "03 14, 2000" }
{ "_id" : { "categories" : "Musical Instruments" }, "date" : "09 18, 2004" }
{ "_id" : { "categories" : "Digital Music" }, "date" : "04 28, 1998" }
>
```

Summary of the Results of the Query:

We have initially sorted the entire collection based on unixReviewTime(The review time field is a string field and hence cannot be sorted) field in ascending order, as the final output requires the oldest review per category. The group operator, groups the documents basis category. To find the oldest review date we have used $first operator, which returns the first element of each group. $project helps project the expected output fields, i.e Categories, and the oldest Review dates

e- Average review per product – display the results for 50 products only

Query:

```
db.review_data.aggregate([
{$group:{_id: {product:"$asin"},avgReview: {$avg : "$overall"}}},
{$limit: 50}
])
```

Output:

```
> db.review_data.aggregate([
... {$group:{_id: {product:"$asin"},avgReview: {$avg : "$overall"}}},
... {$limit: 50}
... ])
{ "_id" : { "product" : "B000AYDQZW" }, "avgReview" : 4.833333333333333 }
{ "_id" : { "product" : "B000AN0UH0" }, "avgReview" : 4.428571428571429 }
{ "_id" : { "product" : "B000AMYSOM" }, "avgReview" : 4.333333333333333 }
{ "_id" : { "product" : "B000A7WBGY" }, "avgReview" : 4.4 }
{ "_id" : { "product" : "B000A6X9CU" }, "avgReview" : 3.6666666666666665 }
{ "_id" : { "product" : "B000A3IAHM" }, "avgReview" : 4.875 }
{ "_id" : { "product" : "B000A2BJR6" }, "avgReview" : 4.616666666666666 }
{ "_id" : { "product" : "B000A2BJ76" }, "avgReview" : 4.285714285714286 }
{ "_id" : { "product" : "B000A0OFXS" }, "avgReview" : 3.926829268292683 }
{ "_id" : { "product" : "B0009WPKY0" }, "avgReview" : 4.5 }
{ "_id" : { "product" : "B0009VGD24" }, "avgReview" : 4.375 }
{ "_id" : { "product" : "B0009ORUZU" }, "avgReview" : 4 }
{ "_id" : { "product" : "B0009K6UX2" }, "avgReview" : 4.5 }
{ "_id" : { "product" : "B00094H4LU" }, "avgReview" : 5 }
{ "_id" : { "product" : "B00094GJR0" }, "avgReview" : 4.4 }
{ "_id" : { "product" : "B00094C6AO" }, "avgReview" : 3.76 }
{ "_id" : { "product" : "B00092RJX0" }, "avgReview" : 4.660714285714286 }
{ "_id" : { "product" : "B0008GQ26C" }, "avgReview" : 4.75 }
{ "_id" : { "product" : "B0008GNVKC" }, "avgReview" : 4.6 }
{ "_id" : { "product" : "B0008FULBK" }, "avgReview" : 4.4 }
Type "it" for more
>
```

Summary of the Results of the Query:

The group operator, groups the documents, basis product and $avg help find the average product rating per product. $limit, limits the projection of the output to 50 products only.

f- Histogram of review ratings (5's, 4's, 3's, 2's and 1's) per product – display the results for 10 products only. Note that there is no requirement for a graphical histogram (like the one on Amazon)

Query:

```
db.review_data.aggregate([
{$group: {_id: {product:"$asin",rating : "$overall"}}},
{$group: {_id : {productId : "$_id.product"}, ratings : {$push:{rate: "$_id.rating", count : "$count"}}}},
{$limit: 10}
])
```

Output:

```
> db.review_data.aggregate([
... {$group: {_id: {product:"$asin",rating : "$overall"}}},
... {$group: {_id : {productId : "$_id.product"}, ratings : {$push:{rate: "$_id.rating", count : "$count"}}}},
... {$limit: 10}
... ])
{ "_id" : { "productId" : "B0000AXTUY" }, "ratings" : [ { "rate" : 5 } ] }
{ "_id" : { "productId" : "B000SM8GWY" }, "ratings" : [ { "rate" : 1 } ] }
{ "_id" : { "productId" : "B000MW6PHS" }, "ratings" : [ { "rate" : 1 } ] }
{ "_id" : { "productId" : "B000PU4O1Q" }, "ratings" : [ { "rate" : 5 } ] }
{ "_id" : { "productId" : "B000QTBD6A" }, "ratings" : [ { "rate" : 1 } ] }
{ "_id" : { "productId" : "B0033P1O6S" }, "ratings" : [ { "rate" : 5 } ] }
{ "_id" : { "productId" : "B000NPM2WQ" }, "ratings" : [ { "rate" : 5 } ] }
{ "_id" : { "productId" : "B00008RW9U" }, "ratings" : [ { "rate" : 4 }, { "rate" : 5 }, { "rate" : 2 }, { "rate" : 3 } ] }
{ "_id" : { "productId" : "B00008RW9V" }, "ratings" : [ { "rate" : 5 }, { "rate" : 3 } ] }
{ "_id" : { "productId" : "B0000AXU02" }, "ratings" : [ { "rate" : 4 }, { "rate" : 5 } ] }
>
```

Summary of the Results of the Query:

The first group operator, groups the document basis product id and product rating. The second group operator, takes the output of the first as its input and groups it again by productid and the $push helps to form an array of count of ratings, per rating category(5's, 4's, 3's, 2's and 1's) for a product. $limit, limits the output to 10 product.

g- Top 10 most helpful reviews per product – display the results for 10 products only (you will have to first provide your own text-based definition of what helpful means and thus not use the Helpful field)

Index Creation:

```
db.review_data.createIndex({reviewText: "text"})
```

Query:

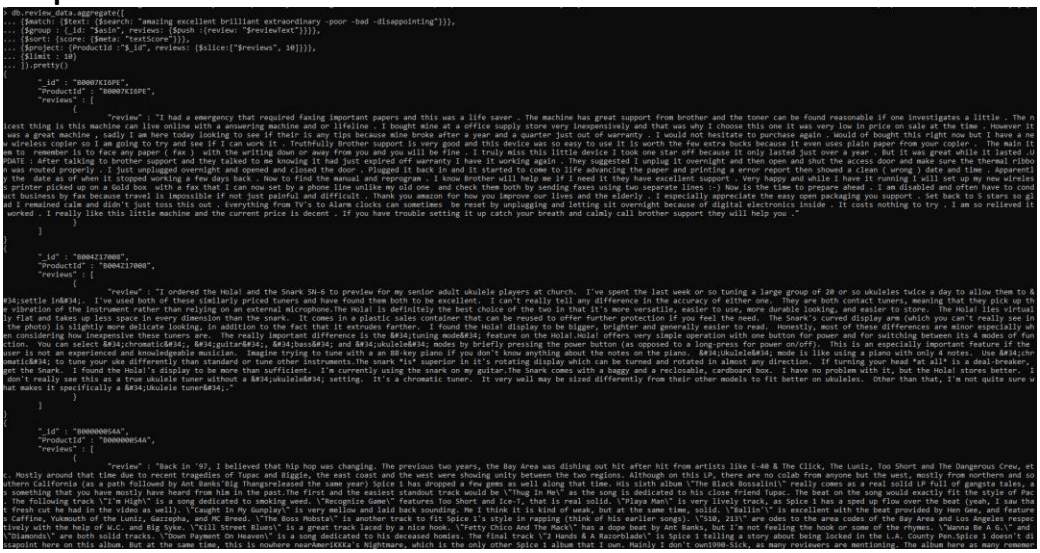
```
db.review_data.aggregate([
{$match: {$text: {$search: "amazing excellent brilliant extraordinary -poor -bad -
disappointing"}}},
```

```
{$group : {_id: "$asin", reviews: {$push :{review: "$reviewText"}}}},
{$sort: {score: {$meta: "textScore"}}},
{$project: {ProductId :"$_id", reviews: {$slice:["$reviews", 10]}}},
{$limit : 10}
])
```

Summary of the Results of the Query:

Text indexes are created to support text search queries on string content. The index here enables the searching of keywords in the review text. $match operator searches and filters the review text based on keywords provided in the $search operator, which includes positive keywords like (amazing, excellent, brilliant, extraordinary)and excludes negative keywords like (poor, bad, disappointing). $group groups the documents after the filter operation, basis product id, and finds the corresponding helpful review.The output is then sorted based on its textScore (found using $meta operator) to find the most helpful reviews. The $project operator projects the product id and the top 10 most helpful reviews. $slice operator finds the top 10 reviews in the array and $limit, limits the output to 10 products.

Output:

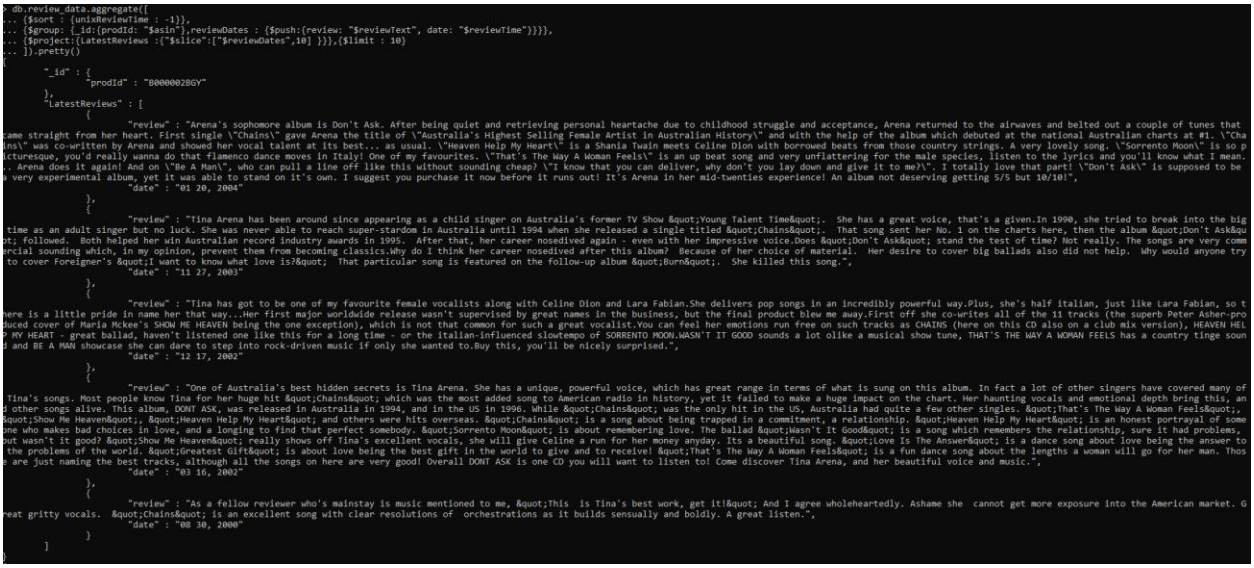


h- Top 10 most recent reviews per product – display the results for 10 products only

Query:

```
db.review_data.aggregate([
{$sort : {unixReviewTime : -1}},
{$group: {_id:{prodId: "$asin"},reviewDates : {$push:{review: "$reviewText", date: "$reviewTime"}}}},
{$project:{LatestReviews :{"$slice":["$reviewDates",10] }}},{$limit : 10}
]).pretty()
```

Output:



Summary of the Results of the Query:

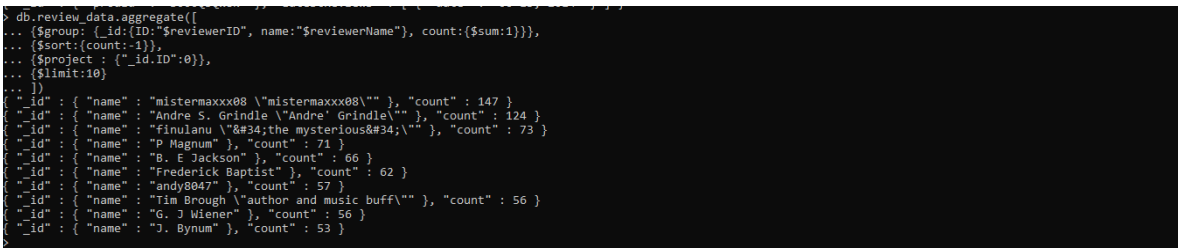
The collection has been sorted in descending order basis the unixReviewTime field as the final output requires the most recent review per category. The group by command groups the documents of sorted collection by product id, the $push operator creates an array of reviews and review date for each product. The output array projection is limited to the latest 10 reviews using $slice. The number of products displayed is limited to 10 products using $limit.

i- Top 10 most prolific reviewers (i.e. with most number of reviews)

Query:

```
db.review_data.aggregate([
{$group: {_id:{ID:"$reviewerID", name:"$reviewerName"}, count:{$sum:1}}},
{$sort:{count:-1}},
{$project : {"_id.ID":0}},
{$limit:10}
])
```

Output:



Summary of the Results of the Query:

The group operator groups the documents basis reviewer id and $sum returns the number of reviews per reviewer. The output is sorted in descending order basis the number of reviews per reviewer. Only the reviewer name and the count of reviews is projected with help of $project operator and the output is limited to 10 reviewers using $limit.

j- Top 10 most verbose reviewers (i.e. that write the most text)

Query:

```
db.review_data.aggregate([
{$project: {reviewerID :1, reviewerName:1, length: { $strLenCP: "$reviewText" }}},
{$sort: {length: -1}},
{$group: {_id: {id: "$reviewerID", name:"$reviewerName"},len: { $first: "$length" }}},
{$sort: {len: -1}},
{$limit: 10} ]
```

Output:

```
> db.review_data.aggregate([
... {$project: {reviewerID :1,reviewerName:1, length: { $strLenCP: "$reviewText" }}},
... {$sort: {length: -1}},
... {$group: {_id: {id: "$reviewerID", name:"$reviewerName"},len: { $first: "$length" }}},
... {$sort: {len: -1}},
... {$limit: 10}
... ])
{ "_id" : { "id" : "A335HWVBHCBNA9", "name" : "BAP" }, "len" : 16125 }
{ "_id" : { "id" : "AAHM9PTW1M2SN", "name" : "semaj emorej" }, "len" : 14301 }
{ "_id" : { "id" : "A16QODENBJVUI1", "name" : "Robert Moore" }, "len" : 13766 }
{ "_id" : { "id" : "AI0BCEWRE04G0", "name" : "Stoney" }, "len" : 13301 }
{ "_id" : { "id" : "A1X4JO8EJ1U5BR", "name" : "Mike London \"MAC\"" }, "len" : 12789 }
{ "_id" : { "id" : "A20JJ8634DG3F5", "name" : "Johnny Guitar \"J.F. Guitar\"" }, "len" : 11310 }
{ "_id" : { "id" : "A1TZDTEULS0618", "name" : "Elianna Greenleaf \"PokeManiac\"" }, "len" : 11213 }
{ "_id" : { "id" : "A3SR2FSNU40ZSD", "name" : "L. Boki \"L. Boki\"" }, "len" : 10667 }
{ "_id" : { "id" : "APPV1ZDETO7B", "name" : "beatlenik49 \"Fixing A Hole Where The Rain Get..." }, "len" : 10488 }
{ "_id" : { "id" : "ACY9QYNDFLVBI", "name" : "G. Farnsworth" }, "len" : 10275 }
```

Summary of the Results of the Query:

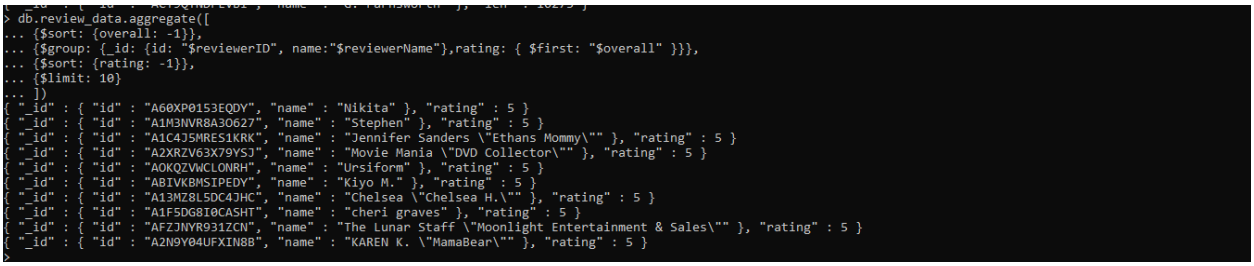
$strLenCP returns the length of the String(Review Text). Combined with $project operator it projects the Reviewer id , reviewer name and the string length of their review. The output is then sorted in descending order basis the length of the review string. $Group helps to group the document by reviewer id and reviewer name, $first fetches the first element of each group. The output is then sorted in the descending order basis the length of the review string to find the reviewers who wrote the most text.

k- Top 10 most positive reviewers based on the ratings of their reviews

Query:

```
db.review_data.aggregate([
{$sort: {overall: -1}},
{$group: {_id: {id: "$reviewerID", name:"$reviewerName"},rating: { $first: "$overall" }}},
{$sort: {rating: -1}},
{$limit: 10}
])
```

Output:



Summary of the Results of the Query:

The collection is sorted in descending order basis the overall rating. $Group groups the collection basis reviewer id and reviewer name, $first helps fetch the first element of each group. The output is sorted in descending order basis the ratings to get the most positive reviewer, based on the rating of their reviews.

l- The total number of helpful votes received for each product (across all reviews) – display the results for 10 products only.

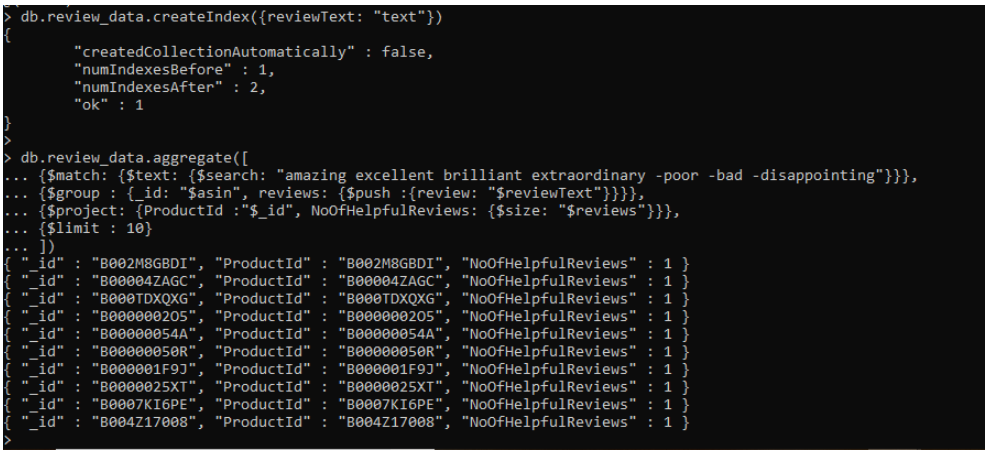
Index Creation:

db.review_data.createIndex({reviewText: "text"})

Query:

db.review_data.aggregate([
{$match: {$text: {$search: "amazing excellent brilliant extraordinary -poor -bad -disappointing"}}},
{$group : {_id: "$asin", reviews: {$push :{review: "$reviewText"}}}},
{$project: {ProductId :"$_id", NoOfHelpfulReviews: {$size: "$reviews"}}},
{$limit : 10}
])

Output:



Summary of the Results of the Query:

Text indexes are created to support text search queries on string content. The index here enables the searching of keywords in the review text. $match operator searches and filters the review text based on keywords provided in the $search operator, which includes positive keywords like (amazing, excellent, brilliant, extraordinary)and excludes negative keywords like (poor, bad, disappointing). $group groups the documents, basis product id, and finds the corresponding helpful review. The array of helpful reviews is formed using $push operator, $size operator is used to find the number of helpful reviews of each product. The $project operator projects the product id and the number of helpful reviews per product. $limit,limits the output to 10 products.

Purpose 2:

Introduction

With more and more systems transitioning into MongoDB, security becomes a far more pertinent question. Security concerns related to MongoDB first surfaced around the year 2012 and it was only by early 2015 the fact, that there were around 30,000 insecure MongoDB installations, was realized[1] and through 2016 Christmas to 2017 February there had been multiple cases of ransom attacks on the MongoDB databases. While access control and other security mechanism were part of MongoDB, the default settings were simple and allowed access to unauthorized users. These loopholes in security allowed multiple ransomware attacks during the same time frame. With the advent of the newer version, security in MongoDB was further strengthened, details of which are discussed in the following paragraph.

Security Feature in Mongo DB:

MongoDB can be designed in a way to reduce its vulnerability to attacks. In the below section we will be covering the current security features.

Authentication in MongoDB:

There is multiple authentication mechanism currently supported by MongoDB. These mechanisms can be implemented either at the Database level or through external mechanism. Default authentication in MongoDB requires a username, a password and the authentication databases related to the user[2]

- **Database Authentication:** MongoDB authenticates entities at a database level using Salted Challenge Response Authentication Mechanism (SCRAM IETF RFC 5802) standard. Using SCRAM, MongoDB verifies the supplied user credentials against the username password and the authentication database[2].
- **LDAP Authentication:** Lightweight Directory Access Protocol follows the client/server model authentication [3]. LDAP. MongoDB Enterprise Advance supports querying an LDAP server for the user groups the authenticated member is a member of. This allows MongoDB to leverage the existing LDAP infrastructure within a organization to both authenticate and authorize users
- **Kerberos Authentication**: Kerberos is an industry standard authentication protocol for large client/server systems. With Kerberos support, MongoDB can leverage any existing authentication infrastructure like Windows Active Directory [4]. In a Kerberos system every participant in the authenticated communication is known as principal and every principal must have a unique name. Principal belongs to a Realm. For each realm, the Kerberos Key Distribution Center (KDC) maintains a database of the realm's principal and the principals' associated "secret keys". For a client-server authentication, the client requests from the KDC a "ticket" for access to a specific asset. KDC uses the client's secret and the server's secret to construct the ticket which allows the client and server to mutually authenticate each other, while keeping the secrets hidden. [5]
- **x.509 Certificate Authentication:** It enables MongoDB to combine existing Security infrastructure with certificate authorities and support both user and inter-node authentication. User and Inter-Cluster authentication is done via certificates which ensure more control and less overhead. If a client x.509 certificate's subject has the same O, OU, and DC combination as

the Member x.509 Certificate, the client will be identified as a cluster member and granted full permission on the system.[6]

- **Red Hat Identity Management:** This security feature enables MongoDB to integrate with Identity Management feature of Red Hat Enterprise Linux (RHEL). The Red Hat Enterprise Linux Identity Management solution, RHEL IdM integrates Kerberos authentication, directory services, certificate management, DNS and NTP in a single service This allows a central management of entities, their authentication, authorization and privileges. [7]

Authorization in MongoDB:

- **Role Based Access Control:** By default, MongoDB provides more than 10 predefined roles to govern access. Apart from these, further user defined roles can be defined to gain further control in the system. User privileges can also be defined at the database level and collection level. Privileges can then be assigned to roles and roles can be assigned to user/team. A role can include one or more existing roles in its definition, in which case the role inherits all the privileges of the included roles[8].
- **LDAP Authorization:** MongoDB enterprise advance can leverage existing LDAP user privileges and directly map them to MongoDB roles[3]
- **Field Level Security with Read-Only Views:** In MongoDB Views can be defined which will only showcase a subset of the original collection. This allows further encapsulation of personal information related fields and provides organization an intuitive way to meet compliance standards.
- **Log Redaction:** MongoDB enterprise Advanced supports Log Redactions, that prevents sensitive information to be written down into the logs. This way the administrators will have access to the metadata in the log but will be restricted to view any personal data associated with the database events**.** Log Redaction can however make troubleshooting and diagnostics more difficult due to the lack of data related to the log event. For this reason, debug messages are not redacted even when log redaction is enabled [9]

Auditing in MongoDB

- MongoDB doesn't log all the read and write event by default. In order to enable the logging, the *auditAuthorizationSucess* parameter has to be configured and then the logs will be logged under the *authcheck* event. Logs can be written to multiple destination, and in variety of formats (JSON/BSON – preferably BSON as the cost is low) [10]. MongoDB allows administrator to create audit trails for all DML, DDL and DCL commands but enabling logging has a significant performance cost associated with it, the cost depends on the events that are being logged along with the storage device and log format [10][11]. MongoDB enterprise Advanced version also supports role-based auditing.

Encryption in MongoDB: The default encryption mode that MongoDB Enterprise uses is the AES256-CBC (or 256-bit Advanced Encryption Standard in Cipher Block Chaining mode) via OpenSSL.

- **Securing Data in Transit (Network Encryption):** Support for TLS/SSL help MongoDB to encrypt network traffic. Use of TSL/SSL requires a .pem file containing a public key certificate and its associated private key. MongoDB is capable to use TSL/SSL certificates by a certificate authority or a self-signed certificate. Usage of Self-Signed certificate leaves the system vulnerable as

there will be no server identity validation. MongoDB Enterprise Advanced supports FIPS 140-2 encryption if it runs in FIPS Mode with a FIPS validated Cryptographic module.[11][12]

- **Securing Data at Rest (Disk Encryption):** With the help of MongoDB encrypted storage engine native encryption of database file on disk can be done. This helps eliminate the management and performance overhead of external encryption. Using the encrypted storage engine, the raw database content is encrypted using an algorithm that takes random encryption key as input and generates ciphertext that only be read if decrypted with the decryption key.

What makes MongoDB Susceptible to attacks:

While the above highlighted security features may lead to a perception that MongoDB is highly secured, there are still certain security loopholes that makes it vulnerable. MongoDB has limited authentication when it runs in sharded mode. With this limitation, implementation of access control and authorization for new users becomes very limited, thus increasing the vulnerability of the system. MongoDB is susceptible towards injection attacks, as it stores user credential in datafiles, administrators with access to the data files can easily recover credentials for users defined in the database [13]. There are no off the rack feature of MongoDB that allows to combine event logs across multiple nodes to a single audit log[9]. This may lead to a myopic view of operations that affects multiple nodes and can be a limitation during exigency scenarios impacting multiple nodes.

Security features of MongoDB VS RDBMS[14] [15][16]-

Feature	MongoDB	RDBMS
Authentication	The absence of a default authentication mechanism makes MongoDB comparatively more prone to attacks. Access controls have to be enabled to enforce Authentication in MongoDB.	In RDBMS, Operating System authentication is a default mode. It provides the feature of Authentication at different levels and different types like Operating Systems and mixed mode
Authorization	MongoDB does not enable access control by default. "–auth" can be used for addressing such cases. In case where the access control is not enabled the user have access to almost all database.	When you create database objects, you must explicitly grant permissions to make them accessible to users. The authorization processes allow or limit the level of access to users. These levels of access are privilege based.
Network Encryption	The encryption in MongoDB is done through open TLS/SSL libraries. To use these libraries, TLS/SSL certificates are required. This means for any self-certified SSL the data can be captured in Man-In-The-Middle(MITM) attack.	SQL Server provides functions to encrypt and decrypt data using a certificate, asymmetric key, or symmetric key. It manages all of these in an internal certificate store. This feature is called secret store. SQL Server supports several symmetric key encryption

		algorithms implemented by Crypto API.
Backup Logs	MongoDB doesn't log all the read and write event by default, logging has to be enabled by user. Also, it has an ability that allows the user to rotate the logs using 'logRotate' command and that prevents a single logfile from consuming too much disk space. Logs can be written to multiple destination, and in variety of formats.	The transaction log is one of the critical components of the database. All SQL Server database has a transaction log, which records all transactions and the database modifications made by each transaction. The log file in SQL can consume all disk space and cause Server to run slowly.
Injection Attacks	NoSQL injection attacks execute where the attack string is parsed, evaluated, or concatenated into a NoSQL API call. This makes it potentially more vulnerable. Since these attacks execute within a procedural language, the potential impact is higher.	SQL injection attacks typically execute within the database engine. Because of this the vulnerability of the database is lesser. Also, SQL injection attacks may execute within a declarative language, making the potential impact lesser.

LINKS:

1- https://snyk.io/blog/mongodb-hack-and-secure-defaults

2 - https://docs.mongodb.com/manual/core/authentication/

3 - https://jumpcloud.com/blog/ldap/what-is-ldap-authentication/

4 - https://docs.mongodb.com/manual/core/authentication-mechanisms-enterprise/#security-auth-ldap

5 - https://docs.mongodb.com/manual/core/kerberos/

6 - https://docs.mongodb.com/manual/core/security-x.509/#security-auth-x509

7 - https://docs.mongodb.com/ecosystem/tutorial/manage-red-hat-enterprise-linux-identity-management/

8 - https://docs.mongodb.com/manual/core/authorization/

9 - https://www.percona.com/doc/percona-server-for-mongodb/LATEST/log-redaction.html

10 - https://www.percona.com/blog/2017/03/03/mongodb-audit-log-why-and-how/

11 - https://webassets.mongodb.com/_com_assets/collateral/MongoDB_Security_Architecture_WP.pdf?_ga=2.81011799.1921314008.1540766227-283797613.1540345942

12 - https://docs.mongodb.com/manual/tutorial/configure-ssl/

13 - https://www.computer.org/csdl/proceedings/trustcom/2011/2135/00/06120863.pdf

14 - https://www.trustwave.com/Resources/SpiderLabs-Blog/Mongodb---Security-Weaknesses-in-a-typical-NoSQL-database/

15 - https://docs.microsoft.com/en-us/dotnet/framework/data/adonet/sql/authentication-in-sql-server

16- https://www.infoworld.com/article/3164504/security/the-essential-guide-to-mongodb-security.html