



NoSQL Data Modeling with Redis



Will Johnston
August 17, 2022



"8 Data Modeling Patterns in Redis," a comprehensive e-book on data modeling in NoSQL, thoroughly examines eight data models that developers can utilize in Redis to build modern applications without the obstacles presented by traditional relational databases. Here's a bit of what you can expect within its pages.

Download **"8 Data Modeling Patterns in Redis"** →

NoSQL Database Design

Unlike relational databases, you don't have to have your schema design figured out at the onset. That's what makes NoSQL such a sound choice for many developers. In SQL, your data has to fit the data model, which can lead to lost data. In a [schemaless design](#), your data remains untouched and remains accessible whenever needed.

NoSQL Data Modeling Techniques

When modeling data within a relational database (SQL database), developers often need to anticipate the future of the application from the beginning. What are the possible features? How can you design a data model such that it is flexible enough to accommodate any change that the business might require?

Relational databases are inflexible. They require lots of upfront knowledge and make storing large sets of data in varying models more than just a little difficult. From transferring data into tables or collections, and the complex queries required to disperse it, the traditional SQL route of data modeling is losing popularity in the face of all the nuances required to build modern enterprise applications.

Here are eight prime data modeling techniques developers use in [Redis and NoSQL](#).

The Embedded Pattern

The Embedded Pattern allows for two separate tables or collections to be bundled together, with one table embedded into the other. The Embedded Pattern is a great model for keeping different tables with information that relates to one another, like a table of product names and one of the product details, in the same place. This makes it easier to find all relevant data and understand what their relationship to each other is in the data structure. It also helps your application retrieve the data in both tables in one query, enhancing its overall read performance.



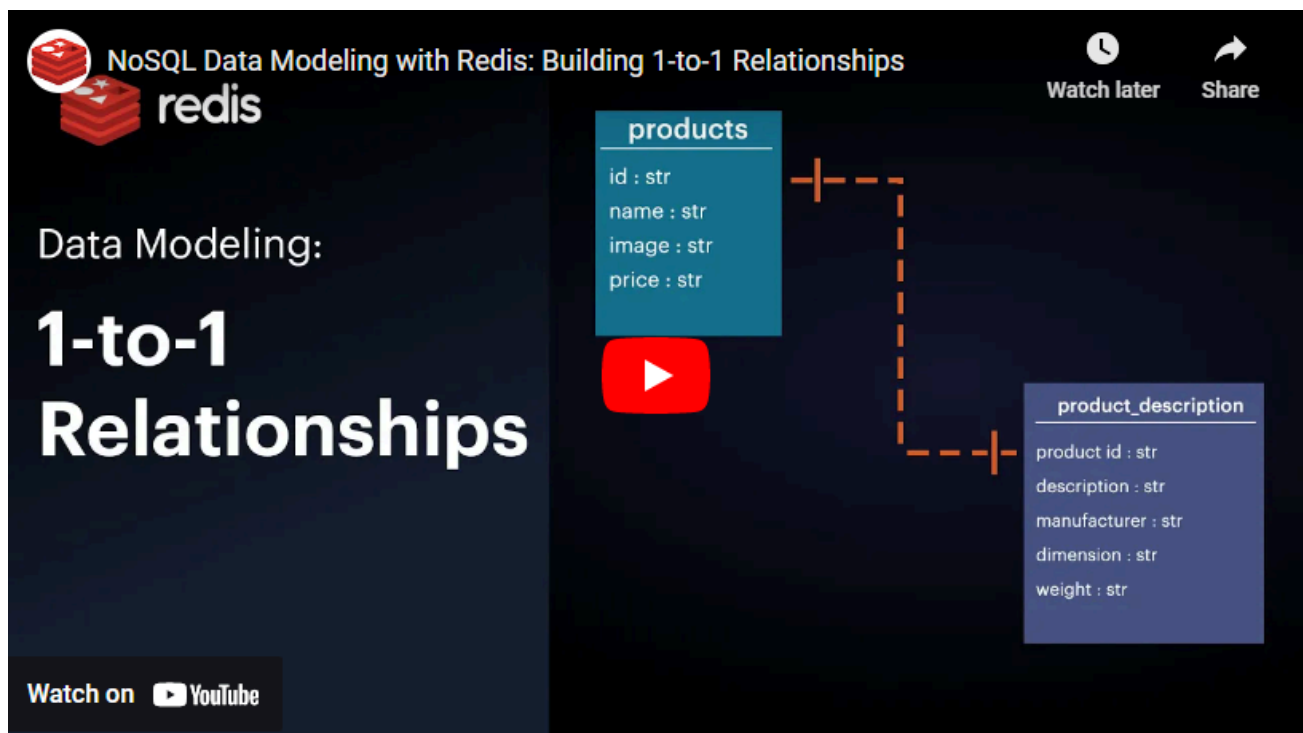
One-to-one relationships

As you'd find in a relational database, the above description describes how a one-to-one data model works.

Say you're building a product catalog for a clothing retailer. The first table could contain the product ID number, name, image, and price, while the other table stores dimensions, specifications, etc.

So, why two tables for data that seem inextricably linked? Why not just one table for data storage and call it a day? Namely, because each table functions as its own view in an application. When showing a list of products, you might only show name, image, and price. The detailed view of a single product would show the rest of the fields. In SQL, the easiest way would be to use two tables, so it's easier to query for only the specific fields you need. In NoSQL, you would only need a single collection since you can nest the details inside a "details" field and not retrieve it if you don't need it.

Take a look at a quick one-to-one relationship model in Redis:



Partial Embed Pattern


One-to-many relationships

First, it's worth noting that one-to-many and many-to-many use both the Embedded and the Partial Embed Pattern.

When you want to model one-to-many relationships, you embed for bounded lists (i.e. lists of a known size) and keep separate collections for unbounded lists.

When building an app that contains products, you typically need real product feedback on your app to establish credibility with your customers. In this case, the product is the "one," and the many reviews, which include an author name, date of publication, rating, and comment, are the 'many' variables in question.

See it in action when you import the [Redis OM](#) library and start modeling:




NoSQL Data Modeling with Redis: Building 1-to-Many Relationships

Watch later Share


Data Modeling:

1-to-many Relationships

Watch on  YouTube


products

- id : str
- name : str
- image : str
- price : str



product_reviews

- product id : str
- name : str
- rating : str
- published_date : datetime
- comment : str



Many-to-many relationships

Pattern 1: Many-to-many relationships with bounded sides

SQL would require you to create a separate table to store the relationship between the datasets of two tables, whereas, in NoSQL, you can create what's called **two-way embedding**.

The video below presents an instructor/course example to illustrate how this model works, using one table for instructors and another for courses. NoSQL simplifies the relational database solution by embedding a list of instructor keys in the course JSON document and a list of course keys in the instructor JSON document.

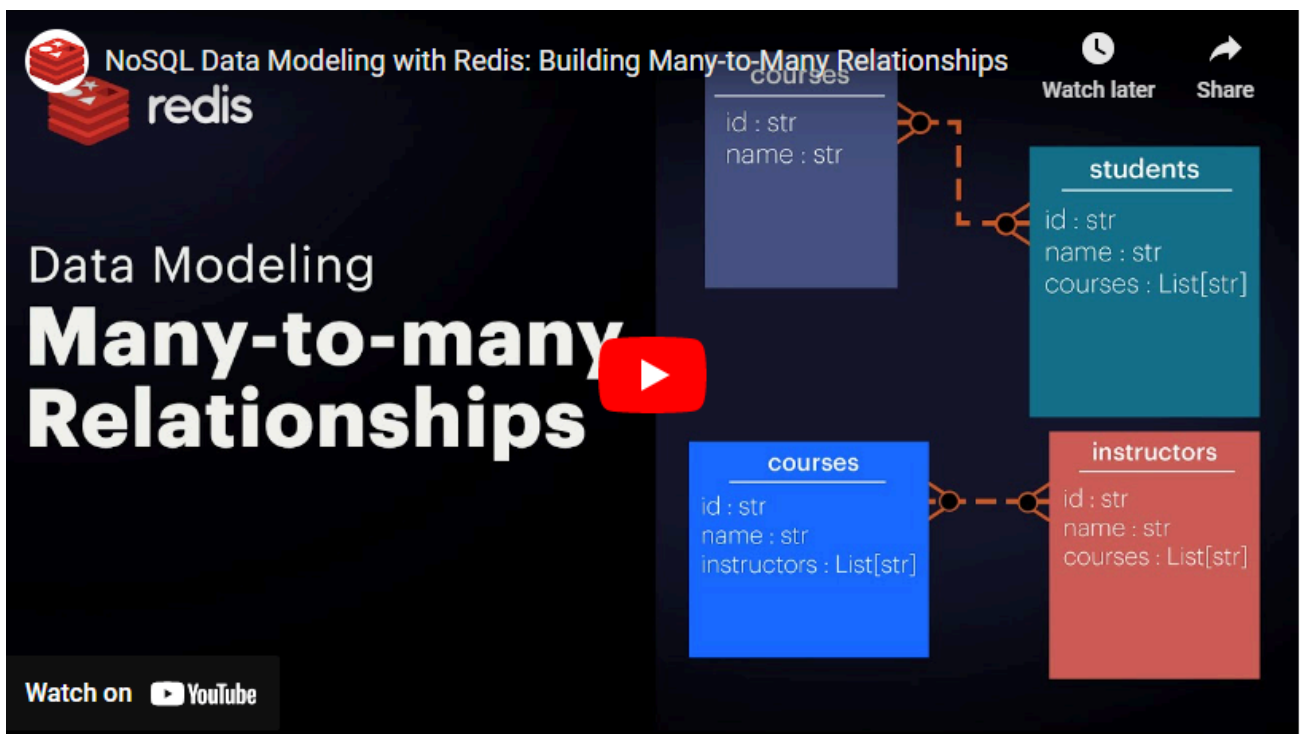
Bounded sides mean there's a limitation in how many data points there are in a dataset. We know there is not an infinite number of professors or courses. Bounded just means it's not infinite.

Pattern 2: Many-to-many relationships with unbounded sides

In SQL, the unbounded sides of a many-to-many relationship could take shape as a table of instructors, and another table with students, where the number enrolled could be infinite in this example. In non-relational databases, what you'd want to do is embed your list on the bounded side.

So, if both sides are bounded, it's possible to embed on both sides, but if only one side is bounded, that would be the collection in which to embed your list.

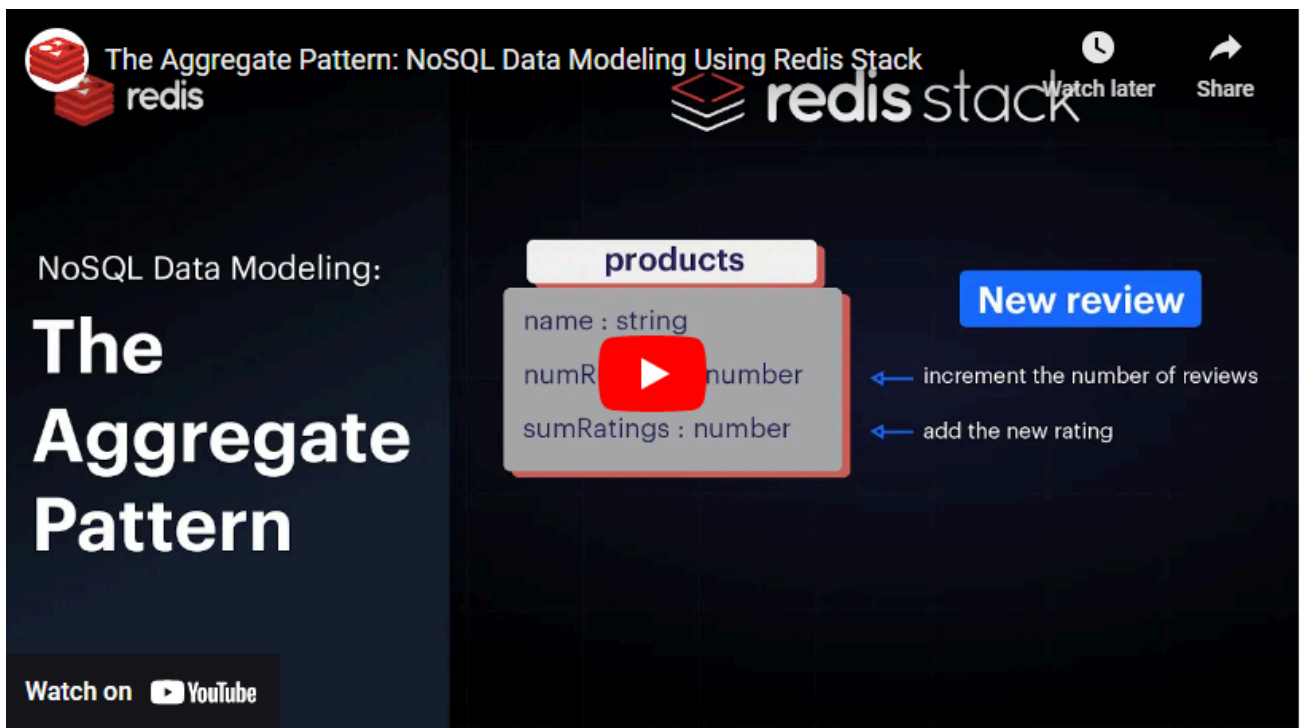
Take a look at what many-to-many relationships look like in [RedisInsight](#):



The Aggregate Pattern

When building read-heavy applications, consider using the Aggregate Pattern to reduce the overhead at read time created by calculating aggregate information on-the-fly. Also known as the Computed Pattern, this model precalculates certain fields during writes rather than reads, which saves read time and overhead on the server and database.

Watch how we model with the aggregate pattern, importing Redis OM for Node.js in Redis Stack:

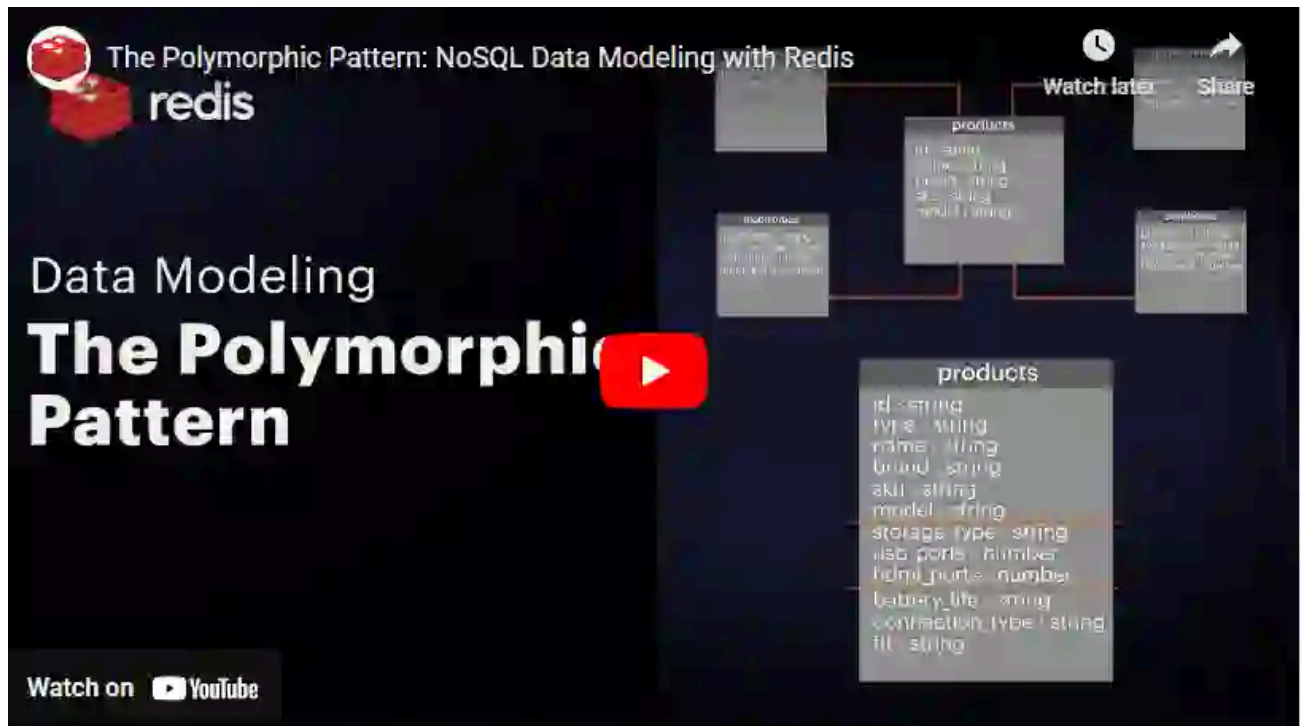


The Polymorphic Pattern

In a relational database, you need separate tables to store specifics about the different types of products you are working with. To get all your products, all tables with all their separate specifics must be joined together, which creates serious overhead.

When building data models, the polymorphic pattern allows you to store many different kinds of products and their unique fields all within the same collection.

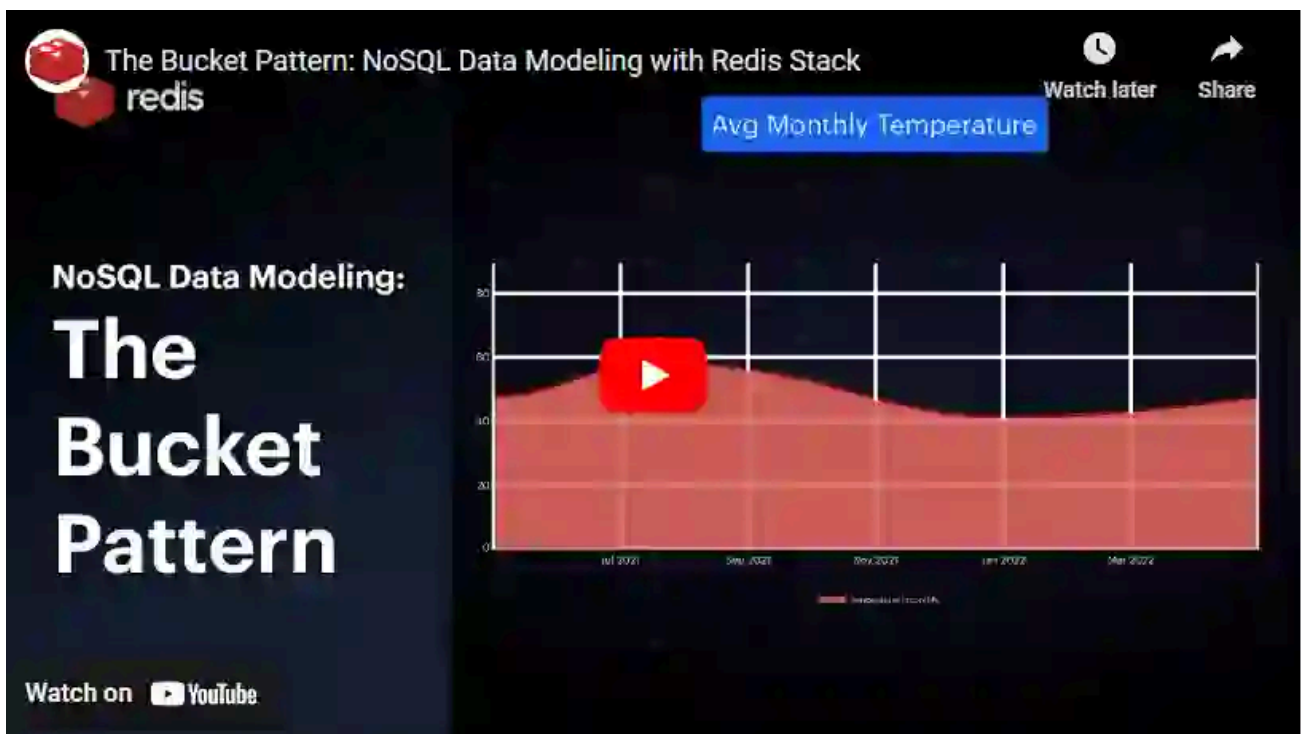
With Redis Stack, all schemas are flexible and let you distinguish type fields that group collections together. Start reducing the number of collections used to store data and simplify app logic in queries using the polymorphic pattern with Redis Stack:



The Bucket Pattern

When building read-heavy applications, consider using the Bucket Pattern to reduce the overhead at read time created by storing and aggregating [time-series data](#) as you go. Storing each bit of data as it trickles puts extra strain on your system. Instead, the Bucket Pattern helps to – you guessed it – “bucket” your time series data according to certain spans of time, lowering your processing time.

Watch as we demonstrate the Bucket Pattern using Node.js and Redis Stack’s built-in time-series capabilities:



The Revision Pattern

How are you tracking your ongoing document changes? That's where the Revision Pattern applies.

Chances are, you've probably used Google Docs in your professional life. Imagine you want to post about the latest product you and your team just developed. The living document displays the post as it stands with the latest revisions, but all the other revisions from all past contributors are still stored and available if needed. This is a perfect example of how the Revision Pattern works. With it, you can store all the revisions you've made to a post, and the post itself, into one single document, simplifying your queries and finessing your schemas.

From the legal industry to [financial services](#), [healthcare](#), publishing, and insurance industries, the Revision Pattern can be leveraged in many use cases reliant on real-time data.

See how we model using the revision pattern in Redis OM for Python:

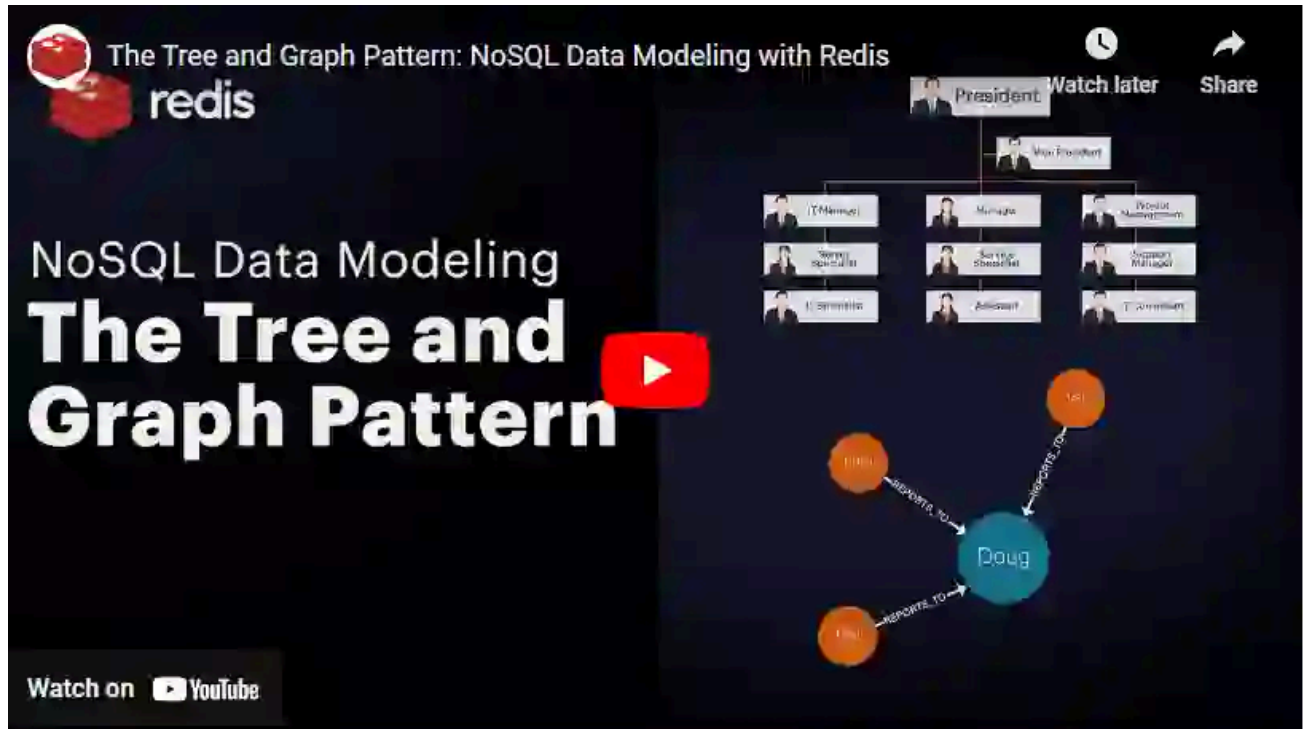


The Tree and Graph Pattern

Typically, you want to avoid JOIN operations in non-relational databases. The Tree and Graph Pattern is particularly useful when those are unavoidable in your schema, and you're working with heavy JOIN-based operations like HR systems, CMSs, product catalogs, and social networks.

The tree pattern isn't a stranger to the relational model – you'll often see the tree pattern like you would in an organization chart or lineage map. In NoSQL, you can leverage these patterns to cut through the complications of JOIN operations without the complexity of SQL queries.

Watch how we model an organizational chart using the Tree and Graph Pattern in Redis Stack:

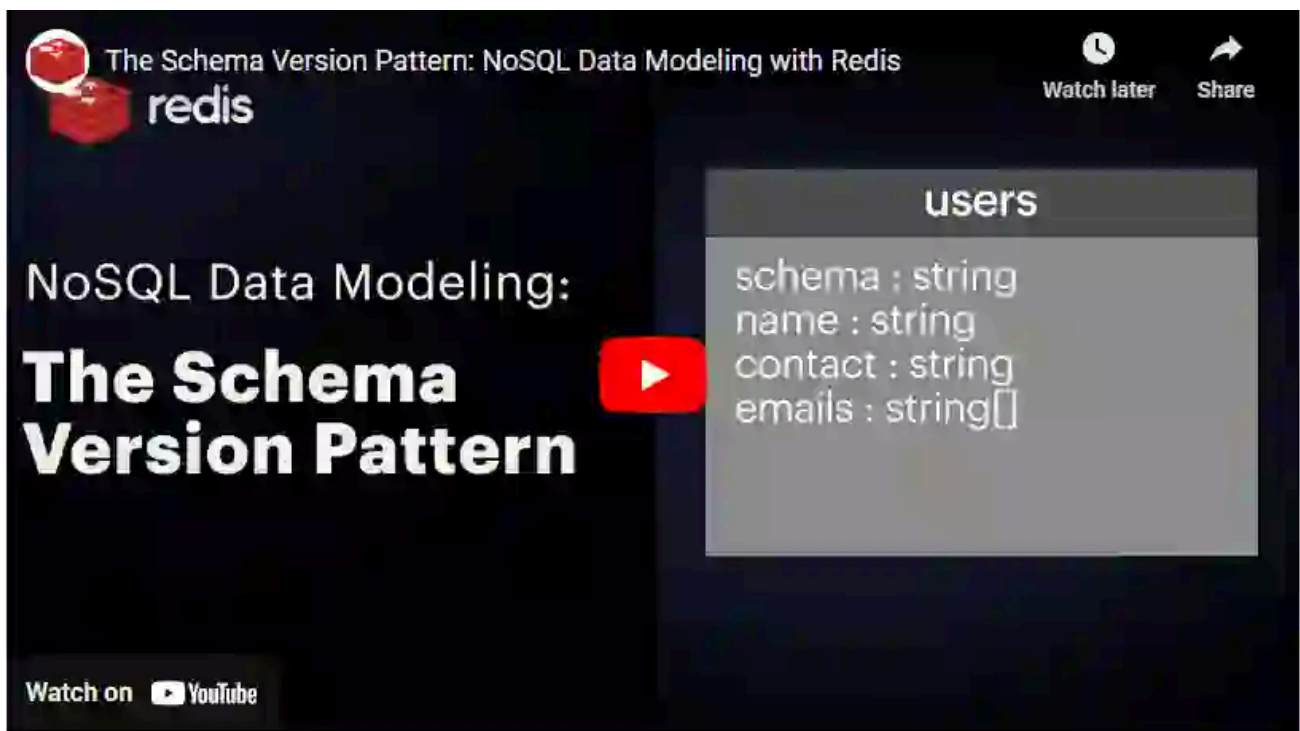


The Schema Version Pattern

One useful protocol to change your data model and upgrade your data is the Schema Version Pattern. When building applications with a SQL database, it's very difficult to make any changes to your model and application logic once you've started building your schema. In NoSQL, you can leverage this model (try Node.js in Redis Stack) to pivot more easily.

It is recommended you always assign a version to your documents so that you can change them in the future without having to worry about immediately migrating all of your data and code. The Schema Version Pattern is a way of assigning a version to your data model, usually done at the document level. You can also choose to version all of your data as part of an API.

You can apply the Schema Version Pattern to your existing code. See how Redis Stack makes it possible:



FAQs

What is a NoSQL data model?

It's a model that is not reinforced by a Relational Database Management System (RDBMS). Therefore, the model isn't explicit about how the data relates – how it all connects together.

How is data stored in NoSQL?

In one of the main non-relational database models, such as a key-value store, document store, graph data model, time series store, column-oriented. Data can be stored on disk, in-memory, or both.

Does NoSQL have a schema?

Yes, it does. When people say NoSQL is "schemaless," they really mean the schema is flexible and determined by the developer and application needs over time. . A schema is eventually settled upon – it doesn't exist at the onset, as is the case with a SQL database.

Can you use Redis as a NoSQL database?

Yes, you can use Redis Enterprise as a NoSQL, in-memory, multi-model database.

Is there an example of a complex NoSQL?

Several companies are using NoSQL for various use cases and to varying levels of complexity. [Redis Launchpad](#) has several examples of applications written using Redis and NoSQL.

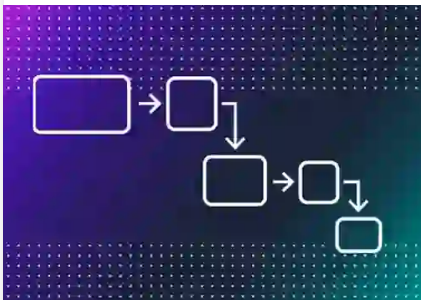
Why does Big Data have a bias toward NoSQL databases?

With Big Data, you typically aggregate huge amounts of data of unknown shape from many different sources. This requires the flexible schema, speed, and scalability that NoSQL databases provide.

Why can we not use machine learning with NoSQL?

This may have been true at one point but is not currently the case.

Related Posts



Schemaless Databases: Pros and Cons

[Learn More →](#)

 **redis stack**

Hello, Redis Stack

[Learn More →](#)

 **redis**
Object Mapping for
spring

What's New in Redis OM Spring?

[Learn More →](#)



PRODUCTS

[Cloud](#)

[Software](#)

[Pricing](#)

[Support](#)

COMPANY

[About Us](#)

[Careers](#)

[Contact us](#)

[Trust Center](#)

[Legal Notices](#)

SOCIAL



 [LANGUAGE](#) 