**TYPES OF DATA BASES**

**Ralational**

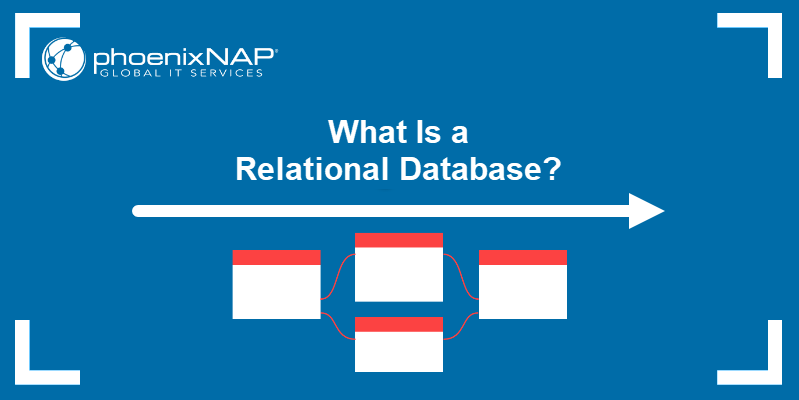
-A relational database is a type of database that stores and provides access to data points that are related to one another. Relational databases are based on the relational model, an intuitive, straightforward way of representing data in tables. In a relational database, each row in the table is a record with a unique ID called the key. The columns of the table hold attributes of the data, and each record usually has a value for each attribute, making it easy to establish the relationships among data points.

<https://www.oracle.com/database/what-is-a-relational-database/>

**Introduction**

With so many options available, it can be challenging to choose a database solution that perfectly fits your needs. When it comes to database types, one popular option is a relational database.

**In this article, we will cover the structure of relational databases, how they work, and the advantages and disadvantages of using them. We will also use examples to illustrate how relational databases organize data.**



## Relational Database Definition

A **relational database** is a type of database that focuses on the relation between stored data elements. It allows users to establish links between different sets of data within the database and use these links to manage and reference related data.

Many relational databases use **SQL** (Structured Query Language) to perform queries and maintain data.

### Relational vs Non-Relational Databases

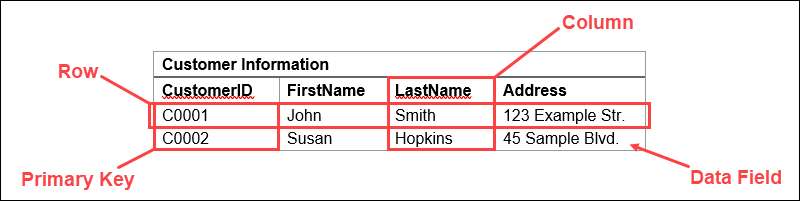
Relational databases focus on relations between data. Hence, relations database need to store data in a highly structured way. This enables faster indexing and query response times and makes the data more secure and consistent.

On the other hand, NoSQL databases don't need to rely on structure as much, which allows them to store large amounts of data, remain flexible, and easily scale storage and performance.

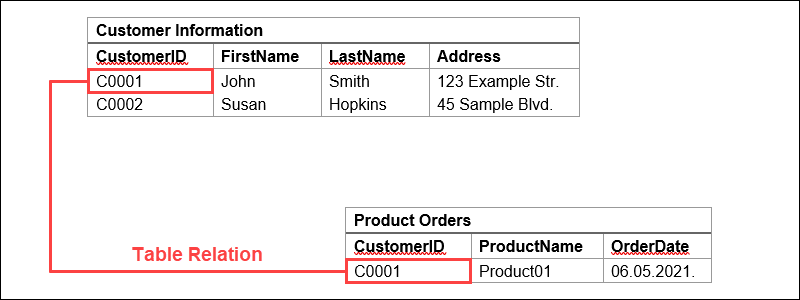
## How Is Data in a Relational Database System Organized?

Relational database systems use a model that organizes data into **tables** of **rows** (also called***records*** or **tuples**) and **columns** (also called **attributes** or **fields**). Generally, columns represent categories of data, while rows represent individual instances.

Let's use a digital storefront as an example. Our database might have a table containing customer information, with columns representing customer names or addresses, while each row contains data for one individual customer.

These tables can be linked or related using **keys**. Each row in a table is identified using a unique key, called a **primary key.** This primary key can be added to another table, becoming a **foreign key.** The primary/foreign key relationship forms the basis of the way relational databases work.

Returning to our example, if we have a table representing product orders, one of the columns might contain customer information. Here, we can import a primary key that links to a row with the information for a specific customer.

This way, we can reference the data or duplicate data from the customer information table. It also means that these two tables are now related.

## Relational Database Examples

Now that we've covered how they work, here are some of the most popular examples of relational databases:

### **MySQL**



MySQL was developed as an open-source management system for relational databases until it was acquired by Sun Microsystems (now Oracle Corporation). It is still available under an open-source license, with the addition of different proprietary licenses.

MySQL features built-in replication support with ACID compliance, shared-nothing clustering, and supports multiple storage engines. However, using some storage engines can cause SQL not to work properly.

MySQL excels at fast data input and scalability while maintaining high availability and performance. This makes it extremely useful for web and application development.

### **PostgreSQL**



PostgreSQL is a free relational database manager available under an open-source license. It shares some features with MySQL, with the notable addition of MVCC (multi-version concurrency control), making it ACID compliant.

PostgreSQL retains a high level of performance and flexibility, even when handling large databases. It's the right choice for users that need high read/write speeds and extensive data analysis.

Some notable users of PostgreSQL include Reddit, Skype, and Instagram.

**Note:** Also, have a look at our [guide to installing PostgreSQL on Ubuntu](https://phoenixnap.com/kb/how-to-install-postgresql-on-ubuntu) and learn [how to create your first database](https://phoenixnap.com/kb/postgres-create-database). And if you want to find out more about different built-in data types available in PostgreSQL, read our article [PostgreSQL data types](https://phoenixnap.com/kb/postgresql-data-types).

### MariaDB, a MySQL fork.MariaDB

MariaDB started as a community-driven fork of MySQL after the latter was purchased by Oracle. It is still open-source, available under the GNU General Public License.

MariaDB builds upon the MySQL base by adding support for even more storage engines and fixing storage engine limitations. This allows it to perform even faster than MySQL and run both SQL and NoSQL in a single database.

Notable MariaDB users include Google, Mozilla, and the Wikimedia Foundation.

### **SQLiteSQLite**

Unlike other entries on this list, SQLite is not a client-server database manager but rather embedded into the end application. This makes it lightweight and able to work with a wide array of systems and platforms.

It also causes some limitations, as SQLite only partially provides triggers, has a limited **ALTER TABLE** function, and cannot write to views. It also limits the maximum size of the database to 32,000 columns and 140 TB.

SQLite is, therefore, best used as a database component for other applications. Notable uses include popular browsers, such as Google Chrome, Mozilla Firefox, Opera, and Safari.

## What Is Relational Database Management System?

A **database management system** (DBMS) is a software solution that helps users view, query, and manage databases.

**Relational database management systems** (RDBMS) are a more advanced subset of DBMS, handling relational databases.

### DBMS vs RDBMS

Here are some of the differences between more general DBMS solutions and RDBMS:

|  |  |
| --- | --- |
| **DBMS** | **RDBMS** |
| Stores smaller amounts of data as files, with no relations. | Stores large amounts of data as tables that are related to each other. |
| Can only access one data element at a time. | Can access multiple data elements at the same time. |
| Working with large amounts of data makes fetching slower. | Relational approach allows data fetching to remain fast even for large databases. |
| No database normalization. | Allows [database normalization](https://phoenixnap.com/kb/database-normalization). |
| Does not support distributed databases. | Supports [distributed databases](https://phoenixnap.com/kb/distributed-database). |
| Supports a single user. | Supports multiple users. |
| Lower security level. | Multiple security levels. |
| Low software and hardware requirements. | High software and hardware requirements. |

## Relational Database Advantages and Disadvantages

Like any other database model, there are advantages and disadvantages to using relational databases:

### Advantages

Since relational databases use tables of rows and columns, they display data more simply than some other database types, making them easier to use.

This tabular structure shifts the focus to handling data, which allows faster performance and the use of complex, high-level queries.

Finally, relational databases make it easy to scale data by simply adding rows, columns, or entire tables without changing the overall database structure.

### Disadvantages

There are limits to how well relational databases can scale. In terms of sheer size, some databases have fixed limits on column lengths. If your database is built on a single [dedicated server](https://phoenixnap.com/servers/dedicated), scaling requires buying more server space, proving expensive in the long run.

Also, constantly adding new elements to a database can make it so complex it becomes difficult to form relations between new pieces of data. Complicated data relations also slow down querying and negatively affect performance.

<https://phoenixnap.com/kb/what-is-a-relational-database>

**Analytical (OLAP)**

-OLAP (Online Analytical Processing) is a category of database processing that facilitates business intelligence.

OLAP provides analysts, managers, and executives with the information they need to make effective decisions about an organization’s strategic directions. OLAP can provide valuable insights into how their business is performing, as well as how they can make improvements.

OLAP tools are optimized for querying and reporting. This is in contrast to Online Transactional Processing (OLTP) applications, which are mainly concerned with transaction based tasks. OLAP tools enable users to analyze multidimensional data interactively from multiple perspectives. OLAP can be used to find trends and get a big picture view of the data. It can also be used for complex number crunching, and to create “what if” scenarios for forward planning.

Typical OLAP applications include business reporting for sales, marketing, management reporting, business process management, budgeting and forecasting, financial reporting, and more.

<https://database.guide/what-is-olap/>

# **WHAT IS THE DEFINITION OF OLAP?**

## Definition of OLAP, Advantages and Uses

## Easy OLAP Definition

**OLAP (Online Analytical Processing)** is the technology behind many Business Intelligence (BI) applications. OLAP is a powerful technology for data discovery, including capabilities for limitless report viewing, complex analytical calculations, and predictive “what if” scenario (budget, forecast) planning.

## How is OLAP Technology Used?

## https://i2.wp.com/olap.com/wp-content/uploads/2019/06/olap-3d-cube.png?w=1080-OLAP is an acronym for **Online Analytical Processing**. OLAP performs multidimensional analysis of business data and provides the capability for complex calculations, trend analysis, and sophisticated data modeling.

It is the foundation for many kinds of business applications for Business Performance Management, Planning, Budgeting, Forecasting, Financial Reporting, Analysis, Simulation Models, Knowledge Discovery, and Data Warehouse Reporting. OLAP enables end-users to perform ad hoc analysis of data in multiple dimensions, thereby providing the insight and understanding they need for better decision making.

## Advantages of OLAP

-Knowledge is the foundation of all successful decisions. Successful businesses continuously plan, analyze and report on sales and operational activities in order to maximize efficiency, reduce expenditures and gain greater market share. Statisticians will tell you that the more sample data you have, the more likely the resulting statistic will be true. Naturally, the more data a company can access about a specific activity, the more likely that the plan to improve that activity will be effective. All businesses collect data using many different systems, and the challenge remains: how to get all the data together to create accurate, reliable, fast information about the business. A company that can take advantage and turn it into shared knowledge, accurately and quickly, will surely be better positioned to make successful business decisions and rise above the competition.

**OLAP technology has been defined as the ability to achieve “fast access to shared multidimensional information.”**  Given OLAP technology’s ability to create very fast aggregations and calculations of underlying data sets, one can understand its usefulness in helping business leaders make better, quicker “informed” decisions.

I'd like to try out an OLAP Service

I'd like to hire an OLAP expert

Unlike relational databases, OLAP tools do not store individual transaction records in two-dimensional, row-by-column format, like a worksheet, but instead use multidimensional database structures—known as ***Cubes*** in OLAP terminology—to store arrays of consolidated information. **The data and formulas are stored in an optimized multidimensional database, while views of the data are created on demand.**

## OLAP for Multidimensional Analysis

Business is a multidimensional activity and businesses are run on decisions based on multiple dimensions. Businesses track their activities by considering many variables. When these variables are tracked on a spreadsheet, they are set on axes (x and y) where each axis represents a logical grouping of variables in a category.

For example, sales in units or dollars may be tracked over one year’s time, by month, where the sales measures might logically be displayed on the y axis and the months might occupy the x axis (i.e., sales measures are rows and months are columns).

To analyze and report on the health of a business and plan future activity, many variable groups or parameters must be tracked on a continuous basis—which is beyond the scope of any number of linked spreadsheets. These variable groups or parameters are called ***Dimensions*** in the On-Line Analytical Processing (OLAP) environment. Nowadays, many spreadsheet users have heard about OLAP technology, but it is not clear to them what OLAP means.

Unlike relational databases, OLAP tools do not store individual transaction records in two-dimensional, row-by-column format, like a worksheet, but instead use multidimensional database structures—known as ***Cubes*** in OLAP terminology—to store arrays of consolidated information. **The data and formulas are stored in an optimized multidimensional database, while views of the data are created on demand.**

Analysts can take any view, or Slice, of a Cube to produce a worksheet-like view of points of interest. Rather than simply working with two dimensions (standard spreadsheet) or three dimensions (for example, a workbook with tabs of the same report, by one variables), companies have many dimensions to track—-for example, a business that distributes goods from more than a single facility will have at least the following Dimensions to consider: Accounts, Locations, Periods, Salespeople and Products. These Dimensions comprise a base for the company’s planning, analysis and reporting activities. Together they represent the “whole” business picture, providing the foundation for all business planning, analysis and reporting activities.

The capability to perform the most sophisticated analyses—-specifically, the multidimensional analysis provided by OLAP technology—is an organizational imperative. Analysts need to view and manipulate data along the multiple dimensions that define an enterprise—essentially, the dimensions necessary for the creation of an effective business model.

## Implementing an OLAP Solution

**-OLAP technology implementations** depend not only on the type of software, but also on underlying data sources and the intended business objective(s).

Each industry or business area is specific and requires some degree of customized modeling to create multidimensional “cubes” for data loading and reporting building, at minimum.

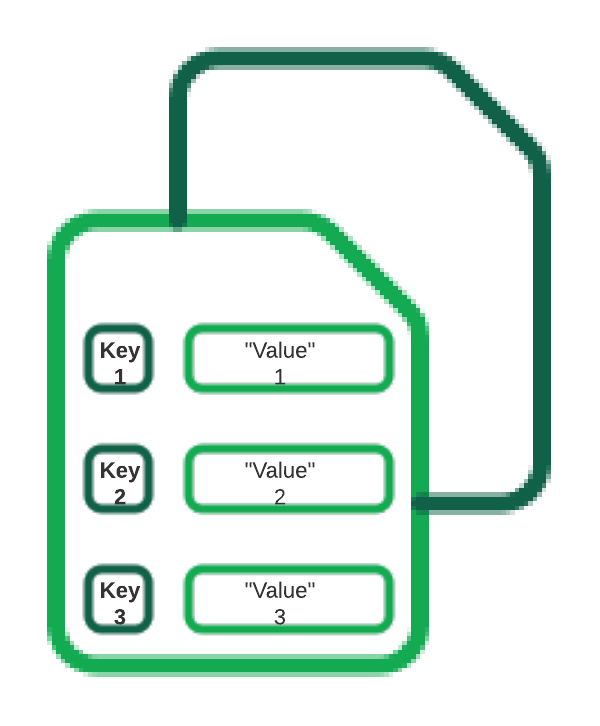
An OLAP solution might be intended for dynamic reporting for finance professionals, with source data originating in an ERP system. Or a solution might address a medical institution’s activities as concerns patient analysis. All of which is to say that customers need to have clear objectives in mind for an intended solution, and start to consider product selection on that basis.

Another factor to consider in an OLAP implementation is the delivery to end users: does the initial user base want to adopt a new front end, or is there a preference for utilizing a dashboard? Or perhaps users are better served by a dynamic spreadsheet “delivery” system to achieve, for example, a collaborative budgeting and forecasting solution.

**Power Excel from PARIS Technologies** [the sponsor of OLAP.com] is one such product that features Excel as a front end, for a wide variety of uses.

<https://olap.com/olap-definition/>

**Key-Value Databases**

- Key value databases, also known as key value stores, are database types where data is stored in a “key-value” format and optimized for reading and writing that data. The data is fetched by a unique key or a number of unique keys to retrieve the associated value with each key. The values can be simple data types like strings and numbers or complex objects.

MongoDB covers a wide range of database examples and use cases, supporting key-value pair data concepts. With its flexible schema and rich query language with secondary indexes, MongoDB is a compelling store for “key-value” data. Learn more in this article and try it with MongoDB Atlas, MongoDB’s Database-as-a-Service platform.

## What is a key-value database?

Over the years, database systems have evolved from legacy relational databases storing data in rows and columns to NoSQL distributed databases allowing a solution per use case. Key-value pair stores are not a new concept and were already with us for the last few decades. One of the known stores is the old Windows Registry allowing the system/applications to store data in a “key-value” structure, where a key can be represented as a unique identifier or a unique path to the value.

Data is written (inserted, updated, and deleted) and queried based on the key to store/retrieve its value.

## How do key-value databases work?

A key-value database, AKA key-value store, associates a value (which can be anything from a number or simple string to a complex object) with a key, which is used to keep track of the object. In its simplest form, a key-value store is like a dictionary/array/map object as it exists in most programming paradigms, but which is stored in a persistent way and managed by a Database Management System (DBMS).

Key-value databases use compact, efficient index structures to be able to quickly and reliably locate a value by its key, making them ideal for systems that need to be able to find and retrieve data in constant time. Redis, for instance, is a key-value database that is optimized for tracking relatively simple data structures (primitive types, lists, heaps, and maps) in a persistent database. By only supporting a limited number of value types, Redis is able to expose an extremely simple interface to querying and manipulating them, and when configured optimally is capable of high throughput.

## What are the features of a key-value database?

A key-value database is defined by the fact that it allows programs or users of programs to retrieve data by keys, which are essentially names, or identifiers, that point to some stored value. Because key-value databases are defined so simply, but can be extended and optimized in numerous ways, there is no global list of features, but there are a few common ones:

* Retrieving a value (if there is one) stored and associated with a given key
* Deleting the value (if there is one) stored and associated with a given key
* Setting, updating, and replacing the value (if there is one) associated with a given key

Modern applications will probably require more than the above, but this is the bare minimum for a key-value store.

## When to use a key-value database

There are several use-cases where choosing a key value store approach is an optimal solution:

* Real time random data access, e.g., user session attributes in an online application such as gaming or finance.
* Caching mechanism for frequently accessed data or configuration based on keys.
* Application is designed on simple key-based queries.

<https://www.mongodb.com/databases/key-value-database>

**[Column (Family) Databases](https://ayende.com/blog/4500/that-no-sql-thing-column-family-databases)**

Column family databases are probably most known because of Google’s BigTable implementation. They are very similar on the surface to relational database, but they are actually quite different beast. Some of the difference is storing data by rows (relational) vs. storing data by columns (column family databases). But a lot of the difference is conceptual in nature. You can’t apply the same sort of solutions that you used in a relational form to a column database.

That is because column databases are not relational, for that matter, they don’t even have what a RDBMS advocate would recognize as tables.

Nitpicker corner: this post is about the concept; I am going to ignore actual implementation details where they don’t illustrate the actual concepts.

Note: If you want more information, I highly recommend this post, explaining about data modeling in a column database.

The following concepts are critical to understand how column databases work:

Column family

Super columns

Column

Columns and super columns in a column database are spare, meaning that they take exactly 0 bytes if they don’t have a value in them. Column families are the nearest thing that we have for a table, since they are about the only thing that you need to define upfront. Unlike a table, however, the only thing that you define in a column family is the name and the key sort options (there is no schema).

Personally, I think that column family databases are probably the best proof of leaky abstractions. Just about everything in CFDB (as I’ll call them from now on) is based around the idea of exposing the actual physical model to the users so they can make efficient use of that.

Column families – A column family is how the data is stored on the disk. All the data in a single column family will sit in the same file (actually, set of files, but that is close enough). A column family can contain super columns or columns.

A super column is a dictionary; it is a column that contains other columns (but not other super columns).

A column is a tuple of name, value and timestamp (I’ll ignore the timestamp and treat it as a key/value pair from now on).

It is important to understand that when schema design in a CFDB is of outmost importance, if you don’t build your schema right, you literally can’t get the data out. CFDB usually offer one of two forms of queries, by key or by key range. This make sense, since a CFDB is meant to be distributed, and the key determine where the actual physical data would be located. This is because the data is stored based on the sort order of the column family, and you have no real way of changing the sorting (except choosing between ascending or descending).

The sort order, unlike in a relational database, isn’t affected by the columns values, but by the column names.

Let assume that in the Users column family, in the row “@ayende”, we have the column “name” set to “Ayende Rahine” and the column “location” set to “Israel”. The CFDB will physically sort them like this in the Users column family file:

@ayende/location = “Israel”

@ayende/name = “Ayende Rahien”

This is because the sort “location” is lower than “name”. If we had a super column involved, for example, in the Friends column family, and the user “@ayende” had two friends, they would be physically stored like this in the Friends column family file:

@ayende/friends/arava= 945

@ayende/friends/rose = 14

Remember that, this property is quite important to understanding how things work in a CFDB. Let us imagine the twitter model, as our example. We need to store: users and tweets. We define three column families:

Users – sorted by UTF8

Tweets – sorted by Sequential Guid

UsersTweets – super column family, sorted by Sequential Guid

  Let us create the user (a note about the notation: I am using named parameters to denote column’s name & value here. The key parameter is the row key, and the column family is Users):

cfdb.Users.Insert(key: “@ayende”, name: “Ayende Rahine”, location: “Israel”, profession: “Wizard”);

You can see a visualization of how below. Note that this doesn’t look at all like how we would typically visualize a row in a relational database.

[](http://ayende.com/Blog/images/ayende_com/Blog/WindowsLiveWriter/ThatNoSQLThingColumnFamilyDatabases_109EE/image_6.png)

Now let us create a tweet:

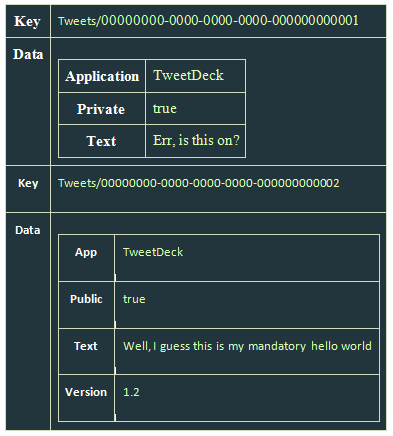
var firstTweetKey = “Tweets/” + SequentialGuid.Create();

cfdb.Tweets.Insert(key: firstTweetKey, application: “TweekDeck”, text: “Err, is this on?”, private: true);

var secondTweetKey = “Tweets/” + SequentialGuid.Create();

cfdb.Tweets.Insert(key: secondTweetKey, app: “Twhirl”, version: “1.2”, text: “Well, I guess this is my mandatory hello world”, public: true);

And here is how it actually looks:

[](http://ayende.com/Blog/images/ayende_com/Blog/WindowsLiveWriter/ThatNoSQLThingColumnFamilyDatabases_109EE/image_8.png)

There are several things to notice here:

In this case, the key doesn’t matter, but it does matter that it is sequential, because that will allow us to sort of it later.

Both rows have different data columns on them.

We don’t actually have any way to associate a user to a tweet.

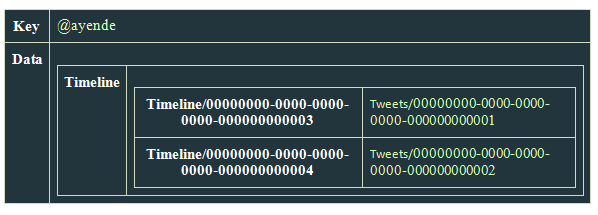
That last bears some talking about. In a relational database, we would define a column called UserId, and that would give us the ability to link back to the user. Moreover, a relational will allow us to query the tweets by the user id, letting us get the user’s tweets. A CFDB doesn’t give us this option, there is no way to query by column value. For that matter, there is no way to query by column (which is a familiar trick if you are using something like Lucene).

Instead, the only thing that a CFDB gives us is a query by key. In order to answer that question, we need the UsersTweets column family:

cfdb.UsersTweets.Insert(key: “@ayende”, timeline: { SequentialGuid.Create(): firstTweetKey } );

cfdb.UsersTweets.Insert(key: “@ayende”, timeline: { SequentialGuid.Create(): secondTweetKey } );

On the CFDB, it looks like this:

[](http://ayende.com/Blog/images/ayende_com/Blog/WindowsLiveWriter/ThatNoSQLThingColumnFamilyDatabases_109EE/image_12.png)

And now we need more explanation about the notation. Here we insert into the UsersTweets column family, to the row with the key: “@ayende”, to the super column timeline two columns, the name of each column is a sequential guid, which means that we can sort by it. What this actually does is create a single row with a single super column, holding two columns, where each column name is a guid, and the value of each column is the key of a row in the Tweets table.

Question: Couldn’t we create a super column in the Users’ column family to store the relationship? Well, yes, we could, but a column family can contain either columns or super columns, it cannot contain both.

Now, in order to get tweets for a user, we need to execute:

var tweetIds =

cfdb.UsersTweets.Get(“@ayende”)  
 .Fetch(“timeline”)

.Take(25)  
 .OrderByDescending()

.Select(x=>x.Value);

var tweets = cfdb.Tweets.Get(tweetIds);

In essence, we execute two queries, one on the UsersTweets column family, requesting the columns & values in the “timeline” super column in the row keyed “@ayende”, then execute another query against the Tweets column family to get the actual tweets.

Because the data is sorted by the column name, and because we choose to sort in descending order, we get the last 25 tweets for this user.

What would happen if I wanted to show the last 25 tweets overall (for the public timeline)? Well, that is actually very easy, all I need to do is to query the Tweets column family for tweets, ordering them by descending key order.

Nitpicker corner: No, there is not such API for a CFDB for .NET that I know of, I made it up so it would be easier to discuss the topic.

Why is a CFDB so limiting?

You might have noticed how many times I noted differences between RDBMS and a CFDB. I think that it is the CFDB that is the hardest to understand, since it is so close, on the surface to the relational model. But it seems to suffer from so many limitations. No joins, no real querying capability (except by primary key), nothing like the richness that we get from a relational database. Hell, Sqlite or Access gives me more than that. Why is it so limited?

The answer is quite simple. A CFDB is designed to run on a large number of machines, and store huge amount of information. You literally cannot store that amount of data in a relational database, and even multi-machine relational databases, such as Oracle RAC will fall over and die very rapidly on the size of data and queries that a typical CFDB is handling easily.

Do you remember that I noted that CFDB is really all about removing abstractions? CFDB is what happens when you take a database, strip everything away that make it hard to run in on a cluster and see what happens.

The reason that CFDB don’t provide joins is that joins require you to be able to scan the entire data set. That requires either someplace that has a view of the whole database (resulting in a bottleneck and a single point of failure) or actually executing a query over all machines in the cluster. Since that number can be pretty high, we want to avoid that.

CFDB don’t provide a way to query by column or value because that would necessitate either an index of the entire data set (or just in a single column family) which in again, not practical, or running the query on all machines, which is not possible. By limiting queries to just by key, CFDB ensure that they know exactly what node a query can run on. It means that each query is running on a small set of data, making them much cheaper.

It requires a drastically different mode of thinking, and while I don’t have practical experience with CFDB, I would imagine that migrations using them are… unpleasant affairs, but they are one of the ways to get really high scalability out of your data storage.

Waiting expectantly to the commenters who would say that relational databases are the BOMB and that I have no idea what I am talking about and that I should read Codd and that no one really need to use this sort of stuff except maybe Google and even then only because Google has no idea how RDBMS work (except maybe the team that worked on AdWords).

<https://ayende.com/blog/4500/that-no-sql-thing-column-family-databases>

# **Graph databases explained**

For a long time, ‘big data’ has been a byword for our society’s **ongoing digitalization**. The wide availability of large quantities of data, however, also provides us with a few challenges: That is to say, rapidly growing, fast-paced and weakly structured volumes of data also require **high-performance IT solutions** so that they can be effectively analyzed and used.

One database model that can handle highly interconnected information is the graph database. It also provides an answer to the problems of the classical, relational database, which quickly comes up against its limits when handling large and complex data sets. Graph databases, then, rank among the modern database alternatives that are free of the traditional, relational approach, and that are brought together under the umbrella term NoSQL (‘Not only SQL’). But how exactly does a graph database work, and what advantages does its structure offer?

**Contents**

1. What is a graph database?
2. [How do queries work in a graph database?](https://www.ionos.com/digitalguide/hosting/technical-matters/graph-database/#c209949)
3. [Differentiation from relational databases and other NoSQL databases](https://www.ionos.com/digitalguide/hosting/technical-matters/graph-database/#c209950)
4. [What are graph databases used for?](https://www.ionos.com/digitalguide/hosting/technical-matters/graph-database/#c209951)
5. [The advantages and disadvantages of graph databases](https://www.ionos.com/digitalguide/hosting/technical-matters/graph-database/#c209952)
6. [An overview of the best-known graph databases](https://www.ionos.com/digitalguide/hosting/technical-matters/graph-database/#c209955)

## What is a graph database?

As its name suggests, a graph database is **modeled based on graphs**. These graphs represent complex, interconnected information as well as the relationships within it in a clear way, and they store this data as a large, coherent data set. The graphs are made up of **nodes –** clearly labeled and identifiable data entities and objects – and **edges**. The latter involves the relationships between the objects. Both components are **represented** visually**as points and lines**. Edges each have a start and end point, while each node always has a certain number of relationships to other nodes, whether incoming, outgoing, or undirected.

Established concepts for constructing such graph databases are the **labeled property graph** and the **resource description framework (RDF)**: With the former, certain properties are assigned to both the nodes and the edges. In the resource description framework (RDF), meanwhile, the modeling of the graph is regulated using **triples and quads**. Triples consist of three elements in the pattern **node-edge-node**. Quads complement triples **withadditional contextual information**, which makes it easier to organize the latter into groups.

## How do queries work in a graph database?

There is a wide range of query possibilities that can be exploited when using a graph database. The main reason for this is that there is no uniform query language. Unlike traditional models, graph databases count on **special algorithms** to fulfill their essential function: simplifying and speeding up complicated data queries.

Two of the most important algorithms are the **depth-first search**and the **breadth-first search**: The depth-first searches for the next node below in each case, while the breadth-first search moves from layer to layer. The algorithms make it possible to find graph patterns as well as direct and indirect adjacent nodes. Other algorithms make it possible to calculate the shortest path between two nodes, and to identify cliques (subsets of nodes) and hotspots (information that is particularly highly interconnected). One of the strengths of the graph database is that **relationships are stored in the database itself**, so they don’t need to be calculated in the query. This results in a high performance speed, even for complicated queries.

## Differentiation from relational databases and other NoSQL databases

[Relational databases](https://www.ionos.com/digitalguide/hosting/technical-matters/relational-databases/) have become established as the standard in databases since they first appeared in 1970. Unlike graph databases, they work **based on tables**that organize the relations of data sets, called tuples, into individual rows. In the columns, meanwhile, characteristics with varying attribute values can be illustrated. Except with regard to **structure** and the **composition**, their **functioning** is also fundamentally different from representation by graph. In order to be able to represent and store relationships with highly interconnected information, several tables must be laboriously linked and offset with one another. With large quantities of data, this can often prove time-consuming and expensive.

While table-based databases exclusively use the query language SQL (“structured query language”), the more modern NoSQL databases are increasingly moving away from this query language and the relational concept it is affiliated with – an approach that graph databases, as a member of the NoSQL family, also follow. Alongside graph databases, lots of other models, such as key-value databases, column-oriented databases and document-oriented databases also belong to this family. These principally process and store more structured and less interconnected data sets.

## What are graph databases used for?

Graph databases can be used for many different sectors and purposes. They allow interconnected information to be analyzed, and processes and connections to be understood, evaluated and made useful.

A typical example use of graph databases is in analyzing user relationships in social networks or users’ buying behavior in online shops. Targeted **product and friend suggestions** can be made based on different data and relationships, for example, allowing individual personal and product networks to be built up. Businesses also benefit from the possibility of creating **comprehensive customer profiles** based on information from search queries, click histories and other components. Graph databases are used in supply chain management to track all processes, from design right through to sales. Finally, the databases are used for**risk assessments**, **fraud detection**and **debugging**.

## The advantages and disadvantages of graph databases

The strength of a database can be measured using four principal factors: **Integrity, performance, efficiency** and **scalability**. The data query ought to become quicker and simpler – the main purpose of graph databases can be roughly summarized in this way. Where relational databases reach their capacity limits, the graph-based model is particularly agile, because complexity and the quantity of data don’t negatively influence the query process in this model.

Also, with the graph database model, **real facts can be stored in a natural way**. The structure used is very similar to human thinking, and this is why the links are so clear. Graph databases are not a complete solution, though. They are **limited**, for example, where **scalability** is concerned. As they are principally designed for one-tier architecture, growth represents a (mathematical) challenge. Plus, there is still no uniform query language.

An overview of the advantages and disadvantages of graph databases:

| Advantages | Disadvantages |
| --- | --- |
| Query speed only dependent on the number of concrete relationships, and not on the amount of data | Difficult to scale, as designed as one-tier architecture |
| Results in real time | No uniform query language |
| Clear and manageable representation of relationships |  |
| Flexible and agile structures |  |

Graph databases should not be considered generally to be an absolute better replacement for conventional databases. Relational structures remain entirely reasonable standard models, guaranteeing high data integrity and stability, and permitting flexible scalability. As so often, the same applies here: It all depends on the intended purpose!

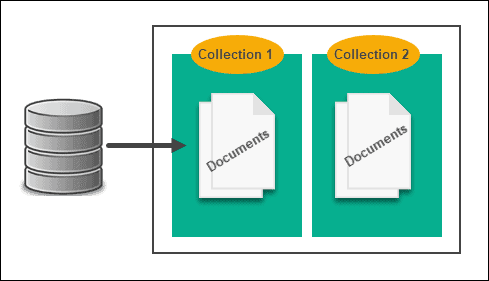
## An overview of the best-known graph databases

* **Neo4j**: Neo4j is the most popular graph database and is conceived as an open-source model.
* **Amazon Neptune**: This graph database can be used with the public cloud for Amazon Web Services and was released in 2018 as a high-performance database.
* **SAP Hana Graph**: With SAP Hana, the developer SAP has created a platform that builds upon a relational database management system and that is complemented by the integrated, graph-oriented model SAP Hana Graph.
* **OrientDB**: This graph database is one of the quickest models currently available.

<https://www.ionos.com/digitalguide/hosting/technical-matters/graph-database/>

**Document Database**

A document database is a type of NoSQL database which stores data as JSON documents instead of columns and rows. JSON is a native language used to both store and query data. These documents can be grouped together into collections to form database systems.



Each document consists of a number of key-value pairs. Here is an example of a document that consists of 4 key value pairs:

{

"ID" : "001",

"Book" : "Java: The Complete Reference",

"Genre" : "Reference work",

"Author" : "Herbert Schildt",

}

Using JSON enables app developers to store and query data in the same document-model format that they use to organize their app’s code. The object model can be converted into other formats, such as JSON, BSON and XML.

**Features of Document Databases**

Document databases provide fast queries, a structure well suited for handling big data, flexible indexing and a simplified method of maintaining the database. It’s efficient for web apps and has been fully integrated by large-scale IT companies like Amazon.

Although SQL databases have great stability and vertical power, they struggle with super-sized databases. Use cases that require immediate access to data, such as healthcare apps, are a better fit for document databases. Document databases make it easy to query data with the same document-model used to code the application.

Document Databases Use Cases

| General Use Cases |
| --- |
| User profiles | Extracting real-time big data |
| Book databases | Data of varying structures |
| Content management | Catalogs |
| Patients' data |  |

We'll cover some of the above-mentioned use cases in greater detail in the following sections.

Book Database

Both relational and NoSQL document systems are used to form a book database, although in different ways.

The relational approach would represent the relationship between books and authors via tables with IDs – an Author table and a Books table. It forces each author to have at least one entry in the Books table by disallowing null values.

By comparison, the document model lets you nest. It shows relationships more naturally and simply by ensuring that each author document has a property called Books, with an array of related book documents in the property. When you search for an author, the entire book collection appears.

Content Management

Developers use document databases to created video streaming platforms, blogs and similar services. Each file is stored as a single document and the database is easier to maintain as the service evolves over time. Significant data modifications, such as data model changes, require no downtime as no schema update is necessary.

Catalogs

Document databases are much more efficient than relational databases when it comes to storing and reading catalog files. Catalogs may have thousands of attributes stored and document databases provide fast reading times. In document databases, attributes related to a single product are stored in a single document. Modifying one product's attributes does not affect other documents.

Document Database Advantages and Disadvantages

Below are some key advantages and disadvantages of document databases:

| Document Database Advantages | Document Database Disadvantages |
| --- | --- |
| Schema-less | Consistency-Check Limitations |
| Faster creation and care | Atomicity weaknesses |
| No foreign keys | Security |
| Open formats |  |
| Built-in versioning |  |

The advantages and disadvantages are further explained in the sections below.

Advantages

Schema-less. There are no restrictions in the format and structure of data storage. This is good for retaining existing data at massive volumes and different structural states, especially in a continuously transforming system.

Faster creation and care. Minimal maintenance is required once you create the document, which can be as simple as adding your complex object once.

No foreign keys. With the absence of this relationship dynamic, documents can be independent of one another.

Open formats. A clean build process that uses XML, JSON and other derivatives to describe documents.

Built-in versioning. As your documents grow in size they can also grow in complexity. Versioning decreases conflicts.

Disadvantages

Consistency-Check Limitations. In the book database use case example above, it would be possible to search for books from a non-existent author. You could search the book collection and find documents that are not connected to an author collection.  
Each listing may also duplicate author information for each book. These inconsistencies aren’t significant in some contexts, but at upper-tier standards of RDB consistency audits, they seriously hamper database performance.

Atomicity weaknesses. Relational systems also let you modify data from one place without the need for JOINs. All new reading queries will inherit changes made to your data via a single command (such as updating or deleting a row).  
For document databases, a change involving two collections will require you to run two separate queries (per collection). This breaks atomicity requirements.

Security. Nearly half of web applications today actively leak sensitive data. Owners of NoSQL databases, therefore, need to pay careful attention to web app vulnerabilities.

https://phoenixnap.com/kb/document-database