SQL and NoSQL Interview Questions

Your essential guide to acing SQL and NoSQL job interviews

Vishwanathan Narayanan

bpb

SQL and NoSQL Interview Questions

Your essential guide to acing SQL and NoSQL job interviews

Vishwanathan Narayanan

www.bpbonline.com

First published: 2023

Published by BPB Online
WeWork
119 Marylebone Road
London NW1 5PU

UK | UAE | INDIA | SINGAPORE

ISBN 978-93-55518-583

www.bpbonline.com

Dedicated to

My beloved mom Kalyani Narayanan
and
my sister

Ishwarya, my brother-in-law Sridhar
and
my niece Durga

My aunt Vijayalakshmi and my uncle Jayaram
and mami Rathna

About the Authors

Vishwanathan Narayanan has overall 20 years of experience in Software development and Management.

He has worked on various technologies like Java, Python, R, React, Angular, and Machine learning.

He has worked in different roles in organizations ranging from developer all the way to architect and designing solutions to complex problems.

He has worked in various roles and was one of the pioneers in using NoSQL right from proof of concept to deployment in production along with maintenance and tuning.

Selecting the right set of tools to be used for a particular use case is his expertise area.

Adapting NOSQL to areas where it fits in is his expertise.

About the Reviewer

Nadir Doctor is a database and data warehousing architect, and a DBA, who has worked in various industries with multiple OLTP and OLAP technologies. He has also worked on primary data platforms, including Snowflake, Databricks, CockroachDB, DataStax, Cassandra, ScyllaDB, Redis, MS SQL Server, Oracle, Db2 Cloud, AWS, Azure, and GCP. His major focus is health-check scripting for security, high availability, performance optimization, cost reduction, and operational excellence. He has presented at several technical conference events, is active in user group participation, and can be reached on LinkedIn.

Thank you to the author and the staff at BPB. I'm grateful for the immense support of my loving wife, children, and family during the technical review of this book. I hope that you all find the content enjoyable, inspiring, and useful.

Acknowledgements

I want to express my deepest gratitude to my family and friends for their unwavering support and encouragement throughout this book's writing,

I am also grateful to BPB Publications for their guidance and expertise in bringing this book to fruition. It was a long journey of revising this book, with valuable participation and collaboration of reviewers, technical experts, and editors.

I would also like to acknowledge the valuable contributions of my colleagues and co-worker during many years working in the tech industry, who have taught me so much and provided valuable feedback on my work.

Finally, I would like to thank all the readers who have taken an interest in my book and for their support in making it a reality. Your encouragement has been invaluable.

Preface

The last few years have seen a dynamic shift in paradigm from the traditional relational database system to the new world of Big data and NoSQL.

The advent of various NoSQL categories and their specialization in solving different types of problems has given rise to limitless opportunities in the world of software development.

When NoSQL was introduced there was resistance to adoption by many organizations but by seeing the benefit it gives, many have moved to NoSQL due to many advantages in terms of distribution and cost it provides.

In this book, we cover a variety of NoSQL as well as Relational databases and their corresponding interview questions.

The focus of the book is to help in last-minute revision for candidates appearing in interviews.

We strongly believe this book will help people from different experience groups as we have covered a wide variety of questions ranging from basics all the way to performing complex tuning.

1. **Chapter one: Relational Database interview questions**

   This chapter lays the foundation for the book. In this chapter we have covered the basics of relational databases as well have pointed out corresponding similarities and differences in various well-renowned databases like Oracle, Postgres, MySQL, and so on. This chapter will help people grasp the relational database questions and the corresponding queries which are generally asked in any interview question

2. **Chapter two: NoSQL interview questions**

   In this chapter, we have a holistic view of NoSQL and its subcategories.We see the different types of NoSQL, the working

principle, differences as compared to relational databases, and the architectural paradigm shift. This chapter will surely serve as a backbone as many interview questions will be related to this topic.

3. **Chapter three: MongoDB interview questions**

   This chapter covers MongoDB which is one the most adopted NoSQL. This chapter will not only help normal database individuals but also developers belonging to Full stack development as MongoDB is used there. We will get an overview of MongoDB, its working principle, architecture, sharding mechanism as well as queries used with it.

4. **Chapter four: Cassandra interview questions**

   This chapter deals with Cassandra, a wide column database which is especially used in reporting based applications.We will learn the use of partioning key and clustering key and see how data is divided. We will also have a look at query mechanism and how data is distributed and various levels of accuracy can be attained with Cassandra.

5. **Chapter five: Redis interview questions**

   In this chapter, we look at Redis which is an adopted NoSQL in secaniors involving cache. We will have a look on the working nature of Redis, how can it be distributed, its architecture and different mechanism of performing queries on it.

   Application areas of Redis is also discussed in this chapter.

6. **Chapter six: HBase interview questions**

   In this chapter we will have a look at HBase which forms the important components in the Big data/Hadoop world. We will look how it is integrated with Hadoop and how can we perorm query on it. Also various helper tools which are used by adminstartors to monitor HBase are also covered.

7. **Chapter seven: Elasticsearch interview questions**

   Most organizations have adopted Elasticsearch for performing various activities including searching as well as log maintenance. In this chapter, we will have a look at Elasticsearch as well as supporting ecosystems including Kibana and Logstash.

   Use cases like log storage is also covered.

8. **Chapter eight: Neo4j interview questions**

   In this chapter, we will have a look at graph-based NoSQL with the name Neo4j which is the market leader in this space. Starting from basic working to overall development of nodes, edges and relationships along with application in real-world scenarios like social media is covered.

Code Bundle and Coloured Images

Please follow the link to download the
Code Bundle and the ***Coloured Images*** of the book:

https://rebrand.ly/6hhy5nr

The code bundle for the book is also hosted on GitHub at **https://github.com/bpbpublications/SQL-and-NoSQL-Interview-Questions**. In case there's an update to the code, it will be updated on the existing GitHub repository.

We have code bundles from our rich catalogue of books and videos available at **https://github.com/bpbpublications**. Check them out!

Errata

We take immense pride in our work at BPB Publications and follow best practices to ensure the accuracy of our content to provide with an indulging reading experience to our subscribers. Our readers are our mirrors, and we use their inputs to reflect and improve upon human errors, if any, that may have occurred during the publishing processes involved. To let us maintain the quality and help us reach out to any readers who might be having difficulties due to any unforeseen errors, please write to us at :

errata@bpbonline.com

Your support, suggestions and feedbacks are highly appreciated by the BPB Publications' Family.

Piracy

If you come across any illegal copies of our works in any form on the internet, we would be grateful if you would provide us with the location address or website name. Please contact us at **business@bpbonline.com** with a link to the material.

If you are interested in becoming an author

If there is a topic that you have expertise in, and you are interested in either writing or contributing to a book, please visit **www.bpbonline.com**. We have worked with thousands of developers and tech professionals, just like you, to help them share their insights with the global tech community. You can make a general application, apply for a specific hot topic that we are recruiting an author for, or submit your own idea.

Reviews

Please leave a review. Once you have read and used this book, why not leave a review on the site that you purchased it from? Potential readers can then see and use your unbiased opinion to make purchase decisions. We at BPB can understand what you think about our products, and our authors can see your feedback on their book. Thank you!

For more information about BPB, please visit **www.bpbonline.com**.

Join our book's Discord space

Join the book's Discord Workspace for Latest updates, Offers, Tech happenings around the world, New Release and Sessions with the Authors:

https://discord.bpbonline.com

Table of Contents

Chapter 1
Relational Database

Introduction

Before deep diving into the world of NoSQL, understanding relational databases plays a very important role in identifying when to use what and why. It is very vital in transactional systems and will continue to be so in the near future.

Hence, understanding the basics and functioning of relational database management system is very important.

This chapter will introduce you to various interview questions related to Database system, SQL, Oracle, and MySQL.

Structure

In this chapter, we will discuss the following topics:

- Basic questions on the Database system
- Structured Query language (SQL) questions

- Oracle interview questions
- MYSQL interview questions

Objectives

In this chapter, we will learn the basics of relational database systems and SQL which is a very important query language.

We will also explore Oracle and MySQL; the two most used databases.

Basic questions on a Database system

Question 1: What is the meaning of a database system?

Answer: A database system consists of systems and tools which help in persisting data. Persistence implies the ability of data to live beyond the scope of its creation. Data should be available even after the restart of the system. In very simple terms, a database is a collection of logically related data. A database system helps in the persistence of the data.

Question 2: What type of operations can we do with the database?

Answer: Within the database, we can perform operations live, create, insert, update, select, and delete. The create, update and delete operation is used to create modify or remove data from the database. The select operation is basically used to get the data from the database. In software terminologies, this is known as CRUD operations.

Question 3: What do you think are the differences between a traditional file system and a database?

Answer: In a traditional file system, data is stored in the form of files. It has the following disadvantages:

- Searching for data is time consuming and tedious effort. Because for getting any data, the entire file system needs to be scanned. The problem will

aggregate if the amount of data is more which is the general case in the case of real-world systems.

- Concurrency control is absent in the case of the file system. It is quite possible that inconsistency of data or cause due to lack of concurrency control. The same data cannot be updated by two different people at the same time.
- Data isolation is very difficult to achieve with the file system.
- Integrity checks are also very difficult to achieve with respect to file systems. Alto problems due to corrupt files are very difficult to handle in the case of the file system.

Question 4: What is a relational database?

Answer: A relational database is a sub-type of the database in which data is organized in the form of tables. Data is arranged in the form of rows and columns.

Rows are also known as tuples, and Columns are also known as attributes.

Question 5: Differentiate between RDBMS and DBMS.

Answer: The following table will help you understand the difference between RDBMS and DBMS:

Criteria	DBMS	RDBMS
Storage	File format	Table format
Relationship between data	Does not exist directly	Exist directly
Redundancy / Duplication of data	Exist	Does not exist because RDMS supports normalization
User support	Single user	Multiple users
Record identification	May be or May not be present	Generally present in the form of the primary key
Amount of data stored	Less	More
Distributed support	Not present	Present

Criteria	DBMS	RDBMS
Integrity Constraints	Absent	Present
Query language support	Limited	Extensive
Codd Rule support	Supports up to 8/10 rules	Supports all the 12 rules
Example	Excel, Access	Oracle, Sybase, Postgres, MySQL, DB2

Table 1.1: *Shows the difference between DBMS and RDBMS*

Question 6: Can you list down CODD rules on DBMS and RDBMS?

Answer: The following table lists the CODD rules on DBMS and RDBMS :

Rule effect	Description
Information Rule	Data should be stored in table format.
Guaranteed Access Rule	Every piece of data stored should be accessible.
Systematic Treatment of NULL Values	NULL which represents the absence of value should be uniformly treated.
Active Online Catalog	A data dictionary also known as a catalog should be available to authorized users to query and see. This stores the description of the data.
Data Sub language	Proper query language to perform insert/update/delete and select operations should be available.
View update rule	Data represented as View should be insertable, updatable, and deletable.
High-Level INSERT/UPDATE/DELETE	It should be possible to insert/update and delete multiple rows in one go.
Physical independence	Data stored in the database should be independent of how it is accessed from the application.
Logical independence	Logical data should be independent of the user's view.
Integrity independence	All integrity constraints should be uniquely identifiable and updatable.

Rule effect	Description
Distributed independence	Data distribution should be abstracted from the end-user.
Non-sub version rule	Lower-level data change should follow all the security and integrity constraint rules.

***Table 1.2**: The rule and corresponding description of the same*

Question 7: List down various languages present in the database.

Answer: The following table lists various languages present in the database:

Language	Description
Data manipulation language (DML)	Contains statement to change data. SELECT, UPDATE, INSERT, MERGE, and DELETE fall under this category
Data definition language (DDL)	Contains the statement to create data CREATE, ALTER, DROP, TRUNCATE, RENAME
Data Control Language (DCL)	Allows giving and revoking permission of users GRANT and REVOKE
Transaction Control Language (TCL)	Allows committing and rollbacking data COMMIT, ROLLBACK, and SAVEPOINT

***Table 1.3**: Language and description of languages supported in the database*

Question 8: What are ACID properties?

Answer: The database provides a very important feature known as ACID properties as shown in the following table:

Property name	Description
Atomicity	Any transaction should be completely done or undone.
Consistency	Any transaction should always cause the data to be in a consistent state. An inconsistent state should not be present.
Isolation	Access to data being changed/updated should be controlled with the help of locks.
Durability	Changes done in the database should be always present. Restart or System failures should not affect data availability.

***Table 1.4**: Explaining ACID properties*

Question 9: How can you define a transaction?

Answer: A transaction is the smallest unit of work with respect to a database. A transaction can either be committed or rolled back based on some condition. We can combine multiple transactions to perform certain operations

Question 10: Can you explain the NULL value with reference to the database?

Answer: The NULL value indicates the absence of the value or the value is not available.

It is different as compared to blank space or zero value.

Question 11: Explain different levels of abstraction in a database.

Answer: The different levels of abstraction in the database are mentioned in the following table:

Abstraction	Description
Physical	It consists of a data storage description and it represents the lowest level.
Conceptual or Logical	Relationship between data is established at this layer. Developers and administrators work at this level.
External or View	Shows only part of data as required abstracting Physical or Logical view.

***Table 1.5**: Abstraction and description*

Question 12: What is the use of an Entity relationship diagram?

Answer: An Entity relationship model or ER diagram represents a diagrammatic representation of the database. It represents the table as an entity mapping it to a real-world object. Columns are represented as properties. Foreign key relationship-between entities are represented as the relationship between entities. By looking at the ER diagram, we can get a complete understanding of all the tables present in the database and their relationship.

Question 13: Explain intension with reference to a database.

Answer: Intension refers to database schema which is designed at the start of the database. All the tables along with

their columns and relationship are agreed upon and it remains mostly unchanged during the entire life of the database hence, it is known as intension.

Question 14: Explain extension with reference to a database.

Answer: Extension refers to several rows which can be added to the database. It can change from time to time and hence, it is known as extension.

Question 15: Do you know the use of the DML compiler?

Answer: It is responsible for converting DML statements to lower-level instructions. The converted lower-level instructions are then passed on to the Query evaluation engine.

Question 16: Explain the use of a DDL interpreter.

Answer: The DDL interpreter understands the DDL statement and adds the same to the metadata repository of the database for reference.

Question 17: Explain a weak entity.

Answer: In a database, a weak entity does not have a primary key of its own and depends on the primary key of another entity/table. As a result of this dependence, it is known as a weak entity.

Question 18: View helps in logical data independence. Is this statement correct?

Answer: View accounts for logical data independence because it does not hold data of its own but is dependent on underlying database tables for data.

Since it provides a view of data, it is known as logical data independence.

Question 19: Explain benefits and problems with respect to viewing.

Answer: Following are the benefits and problems of view:

Benefits:

- No involvement in physical storage
- Access to data can be restricted via permissions

Problems:

- More memory usage.
- On dropping a base table, views created on the top of that table will not be useful, since they become invalid.

Question 20: Define a query.

Answer: A request for information from a database is called a query. A query can search for information from single or multiple tables. How to perform a query is specified by a structured language which is known as SQL (which stands for Structured Query Language).

Question 21. What is meant by SQL?

Answer: SQL stands for Structured Query Language stands for a way to access data from a database table.

The benefits of SQL are that it is easy to use and understand. Also, we can use nested queries to do complex tasks. Not only SQL developers but also data and business analysts find it easy to deal with SQL.

Question 22: Explain function dependency with respect to the database.

Answer: When we can uniquely define one attribute using another, we call it functional dependent.

For example, an SSN can uniquely identify a person's name.

Question 23: What is a fully functional dependency?

Answer: Fully functional dependency is said to occur when a composite key can uniquely identify an instance.

A subset of keys forming a composite key should not be able to identify instances.

For example, if the composite key consists of A and B, then AB should identify C, but individually A or B should not identify C.

Question 24: How does the database support isolation?

Answer: The database supports isolation with the help of locks. There are two types of locks which are supported in the database: Shared locks and exclusive locks Shared Locks are generally preferred for read-in data and exclusive locks are preferred for writing data.

Question 25: Explain Index hunting.

Answer: It is a mechanism in which a collection of indexes are boosted together to improve query performance. Once an index is created on a column, the query optimizer will make use of the index to perform the search faster.

The query optimizer performs various operations on the given query to determine the best way to achieve performance based on an existing workload. To do this, it makes use of an index.

Question 26: Explain fragmentation.

Answer: In the case of a distributed database, the data needs to be divided into smaller units known as fragments which can be then placed across different machines called nodes. The process of doing this is known as fragmentation.

Question 27: Explain various constraints with reference to the database.

Answer: The following table explains the constraints with respect to the database:

Constraints type	Description
Not Null	This allows values other than null.
Check	Values for this column satisfy certain criteria.
Default	In case the column is not assigned a value, the default value will be used.
Unique	This ensures value for the column will be unique.
Index	This allows faster retrieval of values.
Primary key	This should satisfy the primary key which should be not null and have unique values.

Constraints type	Description
Foreign key	Refers to referential integrity in which the value of a column comes from looking up into another table.

***Table 1.6**: Various types of constraints which can be injected into a database table*

Question 28: Explain the difference between clustered and non-clustered indexes.

Answer: Clustered indexes are indexes in which the data in a table is ordered based on this index. The order of data in the table is determined by a clustered index. As a result, a given table can have only one clustered index. There is no such order restriction in the case of non-clustered indexes. If a table does not have a clustered key and has a primary key, then the ordering happens via the primary key.

Question 29: What is the meaning of normalization?

Answer: Normalization is the way of arranging data in a database in such a way that it does not cause any duplication or redundancy.

Since duplication is avoided, storage becomes very efficient. Most transactional systems prefer normalization.

Question 30: What is the meaning of denormalization?

Answer: Denormalization is a way of arranging data in a database in which data is arranged with duplicates or redundancy.

Denormalization causes duplication of data and hence storage is not very efficient. Most analytical systems prefer denormalization.

Question 31: Can you explain the first normal form (also known as 1NF)?

Answer: First normal form deals with multivalued columns.

A very simple example of the multivalued column is the phone number. A single person can have more than one phone number. Hence, instead of storing the phone number in the base table, we will create a new table for the phone number and we will have a relation with the base table.

Thus, a single-valued column should be used instead of multivalued columns.

Question 32: Can you explain the second normal form (also known as 2 NF)?

Answer: Second normal form deals with partial dependency.

This is related to the concept of fully functional dependency which is further related to a composite key.

A composite key is a key which can consist of more than one column.

Consider a classroom table. In this table, consider a composite primary key which is the roll number and class number. Only a class number cannot uniquely identify a student as a single classroom consists of more than one student and only a roll number cannot uniquely identify a student because many students can have the same roll number across classrooms.

Only the combination of a class number and roll number can uniquely identify a student and hence, follows 2 NF.

Question 33: Explain the third normal form (also called 3 NF).

Answer: Third normal form deals with transitive dependency.

As per the third normal form, there should be no transitive dependency.

A transitive dependency is said to exist if column 1 can determine column 2 and column 2 can determine column 3 and column 3 can again determine column 1.

To avoid the above problem, we will need to create a new table: one having column 1 and column 2 and the other having column 1 and column 3.

Let's assume the following employee table with **`employee_id`** as the primary key and having columns like designation and salary.

In the above-given table, **`employee_id`** can determine the designation and the designation can determine the salary. As a result, we have a transitive dependency. Hence, the above example is in 2NF but not in 3NF.

Question 34: Explain Boyce Codd's normal form.

Answer: **Boycee Codd normal form** (**BCNF**) is a stricter version of the third normal form in which for every function dependency X-> Y (X determines Y), X should be the super key.

Question 35: Explain the Fourth normal form.

Answer: **Fourth normal form** (also known as **4NF**) says that multi-valued dependency should be avoided.

Please note 4NF takes care of multi-valued dependency and 1 NF takes care of multi-valued column.

Suppose for column 1 which is the primary key, there can be multiple non-related columns, column 2 and column 3 which can have repetitive values across column 1, then we will need to divide the single table into multiple tables: one having column 1 and column 2 and the other having column 1 and column 3.

Question 36: Explain the fifth normal form.

Answer: The fifth normal form is known as 5NF and/or project/join/normal form.

This normal form tries to make more tables by dividing existing tables. It is very seldom used in the real world.

Question 37: What is the impact of normalization?

Answer: The impact of normalization is as follows:

- More number of tables created

- Redundancy / Duplication removed
- To get data from various tables, the Join operation is required

Structured Query Language (SQL) questions

In any interview, whether it is backend services or from end services; junior role or senior role, SQL questions will always be asked. This is one of the most important topics which I request readers to understand and be very comfortable with. Following are some SQL-related interview questions:

Question 1: How to create a table in a database?

Answer: We can create a table in the database using the "**`create table`**" command where we can specify all the fields we require in the table along with the corresponding data types.

A general syntax is as follows:

```
create table table_name{
  column_one datatype,
  column_two datatype,
primary key( column_one)
};
```

For example, we can create a table with the name university as follows:

```
  create table university{
  id int,
  university_name varchar(100),
primary key(id)
};
```

Question 2: Is SQL case sensitive?

Answer: SQL is not case sensitive, but generally lowercase statements are preferred.

Question 3: What is the general best practice with respect to naming columns?

Answer: Columns are named generally in lowercase. Connecting words in column names have _ For example, **emp_id.**

Question 4: Can we create a table from another table?

Answer: Many database engines allow you to copy the structure from one table to another table.

In the following example, we are copying all the columns and data from the source table to the target table:

```
CREATE TABLE target AS (SELECT * FROM source);
```

If the requirement is to copy only the structure, then we can give a where clause which will result in no rows:

```
CREATE TABLE target AS (SELECT * FROM source
where 10=20);
```

Please note you can give any condition instead of 10 = 20, the only rule is, it should not return any row.

Question 5: Can we create a table structure from multiple tables?

Answer: It is possible to create a table by selecting columns from multiple tables:

```
CREATE TABLE target AS (SELECT column_1,
column2, ... column_n FROM source_1, source_2,
... source_n);
```

Question 6: Once a table is created, can we add a new column?

Answer: Alter table with add column can be used for this purpose:

```
ALTER TABLE table_name ADD column_name column_
datatype;
```

Similarly, multiple columns have been added:

```
ALTER TABLE table_name ADD (column_name1 column_
datatype,column_name2 column_datatype);
```

Question 7: Can we alter the column once created?

Answer: Yes, the column once created can be altered. Alter tables command differs from the database engine.

The following table shows different engines and corresponding alter statements:

Engine	Alter command
MySQL, MariaDB, Oracle	`ALTER TABLE table_name MODIFY column_name data_type;`
SQL Server	`ALTER TABLE table_name ALTER COLUMN column_name data_type;`
PostgreSQL	`ALTER TABLE table_name ALTER COLUMN column_name TYPE data_definition;`

Table 1.7*: Databases and the corresponding alter statement syntax*

Question 8: Can we modify multiple columns once created?

Answer: Yes, multiple columns once created can be altered by using a single alter statement. Alter tables command differs from the database engine.

The following table shows different engines and corresponding alter statements:

Engine	Alter command
Oracle	`ALTER TABLE target_table_name MODIFY (column_1 data_type...column_n data_type);`
MySQL, MariaDB	`ALTER TABLE target_table_name MODIFY column_1 data_type, MODIFY column_2 data_type... ;`
SQL Server	No direct support for multiple alter statements required
PostgreSQL	`ALTER TABLE target_table_name ALTER COLUMN column_name TYPE data_type, ALTER COLUMN column_name TYPE data_type,`

Table 1.8*: Databases and their corresponding alter table with multiple columns*

Question 9: Can we drop a table from the database?

Answer: We can drop a table from the database using the drop table **`table_name`** command.

Question 10: Can we rename a column in the database?

Answer: Column rename is possible, but it differs from database to database.

In the following table, we have shown how to achieve the same using different databases:

Engine	Rename command
Oracle	`ALTER TABLE target_table_name RENAME COLUMN old_column_name TO new_column_name;`
MySQL, MariaDB	`ALTER TABLE target_table_name CHANGE COLUMN old_column_name TO new_column_name;`
SQL Server	`sp_rename 'target_table_name.old_column_name', 'new_column_name', 'COLUMN';`
PostgreSQL	`ALTER TABLE target_table_name RENAME COLUMN old_column_name TO new_column_name;`

***Table 1.9**: Databases and their corresponding alter table for renaming columns*

Question 11: How can we use an alter statement to create a primary key?

Answer: We can achieve that by using the following syntax:

```
ALTER TABLE target_table_name ADD CONSTRAINT
primary_constraint_name PRIMARY KEY (column1,
column2, ... column_n);
```

Question 12: How can we drop a primary key?

Answer: The Alter statement can be used to drop a primary key:

```
ALTER TABLE target_table_name DROP PRIMARY KEY;
```

Question 13: What query do we use to get data from a database table?

Answer: The SELECT query is used for this purpose.

Question 14: Can we filter data from a table?

Answer: Filtering is allowed in a database table with the help of the where clause.

Within the where clause, we can specify various conditions. More than one condition can also be used by using operators like AND and OR.

Question 15: Can you let us know how we filter on more than one condition?

Answer: Filtering on more than one condition is possible with the help of the AND and OR clause.

The AND clause will return data if both the conditions which it connects will result in true. For example, get all students who have taken courses as Database and have received A grades.

The syntax for the AND clause is as follows:

```
Select * from table_name where condition1 and
condition2
```

The OR clause will return data if any condition that it connects will return true. For example, find students who stay in Arizona or Dallas.

Question 16: Write a query to fetch the **`student_name`** from the student table in uppercase and use the ALIAS name as **`student_name_upper`**?

Answer: For doing this, we will need to make use of the UPPER function which converts the data in a column to uppercase:

```
SELECT UPPER(student_name ) AS student_name_
upper FROM student;
```

Question 17: Write a query to count the number of students in the student table.

Answer: We will need to make use of the count function to achieve the requested data:

`SELECT COUNT(*)` from student

Question 18: Write a query to find all the students from the student table whose marks are between 60 to 100.

Answer: The **`BETWEEN`** clause can be used to get data which falls in a range:

```
SELECT * FROM student WHERE marks BETWEEN 60
AND 100;
```

Question 19: How can you fetch the top N rows?

Answer: Fetching top N rows differs from database to database. This can be seen from the following table:

Database	Query
MYSQL MARIADB	`SELECT * FROM student order by id desc LIMIT N;`
SQL Server	`SELECT TOP N * FROM student order by id desc;`
Oracle 12c POSTGRES	`SELECT * FROM  student order by id desc ending FETCH FIRST 5 ROWS ONLY;`
Oracle	`SELECT * FROM student WHERE rownum <= 5 order by id desc`

***Table 1.10**: Databases and their corresponding queries to get n rows*

Question 20: Imagine a student table with two columns fname and lname. Can you write a query to combine them as **`student_name`** which is separated by :?

Answer: To achieve the above requested functionality, we will need to make use of the CONCAT function:

```
SELECT CONCAT(fname, ' :', lname) AS student_
name FROM student;
```

In the preceding query, you can see we are combining fname and lname separated by :

Question 21: Can you write a query which will fetch all the students from the student table, the name of students that ends with Z?

Answer: We can make use of the like clause in which we will use %Z since we want the names of students ending with Z.

```
SELECT * from student where name like '%Z'
```

Similarly, if you want students starting with Z, we could write the following query:

```
SELECT * from student where name like 'Z%'
```

Similarly, if you want students containing Z, we could write the following query:

```
SELECT * from student where name like '%Z%'
```

Question 22: Can you write a query which will fetch all the students from the student table other than the student with **roll_no** 10 and 30?

Answer: The NOT IN clause can be used in such cases to exclude certain data:

```
SELECT * from student where roll_no not in
(10,30)
```

Question 23: Can you write a query which will fetch all the students from the student table with **roll_no** 10,20 and 30?

Answer: The IN Clause can be used for such a scenario where we want to get data based on conditions involving the same field with different values:

```
SELECT * from student where roll_no in (10,20,30)
```

Question 24: Imagine a student table with **student_name** and **dep_name** which indicates the name of the student and department, respectively. How can you write a query to indicate **dept_name** and count of students belonging to the same department?

Answer: To achieve this, we will have to make use of the group by and count function.

Group by will be used to group the data based on **dept_name** and count will then be used to measure the count of students belonging to that group:

```
SELECT dept_name, count(*) as dept_count from
student group by dept_name.
```

Question 25: Imagine a student table with **student_name** and **dep_name** which indicates the name of the student and department, respectively. How can you write a query to indicate the **dept_name** and count of students belonging to the same department? Also, order the result in descending order of count.

Answer: To achieve this, we will need to make use of the group by and count function.

Group by will be used to group the data based on **dept_name** and count will then be used to measure the count of students belonging to that group.

To arrange the data on the count in descending order, we can make use of **ORDER BY,**

```
SELECT dept_name, count(*) as cnt from student
group by dept_name order by cnt desc
```

Question 26: Imagine a student table with columns **stud_name** and marks.

Can you write a query to get three minimum and maximum marks?

Answer: 3 minimum marks for students can be retrieved using

```
SELECT DISTINCT mark FROM student s1 WHERE 3 >=
(SELECT COUNT(DISTINCT mark )FROM student s2
WHERE s1.mark >= s2.mark) ORDER BY s1.mark DESC;
```

Similarly, 3 maximum marks can be retrieved by using the following query:

```
SELECT DISTINCT mark FROM student s1 WHERE 3 >=
(SELECT COUNT(DISTINCT mark )FROM student s2
WHERE s1.mark <= s2.mark) ORDER BY s1.mark DESC;
```

Question 27: Write a query to find N top mark without using the TOP function.

Answer: This query will be resolved using the nested query where N-1 can be passed as follows:

```
SELECT mark
FROM student s1
```

```
WHERE N-1 = (
      SELECT COUNT( DISTINCT ( s1.Salary ) )
      FROM student s2
      WHERE s2.mark >  s1.mark );
```

Question 28: Imagine a database table with the name student having columns **student_name** and **dep_name** which indicates the name of the student and department, respectively. Can you write a query that will print duplicate records present for the above table?

Answer: This will need finding the count of every row by combining all the columns and grouping them and then correspondingly using having clause to remove non-duplicated records:

```
SELECT student_name, dept_name, COUNT(*)  FROM
student GROUP BY student_name, dept_name HAVING
COUNT(*) > 1;
```

Question 29: Imagine a student table with **student_name** and **dep_name** which indicates the name of the student and department, respectively. Can you write a query which will print all the students belonging to the same department?

Answer: In such cases, we will have to perform join the same table:

```
select distinct student_name from student s1,
student s2 where s1.dep_name=s2.dept_name and
s1.student_name!= s2.student_name
```

beThe first condition is to check the similarity of **dept_name** and the second condition is to ensure unique **student_name**.

Question 30: Can you list down different types of JOIN in the database and explain how each of them differs from the other?

Answer: The following table describes the differences between different types of JOIN:

JOIN	Description
INNER	Only matching data from two tables involved in join will come.
LEFT OUTER JOIN	All the data from the left table will appear and only matching ones from the right should come. In case of no match is found in the right table, null would come.
RIGHT OUTER JOIN	All the data from the right table will appear and only matching ones from the left should come. In case of no match is found in the left table, null would come,
FULL OUTER JOIN	All the data from both tables will appear. Wherever no matches occur null values will come

***Table 1.11**: Types of join and their corresponding explanation*

Question 31: Image a student table with columns **`stud_name`**, **`dept_name`** and mark. Can you write a query to find the average marks department-wise?

Answer: For performing the preceding query, we will need to group **`dept_name`** and then find the average of marks:

```
select dept_name, average(mark) from student
group by dept_name
```

Question 32: Consider a table containing information about students with autoincrement ID fields. Write a query to fetch 50% of records from the student table.

Answer: For solving the preceding problem, we will need to find the count and then divide the count by 2.

Post dividing by 2, we will need to restrict the ID based on the value:

```
SELECT * FROM student WHERE id  <= (SELECT
COUNT(id)/2 from student);
```

Question 33: Image a student table with columns **`student_name`**, **`dept_name`** and mark. Can you write a query to list **`dept_name`** having less than 10 students?

Answer: To solve the preceding problem, we will need to group the data in the table by **dept_name** and then count the number of students. Post getting the number of students per department, we will need to check for a count of less than 10:

```
select dept_name,count(*) from student group by
dept_name having count(*) < 10
```

Question 34: Consider a table student with autoincrement ID field. Write a query to print the first and last record from the table.

Answer: Since the ID is an auto-increment field, if we get the minimum value of the ID field, then we can get the first record:

```
select * from student where id = select distinct
min(id) from student
```

Same way, if we find the maximum value of the ID field, we will get the last record of the table:

```
select * from student where id = select distinct
max(id) from student
```

Question 35: Imagine a student table with columns **student_name**, **dept_name** and mark. Can you write a query to find the average marks, minimum marks, and maximum marks department-wise?

Answer: For this, we will need to make use of aggregate operators like min, max, and avg:

```
Select  dept_name,min(mark),max(mark),avg(mark)
from student group by dept_name
```

Question 36: Imagine a student table with columns **student_name**, **dept_name** and mark. Write a query to display students who have two characters followed by sh which can further be followed by any combination.

Answer: In the case of SQL query if you want to indicate a character, we leave single _. For two characters, we will have to leave two __ which will be followed by sh which will be further followed by %:

```
select * from student where student_name like
'__sh%';
```

Question 37: Imagine we have two tables table 1 and table 2 having a common set of columns col1,col2, and col3. Can you write a query to get a common set of records between these two tables?

Answer: To get a common set of records from two different tables having the same columns, and have different implementations in different databases which are illustrated in the following table:

Database	Query
MYSQL Maria DB	SELECT * FROM table1 WHERE col1 IN (SELECT col1 from table2)
SQL Server/Postgres SQL/Oracle	SELECT * FROM table1 INTERSECT SELECT * FROM table2

***Table 1.12**: Databases and their corresponding select query to get a common set of data*

Question 38: What is the basic difference between **`UNION`** and **`UNION ALL`**?

Answer: When we perform **`UNION`** on two tables, duplicates are removed.

When we perform **`UNION ALL`** on two tables, duplicates are retained.

Hence, **`UNION ALL`** performs better than union.

Question 39: Write an SQL query to update the **`student_name`** in the student table by removing leading and trailing spaces.

Answer: For removing space before and after in the **`student_name`** field, we will need to use LTRIM and RTRIM methods, respectively:

```
UPDATE student SET studen_name= LTRIM(RTRIM(stud_
name));
```

Question 40: Imagine a student table with columns **`student_name`**, **`dept_name`** and mark. Can you select those students who do not have any department?

Answer: The absence of a value in a column is indicated by null. We will be using is null to check whether the column contains a null value:

```
SELECT * FROM student WHERE dept_name IS NULL;
```

Question 41: How can you get the current date and time in different databases?

Answer: The current date time methods are different for different databases. Shown in the following table are different databases and their corresponding implementations:

Databases	Time implementation
MySQL/ Maria db	Select now()
SQL Server	SELECT getdate();
Oracle	SELECT SYSDATE FROM DUAL;
Postgres	SELECT LOCALTIMESTAMP ;

***Table 1.13**: Databases and their corresponding query to get date and time*

Question 42: Imagine two tables table 1 and table 2. We need to fetch only those records from table 1 which have value in table 2. How can we achieve that?

Answer: For getting data from one table based on entries from another table, we make use of the exists clause:

```
Select * from table1 where exists (select * from
table2 where table1.id=table2.id)
```

Question 43: How can we fetch only records of odd numbers from the database?

Answer: We can make use of the MOD function which returns the remainder after division with 2. Also, we will need an autoincrement field:

```
SELECT * FROM table1 WHERE MOD (id, 2) <> 0;
```

The preceding query will give odd rows.

Similarly, we can also get even rows:

```
SELECT * FROM table1 WHERE MOD (id, 2) = 0;
```

Question 44: Imagine a student table with columns **student_name**, **dept_name** and mark. Can you write queries using the TOP or LIMIT function to get the N'th mark?

Answer: You can write the following query to get the N'th mark:

```
SELECT TOP 1 col1
FROM (
      SELECT DISTINCT TOP N col1
      FROM table1
      ORDER BY col1 DESC
      )
ORDER BY col1 ASC;


SELECT col1
FROM table1
ORDER BY col1 DESC LIMIT N-1,1;
```

Question 45: Can you explain the difference between **TRUNCATE** and **DELETE**?

Answer: Delete is majorly used to delete certain rows based on condition. It is also tied to transactions and hence even if you are deleting the entire table, it will perform the delete row by row.

On the other hand, Truncate is not tied to any transactions and hence will be preferred whenever we need to delete the entire contents of the given table.

Also, delete operations are logged while truncate operations are not logged.

Question 46: Can you explain the different types of **LOCKS** in SQL?

Answer: Locks in the case of SQL can be classified into two types:

Implicit: These are locks that are implicitly used for queries other than select.

Explicit: These are locks that are set explicitly by the developer to indicate the critical section.

Also, locks can be further classified as exclusive locks which can be used by only one user at a time or shared locks which can be used by multiple users.

Question 47: When to use `COMMIT` and `ROLLBACK`?

Answer: Commit is used to mark the changes done as a part of the transaction as complete and permanent.

Rollback is used to mark the changes done as a part of a transaction to void, that is, to remove all the changes done as a part of a transaction.

Save point: This is used to divide the transaction into smaller parts and perform rollback /commit in parts known as a save point.

Question 48: What are the uses of triggers?

Answer: Triggers are basically used to automate action in case of an event. An event can be either inserting, updating or deleting an event. A trigger can be created in such a way that an event occurring on one table can cause an action on another table. A simple example is an Audit table which can be used to store logs of events occurring in one table.

Triggers can be classified as:

- Statement level trigger: This happens when a statement gets executed irrespective of how many rows are affected. As a result, with this type of trigger, we will not get the before and after values of the table under effect. The major use of this type of trigger is additional security.
- Row level trigger: In this type of trigger, we will have an action defined for each event on every row in a table. As a result, we will be able to retrieve

the before and after values for each row. The major application of such a trigger is to perform an audit.

- BEFORE and AFTER trigger: As the name suggests, these are used for performing some action before or after the occurrence of an event.

Question 49: What is the use of a stored procedure and functions?

Answer: A stored procedure is a group of SQL statements that are pre-compiled database objects which execute the compiled code whenever they are called. A stored procedure can take inputs and return output values though both input and output are optional.

Similar to the stored procedure are functions, which are also pre-compiled database objects, but have to compulsorily return value or result. Another difference is in the case of functions input parameters cannot be changed.

Similarly, a procedure cannot be called from a function, but a function can be called from a stored procedure.

The benefit of the stored procedure is that we can include conditional logic that resides in the database.

Functions can operate on any row returned as a result set of an SQL statement.

Question 50: What is the use of database view?

Answer: A view is a database object which is used to get data from a given query that can be from one or more tables. They are basically used to display data from complex queries.

Views can be generally classified as dynamic view which gets automatically updated on changes in the base tables or static view which requires manual effort in updating the data.

Question 51: What is the partitioned table?

Answer: Some databases allow us to divide a table into smaller parts known as partitions. It provides a logical structure to store a big table into smaller parts.

Oracle interview questions

Question 1: What are SYSTEM tablespace and its relevance with respect to Oracle?

Answer: SYSTEM tablespace is an important table space that contains data dictionary tables for the entire database.

Question 2: Given the Oracle version A.B.C.D.E, what does each label mean?

Answer: The following table shows the version part and their corresponding description:

Version part	Description
A	Major database release number
B	Database maintenance release number
C	Application server release number
D	Component Specific release number
E	Platform Specific release number

***Table 1.14**:Mapping of version part with a description*

Question 3: Why is a bulk copy or BCP used?

Answer: Bulk copy or BCP in Oracle allows performing import and export easily for backup as it copies data without copying the structure of the table.

Question 4: Can you establish the relationship between a tablespace and a data file?

Answer: The entire Oracle database is divided into tablespace. The data in each tablespace is stored in a physical structure known as a data file. So, a single tablespace can correspond to one or more data files

The data files stored depend upon the nature of the operating system in which they are stored.

Question 5: How does the Oracle database ensure reliability?

Answer: One way of ensuring reliability in Oracle is by using snapshots.

In the case of a snapshot, the data from the database is stored on a remote system for later retrieval and usage.

In case of severe errors in the database, we can recreate the entire database by using snapshots.

Question 6: How can tables be classified in Oracle?

Answer: Tables in Oracle can be classified as user tables, that is, they are created by users to store their data and another as data dictionaries, that is, they are created automatically by Oracle to store metadata.

Question 7: Can you explain how many types of backups exist in Oracle?

Answer: Oracle database supports the following two main types of backup:

- Cold backup: In this type of back up the database is shut down using the **`SHUTDOWN`** command. Since the database is shut down it is also known as an offline backup.
- HOT backups: This is done when the database system is on and working. For certain critical applications, there can be no downtime. Hot backups are used for this purpose. Before attempting a hot backup at least one cold backup is needed.

Question 8: What does Oracle use to commit / rollback transactions?

Answer: Save points can be used for the above purpose.

By dividing the given transaction into a much smaller transaction, it becomes easy to capture or roll back the transaction. Oracle database supports 5 save points.

Question 9: Do you understand the difference between **`VARCHAR`** and **`VARCHAR2`**?

Answer: Length difference is one of the major differences between **`VARCHAR`** (roughly around 2000 characters) and **`VARCHAR2`**(roughly around 4000 characters).

The second major difference is that **VARCHAR2** optimizes by storing only as much as stored, thus leaving unused space thus saving space.

Question 10: Is it possible to store binary data or byte data in Oracle?

Answer: RAW data type is typically used to store binary data of variable length or byte data.

Question 11: Which string function can be used to get a subpart of a string identified by position?

Answer: SUBSTR is an inbuild function available in Oracle for performing substring based on numeric values. We can give starting and ending positions.

Question 12: How to find whether a substring exists in a string?

Answer: This can be achieved with the help of the **INSTR** function which returns the occurrence of a string within another string.

Question 13: How can we replace a null value in a select query with a corresponding new value?

Answer: NVL function can be for this purpose. Whenever NVL encounters a null value in a query, it replaces the same with a corresponding new value.

Question 14: Explain the difference between **TRANSLATE** and **REPLACE**.

Answer: **TRANSLATE** replaces any matching character with the corresponding entire string.

REPLACE matches the whole matching character and replaces the entire matching string:

```
TRANSLATE("vishwa","vi","78") results in
7878shwa

REPLACE("vishwa","vi","78") results in 78shwa
```

Question 15: What is the use of **COALESCE**?

Answer: Sometimes, in the select query, we can have multiple expressions to be evaluated as a part of select and the first non-null expression needs to be evaluated as a

result, this is what **COALESCE** does. It returns the first nonnull expression.

Question 16: Where can **ROWID** be used?

Answer: ROWID provides the fastest way to retrieve data from a table since it represents the physical location of a row. One important point to note is that ROWID can change over time whenever the table is rebuilt or whenever a row gets deleted.

Question 17: Do you know how **ROWID** and **ROWNUM** differ?

Answer: **ROWID** represents the physical storage details of a row, it is generally a 16-bit hexadecimal number and indicates block, row, and file in that order.

ROWNUM represents a sequence that gets allocated when data is retrieved and it is a numeric sequence number and this is not related to accessing data from storage.

Question 18: How can you do conditional decisions (if else in Oracle)?

Answer: To handle such cases, we can make use of the **DECODE** or **CASE** function:

```
select country_code,
DECODE (country_code,'UK', 'United
kingdom','USA', 'United states')
FROM country;

select country_code
, CASE (WHEN country_code ='UK' then 'United
kingdom'
WHEN country_code ='USA' then 'United states'
ELSE 'NOTHING') END
FROM country;
```

In the examples shown, we get country names based on code.

Question 19: Explain the use of merge statements in Oracle.

Answer: The use of merge statements is to perform insert or update on one table based upon another table. Thus, it can be used for conditional insert or update

Here is the insertion/updation of data in table1 based upon the values in table t2:

```
MERGE INTO table t1
    USING (SELECT * FROM table2 t2 where
    t2.id> 100) h
    ON (t1.id = t2.emp_id)
  WHEN MATCHED THEN
    UPDATE SET t1.column2 = t1.column2
  WHEN NOT MATCHED THEN
    INSERT (id, column2)
    VALUES (t2.id,t2.column2);
```

Question 20: Can you explain the use of a Dual table?

Answer: A dual table is present under the ownership of the **SYS** user and is accessible to all users. This table contains a column with the name **DUMMY** and is used whenever we want to return value only once.

Question 21: Will you be able to list down all the functions which are used for conversion in the Oracle database?

Answer: Conversions in Oracle are of the following two types:

- Implicit conversions:

  These are conversions that are automatically done by Oracle with data types which are easily convertible.

  Some of them are as follows:

  From VARCHAR2/CHAR to NUMBER, DATE

  From NUMBER To VARCHAR2

  From DATE To VARCHAR2

- Explicit conversions:

  These are conversions that require explicit functions to convert from one data type to another.

  Some of them are as follows:

  TO_NUMBER is used for converting from character to number.

  TO_CHAR is used for converting date or number to char.

  TO_DATE is majorly used for converting from **VARCHAR** date to actual date.

Question 22: Do you know the difference between count(column) and count(*)?

Answer: count(column) returns the count of non-null values. Duplicates are included.

count(distinct column) returns the count of non-null values and with duplicates excluded.

count(*) return the count of the table, including null values and duplicate values.

Question 23: What is PLSQL?

Answer: PLSQL stands for procedural language extension to structured query language.

It allows developers to make use of procedural constructs for SQL statements and provides the ability to perform various operations. They are block-structured solutions providing better exception handling and better performance.

Question 24: Can you classify SQL functions?

Answer: SQL functions can be broadly classified as follows:

Single row function: They operate on a single row and give a single row as a result. For example, Conversion functions, Date functions fall under this category.

Multiple row functions: They operate on multiple rows and give results based on a group of rows. For

example, AVG, COUNT, MAX, MIN, SUM, STDDEV, and VARIANCE falls under this category.

MYSQL DB Questions

Question 1: Can you list down various types of storage engines in MYSQL?

Answer: The following table has various types of storage engines in MYSQL:

Storage	Description
Innodb	• Was the default storage engine once upon a time in MYSQL • Supports transactions • Crash recovery, row-level locking possible with MYSQL
MYISAM	• Supports table-level locking • Does not support transactions • Used in Web/ data warehousing
Memory storage/ Heap storage	• In-memory storage • Faster access
CSV	• Majorly used for dealing with CSV files
Merge	• Consists of a series of MYISAM-related database objects mainly intended for dealing with volumes
Archive storage	• Provides compression of data stored
Blackhole	• Does not store data • Preferred for testing
Federated	• For connecting to remote databases • Used for the distributed environment

Table 1.15*: Shows the storage engine and the corresponding description*

Question 2: What is the use of heap tables in MYSQL?

Answer: Heap tables are tables that are used for faster operations and hence, these tables store data in memory.

A few rules around heap tables are:

- Non-usage of indexes
- An auto-increment field
- Only use of comparison operators and non-usage of blob or text fields

Question 3: What are the uses of the `LENGTH` and `CHAR_LENGTH` functions?

Answer: The `LENGTH` function returns the size in bytes occupied while `CHAR_LENGTH` is used to return the length of characters stored.

Question 4: How can we enforce only certain values for a column?

Answer: Enum in MYSQL is typically used where a column can have values only from a certain set of values.

Question 5: How can we get the version of MYSQL?

Answer: We can get that with the help of the select version() query.

Question 6: What is the use of myisamchk?

Answer: It stands for `MYISAM` check which is used for compression of data thus saving disk or memory.

Question 7: How can we access tables from other remote databases?

Answer: We can achieve that with the help of a federated table which enables getting data from other remote MYSQL databases without having to use replication or cluster technology

Question 8: Is there any difference between BLOB and TEXT data types?

Answer: Sorting and searching are case sensitive for BLOB but not for TEXT.

Question 9: Are there any restrictions on how many columns can be used to create indexes?

Answer: Indexes can be created on a maximum of 16 columns.

Question 10: Can you list down commonly used SQL functions from a MYSQL perspective?

Answer: The following table illustrates the most commonly used SQL functions from an MYSQL perspective:

Functions	Description
CONCAT(A, B)	Concatenates two string values to create a single-string output. Often used to combine two or more fields into one single field.
FORMAT(X, D)	Formats the number X to D significant digits.
CURRDATE(), CURRTIME()	Returns the current date or time.
NOW()	Returns the current date and time as one value.
MONTH(), DAY(), YEAR(), WEEK(), WEEKDAY()	Extracts the given data from a date value.
HOUR(), MINUTE(), SECOND()	Extracts the given data from a time value.
DATEDIFF(A, B)	Determines the difference between two dates and it is commonly used to calculate age
SUBTIMES(A, B)	Determines the difference between the two times.
FROMDAYS(INT)	Converts an integer number of days into a date value.

***Table 1.16**: Shows various date time functions and their explanation*

Question 11: How can you optimize queries in MYSQL?

Answer: If you come across a slow-running query, we can make use of **`EXPLAIN`** statement to get the details of why the query is slow.

The output of **`EXPLAIN`** statement will let us know the query and the corresponding indexes being used, thus enabling us to know on which field indexes can be created.

Few more generic optimization, technique involves using normalization, reducing the number of columns,

using the most appropriate data types and avoiding null values.

Specifying the required columns in the select clause is more preferred as compared to select * (* here denotes select all the columns).

Using select caching is also a preferred approach in the case of MYSQL tuning.

Conclusion

In this chapter, we learned about relational databases and structured query language and its application.

In the next chapter, we will learn the basics of NoSQL, its working, and classification

Chapter 2
NoSQL

Introduction

NoSQL is an important advancement in terms of technology, especially the database to deal with a huge amount of data which is generally known as big data.

Here, we will try to understand the basics of NoSQL, its classification, and its use cases.

Structure

In this chapter, we will discuss the following topics:

- Introduction to NoSQL
- Types of NoSQL databases
- Databases for NoSQL
- NoSQL working basics

Objectives

After reading this chapter, you will have a very good understanding of NoSQL both in terms of differences in terms of databases and differences around different types of NoSQL.

Introduction to NoSQL

Question 1: Explain NoSQL in a single line.

Answer: NoSQL is a type of database in which the mechanism for storage and retrieval is different from normal relational databases preferring vertical scaling over horizontal.

Question 2: Does NoSQL not support SQL queries?

Answer: Many NoSQL supports SQL-like query. The name NoSQL is generally associated to indicate it works differently as compared to SQL in terms of data distribution.

Types of NoSQL databases

Question 1: What are the different types of NoSQL databases?

Answer: These are the different types of NoSQL databases:

- Document databases
  - Stores data in JSON(Javascript object notation)/ BSON(Binary object notation) or XML(Extensible markup language) formats.
  - Data is stored in the form of documents which is equivalent to a database record.
  - Provides great flexibility for developers.
  - Examples: Couch DB, Mongo DB

- Key-value stores
  - Key-value store is a type of database in which data is arranged in the form of key-value pair.
  - We can look up the value by using a key.
  - Major applications are in the use of cache-like solutions.
  - Examples: Redis, Dynamo DB
- Column-oriented databases
  - Wide column store and column-oriented in this data is organized in the form of columns.
  - It can store a large volume of data and is much faster as compared to traditional databases.
  - It is majorly used in the case of recommendation engines for deduction and other types of data warehousing.
  - Examples: Cassandra, Hypertable
- Graph databases
  - Graph-oriented NoSQL are a special type of NoSQL databases in which data is arranged in the form of a graph.
  - As we know graph is a collection of edges and nodes; hence, this type of database finds application in the customer relationship model
  - Examples: Neo 4j, IBM Graph

In *Figure 2.1*, we can see illustrated the Taxonomy of NoSQL:

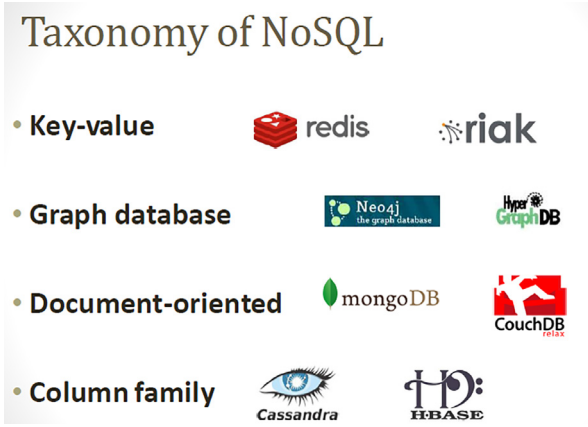


***Figure 2.1**: Shows different types of NoSQL and their corresponding implementations*

Question 2: What is the difference between document-oriented and key-value store?

Answer: A document-oriented store can store data which is mostly in the form of JSON and the lookup of the data can be done on any field. In the case of key-value pair, the lookup can happen only on the key. This is the major difference between these approaches.

Databases for NoSQL

Question 1: What does BASE mean?

Answer: BASE stands for Basically Available, Soft state, and eventually consistent. It is the principle required for the working of NoSQL.

- Basically Available: Availability is a major concern here that can be achieved by replicating or duplicating data on multiple nodes. This can impact consistency.
- Soft State: The value of data stored in nodes can change over a period. Hence, it is called a soft state.

- Eventually Consistent: Data replicated across various nodes may not be consistent immediately but will eventually be consistent when data across nodes are updated completely.

The following figure shows how reading and writing can happen parallelly with different consistencies:

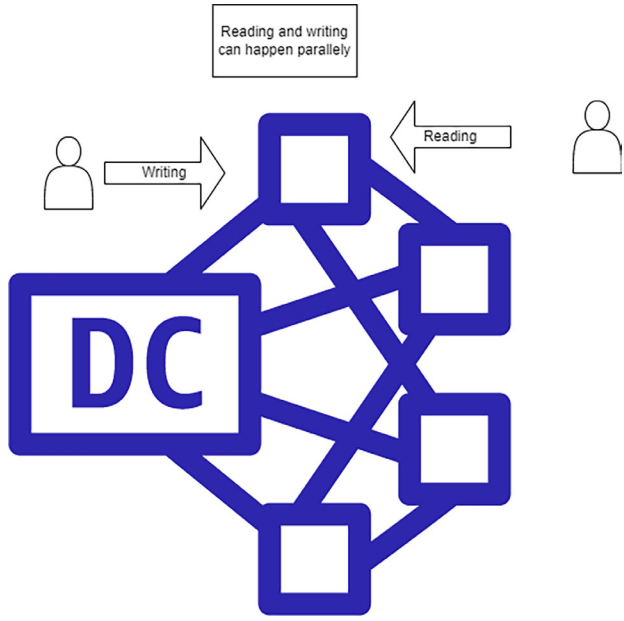


***Figure 2.2**: Reading and writing can happen parallelly affecting consistency (DC in the above figure stands for Datacenter)*

Question 2: Which one is better BASE or ACID properties?

Answer: ACID properties are typically used by databases to ensure atomicity, consistency, isolation, and durability features which are required by transactional-based applications. At the same time, there are applications which are more focused on analytical processing or processing typically on historical data. It is for such systems that we prefer using BASE principles.

So, whether we use ACID or BASE depends upon the nature of the application and the functionality it tries to achieve.

Whenever we prefer higher consistency, we will go for ACID-based relational database and whenever we can compromise on consistency and increased availability, we can go for BASE-based NoSQL

Question 3: Differentiate between vertical and horizontal scaling with reference to databases.

Answer: Vertical and horizontal scaling with reference to databases can be understood from the following table:

Vertical scaling	Horizontal scaling
Increases the capacity of a single machine or node	Capacity is increased by increasing the number of nodes
More RAM or more cores or processing units are added	More number of nodes is added
Data is stored in a single node	Data is partitioned across nodes
Preferred by relational databases	Preferred by NoSQL/Big data

***Table 2.1**: Compares vertical and horizontal scaling*

NoSQL working basics

Question 1: What is the meaning of Polygot persistence?

Answer: Polyglot persistence or hybrid or mixed language persistence is a type of persistence in which diverse technologies work together to achieve a common goal.

The aim of the Polyglot application is to develop a sound solution to a very complex problem keeping in mind higher availability and better results; this is what is used by NoSQL technologies.

Question 2: Explain the use of the Hash table.

Answer: A hash table is a type of table in which we can store data in the form of a key value. The key is calculated based on certain fields with the help of a hash function.

The Hash function will take certain field as input and calculate a hash value which will then be used to lookup the table to get the corresponding value.

This forms the base for many widespread or column-based NoSQL databases.

The following figure illustrates the working of hashing by using columns as input and calculating storage location:

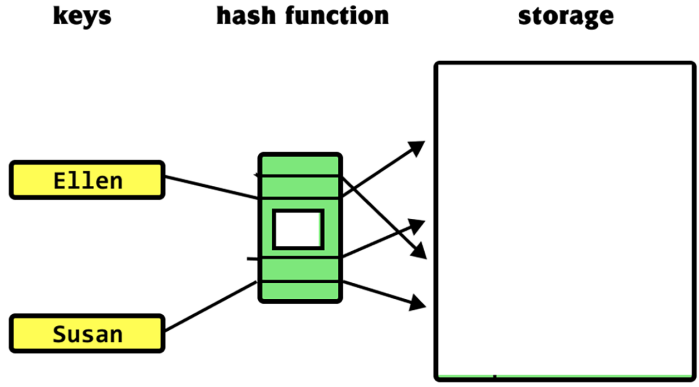


***Figure 2.3**: Working of Hashing by using columns as input and calculating storage location*

Question 3: What is the importance of CAP with respect to the NoSQL database?

Answer: Consistency, availability, and partition tolerance are three properties that are mainly used to control the nature of a database in a distributed environment.

There is a give and take when the value of one of the above properties is increased as compared to others. For example, if we select very high consistency, we cannot have very fast query performance as data is partitioned across multiple nodes and hence will take time to fetch them and check them for consistency

Question 4: Explain why there is a conflict between consistency and availability in the case of distributed systems.

Answer: In the case of distributed replicated systems, data needs to be replicated at different nodes. This replication will take time which will ultimately affect the consistency.

Imagine two nodes in which data is updated in one. While the update process is going on, a read happened

on the second node. This will cause an inconsistent result but will ultimately result in higher availability.

This is the reason why there is always a fight between consistency and availability in the case of distributed systems.

Question 5: What is the meaning of impedance mismatches with respect to the database?

Answer: An impedance mismatch occurs when the nature of the program in language and the database are not in sync.

Typically, programming languages deal with memory and store data in it while databases store the data in the form of files. An improper selection of these two units can cause an impedance mismatch. The high availability systems store the data in memory that is much faster to access as compared to consistent-based systems

Question 6: Have you heard about Big SQL?

Answer: Big SQL is a product developed by IBM which is very secure and easy to use. The solution provides vast analytical processing queries on a very large amount of big data. Passive parallel processing forms the background of Big SQL which allows faster results in a shorter time.

Question 7: What are aggregate-oriented databases?

Answer: Aggregate-oriented databases are the databases which generally sacrifice one of the acid-base properties to achieve higher speed.

The major focus of aggregate-oriented databases is to provide facilities in an aggregate way. They are mainly used for performing OLAP-based operations like drill down or roll up to see data at various levels. An example is a sale-based organization that can see sales at various levels like state, country, or continent.

Question 8: What is the meaning of a flexible data model with respect to NoSQL?

Answer: In a relational schema, data can be added to a row only when a column is created and added to the original table data model or schema. This is called a Stringent data model or tight schema.

An alternative to this approach is to allow a dynamic schema and NoSQL goes one level ahead wherein the data in each row of the table can be different from the previous one. This enables us to add a new field easily without affecting the other existing fields.

Question 9: What kind of data can be managed in a NoSQL?

Answer: NoSQL can handle a variety of data in different formats which can be broadly classified as structured, semi-structured, and unstructured. Within NoSQL, we can store unstructured data like images and videos. We can also store structured data like text files or text files with proper headers.

Semi-structured data like JSON and XML are also supported in NOSQL.

Question 10: What are the advantages of NoSQL over relational database?

Answer: NoSQL provides the following advantages over relational database. It provides dynamic schema, that is, the ability of schema to change dynamically on the fly.

- Cost can be reduced as NoSQL can be deployed on multiple nodes, which can be commodity hardware as compared to the database which requires powerful hardware. Semi-structured data are easily supported in NoSQL.
- Scaling horizontally is possible.
- Very high re-write throughout is possible.
- Data can be nested and even the nested data can be queried.

- The development cycle becomes faster as integration is easier.
- Analytical support for big data-like environments is possible.

Question 11: Are nested fields supported in NoSQL?

Answer: NoSQL can deal with a wide variety of data included in semi-structured data like JSON or XML which can support nesting. Since semi-structured data is supported in NoSQL it is not wrong to say that nested fields are supported.

Question 12: Explain how we know NoSQL can store audit logs on changes. In other words, explain how we can maintain the history of changes in NoSQL.

Answer: There are multiple ways of maintaining an audit log on change history in the case of NoSQL. The most commonly used is versioning in which we can maintain multiple versions of a given document or data.

An alternative to this approach is to capture the change only in the next version. Another way of handling this is by using audit logs as a part of data itself.

Question 13: When are relational databases preferred over NoSQL?

Answer: Relational databases are preferred over NoSQL when high consistency is expected from an application. For example, a net banking application would be expected to have a high consistency of data since it involves money transfers from one account to another. As the database supports ACID (Atomicity, Consistency, Isolation, Durability) properties, consistency becomes easily available. Consistency is also guaranteed on data involving more than one database table. (joins)

Another use case for relational database queries in which data needs to come immediately, also known as real-time queries, is possible only with the help of a relational database.

Question 14: Compare and contrast different types of NoSQL in terms of performance, stability, flexibility, and complexity.

Answer: The comparative understanding can be seen in the following table:

	Key Value	Column Oriented	Document	Graph
Performance	High	High	High	High
Scalability	High	High	Variable	Variable
Flexibility	High	Moderate	High	Moderate
Complexity	Low	Low	Low	Moderate

***Table 2.2**: Comparison of NoSQL with respect to various metrics*

Conclusion

The chapter introduced you to the world of NoSQL. We learned about different types of NoSQL as well as the principles on which they are based.

In the next chapter, we will deal with MongoDB which is a very famous and widely used NoSQL.

Join our book's Discord space

Join the book's Discord Workspace for Latest updates, Offers, Tech happenings around the world, New Release and Sessions with the Authors:

https://discord.bpbonline.com

Chapter 3
MongoDB

Introduction

MongoDB is one the most famous and adopted NoSQL.

It is adopted in a lot of projects involving user interfaces with technological frameworks like React and Angular. In this chapter, you will be introduced to various questions related to MongoDB.

Structure

The questions in this chapter will be on the following topics:

- Introduction to MongoDB
- Mongo architecture
- Data model
- Data types and query language support
- Replication
- Object Id and Advanced queries

- Advanced MongoDB tools

Objectives

In this chapter, we will deep dive into the architecture and working details of MongoDB. We will also study the nature of working in MongoDB as well as advanced queries.

Introduction to MongoDB

MongoDB is one of the famous document databases, founded in 2007. Dwight Merriman, Eliot Horowitz, and Kevin Ryan worked on it. It is very famous and used especially with famous UI technologies like React and Angular. When used with React, it forms **Mongo express react node** (**MERN**). When used with Angular, it forms **Mongo express angular node** (**MEAN**).

Mongo architecture

Mongo follows a distributed architecture and achieves this by using horizontal scaling.

This helps in meeting performance needs even when data increases. The process of achieving the same is known as sharding.

- Thus, the Mongo Db architecture consists of the following:

Shards

- It contains a part of the original data.
- High availability and data consistency is achieved by using shards.

Config servers

- For maintaining information on the cluster, we make use of the config server.
- This information stored in the config server is used by query routers to find the exact location of data.

Query Routers

- They are MongoDB instances to which a client application can connect.
- To support a heavy load, we can add more query routers.
- Query routers get incoming requests and target the corresponding shard from where data can be fetched.

The following figure shows different components of MongoDB:

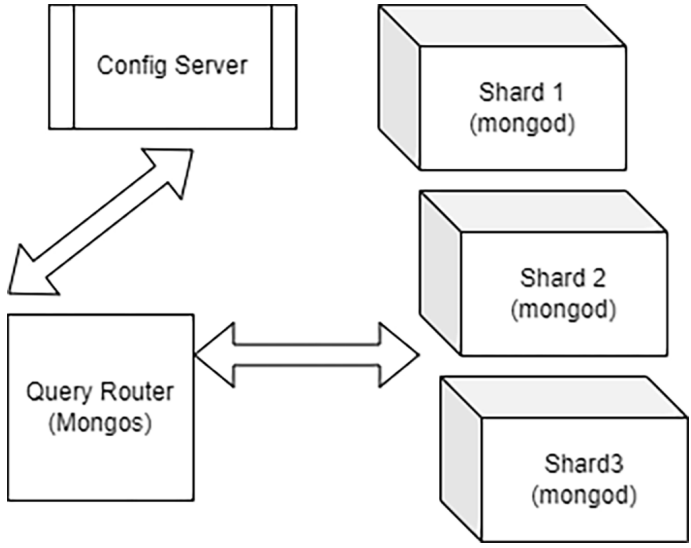


***Figure 3.1**: Shows MongoDB architecture components*

Data model

The data model in MongoDB consists of the following:

Database

- It consists of collections.
- It is equivalent to the schema in relational databases.

Collection

- It consists of records which are also known as documents.
- It is equivalent to a table in relational databases.

Document

- It consists of a row with fields.
- It is equivalent to a row/tuple in relational databases.

Index

- They help in faster searching just like relational databases.

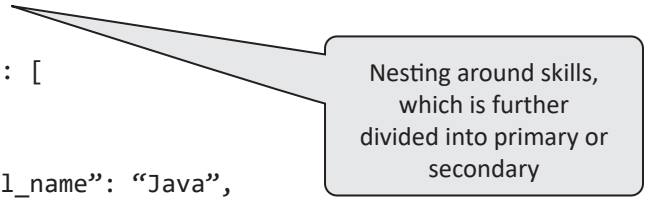
Further, a data model in MongoDB can be:

Embedded

This is nested data:

Example:

```
{
  “emp_name”: “Vishwa”,
  “skills”: [
    {
      “primary”: [
        {
          “skill_name”: “Java”,
          “skill_level”: “Expert”
        },
        {
          “skill_name”: “Python”,
          “skill_level”: “Expert”
        }
      ]
    },
    {
```



```
      “secondary”: [
        {
          “skill_name”: “Oracle”,
          “skill_level”: “Mid”
        },
        {
          “skill_name”: “Postgres”,
          “skill_level”: “Mid”
        }
      ]
    }
  ]
}
```

Normalized

In this case, a single document would be divided into multiple logically related documents.

For example, the Emp document is referred to as **`emp_id`** within each of the primary skill and secondary skill documents.

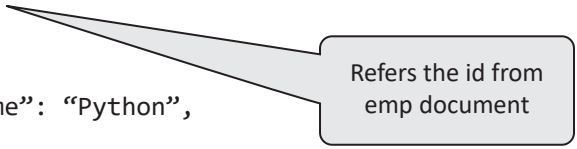
Emp document:

```
{
 id :”ObjectId1 “
  “emp_name”: “Vishwa”
}
```

Primary skill document

```
    {
          “skill_name”: “Java”,
```

```
        "skill_level": "Expert",
         "emp_id":"ObjectId1 "
      },
      {
        "skill_name": "Python",
        "skill_level": "Expert",
  "emp_id":"ObjectId1 "
      }
```



Secondary skills document:

```
      {
        "skill_name": "Oracle",
        "skill_level": "Mid",
     "emp_id":"ObjectId1 "
      },
      {
        "skill_name": "Postgres",
        "skill_level": "Mid",
  "emp_id":"ObjectId1 "
      }
```

Question 1: Why do people call MongoDB schema less?

Answer: Within MongoDB, we can have two rows with different sets of columns or values. Since the structure of data can be changed dynamically at run time, it is known as schema less.

Question 2: Can we map MongoDB topics with corresponding related databases?

Answer: The following table shows the mapping between MongoDB and RDBMS:

MongoDB	RDBMS
Database	Database
Collection	Table
Document Row	Document
Field Column	Field
Index	Index
Embedded document	Join
Shard	Partition

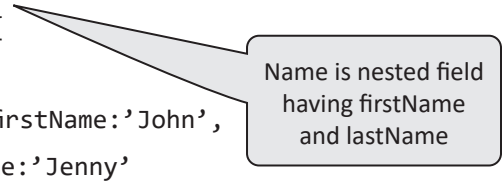
***Table 3.1**: Mapping between Mongo and RDBMS*

Question 3: Can MongoDB support a nested row?

Answer: A nested row is supported in MongoDB. For example, an employee can have nested data as follows:

```
{
    name:
            {
                    firstName:'John',
      lastName:'Jenny'

}
            }
```



Question 4: What is BSON?

Answer: They are a binary-encoded version of JSON. BSON stands for Binary Javascript Object Notation.

Question 5: Are BSON and JSON related?

Answer: BSON and JSON are very closely related.

- JSON is human and machine-readable
- BSON is readable only by machines
- JSON follows UTF-8
- BSON follows binary

- BSON supports more data types compared to JSON like raw binary data and date.
- Also, the representation of a number (that is, integer or float) is better in BSON.

Question 6: What role does Mongo shell play?

Answer: It is a Javascript interpreter using which we can connect to MongoDB running instances. Querying on MongoDB can be done with the help of Mongo shell.

Question 7: Does MongoDB follow Scale-in or Scale-out? Justify.

Answer: MongoDB is a NoSQL that supports Scale-out. Basically, the database system can be either Scale in or Scale-out. The Scale-in system means increase memory or the number of cores in the system. The Scale-out system means increasing the number of servers/ nodes of server for handling the load. Therefore, a scale-out system is cost-effective when using low-cost commodity hardware.

Question 8: What all indexing capabilities are provided by MongoDB?

Answer: It supports various types of indexes like compound and unique advanced indexes like geospatial and full text.

Datatypes and query language support

Following are some questions related to Datatypes and Query language support:

Question 1: What are the list datatypes supported by MongoDB?

Answer: The following table shows the list datatypes supported by MongoDB:

Data type	Example
Boolean	{“isAdult”true}
Number	{“age”:56}
String	{“name”:”Rajesh”}
Date	{“dateOfBirth”:new Date()}
Regular expression	{“findAll”: /vis /g}
Array	{“grades”:[“A’”,”A”,”A+”]
Embedded	{ name: { firstName:”John”, lastName:”’Jenny” } }
Binary data	{“pdf”:/00/x000}

***Table 3.2**: Data types supported and examples*

Question 2: Explain how we can create/insert data into MongoDB.

Answer: MongoDB provides two important methods to perform insert operations:

- **`insertOne`**: This allows the insertion of a single document.
- **`insertMany`**: This allows the insertion of an array of documents.

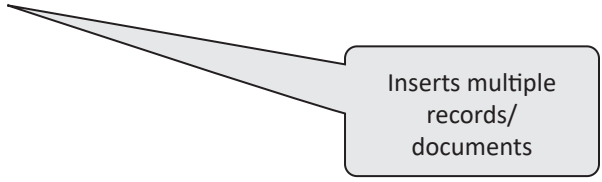
For example:

```
db.student.insertOne({‘rollNo’:1,
’name’:’Rajesh’})
```

Inserts a single record or document

```
db.student.insertMany([
{‘rollNo’:2,’name’:Ram},
```

```
{'rollNo':3,'name':Shyam}
])
```



Question 3: Explain how can you update a document/documents based on the condition in MongoDB.

Answer: Similar to insert queries, MongoDB provides **updateOne** to update a single document and **updateMany** to update multiple documents:

The syntax is as follows:

```
db.collection.updateOne(searchQuery,update-
FieldQuery)
```

```
db.collection.updateMany(searchQuery,update-
FieldQuery)
```

For example:

```
db.student.updateOne({'name':'rajesh'},{$set:{
'name':'Rajesh'}})
```

The above query searches for a document having the name as Rajesh and then updates the name to Rajesh. Since it is using **updateOne**, only the first matching record would be updated.

If we want to update all matching occurrences, we have to use the **updateMany** method.

Question 4: Explain how can you delete a document/documents based on conditions in MongoDB.

Answer: MongoDB provides the **deleteOne** and **deleteMany** methods to correspondingly delete single and multiple records, respectively

The syntax for delete is as follows:

```
db.collection.deleteOne(searchQuery)
```

```
db.collection.deleteMany(searchQuery)
```

For example:

```
db.student.deleteOne({'name':'rajesh'})
```

The preceding query searches for a document having the name as Rajesh and then deletes the same. Since it is using **`deleteOne,`** only the first matching record would be deleted.

If we want to delete all matching occurrences, we need to use the **`deleteMany`** method.

Question 5: How to select all the data present in a collection?

Answer: We can make use of the find command to do the same. The syntax will be like

```
db.collection.find().
```

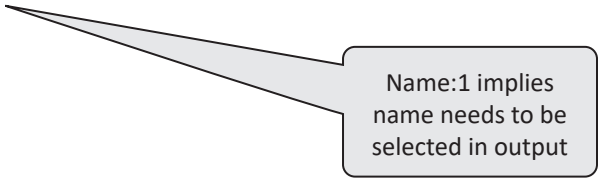
Question 6: Consider a collection with the name student having fields like age, name, and roll_no. How will I print only the name of all students?

Answer: The Find method can be used with projection where we can use specific fields which need to be selected and which need to be ignored. Fields which need to be included are given a value 1 and fields which need to be ignored are given the value 0:

```
Please note that _id field needs to be explicitly
curbed unlike other fields
```

So, the query for the above is:

```
db.student.find({},{name:1,_id:0})
```



Question 7: What is the use of indexes in MongoDB?

Answer: The major use of indexes is that it makes querying faster. This is achieved by storing a subset of data, such as it makes finding data faster.

An order list is maintained which is first looked up before going to the actual data.

Question 8: What is the use of geospatial indexes in MongoDB?

Answer: Geospatial data plays an important role in applications involving maps and geographical locations.

MongoDB supports indexing on geospatial data with the help of indexes.

Two major types of such indexes are supported:

- 2d to deal with a two-dimensional plane.
- 2dsphere: This supports two-dimensional objects like spheres. It allows using points, lines, and polygons.

An example of geospatial data is `[latitude,longitude]`.

A line can also be represented as a collection of points. For example, way: `[(10,2),(11,3),(12,4)…]`.

Question 9: Explain the use of a set modifier.

Answer: A set modifier is generally used in update-related queries to either add a new field in existing data or update the same.

Question 10: What is the aggregation framework in MongoDB?

Answer: The aggregation framework in MongoDB is basically used to compute results or derive values based on the existing value in documents.

It is basically equivalent to a group by operation in a relational database.

Question 11: Can you list the aggregate functions of MongoDB?

Answer: Following is a list of aggregate functions of MongoDB:

- AVG: It is used to find the average.
- Sum: It is used to find the sum.
- Min: It is used to find the minimum.
- Max: It is used to find the maximum.

- First: It is used to find the first value.
- Push: It pushes the data to the generated document.
- addTo Set: It adds to the document but does not duplicate it.
- Last: It is used to get the last record

Question 12: Let us say I want to find and update the values of data in one go; how can that be achieved?

Answer: This is possible with the help of the **`findAndModify`** function. The syntax of this function looks like this:

```
db.collection.findAndModify({
   query:{find query},
   update:{ update query
   }
})
```

The preceding method is preferred when we need to find and change the values in one go.

Question 13: What is the use of a pipeline?

Answer: A pipeline is mainly used in aggregation.

The pipeline allows defining stages, where, in each stage, we take an input process and correspondingly give the output. The stage can repeat itself in the pipeline as shown in the following figure:

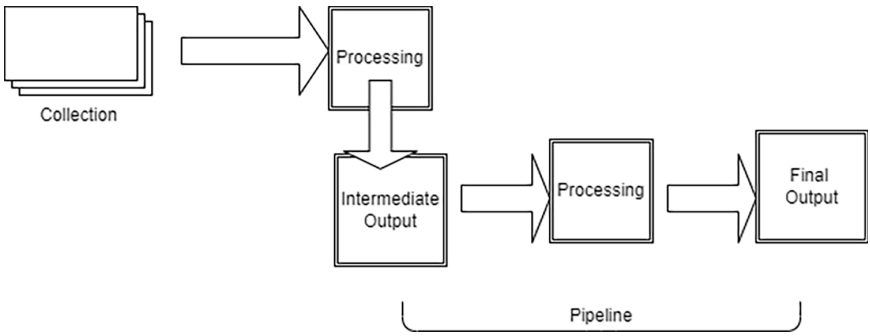


***Figure 3.2**: The pipeline concept*

Question 14: What are the various steps in the pipeline?

Answer: The three main steps in the case of the MongoDB pipeline are:

Steps	Usage
`$match`	To get records of our choice and remove unwanted
`$group`	To combine or group data based on fields
`$sort`	Arrange the data in ascending or descending order
`$project`	Used to select required fields
`$skip`	This is used for skipping forward
`$unwind`	Used to convert an array of objects into individual ones
`$limit`	Limit the number of records

***Table 3.3**: Aggregate functions and their description*

Replication

Question 1: What is the use of replication with respect to MongoDB?

Answer: As MongoDB can run on multiple servers, and servers can fail due to some reason, it is always better to store data at multiple places and this is achieved with the help of replication.

A replica set is a collection of nodes in which one of them acts as a primary node having data and another storing a copy of data which is known as a secondary node. On the crash of the primary node, the secondary node can be promoted to its place.

Question 2: Explain the process of replication in MongoDB.

Answer: Replication is done with the help of special sequential logs which are maintained by the primary server. Whenever the primary server performs any transaction, it stores the details of execution in a log with the name Oplog.

Secondary nodes regularly check for Oplog and whenever it finds a new change, it copies the same. Also, it copies the data files from the primary server or

another secondary server which has updated itself as shown in the following figure:

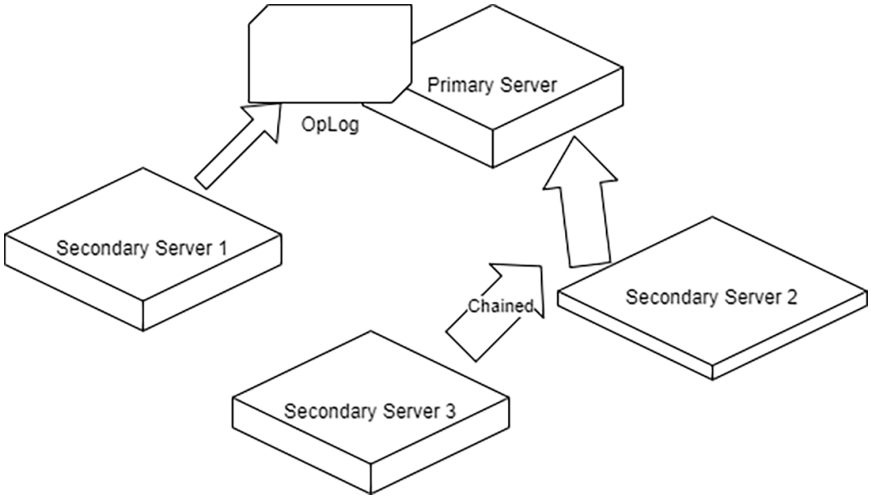


***Figure 3.3**: Replication in MongoDB*

Question 3: What is chained replication?

Answer: Servers can update themselves from another node which can be secondary and hence, they are called chained replication.

Question 4: Can we restrict collection size in MongoDB? If yes, how?

Answer: We can restrict collection size in MongoDB with the help of Capped collection.

We can indicate whether a collection is capped or not while creating the collection and passing the size.

Here is a sample syntax:

```
db.createCollection (name of collection,
{capped: Boolean, size: Number})
```

Question 5: How can we perform joins using MongoDB?

Answer: The **`$lookup`** operator is used for performing joins in MongoDB.

The syntax of look up is as follows:

```
$lookup:
```

```
{
  from: name of collection,
  localField: on which field from input
  should join be applied,
  foreignField: look up field on second
  collection,
  as: how you want the output
}
```

Question 6: Do we have to configure the size of the cache in MongoDB?

Answer: MongoDB automatically makes use of free space in memory to allocate space and hence, separate configuration is not required

Question 7: Your query in Production is running slow in MongoDB. Tell me what steps would you take for the same.

Answer: MongoDB provides the $explain and $hint operators to analyze and give areas of slowness.

Using the explain operator, we can see what all indexes are being used and how are they used:

Sample **`syntax: db.collection.find({}).explain()`**

Post the above step, we can make use of the $hint operator to force an index if that is not being used:

Sample **`syntax: db.collection.find().hind ({specify the column to use for indexing})`**

The preceding steps should help in the slowness scenario.

Question 8: Does MongoDB allow indexing on the sub field?

Answer: Indexing on the sub field is allowed in the case of MongoDB.

For example, we can use the following syntax to create an index in sub field:

```
db.collection.createIndex({'parentfield.
subfield'})
```

Question 9: Does MongoDB support transactions?

Answer: Atomicity Consistency Isolation and Durability are supported by MongoDB.

Object ID and advanced queries

Question 1: Explain **`ObjectId`** in MongoDB.

Answer: When we insert a document MongoDB **`creates _id`** which is a unique identifier for the document.

It is made up of 12 bytes with timestamps, random values, and a counter combination.

Question 2: Is it possible to **`create_id`** in Mongo?

Answer: The **`ObjectId`** function is provided by Mongo which allows us to give either hexadecimal value or integer value as input to **`generate _id`**.

Question 3: Consider the following data set and write a query to perform goals between 2 and 5.

```
{ _id: 1, goals:[1, 3,5,1,0,0]]
{ _id: 2, goals:[0, 0,2,0,0]]
```

Answer: Considering the collection name as a sample, we will use **`elemMatch`** to perform a given task:

```
db.sample.find( { goals:{ $elemMatch: { $gt: 2,
$lt: 5} } } );
```

Question 4: Will you be able to write a customized sort function in MongoDB?

Answer: Consider the following function which will sort data based on salary:

```
db.eval(function() {
return db.scratch.find().toArray().
sort(function(d1, d2) {
return d1.salary - d2.salary
})
});
```

For executing the above function, we will be using the `db.eval()` function.

Question 5: Consider students having fields like:

```
name,age,grade,address,subject,teacher,
classname.
```

How will you write a query to show all fields other than the `classname`?

Answer: Projection would be useful in such cases. Using projection, we can include all fields other than `classname:`

```
db.student.find( { }, { classname: 0 } )
```

Question 6: Using MongoDB will you be able to match an array with elements like 'r' and 'g'?

Answer: MongoDB allows matching arrays.

For example, consider the student collection having a field like `liked_color`, then we can write:

```
db.student.find( { liked_color: [“r”, “g”] } )
```

The preceding command will get elements matching the sequence of r and g.

If sequence matching is not required, we can change it to:

```
db.student.find( { tags: { $all: [“r”, “g”] } } )
```

Question 7: Consider a football collection having an array field with name goals. Will you be able to get records which have goals size equal to 3?

Answer: This is possible by using the $size operator:

```
db.football.find( { “goals”: { $size: 3 } } )
```

The preceding command will return records having goals size as 3.

Question 8: Is it possible to check whether a field is present in the given document?

Answer: `$exists` is used for such cases.

For example, the following command will return only those students who have a name field:

```
db.students.find({"student.
name":{"$exists":true}})
```

Alternatively, we can use null check as it can give better performance:

```
db.students.find({ "student.name": {  $ne: null
} })
```

Advanced MongoDB tools

Question 1: Do you know what **`ObjectId`** is internally made of?

Answer: **`ObjectId`** is made up of a combination of machine ID, process ID, Byte incremented counter, and Timestamp.

Question 2: Give the usage of **`pretty()`**.

Answer: It is used to display the document in a well-formatted way.

Question 3: How can we perform a backup/restore operation?

Answer: The **`mongodump`** command makes a backup of the database.

`mongorestore` is used for restoring the data from the backup.

Question 4: When you don't specify the output directory in the **`mongodump`** command, where will it store the backup?

Answer: The backup is stored in the **`/bin/dump`** folder.

Question 5: Can we back up data in any folder of our choice?

Answer: This is made possible by using the **`--out`** option provided by **`mongodump.`**

Question 6: Why is the save method used in MongoDB?

Answer: Many times, we may want to replace the existing document with a new one. This is made possible with the help of the save method.

Question 7: Does MongoDB provide any tool for knowing database statistics?

Answer: This is possible with the tool **`mongostat`**. Using **`mongostat`**, you can get information like memory used.

Question 8: What type of lock is used by MongoDB?

Answer: MongoDB makes use of read-write locks which enables multiple users to connect at the same time and ensure atomicity and isolation.

Question 9: Explain the significance of MongoDB journaling.

Answer: MongoDB journaling enables write-related information to be stored from memory to disc so that in case of failures and recovery, the same can be used to get data back again.

Question 10: Does MongoDB allows storing images, videos, and audio files?

Answer: MongoDB makes it possible with the help of **`gridfs`**. This is achieved by using **`mongofiles`** tools with the **`gridfs`** option.

```
mongofiles.exe -d gridfs put image
```

The preceding command is used to put an image in **`gridfs`**.

Once stored data will contain **`files_id`** as the primary key.

Reading in chunks is supported by grid fs.

Question 11: How does MongoDB ensure data security?

Answer: It achieves this by using encoding so that only the actual process would be able to receive and understand the data. Also, data is secured at the storage level which is known as storage encryption.

Question 12: Is MongoDB supported on Cloud?

Answer: Yes, MongoDB is supported on the cloud. For e.g. Amazon Web service has MongoDB offering called

Azure, and Atlas and IBM have a cloud offering for MongoDB.

Question 13: What is the use of a covered query?

Answer: Covered queries are ones in which indexes are being used during a query resulting in faster execution of a query.

Question 14: How can you perform a text search using MongoDB?

Answer: The text index is used for performing a search on the text field.

We will have to first create a text index using:

```
db.collection.createIndex({field:"text"})
```

Post creating the index, we will need to use it while searching with the help of $text:

```
db.collection.find({$text:{$search:"text to search"}}).pretty()
```

Question 15: Does indexing have any disadvantages?

Answer: Generally, indexing is talked about as having the advantage of making a query faster but that comes with a price of more RAM usage as well as storage overhead for storing them.

A few MongoDB regular expression cheat sheets are shown in the following table:

Command	Explanation
`db.emp.find({name:{$regex:"Vishwa"}})`	Search for a document containing Vishwa.
`db.emp.find({name:{$regex:/Vishwa/}})`	Search for a document containing Vishwa.
`db.emp.find({name:{$regex:/Vishwa/}},$options:"$i")`	Search for a document containing Vishwa with a case-insensitive nature.
`db.emp.find({holidays:{$regex:"sun"}})`	Search for a document where the holidays array contains the word sun.

Command	Explanation
`db.emp.find({name:{$regex: /^Vi/ }})`	Search for a document where the name starts with Vi.
`db.emp.find({name:{$regex: / wa$/ }})`	Search for a document where the name is starting with wa.
`db.emp.find({ $and:[ {name: /^Vi/ }, {name: /wa$/ } ] })`	Search for documents starting with Vi and ending with wa.

***Table 3.4**: Find Commands using regex in MongoDB and explanation*

Conclusion

By the end of this chapter, we have seen in depth the principles, working, and architecture of MongoDB. We also saw how to insert, update, and delete data in MongoDB.

In the next chapter, we will look at Cassandra which is another very famous NoSQL.

Chapter 4
Cassandra

Introduction

Cassandra is one the most famous NoSQL which is used by many multi-national companies to manage Big data. Influenced by the Big data paradigm, Cassandra provides an easy-to-use SQL-based approach towards handling Big data.

Structure

In this chapter, we will cover the following topics:

- History of Cassandra
- Architecture and design principle
- What is Cassandra?
- Important terms in Cassandra
- Design questions in Cassandra
- Partitioning and Tokens

- Keyspace and Tables
- Protocols used in Cassandra/internal working

Objectives

After reading this chapter, you will get an advanced understanding of the history of Cassandra, Cassandra architecture, Cassandra query language, replication and consistency, and the best practices for using Cassandra.

History of Cassandra

Cassandra was developed in 2008 to aid the search feature of Facebook. The inventors of Cassandra were Avinash Lakshmanan and Prashant Malik. It became an incubator in 2009. At present, we have various versions of Cassandra prominent being versions 3.x and 4.x

Architecture and design principle

This topic forms an important part of Cassandra's interview questions. We will see questions related to the working of Cassandra along with its design principle.

Architecture and design principles will also be covered.

What is Cassandra?

Cassandra is a distributed peer-to-peer NoSQL database that is predominately used to get faster writes. Cassandra is itself influenced by two other NoSQL databases, Amazon DynamoDB (Distribution aspect), and Google Big data (data modeling aspects).

Features offered by Cassandra

Cassandra offers the following features:

- Distributed nature
- Scalable and available

- Ability to tune, that is, increase or decrease consistency based on need
- No single point of failure
- Ability to store a wide variety of data ranging from numbers and text, all the way to complicated data types like Avro/binary data

Tip: The following acronym will help us remember these features:

C

A: Available

S: Scalable

S

A: Ability to store a wide variety of data

N: No single point of failure

D: Distributed

R

A: Ability to tune

The physical architecture of Cassandra

The physical architecture of Cassandra consists of many running instances (which are also known as nodes) connected to each other.

There is no concept of master-slave (primary/replica) in Cassandra, hence they are known as peer-to-peer. The nodes are connected to form a ring. Nodes can exist in the same or different locations. For example, in a 4-node Cassandra system, we can plan two nodes in Atlanta and two in Brazil. Hence, we can say that ring in the case of Cassandra can span across data centers (location). The following figure shows the physical architecture of Cassandra:

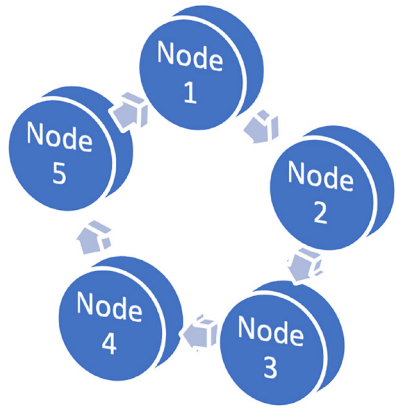


Figure 4.1: *Physical architecture of Cassandra shows peer-to-peer connected nodes*

In *Figure 4.2*, we can see the nodes shared across different data centers:

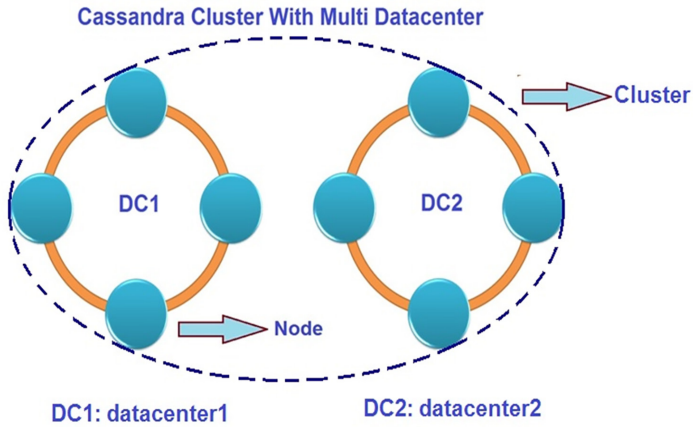


Figure 4.2: *Nodes shared across different data centers*

Logical architecture of Cassandra

The outer component of the logical architecture is the keyspace. This keyspace can be defined as a collection of column families. From a **Relational Database Management System** (**RDBMS**) perspective, we can say that the keyspace is equivalent to the database. Column families are the second one which consist of an ordered collection of rows. From an RDBMS perspective, it is equivalent a to table. In

the following figure, the relationship between various components involved in Cassandra is shown:

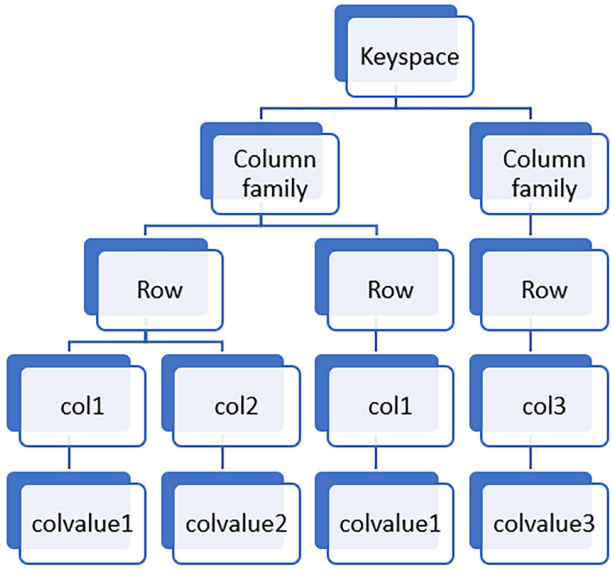


Figure 4.3: *Shows the relationship of keyspace, column family, row and columns*

Important terms in Cassandra

Question 1: How many keyspaces can be present with reference to Cassandra?

Answer: There is no fixed count on the number of keyspaces in Cassandra. As many keyspaces are created by the user, so many keyspaces will exist.

Question 2: Show the correspondence between Cassandra and the Relational Database System.

Answer: The following table shows some of the differences between Cassandra and the Relational Database Management System:

RDBMS	CASSANDRA
Database	Keyspace
Table	Column family / Table

RDBMS	CASSANDRA
Column name	Column key
Column value	Column value

Table 4.1: *Correspondence of Cassandra and RDBMS*

Question 3: What is tunable consistency in Cassandra?

Answer: Generally, NoSQL systems have varying levels of consistency as per application requirements.

Cassandra is an eventually consistent system rather than immediate consistent (like RDBMS).

Cassandra allows us to vary the overall expected consistency. This is what we call tunable consistency.

Question 4: How is tunable consistency obtained?

Answer: This is achieved by setting the expected consistency while starting a session.

For setting the consistency in CSQLsh, we make use of the **`CONSISTENCY`** command in which we can set the consistency level of our choice. Cassandra allows setting different levels of consistency for read and write.

Question 5: Define the replication factor.

Answer: Replication is the process of duplicating data across multiple nodes to ensure better availability of data in case of failures.

The replication factor indicates the number of copies of the same data in the Cassandra cluster.

Question 6: Explain a few consistency levels you are aware of.

Answer: These are a few consistency levels:

- ONE is the lowest level of consistency where data is persisted on only one node.
- ALL is the highest level of consistency where data is persisted on all nodes.

- QUORUM is one of the most discussed consistency, where data has to be persisted in n nodes where n is given as (replication factor/2)+1
- QUORUM consistency can be local or each. In a local case, data consistency is maintained in the same data center while in each case, data consistency is maintained across all data centers.

Question 7: If you need to express tunable consistency as a statement/ formula how would you do it?

Answer: Let n be the desired consistency, and then the results should be returned by minimum n nodes.

Question 8: If the replication factor is 10, how many copies of data would exist in a Cassandra cluster?

Answer: 10 copies of data will exist in a given cluster.

Question 9: What do you think is the difference between database modeling and Cassandra modeling?

Answer: In database modelling, we create a table with columns and the corresponding data type of columns. In Cassandra modeling along with the above details, it also indicates how data will get partitioned across nodes.

Question 10: Explain different types of keys with reference to Cassandra.

Answer: The different types of keys with reference to Cassandra are explained here:

- **Partition keys**: The entire working of Cassandra is based on how can divide a given data across nodes for faster processing. To achieve the same, it makes use of the Partition key which indicates columns using which data can be distributed across nodes. This can be observed in the following figure:

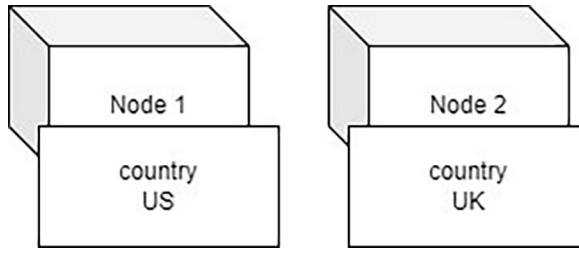


Figure 4.4: *Diagram shows how the partition key works*

In the preceding figure, we can see that data is divided into two different nodes based on the country column, with data having the country as the US going in node 1 and that of the UK going in the second one.

- **Clustering keys**: Once the partition key is selected and data is divided across nodes, we may also want to arrange data in such a way within each partition for faster access. This is done with the help of clustering keys. Thus, the clustering key is used within the node as shown in the following figure:

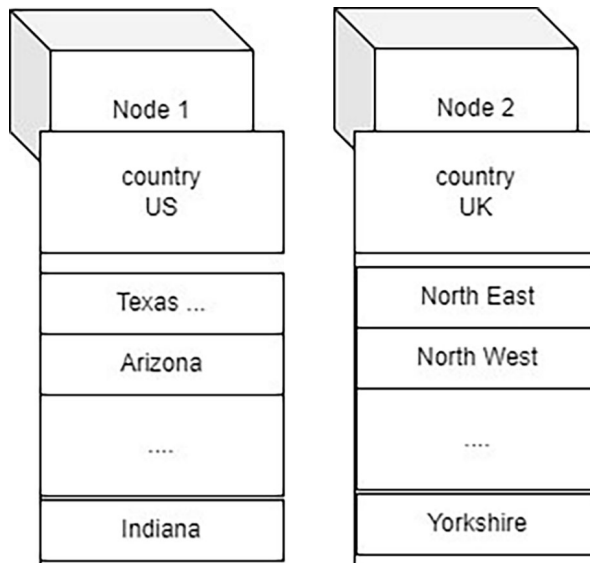


Figure 4.5: *How the clustering key works*

As shown in the preceding figure, each data in individual nodes are further divided based on state.

The combination of the partition key and clustering key is known as the primary key with the partition key always being listed first.

Question 11: Can more than one key form a part of the partition key?

Answer: Cassandra allows more than one key to be a part of the partition and clustering key. In simple words, they can be composite in nature.

Design questions in Cassandra

Question 1: Design a table employee in Cassandra with **`emp_id`** and **`emp_name`** being the partition key. Assume other attributes as required.

Answer: The syntax for creating a table with a partition key is:

```
create table table_name(
column1 datatype,
column2 datatype,
....
column n datatype,
PRIMARY KEY ((partition keys),clustering keys)
```

In the preceding snippet, partition and clustering keys are specified with the help of the primary key option. Within the primary key option, the first part is portioning keys followed by clustering keys. We use brackets to indicate partition keys if there is more than one.

So, the solution to the request problem is as follows:

```
create table employee
(
emp_id int,
emp_name text,
age int,
salary double,
```

```
PRIMARY KEY ((emp_id,emp_name))
)
```

Question 2: Can the clustering key be none? Give any example for the same.

Answer: The clustering key can be none. In the following example of the employee table, we do not have the clustering key:

```
create table employee
(
emp_id int,
emp_name text,
age int,
salary double,
PRIMARY KEY ((emp_id,emp_name))
)
```

Question 3: In the following snippet, can you identify which is the partition key and which is the clustering key?

```
CREATE TABLE sample(
   col1 timestamp,
   col2 text,
   col3 text,
   col4 text,
   PRIMARY KEY (col1, col2)
   )
```

Answer: As per the preceding snippet, **`col1`** is the partition key and **`col2`** is the clustering key.

Question 4: In the following snippet can you identify which is the partition key and which is the clustering key?

```
CREATE TABLE sample(
   col1 timestamp,
   col2 text,
```

```
    col3 text,
    col4 text,
    PRIMARY KEY ((col1,col2))
    )
```

Answer: As per the preceding snippet, **col1** and **col2** combine to form a partition key. There is no clustering key.

Question 5: In the following snippet, can you identify which is the partition key and which is the clustering key?

```
CREATE TABLE sample(
    col1 timestamp,
    col2 text,
    col3 text,
    col4 text,
    PRIMARY KEY ((col1,col2),col3)
    )
```

Answer: As per the preceding snippet, **col1** and **col2** combine to form a partition key and col3 forms the clustering key.

Partitioning and tokens

Question 1: What is a partitioning function?

Answer: The entire work of Cassandra revolves around the partition key. It decides how data gets divided across nodes. To do so, Cassandra internally makes use of a function named partitioning function. The output of the partitioning function is a token which gets mapped to nodes. This is illustrated in the following figure:

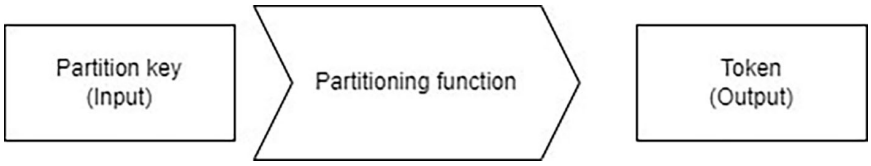


Figure 4.6: *The partitioning function takes a key as input and generates a token as output*

Question 2: Explain the use of a token.

Answer: Cassandra needs to divide data into partition which is then stored in n nodes. It does this with the help of a partitioning function which generates a number called a token.

Each node which is part of the Cassandra cluster is assigned a token range. Post-generation of the token, based on the value of the token, is assigned to a node having that token range. In the following figure, you can see how the token-based approach will work with nodes in Cassandra:

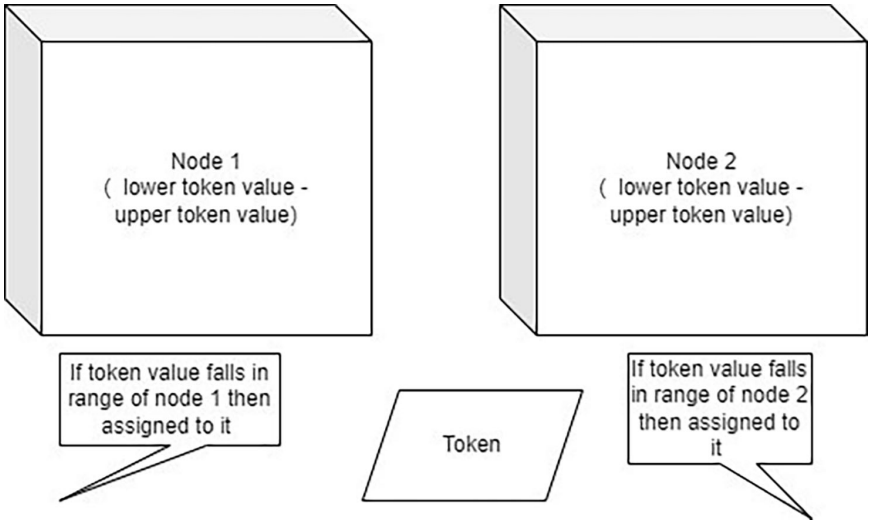


Figure 4.7: *Tokens assigned based on range assigned*

Question 3: What is the general range of a token?

Answer: -2^63 to +2^63 -1

Question 4: What is an unbounded partition?

Answer: An unbounded partition is a partition which will grow in an indefinite volume with respect to time. It is not a desired behavior in Cassandra, as the desired behavior is more partitions whenever we have more data.

Question 5: What is partition skew?

Answer: Partition skew is a behavior in which data goes to one partition more as compared to another partition. As a result, one partition seems more used than others resulting in the hotspot.

Question 6: Why is partition skew caused and how to avoid it?

Answer: Partition skew is caused mostly due to improper keys selected as part of the partition key causing more data in some partitions which tends to be used more

One way of avoiding this is to introduce a new dummy key which will take different values, as a part of the partition key which will force the partition to happen.

Question 7: Considering the following data, please draw the distribution for a 4-node Cassandra system. Consider the city as the partition key. The following figure shows a sample city, state, and country:

City	State	Country
Chandler	AZ	US
Los Angles	CA	US
Bangor	WALES	UK
Munich	BA	GE

Figure 4.8*: Show the data structure to be used for the question*

Answer: The following figure shows that each city would be given an individual node:

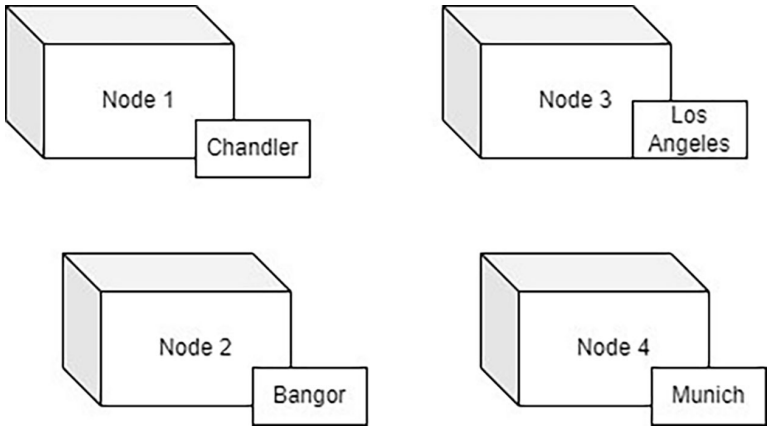


Figure 4.9: *Distribution of data across nodes as per city partition key*

Question 8: Considering the following data, please draw the distribution for a 4-node Cassandra system. Consider the state as the partition key.

City	State	Country
Chandler	AZ	US
Avondale	AZ	US
Bangor	WALES	UK
Munich	BA	GE

Figure 4.10: *Shown above is the sample data to be used for the above question*

Answer: By partition based on the state, we can see that both the rows about the state AZ has landed in the same node.

The following figure shows the partition mechanism for various states in the United States:

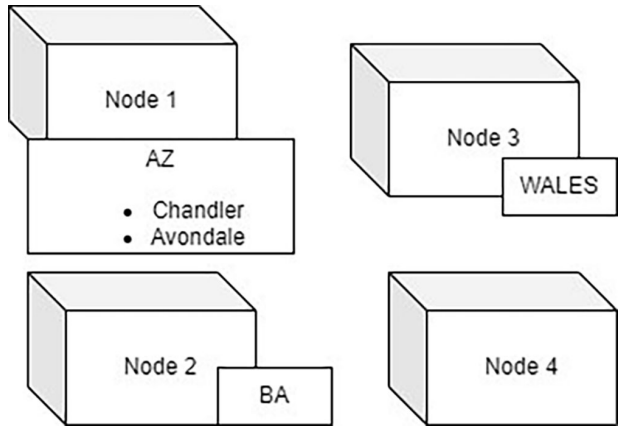


Figure 4.11: *Distribution of data as per state*

Question 9: Considering the following data, as shown in *Figure 4.12*, please draw the distribution for a 4-node Cassandra system. Consider country as the partition key.

City	State	Country
Chandler	AZ	US
Avondale	AZ	US
Chicago	Ilionis	US
San Dieogo	CA	US

Figure 4.12: *Shown above is the sample data to be used for the above question*

Answer: Since we are performing a partition based on the country and it has only one value, all the data would go into a single node. This would be an example of partition skew if all the data belongs to the country US only.

The following figure shows the distribution of data belonging to the United States and the skew effect:

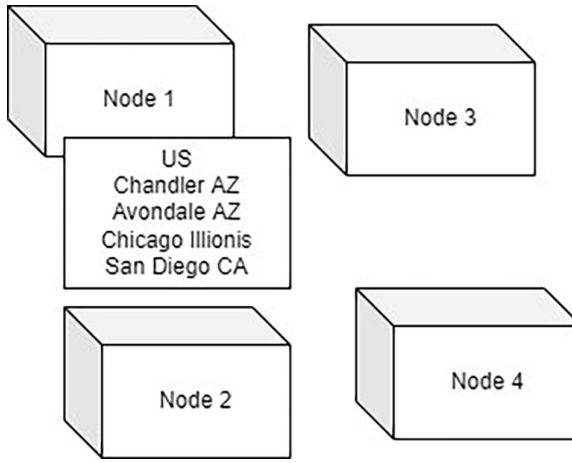


Figure 4.13: *Showing the distribution of data belonging to country US*

Keyspace and tables

Question 1: How can we create a keyspace in Cassandra?

Answer: We can create a keyspace in Cassandra with the help of the create keyspace command. This command takes various parameters such as the replication factor to model the behavior in terms of several copies of data:

```
CREATE KEYSPACE IF NOT EXISTS bpb_keyspace
 WITH REPLICATION = { 'class' : 'SimpleStrategy',
 'replication_factor' : 1 };
```

In the preceding command, we are creating a keyspace with the name **`bpb_keyspace`** having the replication factor of one and strategy as simple (mainly used for single node).

Question 2: How can you create a keyspace in a clustered Cassandra?

Answer: We can create a keyspace in Cassandra with the help of the create keyspace command. This command takes various parameters such as the replication factor to model the behavior in terms of several copies of data:

To create a multi nodekeyspace, we can use the following command

```
CREATE KEYSPACE bpb_multi_node WITH replication
=
 {'class' : 'NetworkTopologyStrategy',
 'datacenter1' : 5,'datacenter2' : 4};
```

The preceding command creates a multi-node keyspace which has a replication factor of 5 in data center 1 and a replication factor of 4 in data center 2. We will use the NetworkTopology strategy as it is a clustered environment.

Question 3: Can we drop a keyspace?

Answer: A keyspace can be dropped with the help of the drop command:

```
drop keyspace bpb_keyspace;
```

The preceding command drops a keyspace with the name **bpb_keyspace**.

Question 4: How to create a table in Cassandra?

Answer: The syntax for creating a table is nearly the same as the SQL counterpart. We specify the columns and corresponding data types.

The primary key which is a combination of the partitioning key and clustering key forms an important part of Cassandra create table structure:

```
CREATE TABLE emp(
emp_id int PRIMARY KEY,
emp_name text,
emp_city text
   );
```

The preceding query creates a table with the name emp having emp_id as the partition key.

The same create statement can be created as follows:

```
CREATE TABLE emp(
emp_idint ,
emp_name text,
emp_city text,
    PRIMARY KEY (emp_id)
   );
```

The output is the same as the preceding query.

If we want to create a table with **emp_id** being the partition key, and **emp_city** as the clustering key, we can do it as follows:

```
CREATE TABLE emp(
emp_idint ,
emp_name text,
emp_city text,
    PRIMARY KEY ((emp_id),emp_city)
   );
```

Question 5: Consider a student table. How can you add the email field which is of type text?

Answer: Alter table can be used for this purpose:

```
ALTER TABLE student ADD email text;
```

Question 6: Consider a table named student having the email field. Write a query to drop the column.

Answer: Alter table can be used for this purpose:

```
ALTER TABLE student DROP email;
```

Question 7: What is the use of list data type?

Answer: List data type falls under the collection data types. It provides the ability to store more than one value for a given column in a single row. Duplicates are allowed in a list.

Question 8: What is the use of a set data type?

Answer: Set data type falls under the collection data types. It provides the ability to store more than one value for a given column in a single row. It does not allow duplicate values to be stored.

Question 9: What is the use of map data type?

Answer: Map data type falls under the collection data type. It provides the ability to store data in the key-value format. Any mapping like the country name to currency mapping can be done using a map. For example, on a country map, we can store the US as key and Dollars as value.

Question 10: Assume we have a candidate table which stores various details of a candidate for a job opening. We are interested in storing skills acquired by the candidate. Which data type should we use?

Answer: We can prefer using the **set** data type for storing the above relationship since duplicates are not possible in skills.

Also, if we wanted to store skills acquired and their rating, we could have made use of a map in which skills would be the key and rating would be the value.

Question 11: Consider we are making an application to store a movie and its rating. Which data type would fit the best for this?

Answer: In Cassandra, wherever mapping is involved, a map is preferred as a data type. Here, we can create a map having the key as the movie and value as a list in which the rating obtained by the movie would be stored:

`map<text,list<text>> review` is the way to go ahead for this.

The following figure shows how we can display a user interface for a movie review app:

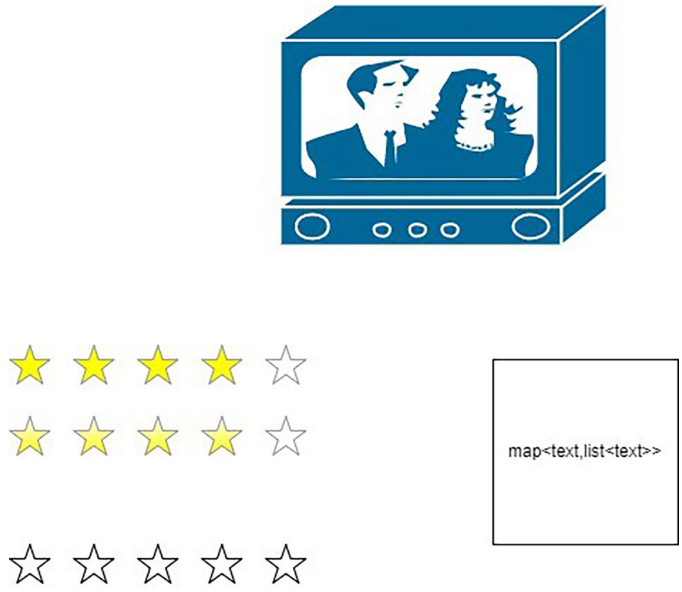


Figure 4.14: *Movies and corresponding rating with a map as the corresponding column*

Question 12: What is the use of a frozen keyword?

Answer: A frozen keyword when used with other data types provides what we call in programming language, immutable. Individual values within frozen data cannot be modified / updated. It will require a complete reinsert. It is mostly preferred for data which would not change regularly and would act as a reference.

Question 13: Does Cassandra help in representing a query in JSON format? If yes, what is the advantage?

Answer: Yes, Cassandra allows sending queries in JSON format. The advantage is the ability to integrate with any application supporting JSON.

In the following example, we are inserting data in a table using JSON format, where the keys represent column names and the values represent corresponding column values:

```
insert into emp JSON '{"emp_id": "123", "name":
"John","salary": 5000}';
```

Question 14: Does Cassandra provide any function to convert the data to JSON directly?

Answer: The **`toJson`** function is available in Cassandra which provides the requested behavior.

```
select toJson(emp_name) from emp;
```

In the preceding query, the **`toJson`** function will convert the column **`emp_name`** to JSON.

Alternatively, we can also use:

```
Select JSON(emp_id,emp_name,salary )from emp;
```

All three columns would be converted to JSON.

Question 15: What are the rules associated with filtering in the case of Cassandra?

Answer: Filtering or the where clause in Cassandra can be used with the partition key followed by the clustering key. We can also add a query on indexed columns.

Cassandra also allows filtering on other columns which are not a part of the above keys by using the allow filtering option. This comes with many disadvantages, especially, performance.

Question 16: What do you think are the differences between querying in the case of RDMS and Cassandra?

Answer: In the case of RDMS, we first design the table to hold data and then correspondingly generate queries to fetch them. Thus, we first generate a model and subsequently generate queries.

In the case of Cassandra, we first design the query and then think of a model. Everything is driven by query expectations and duplication of data is permissible.

Question 17: Can we get the token from given the partition key in Cassandra?

Answer: Cassandra provides the token function which takes the partition key as input and gives the token used for partitioning as output:

```
SELECT emp_id, token(emp_id) FROM emp;
```

In the preceding query, **emp_id** is the partition key and we can use the token function to get the value of the actual token to be used.

Question 18: Does Cassandra support limiting?

Answer: Cassandra supports the limit option which can be used to limit the number of records returned after firing a query:

```
Select emp_id from emp limit 2;
```

As we use limit 2, the output would show only two records.

Question 19: Does Cassandra support in clause?

Answer: Cassandra does support in clause:

```
Select emp_id, emp_name from emp were emp_id in
(2,3,4)
```

The preceding query allows selecting employees with **emp_id** as 2,3 and 4.

Question 20: Is in clause preferred in Cassandra?

Answer: As a general practice, multiple calls to the Cassandra server are preferred as compared to single in clause.

When firing a query with **in clause**, the coordinator of the query as they do more work in getting data from different nodes and then sending them back. This can cause huge load around it.

Another disadvantage associated with in clause is failed queries. In a failed query, the entire query involving in clause must be tried again causing additional load.

Question 21: Explain different types of partitioners supported in Cassandra.

Answer: The different types of partitioners supported in Cassandra are:

- Murmur3Partitioner
  - It is the default
  - Can be used with vnodes

  - Creates a 64-bit hashing function
  - Makes use of the **`MurmurHash`** function which is a simple and powerful hashing technique for lookup
  - Range -2^63 to +2^63-1
- RandomPartitioner
  - Was the default used in earlier versions of Cassandra
  - Distributes data evenly
  - Makes use of the MD5 algorithm
  - Range is 0 to 2^127 -1.
- ByteOrderedPartitioner
  - Orders rows lexically using key bytes
  - Supports ordered partitions
  - Using this is considered an anti-pattern

Protocols used in Cassandra/ internal working

In this section, we will discuss protocols used in Cassandra's internal working.

Question 1: Explain gossip protocol.

Answer: Just like in the real world how people come to know about each other by talking/gossiping, even in peer-to-peer node architecture, each node must identify other nodes. This is done with the help of gossip in which each node shares state information with other nodes. The shared information is known as a gossip message. Each message has an associated version to check which is the latest message. The gossip message contains state information and at the same time information about nodes with which they have gossiped earlier

Question 2: What is the use of a seed node?

Answer: The seed node helps newly joined nodes in the

cluster to start the gossip process. Thus, it helps in bootstrapping.

Question 3: How does failure detection happen in Cassandra?

Answer: Failure detection happens in Cassandra with the help of gossip protocol. During gossip, each node maintains a list of nodes to which it can connect. Cassandra uses an accrual mechanism in which it determines the threshold time for each node to reach other nodes considering various factors like delay, and network congestion.

Question 4: What is the use of snitches?

Answer: Typically, a cluster of Cassandra can span across multiple regions and data centers. Snitches help us in determining which nodes to select for the act of reading and writing.

Question 5: Explain different types of snitches.

Answer: The different types of Snitches are:

- Dynamic snitching

  Determines the best based on the performance of nodes
- SimpleSnitch

  Used on a single data center
- RackInferringSnitch

  Uses Internet protocol address to determine
- PropertyFileSnitch

  Uses rack and data center information
- GossipingPropertyFileSnitch

  Uses Gossip algorithm
- Ec2Snitch

  Used with Amazon EC2 in a single region
- Ec2MultiRegionSnitch

  Used with Amazon EC2 in multiple regions

- GoogleCloudSnitch

  Used with Google Cloud
- CloudstackSnitch

  Used with cloud stack

Question 6: Should distributed joins be preferred in Cassandra or not?

Answer: Distributed joins are not preferred in Cassandra like NoSQL. Instead of preferring joins, use a new table with all the columns included. Denormalization is not discouraged in NoSQL, thus duplication is more.

Question 7: What type of storage engine does Cassandra follow?

Answer: The storage engine followed by Cassandra is a Log structured merge tree. Cassandra stores all inserts and updates in memory and at regular intervals, the changes are flushed from memory onto the disk.

Question 8: Explain the process of write operation in Cassandra.

Answer: The writing operation in Cassandra makes use of various log files and helper tables. The names of the log files and corresponding tables used are listed below.

Commit log: A log file which stores every write made to Cassandra

- Memtable: Writes to Cassandra are made to write back in-memory cache called Memtable.

  Whenever a threshold (**`commitlog_total_space_in_mb`** or **`memtable_heap_space_in_mb`**) set is reached, data is flushed from memtable.
- SSTables: There are files stored on the grid.

  SSTables consist of components, as shown in the following table:

Component	Description
`Data.db`	Data of SSTable
`Index.db`	Primary index

Component	Description
`Filter.db/Bloom filter`	A filter which helps to check whether data exists on memtable before checking for disk
`CompressionInfo.db`	Details of compression
`Statistics.db`	Stores statistical information of sstable
`Summary.db`	Stores summary information of partition index
`TOC.txt`	Stores information of all components

***Table 4.2**: Show the components of SSTables*

The following figure shows how the write operation works in Cassandra:

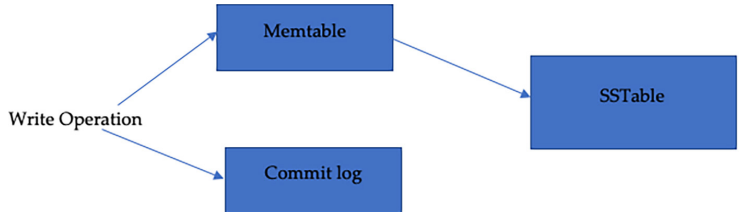


***Figure 4.15:** Write flow in Cassandra*

Conclusion

In this chapter, we focused our questions on Cassandra. We learned how Cassandra's working is different as compared to Relational databases and how there is a thinking paradigm shift in modelling and designing Cassandra database.

In the next chapter, we will focus on another NOSQL, Redis.

Join our book's Discord space

Join the book's Discord Workspace for Latest updates, Offers, Tech happenings around the world, New Release and Sessions with the Authors:

https://discord.bpbonline.com

Chapter 5
Redis

Introduction

In any software-related project, speed/performance is a very important metric. To enhance the speed of the application, we need to use various caching technologies. One of the distributed caching NoSQL solutions available is Redis. In this chapter, we will take a look at Redis, both from architecture and usage perspectives. We will also study various commands which can be used concerning Redis system.

Structure

In this chapter, we will discuss the following topics:

- Introduction to Redis
- Replication
- Data type and working
- Commands in Redis

Objectives

By the end of this chapter, you will have learned the use of Redis as a solution to caching solution along with its architecture and commands used.

Introduction to Redis

This section will discuss questions related to Redis:

Question 1: What is Redis?

Answer: Redis is NOSQL which falls under the category of key value-based storage.

One of the major use cases of Redis is as a cache since data in Redis can be easily stored in memory.

It supports persistence and works on various operating systems, thus making it portable. It can also be integrated with a wide variety of programming languages.

Redis is written in C.

Question 2: What is the full form of Redis?

Answer: It stands for Remote Dictionary service.

Here, Dictionary stands for key value-based services, and Remote stands for the remote machine.

Question 3: Can you list some features of Redis?

Answer: These are some of the features of Redis:

- It is fast and can execute around 100000 queries per second.
- Setting up is quite straightforward.
- Works primarily with memory RAM, hence fast.
- Operations are atomic in nature, that is, they can support use cases revolving around the counter.
- Supports most common data structures supported in most programming languages like list, set, dictionary, and so on.

- Works on multiple operating systems.

Replication

This section will discuss questions related to Replication:

Question 1: Can you explain the different modes in which Redis works?

Answer: Redis can work as a single node mode or, as a one master one replica mode.

For **higher availability** (**HA**) applications, we can have more than one master and replica mode. This is also known as a clustered mode. Some HA modes are shown in the following figure:

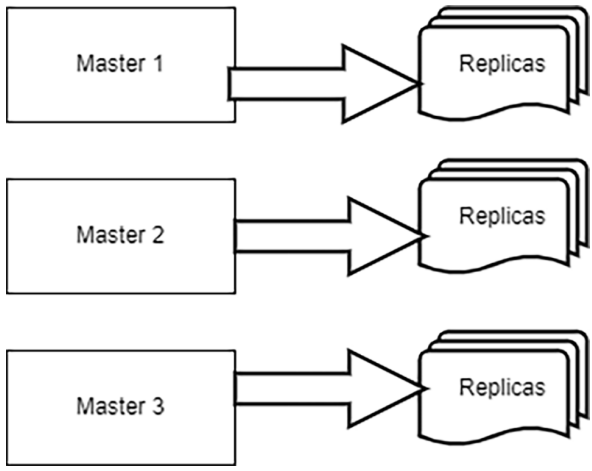


Figure 5.1*: HA modes of Redis*

Question 2: Does Redis support replication?

Answer: Replication is supported in Redis by using master replica architecture. It is one of the most common features expected in a NOSQL environment, hence, Redis also supports it.

Question 3: Can you compare RDBMS and Redis?

Answer: The following table shows a comparison between RDBMS and Redis:

RDBMS	Redis
Belongs to the SQL family	Belong to the NoSQL family
Follows the table structure	Follows the key-value structure
Makes use of secondary memory	Makes use of primary memory
Speed is considered slow when compared to Redis	Speed is faster as compared to a database
Used for data which is big in nature	Used for data which is small in nature
The frequency of data usage is less as compared to Redis	The frequency of data usage is more as compared to RDBMS

***Table 5.1**: Difference between RDBMS and Redis*

Data types and working

This section will discuss questions related to data types and their working:

Question 1: Explain which data types are supported in Redis.

Answer: The data types supported in Redis are, Strings, Hashes, Lists, Sets, and Sorted Sets.

Question 2: Differentiate between the Memory cache and Redis.

Answer: The following table shows the difference between the Memory cache and Redis:

Memory cache	Redis
Does not support native data types readily	Supports native data types readily
Does not support persistence readily	Does support persistence based on files
Clustering is not got out of the box	Support multiple levels of persistence
Supports vertical scaling	Supports horizontal scaling
Data replication is not supported out of the box	Data replication is readily supported
Publisher Subscriber model not readily supported	Publisher Subscriber model readily supported

***Table 5.2**: Difference between the Memory cache and Redis*

Question 3: Can durability be guaranteed in the case of Redis?

Answer: Redis are subject to system failures and can lose data in case of such events.

There is always a possibility of losing data in such cases and hence, durability cannot be guaranteed.

Question 4: What is a snapshot concerning Redis?

Answer: Redis intermediately maintains the copy of data from memory into a disk which is known as a snapshot. These snapshots are helpful in the recovery process to get data back as illustrated in the following screenshot:

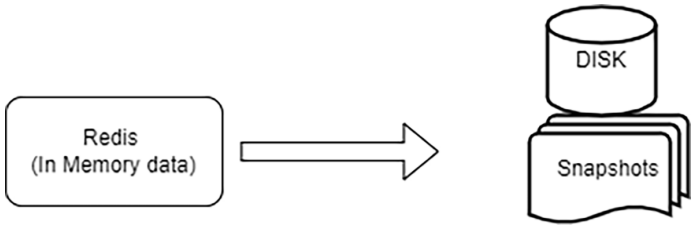


Figure 5.2*: Shows Redis maintaining snapshots in disk*

Question 5: Explain when will Redis put data on disk.

Answer: Redis can save data automatically after intervals based on policies. It can also be made to explicitly save data by calling the BGSAVE function.

Redis also flushes out data into the disk at the time of shutting down.

Question 6: Differentiate between **Redis Database file** (**RDB**) and **Append only file** (**AOF**)?

Answer: The following table shows the difference between RDB and AOF:

Redis Database File (RDB)	Append only file (AOF)
Stands for relational database	Stands for Append only file
Takes a snapshot of data from memory and stores it in disk with extension .rdb	Takes write operations and records them in a file

Redis Database File (RDB)	Append only file (AOF)
SAVE or BGSAVE command can be used	Controlled by the `appendfilesync` mode in the `redis.config` file
Restoration is faster	Restoration can be slow

***Table 5.3**: Shows the difference between RDB and AOF*

Question 7: What is the use of an append log file in Redis?

Answer: An append log file maintains all the write operations which are done on Redis.

As a result, the entire data set can be recovered by just looking at the append log.

Question 8: What are the different policies on the append log associated with Redis?

Answer: Following are the policies associated with Redis:

- **`no`**: When to write to file is decided by the operating system
- **`everysec`**: Write to a file after every second
- **`always`**: Each write to a file is done immediately

Question 9: Why do people call Redis non-true persistent?

Answer: There is no guarantee on whether persistence happens in Redis. This is called best-effort service and is dependent upon external factors like memory and disk space. For example, if while persisting the data on disk, the disk becomes full, then persistence cannot happen and hence data cannot be restored.

Question 10: Someone told me Redis supports transactions. Is that correct?

Answer: Transaction supports implies that either the transaction is committed or not. Data should not be in an inconsistent state.

To support transactions, Redis supports the following serialization of a group of commands which would be executed. It achieves the same with the help of commands like **`EXEC`**, **`MULTI`**, **`WATCH`**, and **`DISCARD`**.

Question 11: What is pipelining from Redis' perspective?

Answer: The collection of commands from the client side and sending them together is known as pipelining. This results in better network usage.

Question 12: Is Redis multi-threaded or single-threaded? How can it handle multiple clients?

Answer: Redis is single-threaded. Each request sent from multi-threaded systems is handled individually and hence, atomicity is guaranteed.

Question 13: Are replication and sharding the same?

Answer: Redis is a key value-based NoSQL. Sharding means dividing data based on some criteria or some condition. From Redis' perspective, we can divide only on the basics of key causing certain keys to be present in certain nodes as compared to others.

Replication, on the other hand, is maintaining the entire copy of data in a different node. Redis has in build support for replication.

The difference between both lies in the granularity of data that we store on different nodes

The difference is illustrated in the following figure:

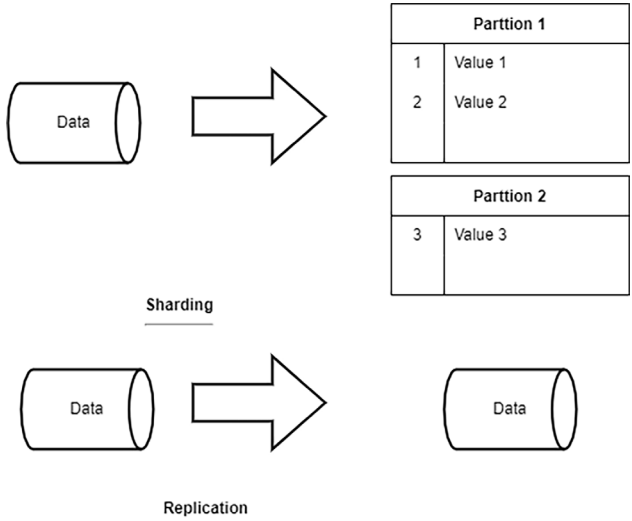


***Figure 5.3**: Difference between Sharding and Replication from Redis perspective*

Question 14: What type of bound is Redis?

Answer: Redis can be bound only by memory or network since Redis makes use of memory to store data. Redis servers can be connected over a network to send data to the client.

Redis does not do a lot of processing involving the CPU.

Question 15: Are rollbacks supported in Redis?

Answer: Rollback is not supported in Redis. This is because Redis is typically meant for faster access; hence, it provides that by not providing certain features like rollback

Question 16: What can we use to connect to the Redis server on a remote host?

Answer: We can make use of redis-cli.

The following command can help:

```
redis-cli -h <ip address> -p <port> -a password
```

Note: For security reasons, the password is not provided directly. The setting of an environment variable using `REDISCLIENT_AUTH` is generally preferred.

Question 18: How can I set a simple key in Redis?

Answer: We can achieve this with the help of the SET command:

```
SET UK UnitedKingdom
```

In the above command, UK is the key and UnitedKingdom is the value.

Commands in Redis

The following section deals with questions related to Redis:

Question 1: How can we set the expiry to a key?

Answer: The following methods can be used to set expiry in either seconds/milliseconds in the Unix format timestamp:

EXPIRE key seconds

PEXPIRE key milliseconds

EXPIREAT key timestamp

PEXPIREAT key milliseconds-timestamp

Question 2: How to get the remaining time to live for a key?

Answer: The **PTTL** command give ttl in milliseconds.

Similarly, the **TTL** command gives ttl in seconds.

Question 3: How to get the data type of a key stored in Redis?

Answer: The **TYPE** command followed by the key can be used to get the data type.

Question 4: Can a key be renamed in Redis?

Answer: The **RENAME** command can be used for this purpose where we can have the original key followed by the new key name.

Question 5: Is it possible to get keys by some pattern?

Answer: Yes, we can make use of the **KEYS** commands with patterns.

For example, **KEYS vishwa*** will list down all the keys starting with vishwa.

Question 6: Is it possible to get the length of the value stored in a key in Redis?

Answer: The **STRLEN** command followed by the key can be used for this purpose.

Question 7: How can you maintain a counter in Redis?

Answer: The **INCR** command is used to increase the value of an integer stored in a key by one.

The **DECR** command is used to decrease the value of an integer stored in a key by one.

Question 8: If I want to decrement or increment by 3 in Redis, is that possible?

Answer: **INCRBY** command is used to increase the value by a given factor.

INCRBY key factor.

DECRBY command performs the same.

Question 9: Is there any way in which we can get and set data (key-value) in one go?

Answer: It is possible by using the GETSET method which will set the new value and at the same time return the old value. GETSET is used to set a key-value pair and at the same time retrieve the value of old data

Question 10: If the values for more than one key are required how can it be obtained?

Answer: We can achieve this with the help of the MGET command in which we can specify more than one key:

MGET KEY1, KEY2….KEYN

Question 11: What is the use of the SETEX command?

Answer: This is used to set a string value in a key and sets the expiry of the same:

SETEX name, that is, **90 vishwa**

In the preceding command, we set the key name to value vishwa and set the expiry time to 90 seconds.

Question 12: What are **HASHES** in terms of Redis?

Answer: **HASHES** are mappers between key and value which is a string.

Real-world objects can be easily represented with the help of **HASHES.**

Up to 4 billion key-value pairs can be stored in Redis.

If you see the following statements, we are using **HASHES** with key students and correspondingly having attributes like name and age:

```
HMSET student name “vishwa nara”

HMSET student age  50
```

We can retrieve all the values stored by using the **HMGET** student.

Question 13: How can we delete a Hash key?

Answer: We can delete a Hash key by using the HDEL command followed by the key to delete.

Question 14: How to check whether the Hash field exists?

Answer: **`HEXISTS`** followed by key, further followed by field can be used.

Question 15: How can we list all the fields in Hashes?

Answer: **`HKEYS`** is used in all the fields in Hashes.

Question 16: Can we set multiple hash fields and values in one go?

Answer: We can achieve the same by using **`HMSET`**, for example.

`HMSET` Key field1 value1 field2 value2

Question 17: Can we get all the values in Hashes?

Answer: We can get all the values by using the **`HVALS`** function which will give us all the values in the given hash.

Question 18: What are lists concerning Redis?

Answer: Lists are collection which maintains the order of insertion and contains strings.

The addition of elements to the list is possible only from the front and back of the list.

Question 19: Is it possible to remove the first element or do we wait till the first element is available and then remove it?

Answer: **`BLPOP`** key is used for removing the element at the start or waiting till they are available.

Question 20: Which command is used to get a list element by using indexes?

Answer: **`LINDEX`** is the command which can fetch from a list based on the index.

Question 21: How to get the length of the list?

Answer: **`LLEN`** is used for this purpose.

Question 22: If I just want to remove the first element from the list, what command to use?

Answer: We make use of the **`LPOP`** command to remove the first element. In the same way, to remove from the end, we make use of the **`RPOP`** command.

Question 23: How to remove elements from the list?

Answer: **`LREM`** is used for this purpose. **`LREM <listkey> <valuetodelete>`**

Question 24: How can we append to the list?

Answer: We can make use of the **`RPUSH`** command.

Question 25: What is the difference between a Set and a Sorted set, in the case of Redis?

Answer: The Sorted set has the score that is used to arrange the data in order from smallest to largest.

The Set data does not have any order.

Question 26: Write a simple code for performing transactions in the case of Redis.

Answer: Here is a simple code snippet to perform such an operation:

```
MULTI
SET user vishwa
INCR users
EXEC
```

In the preceding code, **`SET user`** and **`INCR`** will happen in one transaction.

Question 27: If I want to flush all the previous queued commands in **`MULTI`**, what should we use?

Answer: The **`DISCARD`** command can be used in the above scenario since it flushes out all the previous queued commands.

Question 28: What is the use of the **WATCH** command with respect to **MULTI/EXEC** commands?

Answer: The **WATCH** command is used for performing a transaction based on some condition.

Thus, using **WATCH**, we can get the Check and Set functionality:

```
WATCH numberofusers
count = GET numberofusers
count = count + 1
MULTI
SET numberofusers $count
EXEC
```

In the preceding code, operations pertaining to **numberofusers** would be allowed only when no one else is modifying it. In case of conflicts, the transaction is aborted.

Question 29: If you want to stop the **WATCH** command executed on a key what can we do?

Answer: We can make use of the **UNWATCH** command to do that.

Question 30: How to remove all the keys from all databases?

Answer: We can make use of the **FLUSHALL** command.

Question 31: What if I want to flush all the keys in a given database?

Answer: We can make use of the **FLUSHDB** command.

Question 32: How can we get all the requests made in real-time?

Answer: The **MONITOR** command is useful to get all the requests received at a particular point.

Question 33: How can we get the number of keys in a given database?

Answer: We can make use of the **DBSIZE** command.

Question 34: Can we kill a connection in Redis?

Answer: Killing a connection is possible by using the **`CLIENT KILL`** command in which we give the IP address followed by the port.

Question 35: Can we get a list of all clients connected to Redis?

Answer: The **`CLIENT LIST`** command is used for getting all the clients connected to Redis.

Question 36: In production, we are seeing a lot of commands issued by a particular client, how can we pause them for some time?

Answer: The **`CLIENT PAUSE`** command can be used to stop receiving all the commands for a given time interval.

Question 37: Which command is given to get general statistics of the server?

Answer: The **`INFO`** command is used to get statistics like memory, number of clients, persistence, and so on.

Question 38: How can we perform load testing using Redis?

Answer: Redis provides the **`redis-benchmark`** tool using which we can execute various commands and correspondingly get performance.

We can use various run-time parameters like -t for a set of commands, and -n to know how many commands to run.

We can also give input from a CSV file.

Question 39: How can you set up security on Redis?

Answer: Security can be easily set up on Redis by setting the value of the password in config.

For this, you can make use of the **`requirepass`** value.

By setting the value of **`requirepass,`** you can avoid others connecting without a password.

To connect with a password, we can make use of the **`AUTH`** command followed by the **`requirepass`** value.

Conclusion

In this chapter, we learned the basics and working of Redis along with its corresponding commands which are used in day-to-day projects.

In the next chapter, we will focus on HBase.

Join our book's Discord space

Join the book's Discord Workspace for Latest updates, Offers, Tech happenings around the world, New Release and Sessions with the Authors:

https://discord.bpbonline.com

CHAPTER 6
HBase

Introduction

HBase finds an important place in the tools in the case of Hadoop infrastructure. HBase is a column-oriented NoSQL database which sits on top of Hadoop. It provides random real-time read/write access to Big data.

Structure

In this chapter, we will discuss the following topics:

- Tombstone
- HBase configuration and consistency
- Data operations
- Debugging
- Difference between HBase and other technologies

Objectives

After reading this chapter, you will learn basic to advance level questions on HBase. You will also learn the conceptual to practical level questions on HBase.

Introduction to HBase

Question 1: What are the key components of HBase?

Answer: The region server helps serve a group of regions, where a single table can be divided into multiple regions.

- HMaster helps in coordination among region servers.
- ZooKeeper is the coordinator and helps maintain the server state.

Question 2: When will one prefer HBase?

Answer: It is preferred when:

- Data consistency is required
- Reading and writing are to be done on large tables.

Question 3: Which method is used to read data in HBase?

Answer: The **`get()`** method is used for this purpose.

Question 4: What are column families?

Answer: The column family is a collection of columns. The row is a collection of column families.

Question 5: When should we use a decorating filter?

Answer: It is used for getting additional control over returned data. Here are some examples:

- **`Skip`** filter allows excluding the entire row.
- **`WhileMatchFilter`** abandons the entire scan.
- **`FilterList`** allows more than one filter to be applied.
- **`CustomFilter`** is based on custom requirements.

Question 6: List down some of the important commands in HBase.

Answer: The following table shows the description of the commands in HBase:

Command	Description
`Put`	Used to insert a cell value in a specified row
`Get`	Used for fetching data
`Delete`	Used for deleting cell value
`Deleteall`	Used for deleting all the cells in a given row
`Scan`	Scans and returns the table data
`Count`	Returns the count and number of rows in the table
`Truncate`	Drops and creates a specified table

*Table **6.1:** Commands in HBase with description*

Tombstone

A very important concept related to how deletes happen in HBase and its overall impact with respect to the system is discussed.

Question 1: What is the use of a tombstone?

Answer: Whenever a column or cell is deleted in HBase, it is not deleted immediately; instead, a tombstone marker is used. The tombstone hides all the deleted data from being accessed.

Question 2: When is data deleted from the tombstone?

Answer: It is deleted when major compaction takes place. All the expired data is deleted during this phase, and all smaller data is combined to form a bigger file. This is done to ensure there is no adverse impact on the system. Also, running the compaction can be done in the background, thus ensuring other activities can continue.

Question 3: List down some of the tombstone markers you know.

Answer: Some tombstone markers are:

- Version marker: A particular version of the column is marked for deletion.

- Column marker: All versions of columns are marked for deletion.
- Family marker: The entire column family is marked for deletion.

HBase configuration and consistency

Question 1: What is the default value of the HBase block size?

Answer: The default value is 64KB and is configured per column family level.

Question 2: How can we find the current HBase user?

Answer: We can find this using the **`whoami`** command.

Question 3: What is the use of the Memstore-Local Allocation Buffer?

Answer: It is also known as MSLAB and allows you to set memory at the target region level as compared to the entire heap.

Question 4: How does HBase ensure consistency and removal of corrupt blocks?

Answer: This is done with the help of a tool named HbaseFsck. It allows you to check the consistency at the region level and allows the removal of corrupted blocks which do not match the consistency criteria.

Question 5: Where is metadata information stored in HBase?

Answer: It is stored in tables named catalog.

Question 6: What is the use of compaction in HBase?

Answer: In compaction, files can be combined mainly to save storage and improve disk seeks. The two types of compactions are:

- Minor:
  - o Here, smaller HFiles are selected and combined to form a bigger HFile.

  - Reading a particular row will require more disk seeks if minor compaction was not in action.
- Major:
  - Merging and recommitting of StoreFiles of a region into a single StoreFile.
  - It removes and deletes the expired version.
  - Controlled by the **`hbase.hregion.majorcompaction`** property.

Question 7: What is the use of the **`hbase.hregion.majorcompaction.jitter`** property?

Answer: This ensures that major compaction does not run simultaneously in all nodes.

Question 8: What role does **`HColumnDescriptor`** play?

Answer: **`HColumnDescriptor`** allows us to store information about the column family like compression, number of versions, etc.

Question 9: Can you list the various filters available in HBase?

Answer: The following table shows the filters used in HBase:

Filter name	Description
`ColumnPrefixFilter`	Returns a key-value pair that matches with a column prefix.
`MultipleColumnPrefixFilter`	Takes a list of column prefixes that are matched.
`TimestampsFilter`	Takes a list of timestamp and returns key-value pair matching with the timestamp.
`ColumnPaginationFilter`	Returns a limited number of columns after the offset number.
`SingleColumnValueFilter`	This is mainly used for comparison using a value. Based on the column and the comparator, the matching row will be returned.
`RowFilter`	As the name suggests, the comparison is done at row using the row key.

Filter name	Description
`QualifierFilter`	Make use of the qualifier name for comparison.
`ColumnRangeFilter`	Returns columns that are under the range from minimum to maximum.
`PrefixFilter`	Makes use of prefixes at the row level.
`ColumnCountGetFilter`	Returns a limit on columns.
`InclusiveStopFilter`	Returns all the key values till the matching row key is matched.
`DependentColumnFilter`	It takes a column and the qualifier band returns all the keys that match the timestamp.
`KeyOnlyFilter`	Returns only the key part of the key-value pair.
`FamilyFilter`	Returns all the key values based on the family name.
`CustomFilter`	Customized filter as per requirement.

***Table 6.2**: Commands in HBase with description*

Question 10: What does HBase do on failures?

Answer: Generally, HBase handles failures with the help of a write-ahead log (WAL). A write is supposed to be completed whenever an entry is made in WAL. In case of failures and crashes, the logs are used to recover.

Question 11: How does versioning work in HBase?

Answer: Any operation on the cell-like insert, update, and delete cause a new version of the cell. An upper limit is put on the version count, after which some versions of cells are dropped. The default cell version is 3.

Question 12: Explain the use of the Bloom filter.

Answer: It allows us to search whether the HFile contains certain row or row values. In the absence of this filter, the block index needs to be scanned, which can be less efficient because of various row drops.

Data operations

Question 1: What is the key data structure in HBase?

Answer: Row and column are the key data structures.

Question 2: Explain the steps involved in reading data.

Answer: The following steps are involved in reading data:

1. It looks for the Row cell in the Block cache.
2. If the above does not get data, it moves to Memstore.
3. Finally, it uses the bloom filter.

Question 3: Can you write code to create a schema in HBase?

Answer: This is the code to create a schema in HBase:

```
Configuration config = HBaseConfiguration.create();
HBaseAdmin hbAdmin = new HBaseAdmin(conf);
HTableDescriptor tableDesc= new
HTableDescriptor(TableName.valueOf(“student”));

tableDesc.addFamily(new
HColumnDescriptor(“present”));
tableDesc.addFamily(new
HColumnDescriptor(“old”));

hbAdmin .createTable(tableDesc);
```

Question 4: Can you write code to modify the schema?

Answer: Use this code to modify the schema:

```
HBaseAdmin hbAdmin = new HBaseAdmin(conf);
String tableName = “Student”;
hbAdmin .disableTable(tableName);
hbAdmin .modifyColumn(tableName, columFamily);
hbAdmin .enableTable(tableName);
```

Question 5: Are you aware of an algorithm that focuses on the decompression speed?

Answer: Lempel-Ziv-Oberhumer, also known as LZO, is used for this purpose.

Question 6: How can we open a connection in HBase?

Answer: It can be done using the following:

```
Configuration config= HBaseConfiguration.create();
```

Question 7: What is S3?

Answer: S3 (service provided by AWS) is one of the storage services used by HBase.

Question 8: How many modes can HBase run?

Answer: It can run in a standalone mode, where HDFS is not used, and distributed mode, where HDFS is used. A pseudo-distributed mode is also supported, where everything runs on a single host.

Question 9: How can we list down surgery tools?

Answer: The tool command is used for this purpose. We can display a list of surgery tools using the tool command.

For example: the running tool command can point to surgery tools like assign, close region, compact, flush, major compact, and so on.

Question 10: What is the rule for deleting a table in Hbase?

Answer: First, disable and then delete.

For example, to delete a table named user, we will need to disable it using the disable user command and then delete it using the delete user command.

Question 11: How can we see the description of the table?

Answer: It can be done with the describe command. We can see the description at the table level using the describe command.

For example, running the describe command, the user will print details like data encoding, bloom filter, deleted cells, and data block encoding.

Question 12: What is the smallest unit in HBase?

Answer: The cell is the smallest unit in HBase.

Question 13: What is the use of HRegionServer?

Answer: It is used for managing and serving regions.

Question 14: Can HFile be directly accessed without HBase?

Answer: Yes. It can be done with the help of the main method of HFile.

Question 15: What type of data can be stored in HBase?

Answer: Any data that can be converted into bytes can be stored in Hbase.

Question 16: List down the hierarchy of tables in HBase.

Answer: A single table consists of one or more column families; each column family consists of one or more rows, and a single row consists of one or more columns. The following figure illustrates the relationship between tables, column families, columns, and rows:

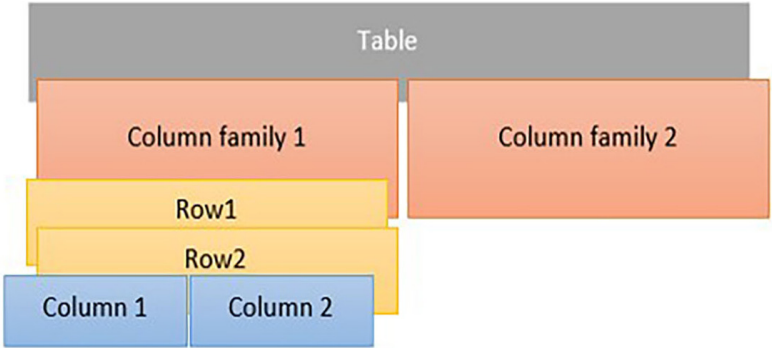


***Figure 6.1**: The relationship between table, column families, columns, and rows*

Debugging

The following section deals with how to debug HBase.

Question 1: Can you list some of the debugging skills you acquired while working with Hbase?

Answer: Logs are the first point to look into for the issue. Another issue is the region server's suicidal problem when the region server is unable to create more threads due to its limit. Another common issue is garbage collection pauses, which can cause time-out issues.

Question 2: How can HBase give high availability?

Answer: It can achieve this with the help of replication of the region, which is controlled by the load balancer.

Question 3: How can we say that HBase is OS-independent?

Answer: We can say that HBase is OS-dependent because it is dependent on platform-independent programming languages like Java.

Question 4: Can an HFile store data belonging to different families?

Answer: Yes.

Question 5: Can you list down commands at the row level and table level?

Answer: Row-level commands are **`get`**, **`put`**, **`scan`**, and **`increment`**. Table-level commands are drop list scans.

Question 6: What do you think is the use of an HBase shell?

Answer: It provides Java API, which allows us to connect to Hbase.

Question 7: Does HBase support row iteration?

Answer: Yes.

Question 8: Is HBase schema-less?

Answer: Yes. It is schema-less because users just need to create column families and can store data on the fly.

Question 9: What is the use of **Time to Live** (**TTL**)?

Answer: TTL indicates the time for data retention, after which the data can be removed to get more storage and space.

Question 10: How can we check whether a table is disabled?

Answer: The **`is_disabled`** command can be used to do this.

Question 11: What is Memstore?

Answer: It is a buffer where data is stored in memory before writing.

Question 12: Can you list some of the problem areas of HBase?

Answer: Inbuilt authentication missing and indexes on key columns only are among the problem areas of HBase.

Question 13: What type of process is HBase?

Answer: It is a scale-out process.

Question 14: How is each row uniquely identified?

Answer: Rowkey is used for this purpose.

Question 15: What are the different types of table design in HBase?

Answer: The different types of table design in HBase are:

- Short and wide: Fewer columns and a large number of rows.
- Tall and thin: A large number of columns and fewer rows.

Question 16: How can we change the default version in HBase?

Answer: `alter 'tablename', {NAME => 'ColFamily', VERSIONS => 50}`

Question 17: Does HBase support SQL?

Answer: No.

Question 18: Does HBase support reverse iteration?

Answer: No.

Question 19: The region server resides with the Data node. State True or False.

Answer: True. It resides on the same nodes as the data node.

Question 20: What is the ideal number of columns in HBase?

Answer: The general rule does not exceed the number of column families per HBase table by 15.

Question 21: Does HBase support join?

Answer: Yes, with the help of a map reduce joins.

Question 22: When do we use the shutdown command?

Answer: We use it to remove all the clusters.

Question 23: What is the use of `THRIFT`?

Answer: It allows schema compilers. This allows you to bind other programming languages other than Java. Other programming languages can connect to Thrift which can in turn connect to HBase to provide required services.

Difference between HBase and other technologies

This section deals with the difference between HBase and other technologies.

Question 1: Differentiate between HBase and RDBMS.

Answer: The following table shows the differentiation:

HBase	RDBMS
Schema-less	Schema oriented
Scalable horizontally	Not scalable horizontally
Non-transaction oriented	Transaction oriented

Table 6.3: *RDBMS vs HBase*

Question 2: Does HBase support sharding?

Answer: Yes. The ability of HBase to divide data in the region is known as sharding.

Question 3: Differentiate between Hive and HBase.

Answer: Refer to the following table:

Hive	HBase
Batch processing	Real-time data streaming
Supports schema model	Schema-less

Hive	HBase
High latency as compared to HBase	Low latency
More costly	Relatively affordable
HQL supports like SQL	Does not support SQL

***Table 6.4**: Hive vs HBase*

Question 4: Differentiate between PIG and HBase.

Answer: The following table explains the difference:

HBase	PIG
Preferred for unstructured	Preferred for structured
Operates on the server-side	Operates on the client-side
Supports key-value pairs	The high-level language that converts to Map reduce

***Table 6.5**: HBase vs PIG*

Conclusion

In this chapter, we learned the basics and configuration details of HBase.

In the next chapter, we will focus on ElasticSearch.

Join our book's Discord space

Join the book's Discord Workspace for Latest updates, Offers, Tech happenings around the world, New Release and Sessions with the Authors:

https://discord.bpbonline.com

Chapter 7
Elasticsearch

Introduction

Have you ever wondered how to achieve faster search? A NoSQL which helps in these areas is Elasticsearch.

Structure

In this chapter, we will discuss questions related to the following topics:

- Introduction to Elasticsearch
- Elasticsearch Logstash and Kibana
- Operations
- Query support
- Connectors and Indices

Objectives

After reading this chapter, you will learn the internal working of Elasticsearch as well as architecture. You will also learn the application areas of and integration points of Elasticsearch.

Introduction to Elasticsearch

Question 1: Can you explain Elasticsearch in a single line?

Answer: Elasticsearch is based on the search engine with the name Lucene which provides restful API for inserting in updating and deleting documents. These documents can be then created and detailed analysis can be performed on the data stored

Question 2: What are some of the features of Elasticsearch?

Answer: The following are some of the features of Elasticsearch:

- It is an open-source number written in Java.
- It allows setting up the index on a variety of data.
- It supports rest APIs for insert/update/delete of data and query data.
- The output can be displayed in the form of JSON.
- It supports full-text search.
- It supports near real-time searches.
- Data can be replicated and sharded.
- It supports searching on replicated and sharded data.
- It supports multiple languages.

Question 3: Explain the key concepts in Elasticsearch.

Answer: The key concepts in Elasticsearch are:

- A single-run instance of Elasticsearch is called a Node
- A cluster collection of one or more nodes is known as a cluster, and it provides indexing and searching capabilities.

- The index can be defined as a collection of documents of different types and their corresponding properties make use of shards.
- The document is a JSON-oriented data structure which consists of a collection of fields in a particular order.
- Shard is the concept of dividing the index horizontally. This implies that each shard contains all the properties of a given document.
- A primary shard is the one where the data is originally stored, and the data is replicated. On secondary shots, replication is done to improve the availability

Question 4: Explain the index with respect to Elasticsearch.

Answer: The equivalent of the database in RDBMS is an index in Elasticsearch.

A single index can consist of multiple data belonging to different types and within each type, we can have many documents with properties.

From an RDBMS perspective, we can say indices are equivalent to the database, types are equivalent to tables and documents are stored in types , which can have two columns or rows.

The index also contains a namespace which is logically mapped to one or more primary shards. It can also contain zero or more replica shards.

Question 5: With respect to Elasticsearch, what is a document?

Answer: In Elasticsearch, a document is the basic unit of data.

It is a JSON key-value pair. Also, each document in Elasticsearch will have a type and a unique ID. For example, **`_id`** – the unique identifier for the document.

Question 6: Can you explain Shard with respect to Elasticsearch?

Answer: Shard is a distribution unit with respect to Elasticsearch.

Any index can be divided across shards in a cluster. Thus, by adding new nodes index can be easily sharded.

Question 7: Can you compare and contrast Elasticsearch and Relational Database Management System?

Answer: The following table shows the difference between Elasticsearch and RDBMS:

Relational Database Management System	Elasticsearch
Database	Cluster
Table	Index
Column	Field
Row	Document

***Table 7.1**: Difference between Elasticsearch and Relational database management system*

Elasticsearch Logstash and Kibana

In this section, we will learn about Logstash and its usage along with Kibana and their interrelationship.

Question 1: What do you mean by the term ELK?

Answer: ELK is an acronym for Elasticsearch Logstash and Kibana. These are the products developed and maintained by the company with the name Elastic.

- Elasticsearch is used for storing logs.
- Logstash provides a mechanism for sending logs and processing logs.
- Kibana stands for a visualization tool which allows us to visualize the logs.

The following diagram explains the relationship between Logstash, Elastic and Kibana:

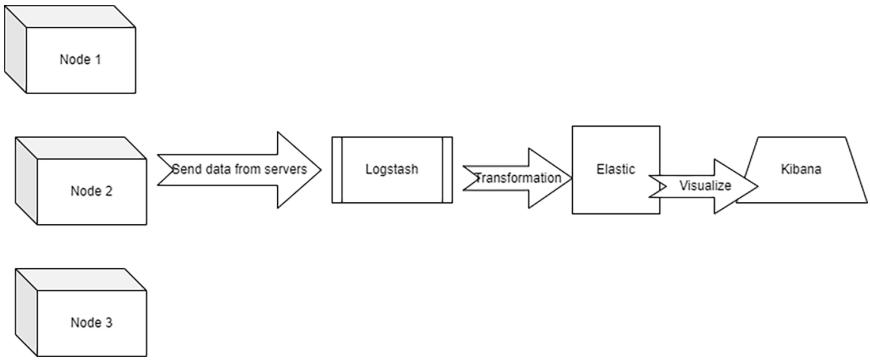


***Figure 7.1**: Different responsibilities in the case of the ELK model*

Question 2: What do you think are the advantages of using ELK?

Answer: These are the advantages of using ELK:

- It helps to provide a central way of storing and analyzing logs from various applications in an organization.
- It can provide detailed insights ranging from a single instance to multiple instances and eliminate the need to get logs from individual machines manually.
- Installation is quite simple.
- It can scale vertically and horizontally.
- A lot of ready-to-use libraries are available in many programming languages

Question 3: What is the use of a tokenizer in Elasticsearch?

Answer: A tokenizer can be considered as a processing unit which receives a stream of characters and breaks down that stream of characters into smaller individual words known as tokens and the output is a stream of tokens.

A tokenizer has many functionalities:

- Maintaining the order or position of each word.
- Maintaining the character of sets, including starting and ending offset.
- Finding the token type.

Elasticsearch supports a wide variety of tokenizers some of which are as follows:

- Word-oriented tokens support a wide variety of token build-in user white space lowercase and classic tokenizer.
- Partial word tokenizers make use of the N-gram algorithm which is used for natural language processing.
- Text tokenizers porch-making tokens using patterns and keywords.

Question 4: What is the use of replica with respect to Elasticsearch?

Answer: One of the major requirements of NoSQL is high availability and fault tolerance.

Elasticsearch achieves this with the help of replicas.

Elasticsearch is made up of shards and for each shard, two or more copies of data are known as replicas.

Operations

Question 1: Explain a few operations which you can perform on documents.

Answer: Some of the operations which can be performed on documents are:

- Inserting a document
- Updating a document
- Deleting a document
- Fetching data from a document
- Indexing a document

Question 2: What is the use of cluster with respect to Elasticsearch?

Answer: A cluster is made up oa f collection of nodes that can hold the data and allows indexing on a federated basis and searching abilities across nodes.

Question 3: Which command can be used to delete an index?

Answer: To delete an index, we can make use of the **`DELETE`** command followed by the index name:

```
DELETE /index name
```

Question 4: Which HTTP method is used to add an index in Elasticsearch?

Answer: The POST method can be used to add an index.

Question 5: Where is the data stored in the case of Elasticsearch?

Answer: The data is stored in:

Linux flavors

`/var/lib/elasticsearch` (CentOS)

`/var/lib/elasticsearch/data` (Ubuntu)

In Windows, it is in a data sub folder.

Question 6: Can you list down the steps used to start an Elasticsearch server?

Answer: Follow the given steps to start the Elasticsearch server:

1. Start a prompt or terminal and go to the Bin directory.
2. The Elasticsearch process can then be started by firing either **`elasticsearch.bat`** or elasticsearch.sh based on the nature of the environment.
3. On firing above, you can open the browser of your choice and in the browser, open **http://localhost:9200**. This will show you the elastic cluster and meta value of the database.

Question 7: Explain the use of Fuzzy Logic in the case of Elasticsearch.

Answer: The major use of Fuzzy query is to find the document that contains a term which is similar to the one being searched. A lot of variations are created by fuzzy query with respect to the solar storms with a specified edit distance.

On firing the search using a fuzzy query, there are near-matching terms returned by the system.

Question 8: What is the use of term queries with respect to Elasticsearch?

Answer: Term queries provide various functionalities like a wild card or regular expression search range search, fuzzy search as well as existing search.

Question 9: What is the meaning of NRT with reference to Elasticsearch?

Answer: Near real-time search supports searching, after storing the document after a delay of one second.

Question 10: What is X-pack?

Answer: X-Pack extension which provides many other capabilities like security/alerting/monitoring/reporting, and machine learning.

Question 11: What is the use of CAT API?

Answer: Cat API provides an easier way of displaying data which is much more readable.

Provided are a few mappings:

`GET /_cat/indices?v =è`displays various indexes.

`GET /_cat/nodes?h=ip,portà` displays nodes details, including IP and port. H stands for the header.

`GET/ _cat/templates?v&sàorder:desc,index_patternsàdisplays` sorting order.

`GET /_cat/count?v-à` gives a count of the number of documents.

Question 12: What is the use of mapping in the document?

Answer: Mapping defines the data type of fields stored in Elasticsearch and the corresponding format of data stored.

It also controls rules around adding new dynamic fields.

Question 13: What are the data types supported by Elastic search?

Answer: It supports two major data types:

- Core data type:

  They are from one of the following:

  text, date, long, double, boolean, keyword and IP

- Complex data types:

  They include objects and arrays which can further store objects.

Question 14: Can we search more than one index in Elasticsearch?

Answer: We can search more than one index by using either the comma-separated notation POST / index1,index2,index3/_search.

Question 15: Can we search all the indices in Elasticsearch?

Answer: We can achieve this by using POST /_all/_search.

Question 16: What is the use of Index API?

Answer: Index API is used to either insert or update data in an index. This is basically done with the help of a PUT request.

Question 17: Is Automatic index creation supported in Elasticsearch?

Answer: Automatic Index creation is supported in Elastic search, that is, whenever an index is not present, it automatically creates an index and corresponding mapping.

Question 18: Can automatic index creation be disabled in Elasticsearch?

Answer: It can be disabled by setting the following properties to false:

`action.auto_create_index` and **`index.mapper.dynamic`**

Question 19: What type of versioning is supported by Elasticsearch?

Answer: Internal versioning and external versioning are two types of versioning supported by Elasticsearch:

- Internal versioning makes use of a default counter to increase by one.
- In external versioning, a third-party library is used.

Query support

In this section, we will learn how to use a query with Elasticsearch.

Question 1: What is Query DSL?

Answer: DSL stands for a domain-specific language which uses JSON to define queries.

Leaf query: These are queries which look for a specific value in a field.

Compound query: It consists of multiple leaf and compound queries to achieve a particular result.

Question 2: What type of full-text queries are in Elasticsearch?

Answer: Elasticsearch supports a wide variety of full-text queries and some of them are as follows:

Match query: It matches a particular word or text:

```
{
   “query”:{
      “match” : {
         “field”:”word”
      }
   }
}
```

Multi-match query: It is used in a multi-match query. We can match a word or phrase in more than one field:

```
{
   “query”:{
      “multi_match” : {
         “query”: “word”,
         “fields”: [ “field1”, “field2” ]
```

```
        }
    }
}
```

Question 3: Is relational query possible in Elasticsearch?

Answer: Range queries are used for such scenarios range queries supports relational operators like greater than less than greater than and equal to or less than or equal to functionalities:

```
{
    "query":{
        "range":{
            "field":{
                "gte":value
            }
        }
    }
}
```

Question 4: Can you give an example of a compound query?

Answer: A compound query can consist of more than one or more leave queries or other compound queries A sample compound query is as follows:

```
{
    "query": {
        "bool" : {
            "must" : {
                "term" : { "field1" : "value" }
            },
            "filter": {
                "term" : { "field2" : "value2" }
            },
            "minimum_should_match" : 1
```

```
        }
    }
}
```

Question 5: Does Elasticsearch support geospatial queries?

Answer: The special queries are supported by Elasticsearch with a mechanism known as Geo queries. Geo queries can check on locations as well as coordinates and on points.

Question 6: What is the use of ingest node in Elasticsearch?

Answer: Any transformations which are required before putting the data in Elasticsearch are achieved with the help of ingest node.

It involves creating a pipeline, post in which a document is created. In the pipeline, we have various processes such as conversion from string to integer before creating the document

A point to be noted is that a document can be created even without having transformation if it is to be stored directly.

Question 7: What are the different policy management APIs present in Elasticsearch?

Answer: Elasticsearch forms various policy management APIs for creating update and delete in policies:

`PUT_ilm/policy/policy_id` for inserting and updating

`GET_ilm/policy/policy_id` for getting policies

`DELETE_ilm/policy/policy_id` for deleting policies

Question 8: For people belonging to a sequel background, does Elasticsearch provide an easy mechanism to use sequel-like queries?

Answer: Elasticsearch sequel is a component in Elasticsearch that allows SQL-like queries. It provides native integration and is lightweight and efficient, and requires no additional hardware to run:

```
POST /_sql?format=txt
{
   “query”: “SELECT * FROM vishwa where name
   like ‘Tiger’”
}
```

Connectors and indices

Question 1: What is the role of connectors and exporters in the case of Elasticsearch?

Answer: Connectors are used to get the status of various APIs in Elasticsearch. It runs after a given interval of time and can create one or more monitoring documents.

Exporters are responsible for collecting the data from elastic sources and giving it to the clusters which are used for monitoring.

Exporters can be in the same cluster (known as local) or they can be in different clusters.

Question 2: What is the use of the role of the job in the case of Elasticsearch?

Answer: The role of the job has the following properties:

- It runs periodically.
- It can perform a summarization of data which it collects from indexes.

Using the above two features, we can specify a pattern and create a new index.

Question 3: What is the meaning of frozen indices?

Answer: Some of the cells which are not frequently used and can be rebuilt on the fly are called frozen indices. The advantage of using frozen indices in devices is that they make use of less heap memory that helps in maintaining a more disk-to-heap ratio.

Question 4: Do you know the difference between the Master eligible node and the master node?

Answer: Master eligible knows what you can think of as backup nodes, and which will be elected to become master nodes in case of Master node failures.

The master node performs many important functionalities such as creating and maintaining indexes.

Thus, master eligible nodes will be promoted to master nodes in case of failures.

Question 5: What is the use of migration API?

Answer: This API is used after an upgrade in Elasticsearch to a new version. This is helpful to get all the Indus updated to the new version of the Elasticsearch cluster with the help of XPAC.

Question 6: What is the use of Elastic stack reporting?

Answer: Elastic stack reporting is used to export data into various formats like images or PDFs or even spreadsheets as per need.

Question 7: What is the use of Beats with respect to Elasticsearch?

Answer: The transfer of data from an existing server to an elastic search is done with the help of Beats. This is used to ensure that the data is processed before it can be shown in Kibana.

Various logs like audit logs or event logs, cloud logs or network traffic logs can be transported with the help of Beats.

Question 8: What is the use of analyzers in Elasticsearch?

Answer: Analyzers play an important role during the indexing of data in Elasticsearch for performing the transformation of data. Different types of analyzers are supported in Elasticsearch; our simple standard whitespace keyword pattern snowball as well as custom analysis.

Question 9: Can we enforce a schema in Elasticsearch?

Answer: This is possible with the help of mappings in Elasticsearch where we can force a schema on documents.

Conclusion

In this chapter, we focused on Elasticsearch.

In the next chapter, we will see how to use a graph-based NoSQL named Neo4j.

Join our book's Discord space

Join the book's Discord Workspace for Latest updates, Offers, Tech happenings around the world, New Release and Sessions with the Authors:

https://discord.bpbonline.com

CHAPTER 8
Neo4j

Introduction

Do you see possible friends on social media? How did the tool know that these are your possible friends? Graph-based NoSQL is one way of achieving this. One of the famous graph-based tools is Neo4j which has a free community version.

Structure

In this chapter, we will discuss the following topics:

- Introduction to Neo4j
- Neo4j commands
- Application and internal working
- Advanced working

Objectives

By the end of this chapter, you will have learned the basics, working, and query language support of Neo4j.

Introduction to Neo4j

In this section, we will deal with questions related to the introduction of Neo4j.

Question 1: What is Neo4j?

Answer: It is schema-free popular graph-based data. It is one of the widely adopted graph-based NoSQL which is used in a variety of applications like social media-based applications and customer relationship management software.

Question 2: What type of data structure does Neo4j use?

Answer: Neo4j makes use of a graph-based data structure that consists of edges and nodes.

Question 3: Do you know in which programming language is Neo4j written?

Answer: Neo4j is written in Java programming language.

Question 4: Do you know the name of the query language used by Neo4j?

Answer: Cypher query language is the name of the query language which is used by Neo4j. This query language allows you to retrieve data from graph-based data. It is equivalent to SQL of NoSQL.

Cypher query language is executed using the $ symbol.

The basic syntax of the query is as follows:

```
START n
MATCH n-[r]- m
RETURN r;
```

```
In the above n stands for node
r stands for relationship
```

Question 5: What type of language is Cypher query language?

Answer: It is a declarative language.

Question 6: For what type of application is Neo4j used?

Answer: It is mainly used for Network and IT operations. It is also used for real-time tracking of social network privacy and risk management.

Neo4j commands

This section discusses questions related to Neo4j commands.

Question 1: Can you list the most commonly used commands in Neo4j?

Answer: The most commonly used commands in Neo4j are:

- **`CREATE`**: It is used for creating nodes or relationships.
- **`MATCH:`** It is used for fetching data.
- **`MERGE`**: It combines create and match.
- **`SET`**: It adds or updates new properties/old properties.
- **`CREATE UNIQUE`**: It is used for maintaining constraints.

Question 2: Can you show the basic syntax of **`MERGE`**?

Answer: The basic syntax of **`MERGE`** is:

```
MERGE (node:label {key1:value, . . . . . })
```

Merge combines a node and label with corresponding key-value pairs.

Question 3: What is stored in a node in Neo4j?

Answer: A node in Neo4j is used to store data/records.

Question 4: Compare terminologies of graph-based NoSQL with RDBMS.

Answer: The following table shows a terminologies comparison of graph-based NoSQL with RDBMS:

Graph NOSQL	RDBMS
Graphs	Tables
Nodes	Rows
Fields and Properties	Column and Data
Relationship	Constraints
Traversal	Joins

***Table 8.1**: Comparison of graph-based NoSQL and RDBMS*

Question 5: What are some of the unique features of Neo4j?

Answer: Some unique features of Neo4j are:

- It supports a native graph processing engine.
- It provides rest API to be used by various programming languages.
- It also supports exporting data into various formats like JSON or Excel.

Question 6: How can we access the Neo4j environment?

Answer: Neo4j Desktop provides a development environment required while dealing with Neo4j. Neo4j Browser is inbuild with Neo4j Desktop and is used to access services exposed. By default, 7474 is the port configured for Neo4jBrowser and hence, it can be accessed using the URL **http://127.0.0.1:7474/.**

Question 7: What is the command used to delete all the nodes?

Answer: The trick is to match all the nodes using the Match command and then call detach with delete:

```
MATCH (n)
DETACH DELETE n
```

Question 8: What is the use of the `DETACH` command?

Answer: It removes the node after making it free of all the related entities.

Question 9: Explain the use of the `MATCH` command.

Answer: It is used for either updating or returning the value of data fetched.

It cannot be used alone and needs to be used either with return or update/delete statements.

Question 10: Is it possible to delete only the relationship?

Answer: Yes, that is possible.

```
MATCH (n { field: value })-[r:KNOWS]->()
DELETE r
```

In the preceding command, we match a node on the field value pair and remove the relationship only.

Question 11: How to delete a single node based on the field value pair?

Answer: You can delete a single note by:

```
MATCH (n:Entity { field: value })
DETACH DELETE n
```

Question 12: Does Neo4j support the cloud?

Answer: Yes, Neo4j supports the cloud because it has rest API which can get exposed over the cloud.

Question 13: How can we add a new property to a node?

Answer: This is possible with the help of the **SET** statement.

Here is the syntax of SET:

```
MATCH (n {field: value})
SET (condition).property = value
RETURN n
```

Question 14: How can we remove labels or properties?

Answer: We can do this with the help of the **REMOVE** command.

Question 15: Do you know what differentiates **REMOVE** and **DELETE**?

Answer: **DELETE** is used for removing nodes.

REMOVE is used for deleting properties or labels.

Question 16: Does NEO4j use any cache?

Answer: Yes, Neo4j makes use of an object cache which can store nodes and relationships for faster access.

Following are the two types of caches:

- **Reference Cache**: It has an entire heap on which objects and relationships are stored.
- **High-Performance Cache** (**HPC**): It has allocated a limit on which objects and relationships are stored.

Question 17: Does Neo4j support limiting the data returned?

Answer: Yes, Cypher Query language has support for the LIMIT clause which can be used for the above purpose.

Applications and internal working

In the following section, we will discuss application areas of graph-based NOSQL along with its architecture and internal working.

Question 1: How can graph databases be used for finding attacks?

Answer: All the attacks done from specific machines or Inter protocol addresses can be represented as nodes.

The relationship between them and the time of the attack can let us easily understand when the attack happened.

Question 2: Do you know how NEO4j internally stores data?

Answer: Internally, NEO4j stores data in the form of files.

There are specific files for storing specific data.

For example:

- **`neostore.nodestore.db`** stores data of the node.
- **`neostore.propertystore.db`** stores property of the node.
- **`neostore.relationshipstore.db`** stores relationships of the node.
- **`neostore.label.db`** stores labels of the node.

Question 3: Does Neo4j support searching a collection of values?

Answer: The IN clause in Neo4j can help with selecting from a collection of values.

Question 4: What are some of the offerings from the Neo4j database?

Answer: Some of the offerings from the Neo4j database are:

- **Neo4j Graph Database**: This is the core engine for storing nodes and relationships.
- **Neo4j Browser**: This supports basic visualization and querying capability over the browser.
- **Neo4j Bloom**: This is a visualization tool for showing graphs without query.
- **Neo4j AuraDB**: This is a database-as-a-service on cloud.
- **Graph Data Science**: This provides data science algorithms over graph databases.
- **Neo4j ETL**: This supports ETL to get data from different data sources.

Question 5: Does NEO4j support indexing?

Answer: NEO4j supports indexing by using create an index on the command which can create an index on the label or properties of nodes.

Question 6: Does NEO4j support any visualization tool?

Answer: NEO4j Bloom provides visualization without any need for querying knowledge.

Question 7: What command do we use for starting the NEO4j database?

Answer: Post installation, we can start NEO4j by going inside the bin folder and running the neo4j bat /.sh file.

Question 8: Does Neo4j have any limit on the number of nodes made?

Answer: The earlier version had a hard limit of 34 billion nodes, but the latest versions make use of compression, thus removing any limit.

Question 9: What are Fabric graphs?

Answer: Neo4j supports sharding with the help of a Fabric database, which can be used as an entry point.

It allows retrieving data from multiple different databases in a single query.

Question 10: What is the use of CUgraph?

Answer: It is a collection of various algorithms revolving around the graph theory which provides various insights into the data stored.

Question 11: What is the use of Neo4j ETL?

Answer: It provides insights about relationships stored in a database.

Question 12: Does Neo4j support replication?

Answer: Replica writes are the standard mechanisms supported by NEO4j in which replica is written across various data centers.

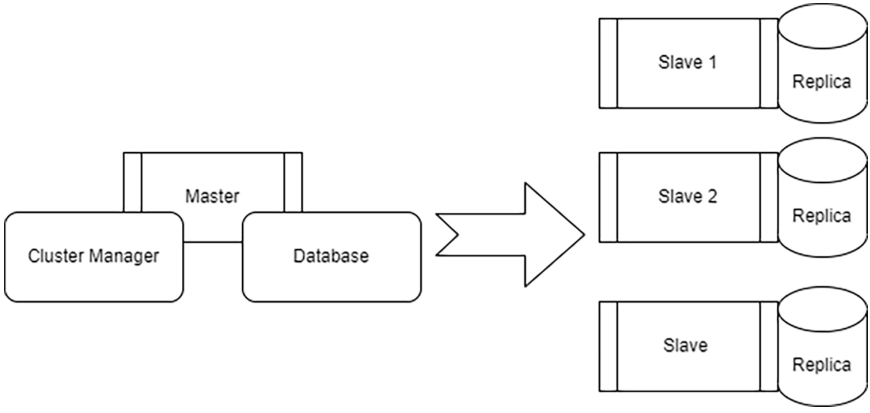


***Figure 8.1**: How data is replicated across nodes*

Question 13: What is the use of Neo4j streams?

Answer: This is used to connect to high-speed streaming solutions like Kafka.

Advanced working

In this section, we will deal with **Create update and delete** (**CRUD**) along with select operations.

Question 1: Can you write an example of how to create nodes and relationships?

Answer: The example is as follows:

```
CREATE (Ricardo:coder{name: "Ricardo Pocha", YOB: 1985, POB: "Mexico"})
CREATE (Mex:Country {name: "Mexico"})
CREATE (Ricardo)-[r:CODER_OF]->(MEX)
```

The preceding code is the creation of two nodes and the correspondingly creating relationship between them.

Question 2: What is the use of FOR each clause?

Answer: It is used to update data based on a list or collection.

It has a starting and ending point on which, we can get each node and perform corresponding actions:

```
MATCH collection = (starting)-[*]->(ending)
WHERE starting.node = value1 AND ending.node = value2
FOREACH (n IN nodes(collection)| SET n.marked = TRUE)
```

Question 3: Can we use where clauses in NEO4j?

Answer: Yes, the syntax of the where clause is as follows:

```
MATCH (label)
WHERE label.field = value
RETURN label
```

Question 4: Does Neo4j supports count?

Answer: Yes, Neo4j supports retrieving using the count.

The syntax is as follows:

```
MATCH (n { field: value })-->(x)
```

```
RETURN n, count(*)
```

Question 5: How does Neo4j support ordering?

Answer: It supports Ordering with the help of the ORDER By clause:

```
MATCH (n)
RETURN n.field1,n.field2 . . . . . .
ORDER BY n.field1
```

Question 6: Can you skip certain records in Neo4j?

Answer: Yes, that is supported with the help of the SKIP function in which we can mention the number of records to be skipped.

Question 7: Does Neo4j support aggregation?

Answer: Yes, it supports aggregate operators like count, avg, min, max, and sum.

Question 8: How can we convert a sequence to n number of rows?

Answer: This can be achieved with the help of **UNWIND** functions.

Question 9: Can we create constraints on the Neo4j graph?

Answer: Yes, constraints can be easily created in the Neo4j graph with the help of the create unique constraint:

```
MATCH (root {field1: value})
CREATE UNIQUE (root)-[:relationship]-(someone)
RETURN someone
```

Question 10: What are the string operations supported in NEO4j?

Answer: Case conversions like upper, lower, substring, and replace operations are supported.

Conclusion

In this chapter, we saw the working, architecture as well as query language related to graph-based NoSQL.

This book made you learn what types of applications in which graph-based NoSQL is used. We learned the working of Relational databases as well as NOSQL databases, how both relational databases and NoSQL work and how they differ from each other. Along with this, we also learned the categories of NoSQL and the application areas of each type of NoSQL. Internal working, replication, partitioning, and architecture were covered for each type of NoSQL.

Join our book's Discord space

Join the book's Discord Workspace for Latest updates, Offers, Tech happenings around the world, New Release and Sessions with the Authors:

https://discord.bpbonline.com

Index

D

E

F

G

H

I

J

L

M

N

O

P

Q

R

S

T

V

W

X

SQL and NoSQL Interview Questions

DESCRIPTION

In every software-based job interview, database systems will undoubtedly be a topic of discussion. It has become customary to ask at least a few database-related questions. As NoSQL technologies continue to gain popularity, asking about their functionality and practical applications during interviews is becoming more commonplace.

This book focuses on these two areas, aiming to familiarize you with the types of questions you may encounter in interviews and providing guidance on preparing and strategizing accordingly. This book thoroughly explores the NoSQL family, covering everything from the fundamentals to advanced topics such as architecture, optimization, and practical use cases. It also includes a selection of frequently asked questions from a query perspective. Moreover, this book is designed to assist you in last-minute revisions. This book also tackles a common interview challenge of effectively communicating complex concepts in a clear and concise manner, even if you have a strong understanding of the subject matter.

By the end of the book, you will be well-equipped to handle interviews and confidently answer queries related to both, database systems and NoSQL.

KEY FEATURES

- Get familiar with different concepts and queries in SQL.
- Comprehensive coverage of different types of NoSQL databases.
- Understand the performance tuning strategies and best practices for NoSQL databases.

WHAT WILL YOU LEARN

- Get an in-depth understanding of Relational Databases.
- Understand the differences between Relational databases and NoSQL databases.
- Explore the architecture for each type of NoSQL database.
- Get insights into the application areas of each type of NoSQL database.
- Understand the paradigm shift in designing NoSQL schema and queries.

WHO THIS BOOK IS FOR

This book is for current and aspiring emerging tech professionals, students, and anyone who wishes to have a rewarding career in emerging technologies such as Relational database and NOSQL.

BPB PUBLICATIONS
www.bpbonline.com

ISBN 978-93-5551-858-3