NoSQL databases are classified into several types based on their data model. Here are the primary types along with examples for each:

<u>Document Stores:</u>

Description: Store data as documents, typically in JSON, BSON format.

Documents can contain complex data structures and nested sub-documents.

Example: MongoDB

Use Case: Content management systems, e-commerce applications, real-time analytics.

```json
{
    "_id": "12345",
    "name": "John Doe",
    "age": 30,
    "address": {
        "street": "123 Main St",
        "city": "Anytown",
        "state": "CA",
        "zip": "12345"
    },
    "hobbies": ["reading", "hiking", "coding"]
}
```

Key-Value Stores:

Description: Store data as key-value pairs, where the key is a unique identifier and the value is the data associated with the key. These are highly performant for simple read and write operations.

Example: Redis

Use Case: Caching, session management, real-time analytics.

```
SET "user:12345:name" "John Doe"
SET "user:12345:age" "30"
HSET "user:12345:address" "street" "123 Main St"
HSET "user:12345:address" "city" "Anytown"
HSET "user:12345:address" "state" "CA"
HSET "user:12345:address" "zip" "12345"
LPUSH "user:12345:hobbies" "reading"
LPUSH "user:12345:hobbies" "hiking"
LPUSH "user:12345:hobbies" "coding"
```

Column-Family Stores:

Description: Store data in columns and rows, similar to relational databases but optimized for reading and writing large volumes of data. Each column family can have a different schema.

Example: Apache Cassandra

Use Case: Time-series data, large-scale distributed systems, real-time data processing.

Example Comparison
Relational Database (MySQL)
Schema Definition:

```sql
CREATE TABLE Users (
    user_id INT PRIMARY KEY,
    name VARCHAR(100),
    age INT,
    email VARCHAR(100)
);

CREATE TABLE Posts (
    post_id INT PRIMARY KEY,
    user_id INT,
    content TEXT,
    timestamp DATETIME,
    FOREIGN KEY (user_id) REFERENCES Users(user_id)
);
```

Query Example:

```sql
SELECT Users.name, Posts.content
FROM Users
JOIN Posts ON Users.user_id = Posts.user_id
WHERE Users.user_id = 1;
```

Column-Family Store (Cassandra)
Schema Definition:

```
CREATE KEYSPACE socialmedia WITH REPLICATION = {
'class' : 'SimpleStrategy', 'replication_factor' : 3 };

CREATE TABLE socialmedia.Users (
    user_id UUID PRIMARY KEY,
    name TEXT,
    age INT,
    email TEXT
);

CREATE TABLE socialmedia.UserPosts (
    user_id UUID,
    post_id UUID,
    post_timestamp TIMESTAMP,
    content TEXT,
    PRIMARY KEY (user_id, post_timestamp)
) WITH CLUSTERING ORDER BY (post_timestamp DESC);
```

Query Example:

```
SELECT * FROM socialmedia.Users WHERE user_id =
some_uuid;

SELECT * FROM socialmedia.UserPosts WHERE user_id =
some_uuid;
```

Summary:

Data Model: Relational databases use a fixed schema with structured rows and tables, while column-family stores use a flexible schema with rows that can have varying columns.

Architecture: Relational databases are typically centralized and vertically scaled, whereas column-family stores are distributed and horizontally scaled.

Use Cases: Relational databases are suited for transactional applications requiring strong consistency and complex queries. Column-family stores are suited for high-throughput applications handling large volumes of data with flexible schemas.

Graph Databases:

Description: Store data in graph structures, with n**odes representing entities** and **edges representing relationships** between entities. These databases are designed to handle complex and interconnected data.

Example: Neo4j

Use Case: Social networks, recommendation systems, fraud detection.

Graph Data Model

In a graph database like Neo4j, data is stored as nodes, relationships, and properties:

- **Nodes:** Represent entities (e.g., users, posts).
- **Relationships:** Represent connections between entities (e.g., friendships, likes).
- **Properties:** Attributes of nodes and relationships (e.g., name, age, post content).

Example: Social Network

1. **Users:** Alice, Bob, and Carol.
2. **Posts:** Created by users.
3. **Relationships:** Friendships between users and "likes" on posts.

Cypher Queries to Create and Query the Graph

Creating Nodes

```
Creating Nodes
CREATE (alice:User {name: 'Alice', age: 30}),
       (bob:User {name: 'Bob', age: 25}),
       (carol:User {name: 'Carol', age: 27}),
       (post1:Post {content: 'Graph databases are cool!',
timestamp: '2024-05-20'}),
       (post2:Post {content: 'Learning Cypher is fun!',
timestamp: '2024-05-19'})
```

Creating Relationships

```
// Creating friendships
CREATE (alice)-[:FRIEND]->(bob),
       (bob)-[:FRIEND]->(carol),
       (carol)-[:FRIEND]->(alice)

// Creating posts by users
CREATE (alice)-[:POSTED]->(post1),
       (bob)-[:POSTED]->(post2)

// Creating likes
CREATE (bob)-[:LIKES]->(post1),
       (carol)-[:LIKES]->(post1),
       (alice)-[:LIKES]->(post2)
```

Creating Relationships

```
// Creating friendships
CREATE (alice)-[:FRIEND]->(bob),
       (bob)-[:FRIEND]->(carol),
       (carol)-[:FRIEND]->(alice)

// Creating posts by users
CREATE (alice)-[:POSTED]->(post1),
       (bob)-[:POSTED]->(post2)

// Creating likes
CREATE (bob)-[:LIKES]->(post1),
       (carol)-[:LIKES]->(post1),
       (alice)-[:LIKES]->(post2)
```

Querying the Graph

Find all friends of Alice:

```
MATCH (alice:User {name: 'Alice'})-[:FRIEND]->(friends)
RETURN friends.name
```

Find all posts liked by Alice:

```
MATCH (alice:User {name: 'Alice'})-[:LIKES]->(posts)
RETURN posts.content
```

Find who likes the post "Graph databases are cool!":

```
MATCH (post:Post {content: 'Graph databases are
cool!'})<-[:LIKES]-(users)
```

```
RETURN users.name
```

Example Result Set

- Friends of Alice:
  Bob
  Carol
- Posts liked by Alice:
  "Learning Cypher is fun!”

- Users who like the post "Graph databases are cool!":
  Bob
  Carol

Explanation

- **Nodes and Properties:** Nodes User and Post have properties such as name, age, content, and timestamp.

- **Relationships:** Relationships like FRIEND, POSTED, and LIKES connect the nodes, defining how users interact with each other and the content.

- **Queries:** Cypher queries are used to traverse the graph and retrieve data based on the relationships and properties defined.

This example demonstrates how Neo4j can be used to model and query a social network, leveraging the power of graph databases to efficiently manage and explore relationships within the data.