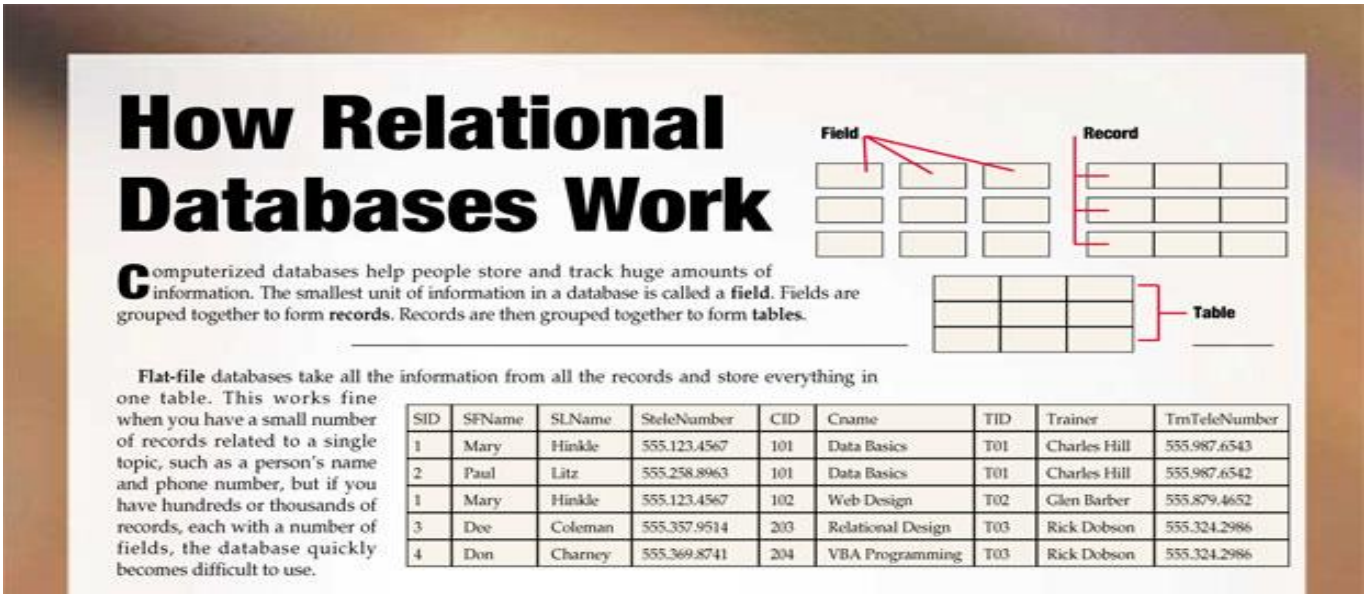
Types Of Databases

1. Relational Database

A relational database is a collection of data items with pre-defined relationships between them. These items are organized as a set of tables with columns and rows. Tables are used to hold information about the objects to be represented in the database. Each column in a table holds a certain kind of data and a field stores the actual value of an attribute. The rows in the table represent a collection of related values of one object or entity. Each row in a table could be marked with a unique identifier called a primary key, and rows among multiple tables can be made related using foreign keys. This data can be accessed in many different ways without reorganizing the database tables themselves.

In 1970, Edgar F. Codd, an Oxford-educated mathematician working at the IBM San Jose Research Lab, published a paper showing how information stored in large databases could be accessed without knowing how the information was structured or where it resided in the database.



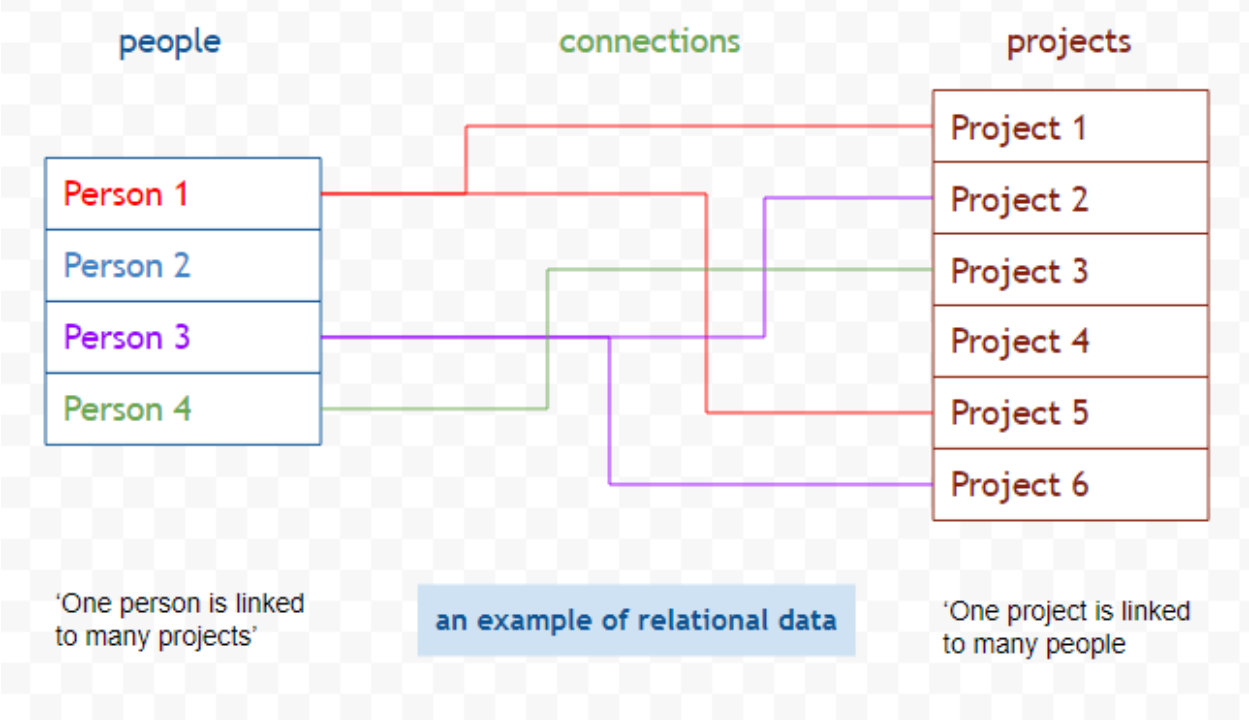
Until then, retrieving information required relatively sophisticated computer knowledge, or even the services of specialists who knew how to write programs to fetch specific information—a time-consuming and expensive task.

Databases that were used to retrieve the same information over and over, and in a predictable way—such as a bill of materials for manufacturing—were well established at the time. What Codd did was open the door to a new world of data independence. Users wouldn't have to be

specialists, nor would they need to know where the information was or how the computer retrieved it. They could now concentrate more on their businesses and less on their computers.

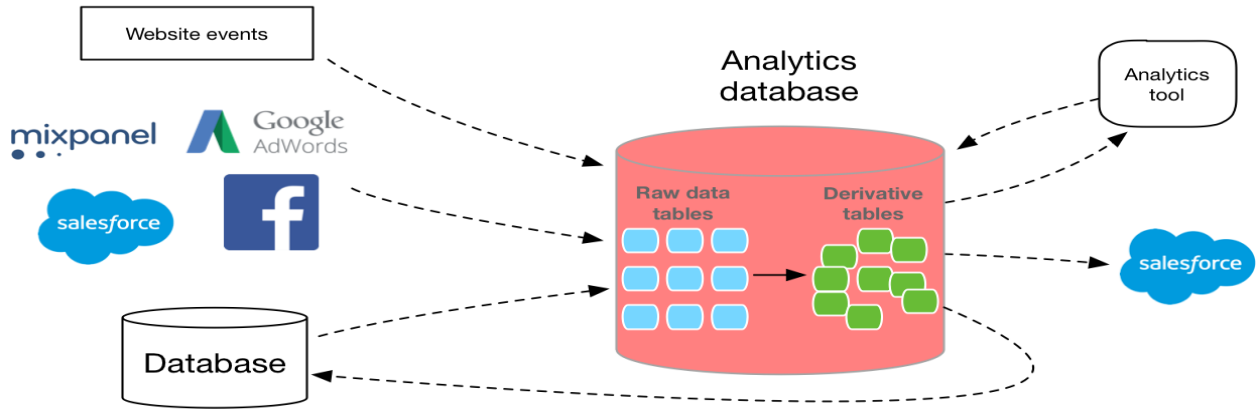
Codd called his paper, “A Relational Model of Data for Large Shared Data Banks.” Computer scientists called it a “revolutionary idea.”

Today, the ease and flexibility of relational databases have made them the predominant choice for financial records, manufacturing and logistical information, and personnel data. Most routine data transactions—accessing bank accounts, using credit cards, trading stocks, making travel reservations, buying things online—all use structures based on relational database theory.

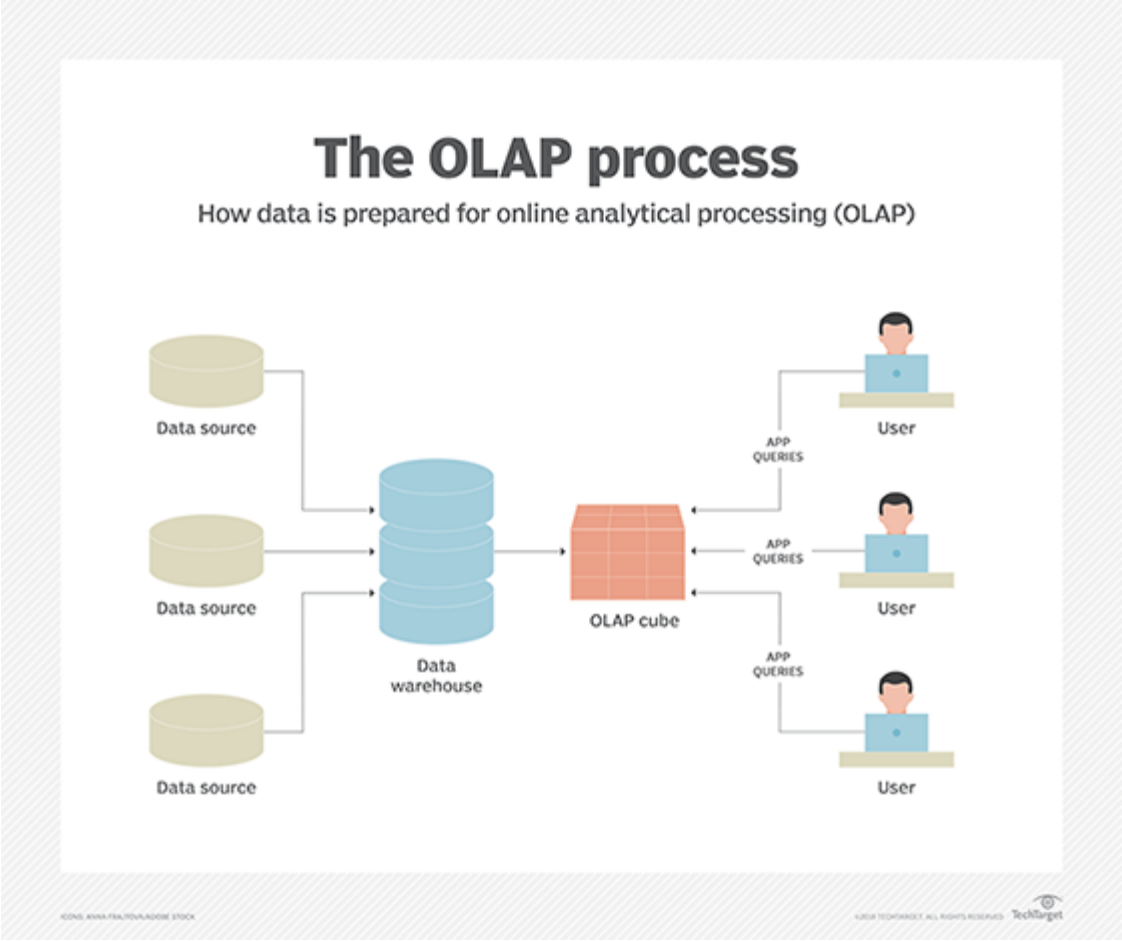


2. Analytical Database

An analytical database stores and manages big data, including business, market and customer data for business intelligence (BI) analysis. Analytical databases are specially optimized for faster queries and scalability.



FAQs



What Is an Analytical Database?

Analytical database software specializes in big data management for business applications and services. Analytical databases are optimized to provide quick query response times and advanced analytics. They are also more scalable than traditional databases and often times are columnar databases that can efficiently write and read data to and from hard disk storage in

order to speed up the time it takes to return a query. Analytical database features include column-based storage, in-memory loading of compressed data and the ability to search data through multiple attributes.

How Are Analytical Databases Used?

Analytic database software is designed to quickly analyze massive amounts of data, performing up to 1,000 times faster than an operational database for demanding analytical workloads. Business analysts, researchers, financial market analysts, big data analysts, geospatial analysts and data scientists rely on the high availability of analytical databases that can handle high data volumes.

The historical data in an analytical database is compared with operational data. Historical data refers to data that is not in real-time but could only be a few hours old. Contrasting analytical and operational data helps determine the best processes for transactions and other business or research decisions.

Examples of Analytical Databases

- Market data — Historical price and volume data for financial markets for testing trading strategies.
- Transactional data — Historical transactions that can include purchasing patterns for improved marketing.
- Sensor data — Historical data from sensors that monitor situations like the weather.
- Natural language data — Study of social media posts for research purposes.
- Process data — Study of processes to better understand logistics and find bottlenecks.
- Machine data — Software and hardware-generated data from products to improve efficiency.

Operational databases contain transactional data while analytical databases are designed for efficient analysis.

Analytical Database Benefits

Interest in analytical databases has risen in the last 15 years with the increased demand for tools that enable data processing in real-time from sources such as: IOT connected devices, mobile devices, remote sensors, biometrics devices and streaming video and media software. Here as some high level benefits of using an analytical database:

- Columnar data storage — A column versus row-based design which allows for very fast analysis of large sets of data point within a column. Traditional row-based designs cannot scale for massive amounts of data the way columns can.

- Efficient data compression — The columnar design of analytical databases allow for the most efficient version of data compression, which is how database space and speed is maximized.

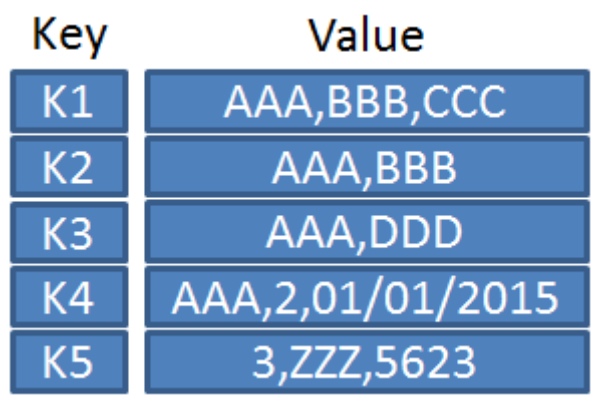
- Distributed workloads — Data is stored on a cluster of servers also called “nodes.” When data is stored across many different parallel servers, queries can be processed across the board. This allows for very efficient processing of large volumes of data.

Other analytical database benefits include:

- Horizontal scalability
- SQL compatibility
- Advanced math and statistical functionality

3. Key-Value Database

A **key–value database**, or **key–value store**, is a data storage paradigm designed for storing, retrieving, and managing associative arrays, and a data structure more commonly known today as a *dictionary* or *hash table*. Dictionaries contain a collection of *objects*, or *records*, which in turn have many different *fields* within them, each containing data. These records are stored and retrieved using a *key* that uniquely identifies the record, and is used to find the data within the database.



A table showing different formatted data values associated with different keys

Key	Value
Name	Joe Bloggs
Age	42
Occupation	Stunt Double
Height	175cm
Weight	77kg

Key–value databases work in a very different fashion from the better known relational databases (RDB). RDBs predefine the data structure in the database as a series of tables containing fields with well defined data types. Exposing the data types to the database program allows it to apply a number of optimizations. In contrast, key–value systems treat the data as a single opaque collection, which may have different fields for every record. This offers considerable flexibility and more closely follows modern concepts like object-oriented programming. Because optional values are not represented by placeholders or input parameters, as in most RDBs, key–value databases often use far less memory to store the same database, which can lead to large performance gains in certain workloads.[*citation needed*]

Performance, a lack of standardization and other issues limited key–value systems to niche uses for many years, but the rapid move to cloud computing after 2010 has led to a renaissance as part of the broader NoSQL movement. Some graph databases, such as ArangoDB,[1] are also key–value databases internally, adding the concept of the relationships (*pointers*) between records as a first class data type.

Types and notable examples[edit]

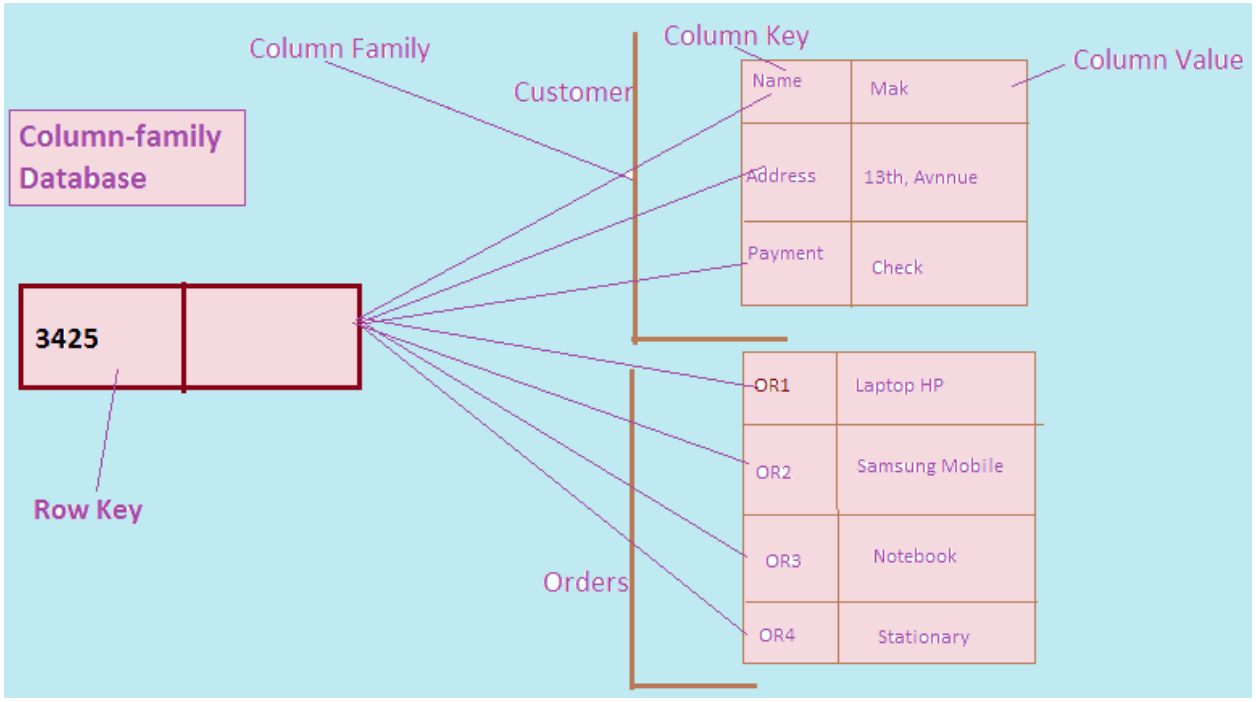
Key–value databases can use consistency models ranging from eventual consistency to serializability. Some support ordering of keys.

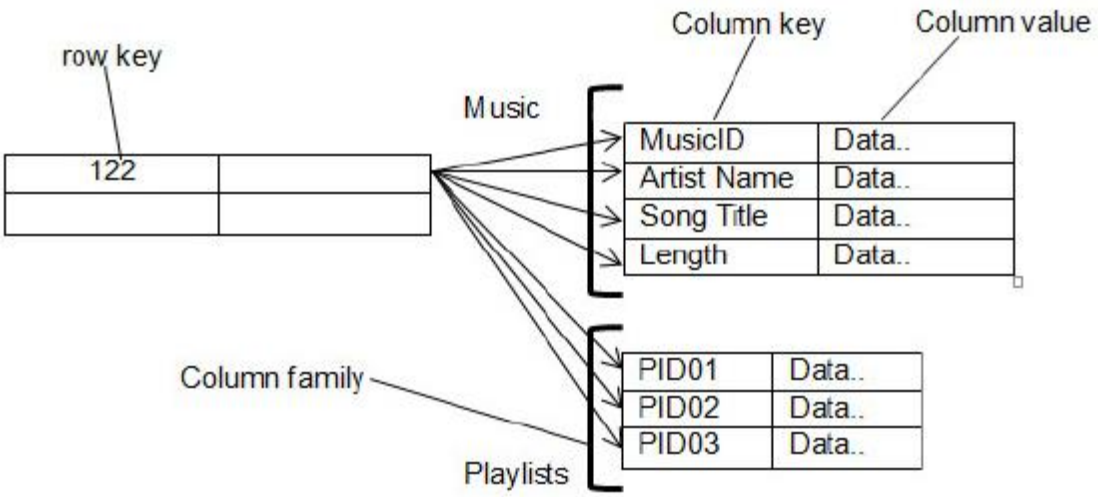
Some maintain data in memory (RAM), while others employ solid-state drives or rotating disks.

Every entity (record) is a set of key–value pairs. A key has multiple components, specified as an ordered list. The major key identifies the record and consists of the leading components of the key. The subsequent components are called minor keys. This organization is similar to a directory path specification in a file system (e.g., /Major/minor1/minor2/). The “value” part of the key–value pair is simply an uninterpreted string of bytes of arbitrary length.[2]

4. Column-Family Database

NoSQL column family database is another aggregate oriented database. In NoSQL column family database we have a single key which is also known as row key and within that, we can store multiple column families where each column family is a combination of columns that fit together.
Column family as a whole is effectively your aggregate. We use row key and column family name to address a column family.
It is, however, one of the most complicated aggregate databases but the gain we have in terms of retrieval time of aggregate rows. When we are taking these aggregates into the memory, instead of spreading across a lot of individual records we store the whole thing in one database in one go.
The database is designed in such a way that it clearly knows what the aggregate boundaries are. This is very useful when we run this database on the cluster.
As we know that aggregate binds the data together, hence different aggregates are spread across different nodes in the cluster.
Therefore, if somebody wants to retrieve the data, say about a particular order, then you need to go to one node in the cluster instead of shooting on all other nodes to pick up different rows and aggregate it.
Among the most popular column family NoSQL databases are Apache HBase and Cassandra.





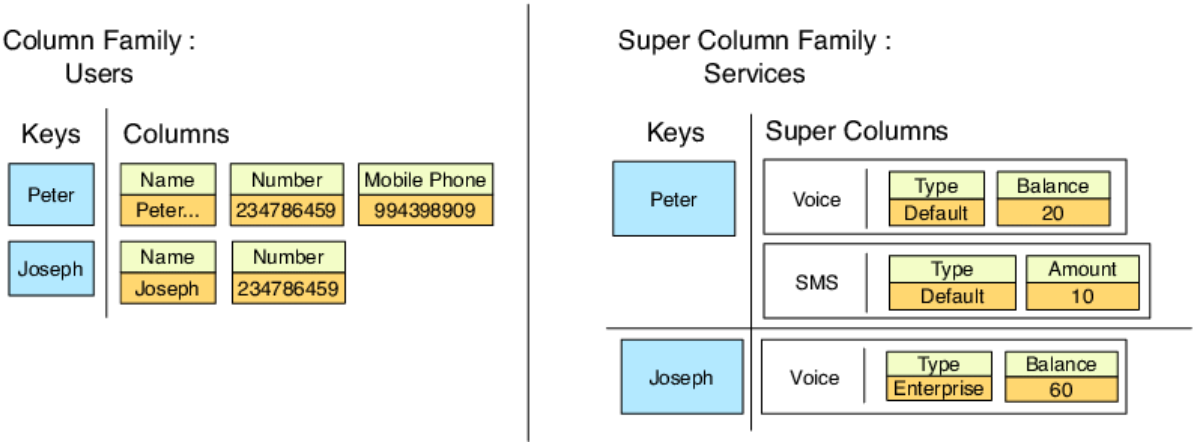
Aggregate orientation is not always a good thing.

Let us consider the user needs the revenue details by product. He does not care about the revenue by orders.
Effectively he wants to change the aggregate structure from order aggregate line item to produce aggregate line items. Therefore, the product becomes the root of the aggregate.
In a relational database, it is straightforward. We just query few tables and make joins and the result is there on your screen. But when it comes to aggregate orientation database it is a pain. We have to run different **MapReduce** jobs to rearrange your data into different aggregate forms and keep doing the incremental update on aggregated data in order to serve your business requirement, but this is very complicated.
Therefore, the aggregate oriented database has an advantage if most of the time you use the same aggregate to push data back and forth into the system. It is a disadvantage if you want to slice and dice data in different ways.

A **column family** is a database object that contains columns of related data. It is a tuple (pair) that consists of a key–value pair, where the key is mapped to a value that is a set of columns. In analogy with relational databases, a column family is as a "table", each key-value pair being a "row". Each column is a tuple (triplet) consisting of a column name, a value, and a timestamp. In a relational database table, this data would be grouped together within a table with other non-related data.

Two types of column families exist:

- Standard column family: contains only columns
- Super column family: contains a map of super columns



5. Graph Database

A graph database is defined as a specialized, single-purpose platform for creating and manipulating graphs. Graphs contain nodes, edges, and properties, all of which are used to represent and store data in a way that relational databases are not equipped to do.
Graph analytics is another commonly used term, and it refers specifically to the process of analyzing data in a graph format using data points as nodes and relationships as edges. Graph analytics requires a database that can support graph formats; this could be a dedicated graph database, or a converged database that supports multiple data models, including graph.

Graph database types

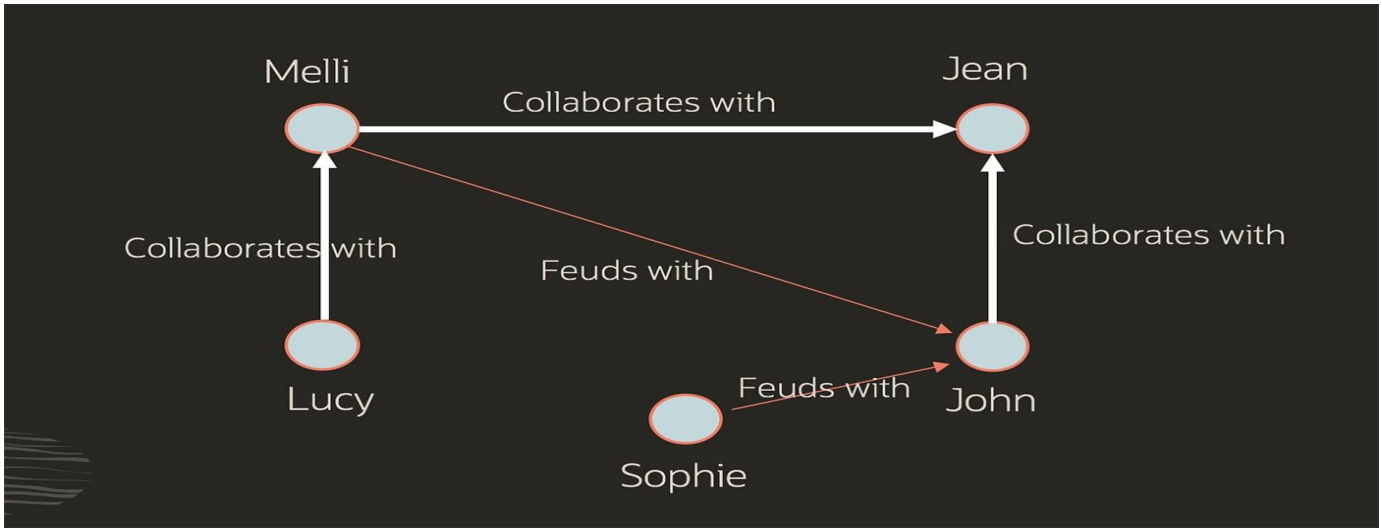
There are two popular models of graph databases: property graphs and RDF graphs. The property graph focuses on analytics and querying, while the RDF graph emphasizes data integration. Both types of graphs consist of a collection of points (vertices) and the connections between those points (edges). But there are differences as well.

There are two popular models of graph databases: property graphs and RDF graphs. The property graph focuses on analytics and querying, while the RDF graph emphasizes data integration. Both types of graphs consist of a collection of points (vertices) and the connections between those points (edges). But there are differences as well.

Property graphs

Property graphs are used to model relationships among data, and they enable query and data analytics based on these relationships. A property graph has vertices that can contain detailed information about a subject, and edges that denote the relationship between the vertices. The vertices and edges can have attributes, called properties, with which they are associated.

In this example, a set of colleagues and their relationships are represented as a property graph.



Because they are so versatile, property graphs are used in a broad range of industries and sectors, such as finance, manufacturing, public safety, retail, and many others.

RDF graphs

RDF graphs (RDF stands for Resource Description Framework) conform to a set of W3C (Worldwide Web Consortium) standards designed to represent statements and are best for representing complex metadata and master data. They are often used for linked data, data integration, and knowledge graphs. They can represent complex concepts in a domain, or provide rich semantics and inferencing on data.

In the RDF model a statement is represented by three elements: two vertices connected by an edge reflecting the subject, predicate and object of a sentence—this is known as an RDF triple. Every vertex and edge is identified by a unique URI, or Unique Resource Identifier. The RDF model provides a way to publish data in a standard format with well-defined semantics, enabling information exchange. Government statistics agencies, pharmaceutical companies, and healthcare organizations have adopted RDF graphs widely.

How graphs and graph databases work

Graphs and graph databases provide graph models to represent relationships in data. They allow users to perform “traversal queries” based on connections and apply graph algorithms to find patterns, paths, communities, influencers, single points of failure, and other relationships, which enable more efficient analysis at scale against massive amounts of data. The power of graphs is in analytics, the insights they provide, and their ability to link disparate data sources.

When it comes to analyzing graphs, algorithms explore the paths and distance between the vertices, the importance of the vertices, and clustering of the vertices. For example, to determine importance algorithms will often look at incoming edges, importance of neighboring vertices, and other indicators.

Graph algorithms—operations specifically designed to analyze relationships and behaviors among data in graphs—make it possible to understand things that are difficult to see with other methods. When it comes to analyzing graphs, algorithms explore the paths and distance between the vertices, the importance of the vertices, and clustering of the vertices. The algorithms will often look at incoming edges, importance of neighboring vertices, and other indicators to help determine importance. For example, graph algorithms can identify what individual or item is most connected to others in social networks or business processes. The algorithms can identify communities, anomalies, common patterns, and paths that connect individuals or related transactions.

Because graph databases explicitly store relationships, queries and algorithms utilizing the connectivity between vertices can be run in sub-seconds rather than hours or days. Users don't need to execute countless joins and the data can more easily be used for analysis and machine learning to discover more about the world around us.

Advantages of graph databases

The graph format provides a more flexible platform for finding distant connections or analyzing data based on things like strength or quality of relationship. Graphs let you explore and discover connections and patterns in social networks, IoT, big data, data warehouses, and also complex transaction data for multiple business use cases including fraud detection in banking, discovering connections in social networks, and customer 360. Today, graph databases are increasingly being used as a part of data science as a way to make connections in relationships clearer.

Because graph databases explicitly store the relationships, queries and algorithms utilizing the connectivity between vertices can be run in subseconds rather than hours or days. Users don't need to execute countless joins and the data can more easily be used for analysis and machine learning to discover more about the world around us.

Graph databases are an extremely flexible, extremely powerful tool. Because of the graph format, complex relationships can be determined for deeper insights with much less effort. Graph databases generally run queries in languages such as Property Graph Query Language (PGQL). The example below shows the same query in PGQL and SQL.

As seen in the above example, the PGQL code is simpler and much more efficient. Because graphs emphasize relationships between data, they are ideal for several different types of analyses. In particular, graph databases excel at:

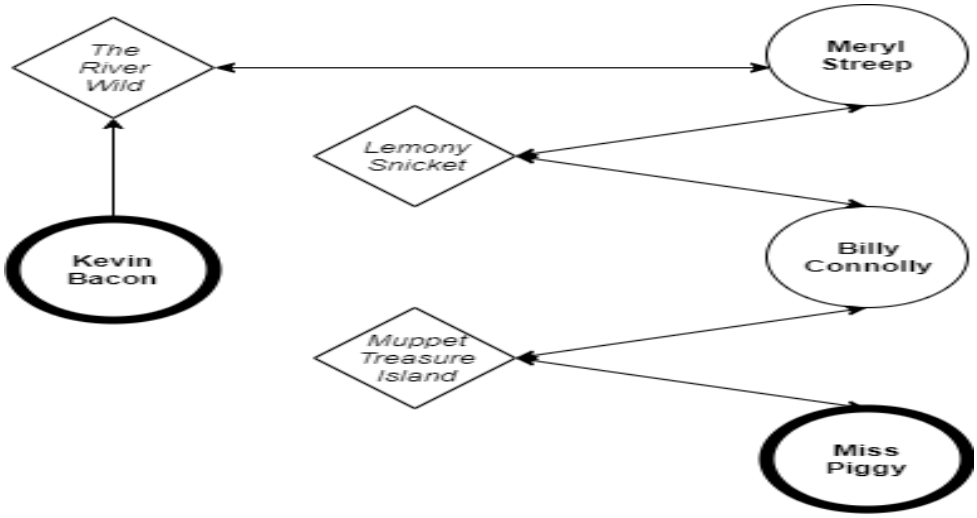
- Finding the shortest path between two nodes
- Determining the nodes that create the most activity/influence
- Analyzing connectivity to identify the weakest points of a network

- Analyzing the state of the network or community based on connection distance/density in a group

How graph databases and graph analytics work

A simple example of graph databases in action is the image below, which shows a visual representation of the popular party game "Six Degrees of Kevin Bacon." For those new to it, this game involves coming up with connections between Kevin Bacon and another actor based on a chain of mutual films. This emphasis on relationships makes it the ideal way to demonstrate graph analytics.

Imagine a data set with two categories of nodes: every film ever made and every actor that has been in those films. Then, using graph, we run a query asking to connect Kevin Bacon to Muppet icon Miss Piggy. The result would be as follows:



In this example, the available nodes (vertices) are both actors and films and the relationships (edges) are the status of "acted in." From here, the query returns the following results:

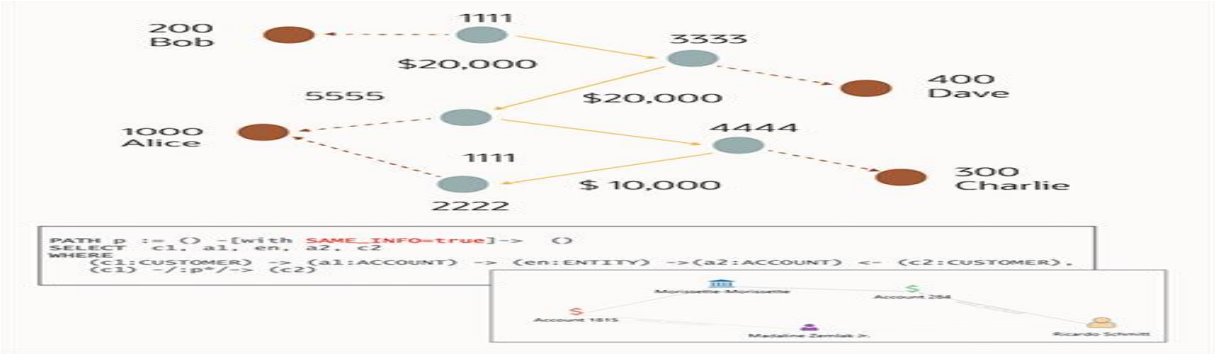
- Kevin Bacon acted in The River Wild with Meryl Streep.
- Meryl Streep acted in Lemony Snicket's A Series of Unfortunate Events with Billy Connolly.
- Billy Connolly acted in Muppet Treasure Island with Miss Piggy.

Graph databases can query many different relationships for this Kevin Bacon example, such as:

- "What is the shortest chain to connect Kevin Bacon to Miss Piggy?" (shortest path analysis, as used in the Six Degrees game above)
- "Who has worked with the largest number of actors?" (degree centrality)
- "What is the average distance between Kevin Bacon and all other actors?" (closeness centrality)

This is, of course, a more amusing example than most uses of graph analytics. But this approach works in nearly all big data—any situation where large numbers of records show a natural connectivity with each other. Some of the most popular ways to use graph analytics is for analyzing social networks, communication networks, website traffic and usage, real-world road data, and financial transactions and accounts.

Graph database use case: money laundering



Conceptually, money laundering is simple. Dirty money is passed around to blend it with legitimate funds and then turned into hard assets. This is the kind of process that was used in the Panama Papers analysis.

More specifically, a circular money transfer involves a criminal who sends large amounts of fraudulently obtained money to himself or herself—but hides it through a long and complex series of valid transfers between "normal" accounts. These "normal" accounts are actually accounts created with synthetic identities. They typically share certain similar information because they are generated from stolen identities (email addresses, addresses, etc.) and it's this related information that makes graph analysis such a good fit to make them reveal their fraudulent origins.

To make fraud detection simpler, users can create a graph from transactions between entities as well as entities that share some information, including the email addresses, passwords, addresses, and more. Once a graph is created, running a simple query will find all customers with accounts who have similar information, and reveal which accounts are sending money to each other.

Graph database use case: social media analysis

Graph databases can be used in many different scenarios, but it is commonly used to analyze social networks. In fact, social networks make the ideal use case as they involve a heavy volume of nodes (user accounts) and multi-dimensional connections (engagements in many different directions). A graph analysis for a social network can determine:

- How active are users? (number of nodes)
- Which users have the most influence? (density of connections)

- Who has the most two-way engagement? (direction and density of connections)

  However, this information is useless if it has been unnaturally skewed by bots. Fortunately, graph analytics can provide an excellent means for identifying and filtering out bots.

  In a real-world use case, the Oracle team used Oracle Marketing Cloud to evaluate social media advertising and traction—specifically, to identify fake bot accounts that skewed data. The most common behavior by these bots involved retweet target accounts, thus artificially inflating their popularity. A simple pattern analysis allowed for a look using retweet count and density of connections to neighbors. Naturally popular accounts showed different relationships with neighbors compared to bot-driven accounts.

6. Document Database

A Document Database (Also known as a document-oriented database or a document store) is a database that store information in documents.

Document databases offer a variety of advantages, including:

- An intuitive data model that is fast and easy for developers to work with.
- A flexible schema that allows for the data model to evolve as application needs change.
- The ability to horizontally scale out.

Because of these advantages, document databases are general-purpose databases that can be used in a variety of use cases and industries.

Document databases are considered to be non-relational (or NoSQL) databases. Instead of storing data in fixed rows and columns, document databases use flexible documents. Document databases are the most popular alternative to tabular, relational databases. Learn more about NoSQL databases.

What are documents?

A document is a record in a document database. A document typically stores information about one object and any of its related metadata.

Documents store data in field-value pairs. The values can be a variety of types and structures, including strings, numbers, dates, arrays, or objects. Documents can be stored in formats like JSON, BSON, and XML

What Are Document Databases

A document database is a NoSQL data stores that is designed to store and query data as JSON-like documents. The data in document databases is stored as documents with their metadata. The document stored is in key/value pair where the key is the unique identifier of the document. Unlike relational databases, document databases are faster to load, access, and parse.

Document database are also referred as document database management systems, document-oriented databases, or document store database.

Here are the key characteristics of document databases:

1. Document DBMSs are NoSQL databases.
2. Document DBMSs use key/value to store and access documents data.
3. Document DBMSs have a flexible schema that can be different for each document. For example, one document can be an Author profile, while other document can be a blog.
4. Common examples of document DBMS include JSON, XML docs, Catalogs, serialized PDFs and Excel docs, Profile data, and serialized objects.

Document database example

Here is a document that stores a book data. As you can see from this document, it's a JSON document that has tags and values that defines a book including year published, book title, author, release date, publisher, and price.

```
[{
   "year": 2001,
   "title": "A Programmer's Guide to ADO.NET",
   "info": {
      "author": "Mahesh Chand",
      "release_date": "2001-02-01",
      "publisher": "APress",
      "price": "44.95",
      "image_url": "ADOBook.jpg"
   }
}, {
   "year": 2003,
   "title": "GDI+ Programming",
   "info": {
      "author": "Mahesh Chand",
      "release_date": "2003-03-01",
      "publisher": "Addison Wesley",
      "price": "49.95",
      "image_url": "GDIPlusBook.jpg"
```

```
20.     }
21. }]
```

When to use document databases

Traditional relational DBMSs are not designed to provide efficient access to large documents or unstructured data. In case of catalogs, or profiles, or document storages, we don't need structured design. For example, storing a document in a CMS does not require a structured format.

Document databases are designed to store large documents in a key/value store that are easy to search and access. The entire document is read into a memory object that is easy to read and present.

User profiles, content management systems, and catalogs are some common use case of document DBMS. One of the perfect example of use of a document database is storing C# Corner articles in a document DB, rather than a DRBMS.