**1.** Describe the key characteristics that differentiate NoSQL databases from traditional relational databases.

**Answer:** The key characteristics that differentiate NoSQL databases from traditional relational databases are:

 **Data Model**:

* **Relational Databases**: Use a structured data model with tables that have predefined schemas, requiring all data to fit into these tables.
* **NoSQL Databases**: Typically have flexible schemas for unstructured data. They can store data in various formats such as key-value pairs, documents, graphs, or wide-column stores, allowing for varied data types and structures.

 **Scalability**:

* **Relational Databases**: Generally scale vertically, meaning they require more powerful hardware to handle increased loads.
* **NoSQL Databases**: Designed to scale horizontally across many servers. They handle increased loads by adding more servers to the database infrastructure, which can be more cost-effective and provide greater scalability.

 **Consistency Model**:

* **Relational Databases**: Employ ACID (Atomicity, Consistency, Isolation, Durability) properties to ensure reliable transactions and data integrity.
* **NoSQL Databases**: Often use BASE (Basically Available, Soft state, Eventual consistency) consistency models which do not guarantee immediate consistency but ensure that all updates will eventually propagate through the system and become consistent.

**2.** Explain the four main types of NoSQL databases: Key-Value Stores, Document Stores, Column-Family Stores, and Graph Databases.

Answer: The four main types of NoSQL databases are:

1. **Key-Value Stores**:
   * **Description**: These are the simplest form of NoSQL databases where each item contains a key and a value. The key is unique, and the associated value can be a string, number, or any serialized form of data.
   * **Use Cases**: Ideal for scenarios where quick access to data is required via a key, such as caching sessions in web applications, storing user preferences, or handling massive volumes of data with simple lookup queries.
   * **Examples**: Redis, Amazon DynamoDB, and Riak.
2. **Document Stores**:
   * **Description**: In document stores, data is stored in documents that are grouped together in collections. Each document can contain complex nested data structures and typically uses a format like JSON, BSON, or XML.
   * **Use Cases**: Suitable for content management systems, e-commerce applications, and any application that requires storing and managing semi-structured data that can vary in structure.
   * **Examples**: MongoDB, CouchDB, and Apache Couchbase.
3. **Column-Family Stores**:
   * **Description**: These databases store data in columns rather than rows, which is beneficial for queries that access large volumes of data from a few columns. Data is stored in column families as rows that have many columns associated with a row key.
   * **Use Cases**: Best suited for analyzing large datasets, real-time analytics, and applications that require high performance and scalability, such as Internet of Things (IoT) and time-series data.
   * **Examples**: Apache Cassandra, HBase, and ScyllaDB.
4. **Graph Databases**:
   * **Description**: Graph databases use graph structures (nodes, edges, and properties) to store, map, and query relationships. They are efficient at traversing and managing highly connected data.
   * **Use Cases**: Ideal for social networks, recommendation engines, fraud detection systems, and any application where relationships and connections between data points are key.
   * **Examples**: Neo4j, JanusGraph, and ArangoDB.

**3. Working with Document Stores – MongoDB**

1. Initialize a collection named ‘games’.

**Query:**

db.createCollection("games")

**(2)** Insert 6 games into the collection. Each document below include four attributes: name, genre, rating and achievements.

**Query**:

db.games.insertMany([

{

"name": "Spy Hunter",

"genre": "Racing",

"rating": 76,

"achievements": ["Speed Demon"]

},

{

"name": "Mario Kart 64",

"genre": "Racing",

"rating": 96,

"achievements": ["Game Master", "Speed Demon"]

},

{

"name": "Tetris",

"genre": "Puzzle",

"rating": 83,

"achievements": ["Puzzle Solver"]

},

{

"name": "Mega Man 5",

"genre": "Platformer",

"rating": 81,

"achievements": ["Robot Master"]

},

{

"name": "Star Fox",

"genre": "Action",

"rating": 71

},

{

"name": "The Legend of Zelda: Ocarina of Time",

"genre": "Action"

},

{

"name": "Banjo-Kazooie",

"genre": "Platformer",

"rating": 92,

"achievements": ["Game Master", "Speed Demon"]

}

]);

// Confirm insertion

print("Games inserted successfully.");

**Output**:

{

"acknowledged" : true,

"insertedIds" : [

ObjectId("6626c23aa620cfbeca17ef2e"),

ObjectId("6626c23aa620cfbeca17ef2f"),

ObjectId("6626c23aa620cfbeca17ef30"),

ObjectId("6626c23aa620cfbeca17ef31"),

ObjectId("6626c23aa620cfbeca17ef32"),

ObjectId("6626c23aa620cfbeca17ef33"),

ObjectId("6626c23aa620cfbeca17ef34")

]

}

Games inserted successfully.

**(3)** Retrieve all the games in the collection.

**Query**:

var allGames = db.games.find();

allGames.forEach(function(game) {

printjson(game);

});

**Output**:

{

"\_id" : ObjectId("6626c8b6c3af7369f67446c2"),

"name" : "Spy Hunter",

"genre" : "Racing",

"rating" : 76,

"achievements" : [

"Speed Demon"

]

}

{

"\_id" : ObjectId("6626c8b6c3af7369f67446c3"),

"name" : "Mario Kart 64",

"genre" : "Racing",

"rating" : 96,

"achievements" : [

"Game Master",

"Speed Demon"

]

}

{

"\_id" : ObjectId("6626c8b6c3af7369f67446c4"),

"name" : "Tetris",

"genre" : "Puzzle",

"rating" : 83,

"achievements" : [

"Puzzle Solver"

]

}

{

"\_id" : ObjectId("6626c8b6c3af7369f67446c5"),

"name" : "Mega Man 5",

"genre" : "Platformer",

"rating" : 81,

"achievements" : [

"Robot Master"

]

}

{

"\_id" : ObjectId("6626c8b6c3af7369f67446c6"),

"name" : "Star Fox",

"genre" : "Action",

"rating" : 71

}

{

"\_id" : ObjectId("6626c8b6c3af7369f67446c7"),

"name" : "The Legend of Zelda: Ocarina of Time",

"genre" : "Action"

}

{

"\_id" : ObjectId("6626c8b6c3af7369f67446c8"),

"name" : "Banjo-Kazooie",

"genre" : "Platformer",

"rating" : 92,

"achievements" : [

"Game Master",

"Speed Demon"

]

}

**(4)** Display only the name and genre of all games, excluding their \_id.

**Query**:

var projection = { \_id: 0, name: 1, genre: 1 };

var games = db.games.find({}, projection);

games.forEach(function(game) {

print("Name: " + game.name + ", Genre: " + game.genre);

});

**Output**:

Name: Spy Hunter, Genre: Racing

Name: Mario Kart 64, Genre: Racing

Name: Tetris, Genre: Puzzle

Name: Mega Man 5, Genre: Platformer

Name: Star Fox, Genre: Action

Name: The Legend of Zelda: Ocarina of Time, Genre: Action

Name: Banjo-Kazooie, Genre: Platformer

A screenshot of a computer

Description automatically generated

**(5)** Locate a specific game by its name, such as “Mario Kart 64”, without employing limit(). Instead, use the findOne function.

**Query**:

var game = db.games.findOne({ name: "Mario Kart 64" });

printjson(game);

**Result**:

A screenshot of a computer code

Description automatically generated

**(6)** Find the top 3 highest-rated games.

**Query**:

var topRatedGames = db.games.find().sort({ rating: -1 }).limit(3);

var rank = 1;

topRatedGames.forEach(function(game) {

print("Rank " + rank + ": " + game.name + " - Rating: " + game.rating);

rank++;

});

**Output**:

Rank 1: Mario Kart 64 - Rating: 96

Rank 2: Banjo-Kazooie - Rating: 92

Rank 3: Tetris - Rating: 83

A screenshot of a number of text

Description automatically generated

**(7)** List all unique genres available in the collection.

**Query**:

var uniqueGenres = db.games.distinct("genre");

printjson(uniqueGenres);

**Output**:

A close up of a sign

Description automatically generated

**(8)** Find games with ratings above 90.

**Query**:

var highlyRatedGames = db.games.find({ rating: { $gt: 90 } });

highlyRatedGames.forEach(function(game) {

printjson(game);

});

**Output**:

{

"\_id" : ObjectId("6626cac15f4a022ac5a070fc"),

"name" : "Mario Kart 64",

"genre" : "Racing",

"rating" : 96,

"achievements" : [

"Game Master",

"Speed Demon"

]

}

{

"\_id" : ObjectId("6626cac15f4a022ac5a07101"),

"name" : "Banjo-Kazooie",

"genre" : "Platformer",

"rating" : 92,

"achievements" : [

"Game Master",

"Speed Demon"

]

}

A computer screen shot of a computer program

Description automatically generated

**(9)** Find all games except those in the ‘Racing’ genre.

**Query**:

var nonRacingGames = db.games.find({ genre: { $ne: "Racing" } });

nonRacingGames.forEach(function(game) {

printjson(game);

});

**Result**:

{

"\_id" : ObjectId("6626cb1ac8a288ca88fbcfac"),

"name" : "Tetris",

"genre" : "Puzzle",

"rating" : 83,

"achievements" : [

"Puzzle Solver"

]

}

{

"\_id" : ObjectId("6626cb1ac8a288ca88fbcfad"),

"name" : "Mega Man 5",

"genre" : "Platformer",

"rating" : 81,

"achievements" : [

"Robot Master"

]

}

{

"\_id" : ObjectId("6626cb1ac8a288ca88fbcfae"),

"name" : "Star Fox",

"genre" : "Action",

"rating" : 71

}

{

"\_id" : ObjectId("6626cb1ac8a288ca88fbcfaf"),

"name" : "The Legend of Zelda: Ocarina of Time",

"genre" : "Action"

}

{

"\_id" : ObjectId("6626cb1ac8a288ca88fbcfb0"),

"name" : "Banjo-Kazooie",

"genre" : "Platformer",

"rating" : 92,

"achievements" : [

"Game Master",

"Speed Demon"

]

}

**(10)** Identify games that are missing the ‘rating’ attribute.

**Query**:

var gamesWithoutRating = db.games.find({ rating: { $exists: false } });

gamesWithoutRating.forEach(function(game) {

printjson(game);

});

**Output**:

{

"\_id" : ObjectId("6626cb86125e33eb5c0ae5b3"),

"name" : "The Legend of Zelda: Ocarina of Time",

"genre" : "Action"

}

**(11)** Count the number of games per genre

**Query**:

var genreCounts = db.games.aggregate([

{ $group: { \_id: "$genre", count: { $sum: 1 } } }

]);

genreCounts.forEach(function(genreCount) {

print("Genre: " + genreCount.\_id + ", Count: " + genreCount.count);

});

**Result**:

Genre: Racing, Count: 2

Genre: Platformer, Count: 2

Genre: Puzzle, Count: 1

Genre: Action, Count: 2

**(12)** Calculate the average rating for each genre.

**Query**:

var averageRatingByGenre = db.games.aggregate([

{

$group: {

\_id: "$genre",

averageRating: { $avg: "$rating" }

}

}

]);

averageRatingByGenre.forEach(function(genreRating) {

print("Genre: " + genreRating.\_id + ", Average Rating: " + genreRating.averageRating.toFixed(2));

});

**Output**:

Genre: Platformer, Average Rating: 86.50

Genre: Racing, Average Rating: 86.00

Genre: Puzzle, Average Rating: 83.00

Genre: Action, Average Rating: 71.00

**(13)** Locate all the games that possesses both the “Game Maser” and the “Speed Demon” achievements.

**Query**:

var gamesWithAchievements = db.games.find({ achievements: { $all: ["Game Master", "Speed Demon"] } });

gamesWithAchievements.forEach(function(game) {

printjson(game);

});

**Output**:

{

"\_id" : ObjectId("6626cc53ccb6f9abe447f4bb"),

"name" : "Mario Kart 64",

"genre" : "Racing",

"rating" : 96,

"achievements" : [

"Game Master",

"Speed Demon"

]

}

{

"\_id" : ObjectId("6626cc53ccb6f9abe447f4c0"),

"name" : "Banjo-Kazooie",

"genre" : "Platformer",

"rating" : 92,

"achievements" : [

"Game Master",

"Speed Demon"

]

}

**14)** Employee the update() function to modify a game named with “Star Fox” by adding two achievements with the following properties:

"name": "Game Master", "points": 100

"name": "Speed Demon", "points": 135

**Query**:

db.games.update(

{ name: "Star Fox" }, // Specify the game to update

{

$push: {

achievements: { $each: [

{ name: "Game Master", points: 100 },

{ name: "Speed Demon", points: 135 }

]}

}

}

);

**Result**:



**(15)** Add a common achievement “Fan Favorite” to all games with a rating of 90 or higher.

**Query**:

db.games.updateMany(

{ rating: { $gte: 90 } }, // Select games with a rating of 90 or higher

{ $push: { achievements: { name: "Fan Favorite" } } } // Add the "Fan Favorite" achievement

);

Output:



**(16)** Delete a game named with “Banjo-Kazooie”.

**Query**:

db.games.deleteOne({ name: "Banjo-Kazooie" });

**Result**:



**4. Working with Graph Database** - **Neo4j and Cypher query language**

1. Retrieve the names of members who enjoy thriller books.

Query:

MATCH (r:Reader)-[:LIKES]->(:Book)-[:IS\_GENRE]->(:Genre {name: 'thriller'})

RETURN r.name

Result:

| **r.name** |
| --- |
| **1** | "Anne HatsAway" |
| **2** | "Robert Bertoli" |
| **3** | "Wilfried Lemahieu" |
| **4** | "Bart Baesens" |
| **5** | "Wilfried Lemahieu" |
| **6** | "Elvis Presley" |
| **7** | "Seppe vanden Broucke" |

**(2)** Determine which book has been liked by the youngest reader.

Query:

MATCH (r:Reader)

WITH MIN(r.age) AS minAge

MATCH (r:Reader)-[:LIKES]->(b:Book)

WHERE r.age = minAge

RETURN b.title

| **b.title** |
| --- |
| **1** | "Where are my Keys?" |
| **2** | "Database Management" |
| **3** | "Laughing and Learning" |
| **4** | "A Wizardly Story" |

**(3)** Determine the genre most beloved by the club members.

Query:

MATCH (r:Reader)-[:LIKES]->(:Book)-[:IS\_GENRE]->(g:Genre)

RETURN g.name, COUNT(r) AS readerCount

ORDER BY readerCount DESC

LIMIT 1

Result:

| **g.name** | **readerCount** |
| --- | --- |
| **1** | "humor" | 19 |

**(4)** Identify the most popular book based on the number of likes.

Query:

MATCH (:Reader)-[l:LIKES]->(b:Book)

WITH b, COUNT(l) AS likeCount

ORDER BY likeCount DESC

LIMIT 1

RETURN b.title AS mostPopularBook, likeCount

Result:

| **mostPopularBook** | **likeCount** |
| --- | --- |
| **1** | **"Database Management"** | **5** |

**(5)** Find the common liked books between ‘Wilfried Lemahieu’ and his friends.

Query:

MATCH (wilfried:Reader {name: 'Wilfried Lemahieu'})-[:FRIEND\_OF]->(friend:Reader)-[:LIKES]->(book:Book)

WITH DISTINCT book AS commonLikedBook

MATCH (wilfried)-[:LIKES]->(commonLikedBook)

RETURN commonLikedBook.title AS CommonLikedBook

Result:

| **CommonLikedBook** |
| --- |
| **1** | "Where are my Keys?" |
| **2** | "Where are my Keys?" |
| **3** | "Where are my Keys?" |
| **4** | "Where are my Keys?" |
| **5** | "A Wizardly Story" |
| **6** | "A Wizardly Story" |
| **7** | "A Wizardly Story" |
| **8** | "Who Says Love is Outdated?" |
| **9** | "Who Says Love is Outdated?" |
| **10** | "Without you I am Nothing" |
| **11** | "Without you I am Nothing" |
| **12** | "Mathematics for the Rest of Us" |
| **13** | "Mathematics for the Rest of Us" |
| **14** | "A Chainsaw Massacre" |
| **15** | "An Unsolved Problem for Detective Whiskers" |
| **16** | "An Unsolved Problem for Detective Whiskers" |
| **17** | "An Unsolved Problem for Detective Whiskers" |

**6)** Calculate the average age of readers who like ‘Detective’ genre books.

Query:

MATCH (r:Reader)-[:LIKES]->(:Book)-[:IS\_GENRE]->(:Genre {name: 'detective'})

WITH r.age AS age

RETURN AVG(age) AS averageAge

Result:

| **averageAge** |
| --- |
| **1** | 35.0 |

**(7)** Find friends of 'Bart Baesens' who liked thriller books, ensuring no name is repeated in your results.

Query:

MATCH (bart:Reader {name: 'Bart Baesens'})-[:FRIEND\_OF]->(friend:Reader)-[:LIKES]->(:Book)-[:IS\_GENRE]->(:Genre {name: 'thriller'})

RETURN DISTINCT friend.name AS FriendName

Result:

| **FriendName** |
| --- |
| **1** | "Wilfried Lemahieu" |
| **2** | "Seppe vanden Broucke" |

**(8)** List the books that are liked by friends of 'Wilfried Lemahieu' but not by Wilfried himself.

Query:

MATCH (wilfried:Reader {name: 'Wilfried Lemahieu'})-[:FRIEND\_OF]->(friend:Reader)-[:LIKES]->(book:Book)

WHERE NOT (wilfried)-[:LIKES]->(book)

RETURN DISTINCT book.title AS LikedByFriendOnly

Result:

| **LikedByFriendOnly** |
| --- |
| **1** | "A Wizardly Story" |
| **2** | "Who Says Love is Outdated?" |
| **3** | "Without you I am Nothing" |
| **4** | "Mathematics for the Rest of Us" |
| **5** | "A Chainsaw Massacre" |
| **6** | "An Unsolved Problem for Detective Whiskers" |

**(9)** Determine which reader has the broadest taste in books, meaning they like the widest variety of genres.

Query:

MATCH (r:Reader)-[:LIKES]->(:Book)-[:IS\_GENRE]->(g:Genre)

WITH r, COUNT(DISTINCT g) AS genreCount

ORDER BY genreCount DESC

LIMIT 1

RETURN r.name AS ReaderWithBroadestTaste, genreCount AS NumberOfGenresLiked

Result:

| **ReaderWithBroadestTaste** | **NumberOfGenresLiked** |
| --- | --- |
| **1** | **"Robert Bertoli"** | 6 |

**(10)** Query for books which haven’t caught anyone’s interest yet. You may get no result for this query.

Query:

MATCH (b:Book)

WHERE NOT (:Reader)-[:LIKES]->(b)

RETURN b.title AS UnlikedBooks

Result:

(no changes, no records)