I'm not a robot

A graph database (GDB) is a database that uses graph structures for storing data. It uses nodes, edges, and properties instead of tables or documents to represent and store data. The edges represent relationships between the nodes. This helps in retrieving data more easily and, in many cases, with one operation. Graph databases are commonly referred to as NoSQL. Ex: Neo4j, Amazon Neptune, ArangoDB etc.Representation:The graph database is based on graph theory. The data is stored in the nodes of the graph and the relationship between the data are represented by the edges between the nodes.graph representation of dataWhen do we need Graph Database?1. It solves Many-To-Many relationship problemsIf we have friends of friends and stuff like that, these are many to many relationships.Used when the query in the relational database is very complex.2. When relationships between data elements are more importantFor example- there is a profile and the profile has some specific information in it but the major selling point is the relationship between these different profiles that is how you get connected within a network.In the same way, if there is data element such as user data element inside a graph database there could be multiple user data elements but the relationship is what is going to be the factor for all these data elements which are stored inside the graph database.3. Low latency with large scale dataWhen you add lots of relationships in the relational database, the data sets are going to be huge and when you query it, the complexity is going to be more complex and it is going to be more than a usual time. However, in graph database, it is specifically designed for this particular purpose and one can query relationship with ease. Why do Graph Databases matter? Because graphs are good at handling relationships, some databases store data in the form of a graph. Example We have a social network in which five friends are all connected. These friends are Anay, Bhagya, Chaitanya, Dilip, and Erica. A graph database that will store their personal information may look something like this: idfirst namelast nameemailphone1AnayAgarwalanay@example.net555-111-55552BhagyaKumarbhagya@example.net555-222-55553ChaitanyaNayakchaitanya@example.net555-333-55554DilipJaindilip@example.net555-444-55555EricaEmmanuelerica@example.net555-555-5555Now, we will also a need another table to capture the friendship/relationship between users/friends. Our friendship table will look something like this:user_idfriend_id12131415212324253132343541424345515253 54We will avoid going deep into the Database(primary key & foreign key) theory. Instead just assume that the friendship table uses id's of both the friends. Assume that our social network here has a feature that allows every user to see the personal information of his/her friends. So, If Chaitanya were requesting information then it would mean she needs information about Anay, Bhagya, Dilip and Erica. We will approach this problem the traditional way(Relational database). We must first identify Chaitanya's id in the User's table:idfirst namelast nameemailphone3ChaitanyaNayakchaitanya@example.net555-333-5555Now, we'd look for all tuples in friendship table where the user_id is 3. Resulting relation would be something like this:Now, let's analyse the time taken in this Relational database approach. This will be approximately log(N) times where N represents the number of tuples in friendship table or number of relations. Here, the database maintains the rows in the order of id's. So, in general for 'M' no of queries, we have a time complexity of M*log(N) Only if we had used a graph database approach, the total time complexity would have been O(N). Because, once we've located Cindy in the database, we have to take only a single step for finding her friends. Here is how our query would be executed: Advantages: Frequent schema changes, managing volume of data, real-time query response time, and more intelligent data activation requirements are done by graph model.Disadvantages: Note that graph databases aren’t always the best solution for an application. We will need to assess the needs of application before deciding the architecture.Limitations of Graph Databases:Graph Databases may not be offering better choice over the NoSQL variations.If application needs to scale horizontally this may introduces poor performance.Not very efficient when it needs to update all nodes with a given parameter. Share — copy and redistribute the material in any medium or format for any purpose, even commercially. Adapt — remix, transform, and build upon the material for any purpose, even commercially. The licensor cannot revoke these freedoms as long as you follow the license terms. Attribution — You must give appropriate credit , provide a link to the license, and indicate if changes were made . You may do so in any reasonable manner, but not in any way that suggests the licensor endorses you or your use. ShareAlike — If you remix, transform, or build upon the material, you must distribute your contributions under the same license as the original. No additional restrictions — You may not apply legal terms or technological measures that legally restrict others from doing anything the license permits. You do not have to comply with the license for elements of the material in the public domain or where your use is permitted by an applicable exception or limitation . No warranties are given. The license may not give you all of the permissions necessary for your intended use. For example, other rights such as publicity, privacy, or moral rights may limit how you use the material. Graph databases are structured through nodes and relationships. Nodes are entities in the graph which can: Relationships provide named connections between two nodes (e.g., Person - LOVES - Person) and they: Must always have a start node, an end node, and exactly one type. Must have a direction. Can have properties, like nodes. Nodes can have multiple relationships of various types without sacrificing performance. In summary, nodes and relationships are an efficient and flexible way to store data since they allow you to: Create traversals in big graphs for both depth and breadth. Scale-up your database to billions of nodes. Design flexible property graph data models that can adapt over time. Database using graph structures for queries This article has multiple issues. Please help improve it or discuss these issues on the talk page. (Learn how and when to remove these messages) This article possibly contains original research. Please improve it by verifying the claims made and adding inline citations. Statements consisting only of original research should be removed. (August 2016) (Learn how and when to remove this message) This article relies excessively on references to primary sources. Please improve this article by adding secondary or tertiary sources. Find sources: "Graph database" – news · newspapers · books · scholar · JSTOR (August 2016) (Learn how and when to remove this message) This article needs additional citations for verification. Please help improve this article by adding citations to reliable sources. Unsourced material may be challenged and removed.Find sources: "Graph database" – news · newspapers · books · scholar · JSTOR (August 2016) (Learn how and when to remove this message) (Learn how and when to remove this message) A graph database (GDB) is a database that uses graph structures for semantic queries with nodes, edges, and properties to represent and store data.[1] A key concept of the system is the graph (or edge or relationship). The graph relates the data items in the store to a collection of nodes and edges, the edges representing the relationships between the nodes. The relationships allow data in the store to be linked together directly and, in many cases, retrieved with one operation. Graph databases hold the relationships between data as a priority. Querying relationships is fast because they are perpetually stored in the database. Relationships can be intuitively visualized using graph databases, making them useful for heavily inter-connected data.[2] Graph databases are commonly referred to as a NoSQL database. Graph databases are similar to 1970s network model databases in that both represent general graphs, but network-model databases operate at a lower level of abstraction[3] and lack easy traversal over a chain of edges.[4] The underlying storage mechanism of graph databases can vary. Relationships are first-class citizens in a graph database and can be labelled, directed, and given properties. Some depend on a relational engine and store the graph data in a table (although a table is a logical element, therefore this approach imposes a level of abstraction between the graph database management system and physical storage devices). Others use a key–value store or document-oriented database for storage, making them inherently NoSQL structures. As of 2021[update], no graph query language has been universally adopted in the same way as SQL was for relational databases, and there are a wide variety of systems, many of which are tightly tied to one product. Some early standardization efforts led to multi-vendor query languages like Gremlin, SPARQL, and Cypher. In September 2019 a proposal for a project to create a new standard graph query language (ISO/IEC 39075 Information Technology — Database Languages — GQL) was approved by members of ISO/IEC Joint Technical Committee 1(ISO/IEC JTC 1). GQL is intended to be a declarative database query language, like SQL. In addition to having query language interfaces, some graph databases are accessed through application programming interfaces (APIs). Graph databases differ from graph compute engines. Graph databases are technologies that are translations of the relational online transaction processing (OLTP) databases. On the other hand, graph compute engines are used in online analytical processing (OLAP) for bulk analysis.[5] Graph databases attracted considerable attention in the 2000s, due to the successes of major technology corporations in using proprietary graph databases,[6] along with the introduction of open-source graph databases. One study concluded that an RDBMS was "comparable" in performance to existing graph analysis engines at executing graph queries.[7] In the mid-1960s, navigational databases such as IBM's IMS supported tree-like structures in its hierarchical model, but the strict tree structure could be circumvented with virtual records.[8][9] Graph structures could be represented in network model databases from the late 1960s. CODASYL, which had defined COBOL in 1959, defined the Network Database Language in 1969. Labeled graphs could be represented in graph databases from the mid-1980s, such as the Logical Data Model.[10][11] Commercial object databases (ODBMSs) emerged in the early 1990s. In 2000, the Object Data Management Group published a standard language for defining object and relationship (graph) structures in their ODMG'93 publication. Several improvements to graph databases appeared in the early 1990s, accelerating in the late 1990s with endeavors to index web pages. In the mid-to-late 2000s, commercial graph databases with ACID guarantees such as Neo4j and Oracle Spatial and Graph became available. In the 2010s, commercial ACID graph databases that could be scaled horizontally became available. Further, SAP HANA brought in-memory and columnar technologies to graph databases.[12] Also in the 2010s, multi-model databases that supported graph models (and other models such as relational database or document-oriented database) became available, such as OrientDB, ArangoDB, and MarkLogic (starting with its 7.0 version). During this time, graph databases of various types have become especially popular with social network analysis with the advent of social media companies. Also during the decade, cloud-based graph databases such as Amazon Neptune and Neo4j AuraDB became available. Graph databases portray the data as it is viewed conceptually. This is accomplished by transferring the data into nodes and its relationships into edges. A graph database is a database that is based on graph theory. It consists of a set of objects, which can be a node or an edge. Nodes represent entities or instances such as people, businesses, accounts, or any other item to be tracked. They are roughly the equivalent of a record, relation, or row in a relational database, or a document in a document-store database. Edges, also termed graphs or relationships, are the lines that connect nodes to other nodes; representing the relationship between them. Meaningful patterns emerge when examining the connections and interconnections of nodes, properties and edges. The edges can either be directed or undirected. In an undirected graph, an edge connecting two nodes has a single meaning. In a directed graph, the edges connecting two different nodes have different meanings, depending on their direction. Edges are the key concept in graph databases, representing an abstraction that is not directly implemented in a relational model or a document-store model. Properties are information associated to nodes. For example, if Wikipedia were one of the nodes, it might be tied to properties such as website, reference material, or words that starts with the letter w, depending on which aspects of Wikipedia are germane to a given database. An example of a Labeled-property graph A labeled-property graph model is represented by a set of nodes, relationships, properties, and labels. Both nodes of data and their relationships are named and can store properties represented by key–value pairs. Nodes can be labelled to be grouped. The edges representing the relationships have two qualities: they always have a start node and an end node, and are directed;[13] making the graph a directed graph. Relationships can also have properties. This is useful in providing additional metadata and semantics to relationships of the nodes.[14] Direct storage of relationships allows a constant-time traversal.[15] Main article: RDF (computer science) An example RDF graph In an RDF graph model, each addition of information is represented with a separate node. For example, imagine a scenario where a user has to add a name property for a person represented as a distinct node in the graph. In a labeled-property graph model, this would be done with an addition of a name property into the node of the person. However, in an RDF, the user has to add a separate node called hasName connecting it to the original person node. Specifically, an RDF graph model is composed of nodes and arcs. An RDF graph notation or a statement is represented by: a node for the subject, a node for the object, and an arc for the predicate. A node may be left blank, a literal and/or be identified by a URI. An arc may also be identified by a URI. A literal for a node may be of two types: plain (untyped) and typed. A plain literal has a lexical form and optionally a language tag. A typed literal is made up of a string with a URI that identifies a particular datatype. A blank node may be used to accurately illustrate the state of the data when the data does not have a URI.[16] Graph databases are a powerful tool for graph-like queries. For example, computing the shortest path between two nodes in the graph. Other graph-like queries can be performed over a graph database in a natural way (for example graph's diameter computations or community detection). Graphs are flexible, meaning it allows the user to insert new data into the existing graph without loss of application functionality. There is no need for the designer of the database to plan out extensive details of the database's future use cases. The underlying storage mechanism of graph databases can vary. Some depend on a relational engine and "store" the graph data in a table (although a table is a logical element, therefore this approach imposes another level of abstraction between the graph database, the graph database management system and the physical devices where the data is actually stored). Others use a key–value store or document-oriented database for storage, making them inherently NoSQL structures. A node would be represented as any other document store, but edges that link two different nodes hold special attributes inside its document; a _from and _to attributes. Data lookup performance is dependent on the access speed from one particular node to another. Because index-free adjacency enforces the nodes to have direct physical RAM addresses and physically point to other adjacent nodes, it results in a fast retrieval. A native graph system with index-free adjacency does not have to move through any other type of data structures to find links between the nodes. Directly related nodes in a graph are stored in the cache once one of the nodes are retrieved, making the data lookup even faster than the first time a user fetches a node. However, such advantage comes at a cost. Index-free adjacency sacrifices the efficiency of queries that do not use graph traversals. Native graph databases use index-free adjacency to process CRUD operations on the stored data. Multiple categories of graphs by kind of data have been recognised. Gartner suggests the five broad categories of graphs:[17] Social graph: this is about the connections between people; examples include Facebook, Twitter, and the idea of six degrees of separation Intent graph: this deals with reasoning and motivation. Consumption graph: also known as the "payment graph", the consumption graph is heavily used in the retail industry. E-commerce companies such as Amazon, eBay and Walmart use consumption graphs to track the consumption of individual customers. Interest graph: this maps a person's interests and is often complemented by a social graph. It has the potential to follow the previous revolution of web organization by mapping the web by interest rather than indexing webpages. Mobile graph: this is built from mobile data. Mobile data in the future may include data from the web, applications, digital wallets, GPS, and Internet of Things (IoT) devices. Since Edgar F. Codd's 1970 paper on the relational model,[18] relational databases have been the de facto industry standard for large-scale data storage systems. Relational models require a strict schema and data normalization which separates data into many tables and removes any duplicate data within the database. Data is normalized in order to preserve data consistency and support ACID transactions. However this imposes limitations on how relationships can be queried. One of the relational model's design motivations was to achieve a fast row-by-row access.[18] Problems arise when there is a need to form complex relationships between the stored data. Although relationships can be analyzed with the relational model, complex queries performing many join operations on many different attributes over several tables are required. In working with relational models, foreign key constraints should also be considered when retrieving relationships, causing additional overhead. Compared with relational databases, graph databases are often faster for associative data sets[citation needed] and map more directly to the structure of object-oriented applications. They can scale more naturally[citation needed] to large datasets as they do not typically need join operations, which can often be expensive. As they depend less on a rigid schema, they are marketed as more suitable to manage ad hoc and changing data with evolving schemas. Conversely, relational database management systems are typically faster at performing the same operation on large numbers of data elements, permitting the manipulation of the data in its natural structure. Despite the graph databases' advantages and recent popularity over [citation needed] relational databases, it is recommended the graph model itself should not be the sole reason to replace an existing relational database. A graph database may become relevant if there is an evidence for performance improvement by orders of magnitude and lower latency.[19] The relational model gathers data together using information in the data. For example, one might look for all the "users" whose phone number contains the area code "311". This would be done by searching selected datastores, or tables, looking in the selected phone number fields for the string "311". This can be a time-consuming process in large tables, so relational databases offer indexes, which allow data to be stored in a smaller sub-table, containing only the selected data and a unique key (or primary key) of the record. If the phone numbers are indexed, the same search would occur in the smaller index table, gathering the keys of matching records, and then looking in the main data table for the records with those keys. Usually, a table is stored in a way that allows a lookup via a key to be very fast.[20] Relational databases do not inherently contain the idea of fixed relationships between records. Instead, related data is linked to each other by storing one record's unique key in another record's data. For example, a table containing email addresses for users might hold a data item called userpk, which contains the primary key of the user record it is associated with. In order to link users and their email addresses, the system first looks up the selected user records primary keys, looks for those keys in the userpk column in the email table (or, more likely, an index of them), extracts the email data, and then links the user and email records to make composite records containing all the selected data. This operation, termed a join, can be computationally expensive. Depending on the complexity of the query, the number of joins, and indexing various keys, the system may have to search through multiple tables and indexes and then sort it all to match it together.[20] In contrast, graph databases directly store the relationships between records. Instead of an email address being found by looking up its user's key in the userpk column, the user record contains a pointer that directly refers to the email address record. That is, having selected a user, the pointer can be followed directly to the email records, there is no need to search the email table to find the matching records. This can eliminate the costly join operations. For example, if one searches for all of the email addresses for users in area code "311", the engine would first perform a conventional search to find the users in "311", but then retrieve the email addresses by following the links found in those records. A relational database would first find all the users in "311", extract a list of the primary keys, perform another search for any records in the email table with those primary keys, and link the matching records together. For these types of common operations, graph databases would theoretically be faster.[20] The true value of the graph approach becomes evident when one performs searches that are more than one level deep. For example, consider a search for users who have "subscribers" (a table linking users to other users) in the "311" area code. In this case a relational database has to first search for all the users with an area code in "311", then search the subscribers table for any of those users, and then finally search the users table to retrieve the matching users. In contrast, a graph database would search for all the users in "311", then follow the backlinks through the subscriber relationship to find the subscriber users. This avoids several searches, look-ups, and the memory usage involved in holding all of the temporary data from multiple records needed to construct the output. In terms of big O notation, this query would be O (log n) + O (1) {\displaystyle O(\log n)+O(1)} time – i.e., proportional to the logarithm of the size of the data. In contrast, the relational version would be multiple O (log n) {\displaystyle O(\log n)} lookups, plus the O (n) {\displaystyle O(n)} time needed to join all of the data records.[20] The relative advantage of graph retrieval grows with the complexity of a query. For example, one might want to know "that movie about submarines with the actor who was in that movie with that other actor that played the lead in Gone With the Wind". This first requires the system to find the actors in Gone With the Wind, find all the movies they were in, find all the actors in all of those movies who were not the lead in Gone With the Wind, and then find all of the movies they were in, finally filtering that list to those with descriptions containing "submarine". In a relational database, this would require several separate searches through the movies and actors tables, doing another search on submarine movies, finding all the actors in those movies, and then comparing the (large) collected results. In contrast, the graph database would walk from Gone With the Wind to Clark Gable, gather the links to the movies he has been in, gather the links out of those movies to other actors, and then follow the links out of those actors back to the list of movies. The resulting list of movies can then be searched for "submarine". All of this can be done via one search.[21] Properties add another layer of abstraction to this structure that also improves many common queries. Properties are essentially labels that can be applied to any record, or in some cases, edges as well. For example, one might label Clark Gable as "actor", which would then allow the system to quickly find all the records that are actors, as opposed to director or camera operator. If labels on edges are allowed, one could also label the relationship between Gone With the Wind and Clark Gable as "lead", and by performing a search on people that are "lead" "actor" in the movie Gone With the Wind, the database would produce Vivien Leigh, Olivia de Havilland and Clark Gable. The equivalent SQL query would have to rely on added data in the table linking people and movies, adding more complexity to the query syntax. These sorts of labels may improve search performance under certain circumstances, but are generally more useful in providing added semantic data for end users.[21] Relational databases are very well suited to flat data layouts, where relationships between data are only one or two levels deep. For example, an accounting database might need to look up all the line items for all the invoices for a given customer, a three-join query. Graph databases are aimed at datasets that contain many more links. They are especially well suited to social networking systems, where the "friends" relationship is essentially unbounded. These properties make graph databases naturally suited to types of searches that are increasingly common in online systems, and in big data environments. For this reason, graph databases are becoming very popular for large online systems like Facebook, Google, Twitter, and similar systems with deep links between records. To further illustrate, imagine a relational model with two tables: a people table (which has a person_id and person_name column) and a friend table (with friend_id and person_id, which is a foreign key from the people table). In this case, searching for all of Jack's friends would result in the following SQL query. SELECT p2.person_name FROM people p1 JOIN friend ON (p1.person_id = friend.person_id) JOIN people p2 ON (p2.person_id = friend.friend_id) WHERE p1.person_name = 'Jack'; The same query may be translated into -- Cypher, a graph database query languageMATCH (p1:person {name: 'Jack'})-[:FRIEND_WITH]-(p2:person) RETURN p2.name SPARQL, an RDF graph database query language standardized by W3C and used in multiple RDF Triple and Quad stores Long form PREFIX foaf: < SELECT ?name WHERE { ?s a foaf:Person . ?s foaf:name "Jack" . ?s foaf:knows ?o . ?o foaf:name ?name . } Short form PREFIX foaf: < SELECT ?name WHERE { ?s foaf:name "Jack" ; foaf:knows ?o . ?o foaf:name ?name . } SPASQL, a hybrid database query language, that extends SQL with SPARQLSELECT people.name FROM (SPARQL PREFIX foaf: < SELECT ?name WHERE { ?s foaf:name "Jack" ; foaf:knows ?o . ?o foaf:name ?name . }) AS people ; The above examples are a simple illustration of a basic relationship query. They condense the idea of relational models' query complexity that increases with the total amount of data. In comparison, a graph database query is easily able to sort through the relationship graph to present the results. There are also results that indicate simple, condensed, and declarative queries of the graph databases do not necessarily provide good performance in comparison to the relational databases. While graph databases offer an intuitive representation of data, relational databases offer better results when set operations are needed.[15] The following is a list of notable graph databases: name currentversion latestreleasedate(YYYY-MM-DD) softwarelicense programming language description Aerospike 7.0 May 15, 2024; 12 months ago (2024-05-15) Proprietary C Aerospike Graph is a highly scalable, low-latency property graph database built on Aerospike’s proven real-time data platform. Aerospike Graph combines the enterprise capabilities of the Aerospike Database - the most scalable real-time NoSQL database - with the property graph data model via the Apache Tinkerpop graph compute engine. Developers will enjoy native support for the Gremlin query language, which enables them to write powerful business processes directly. AgensGraph[22] 2.14.1 2025-01[23] Apache 2 Community version, proprietary Enterprise Edition C AgensGraph is a cutting-edge multi-model graph database designed for modern complex data environments. By supporting both relational and graph data models simultaneously, AgensGraph allows developers to seamlessly integrate legacy relational data with the flexible graph data model within a single database. AgensGraph is built on the robust PostgreSQL RDBMS, providing a highly reliable, fully-featured platform ready for enterprise use. AllegroGraph 7.0.0 December 20, 2022; 2 years ago (2022-12-20) Proprietary, clients: Eclipse Public License v1 C#, C, Common Lisp, Java, Python Resource Description Framework (RDF) and graph database. AmazonNeptune 1.4.0.0 2024-11-06[24] Proprietary Not disclosed Amazon Neptune is a fully managed graph database by Amazon.com. It is used as a web service, and is part of Amazon Web Services. Supports popular graph models property graph and W3C's RDF, and their respective query languages Apache TinkerPop, Gremlin, SPARQL, and openCypher. Altair Graph Studio 2.1 2020-02 Proprietary C, C++ AnzoGraph DB is a massively parallel native Graph Online Analytics Processing (GOLAP) style database built to support SPARQL and Cypher Query Language to analyze trillions of relationships. AnzoGraph DB is designed for interactive analysis of large sets of semantic triple data, but also supports labeled properties under proposed W3C standards.[25][26][27][28] ArangoDB 3.12.4.2 2025-04-09 Free Apache 2, Proprietary C++, JavaScript, .NET, Java, Python, Node.js, PHP, Scala, Go, Ruby, Elixir NoSQL native graph database system developed by ArangoDB Inc, supporting three data models (key/value, documents, graphs, vector), with one database core and a unified query language called AQL (ArangoDB Query Language). Provides scalability and high availability via datacenter-to-datacenter replication, auto-sharding, automatic failover, and other capabilities. Azure Cosmos DB 2017 Proprietary Not disclosed Multi-modal database which supports graph concepts using the Apache Gremlin query language DataStaxEnterpriseGraph v6.0.1 2018-06 Proprietary Java Distributed, real-time, scalable database; supports Tinkerpop, and integrates with Cassandra[29] GUN (Graph Universe Node) 0.2020.1240 2024 Open source, MIT License, Apache 2.0, zlib License JavaScript An open source, offline-first, real-time, decentralized, graph database written in JavaScript for the web browser.[30][31] It is implemented as a peer-to-peer network featuring multi-master replication with a custom commutative replicated data type (CRDT).[32] InfiniteGraph 2021.2 2021-05 Proprietary, commercial, free 50GB version Java, C++, 'DO' query language A distributed, cloud-enabled and massively scalable graph database for complex, real-time queries and operations. Its Vertex and Edge objects have unique 64-bit object identifiers that considerably speed up graph navigation and pathfinding operations. It supports batch or streaming updates to the graph alongside concurrent, parallel queries. InfiniteGraph's 'DO' query language enables both value based queries, as well as complex graph queries. InfiniteGraph is goes beyond graph databases to also support complex object queries. JanusGraph 1.0.0 2023-10-21[33] Apache 2 Java Open source, scalable, distributed across a multi-machine cluster graph database under The Linux Foundation; supports various storage backends (Apache Cassandra, Apache HBase, Google Cloud Bigtable, Oracle Berkeley DB);[34] supports global graph data analytics, reporting, and extract, transform, load (ETL) through integration with big data platforms (Apache Spark, Apache Giraph, Apache Hadoop); supports geo, numeric range, and full-text search via external index storages (Elasticsearch, Apache Solr, Apache Lucene).[35] MarkLogic 8.0.4 2015 Proprietary, freeware developer version Java Multi-model NoSQL database that stores documents (JSON and XML) and semantic graph data (RDF triples); also has a built-in search engine. Microsoft SQL Server 2017 RC1 Proprietary SQL/T-SQL, R, Python Offers graph database abilities to model many-to-many relationships. The graph relationships are integrated into Transact-SQL, and use SQL Server as the foundational database management system.[36] NebulaGraph 3.7.0 2024-03 Open Source Edition is under Apache 2.0, Common Clause 1.0 C++, Go, Java, Python A scalable open-source distributed graph database for storing and handling billions of vertices and trillions of edges with milliseconds of latency. It is designed based on a shared-nothing distributed architecture for linear scalability.[37] Neo4j 2025.04 2025-04-30[38] GPLv3 Community Edition, commercial and AGPLv3 options for enterprise and advanced editions Java, .NET, JavaScript, Python, Go, Ruby, PHP, R, Erlang/Elixir, C/C++, Clojure, Perl, Haskell Open-source, supports ACID, has high-availability clustering for enterprise deployments, and comes with a web-based administration that includes full transaction support and visual node-link graph explorer; accessible from most programming languages using its built-in REST web API interface, and a proprietary Bolt protocol with official drivers. Ontotext GraphDB 10.7.6 2024-10-15[39] Proprietary, Standard and Enterprise Editions are commercial, Free Edition is freeware Java Highly efficient and robust semantic graph database with RDF and SPARQL support, also available as a high-availability cluster. Integrates OpenRefine for ingestion and reconciliation of tabular data and ontop for Ontology-Based Data Access. Connects to Lucene, SOLR and Elasticsearch for Full text and Faceted search, and Kafka for event and stream processing. Supports OGC GeoSPARQL. Provides JDBC access to Knowledge Graphs. OpenLinkVirtuoso 8.2 2018-10 Open Source Edition is GPLv2, Enterprise Edition is proprietary C, C++ Multi-model (Hybrid) relational database management system (RDBMS) that supports both SQL and SPARQL for declarative (Data Definition and Data Manipulation) operations on data modelled as SQL tables and/or RDF Graphs. Also supports indexing of RDF-Turtle, RDF-N-Triples, RDF-XML, JSON-LD, and mapping and generation of relations (SQL tables or RDF graphs) from numerous document types including CSV, XML, and JSON. May be deployed as a local or embedded instance (as used in the NEPOMUK Semantic Desktop), a one-instance network server, or a shared-nothing elastic-cluster multiple-instance networked server[40] Oracle RDF Graph; part of Oracle Database 21c 2020 Proprietary SPARQL, SQL RDF Graph capabilities as features in multi-model Oracle Database: RDF Graph: comprehensive W3C RDF graph management in Oracle Database with native reasoning and triple-level label security. ACID, high-availability, enterprise scale. Includes visualization, RDF4J, and native end Sparql end point. Oracle Property Graph; part of Oracle Database 21c 2020 Proprietary; Open Source language specification PGQL, Java, Python Property Graph; consisting of a set of objects or vertices, and a set of arrows or edges connecting the objects. Vertices and edges can have multiple properties, which are represented as key–value pairs. Includes PGQL, an SQL-like graph query language and an in-memory analytic engine (PGX) nearly 60 prebuilt parallel graph algorithms. Includes REST APIs and graph visualization. OrientDB 3.2.28 2024-02 Community Edition is Apache 2, Enterprise Edition is commercial Java Second-generation[clarification needed] distributed graph database with the flexibility of documents in one product (i.e., it is both a graph database and a document NoSQL database); licensed under open-source Apache 2 license; and has full ACID support; it has a multi-master replication; supports schema-less, -full, and -mixed modes; has security profiling based on user and roles; supports a query language similar to SQL. It has HTTP REST and JSON API. RedisGraph 2.0.20 2020-09 Redis Source Available License C In-memory, queryable Property Graph database which uses sparse matrices to represent the adjacency matrix in graphs and linear algebra to query the graph.[41] SAP HANA 2.0 SPS 05 2020-06[42] Proprietary C, C++, Java, JavaScript and SQL-like language In-memory ACID transaction supported property graph[43] Sparksee 5.2.0 2015 Proprietary, commercial, freeware for evaluation, research, development C++ High-performance scalable database management system from Sparsity Technologies; main trait is its query performance for retrieving and exploring large networks; has bindings for Java, C++, C#, Python, and Objective-C; version 5 is the first graph mobile database. TeradataAster 7 2016 Proprietary Java, SQL, Python, C++, R Massive parallel processing (MPP) database incorporating patented engines supporting native SQL, MapReduce, and graph data storage and manipulation; provides a set of analytic function libraries and data visualization[44] TerminusDB 11.0.6 2023-05-03[45] Apache 2 Prolog, Rust, Python, JSON-LD Document-oriented knowledge graph; the power of an enterprise knowledge graph with the simplicity of documents. TigerGraph 4.1.2 2024-12-20[46] Proprietary C++ Massive parallel processing (MPP) native graph database management system[47] TypeDB 2.14.0 2022-11[48] Free, GNU AGPLv3, Proprietary Java, Python, JavaScript TypeDB is a strongly-typed database with a rich and logical type system. TypeDB empowers you to tackle complex problems, and TypeQL is its query language. TypeDB allows you to model your domain based on logical and object-oriented principles. Composed of entity, relationship, and attribute types, as well as type hierarchies, roles, and rules, TypeDB allows you to think higher-level, as opposed to join-tables, columns, documents, vertices, edges, and properties.[promotion?] Tarantool Graph DB 1.2.0 2024-01-01[49] Proprietary Lua, C Tarantool Graph DB is a graph-vector database. Analyze data connections in real time using a high-speed graph and vector storage AQL (ArangoDB Query Language): a SQL-like query language used in ArangoDB for both documents and graphs Cypher Query Language (Cypher): a graph query declarative language for Neo4j that enables ad hoc and programmatic (SQL-like) access to the graph.[50] GQL: proposed ISO standard graph query language GraphQL: an open-source data query and manipulation language for APIs. Dgraph implements modified GraphQL language called DQL (formerly GraphQL+-) Gremlin: a graph programming language that is a part of Apache TinkerPop open-source project[51] SPARQL: a query language for RDF databases that can retrieve and manipulate data stored in RDF format regular path queries, a theoretical language for queries on graph databases Graph transformation Hierarchical database model Datalog Vadalog Object database RDF Database Structured storage Text graph Wikidata is a Wikipedia sister project that stores data in a graph database. Ordinary web browsing allows for viewing nodes, following edges, and running SPARQL queries.

^ Bourbakis, Nikolaos G. (1998). Artificial Intelligence and Automation. World Scientific. p. 381. ISBN 9789810226374. Retrieved 2018-04-20. ^ Yoon, Byoung-Ha; Kim, Seon-Kyu; Kim, Seon-Young (March 2017). "Use of Graph Database for the Integration of Heterogeneous Biological Data". Genomics & Informatics. 15 (1): 19–27. doi:10.5808/GI.2017.15.1.19. ISSN 1598-866X. PMC 5389944. PMID 28416946. ^ Angles, Renzo; Gutierrez, Claudio (1 Feb 2008). "Survey of graph database models" (PDF). ACM Computing Surveys. 40 (1): 1–39. CiteSeerX 10.1.1.110.1072. doi:10.1145/1322432.1322433. S2CID 207166126. Archived from the original (PDF) on 15 August 2017. Retrieved 28 May 2016. network models [...] lack a good abstraction level: it is difficult to separate the db-model from the actual implementation ^ Silberschatz, Avi (28 January 2010). Database System Concepts, Sixth Edition (PDF). McGraw-Hill. p. D-29. ISBN 978-0-07-352332-3. ^ Robinson, Ian (2015-06-10). Graph Databases: New Opportunities for Connected Data. O'Reilly Media, Inc. p. 4. ISBN 9781491930861. ^ "Graph Databases Burst into the Mainstream". www.kdnuggets.com. Retrieved 2018-10-23. ^ Fan, Jing; Gerald, Adalbert (2014-12-25). The case against specialized graph analytics engines (PDF). Conference on Innovative Data Systems Research (CIDR). ^ Silberschatz, Avi (28 January 2010). Database System Concepts, Sixth Edition (PDF). McGraw-Hill. p. E-20. ISBN 978-0-07-352332-3. ^ Parker, Lorraine. "IMS Notes". vcu.edu. Retrieved 31 May 2016. ^ Angles, Renzo; Gutierrez, Claudio (1 Feb 2008). "Survey of graph database models" (PDF). ACM Computing Surveys. 40 (1): 1–39. CiteSeerX 10.1.1.110.1072. doi:10.1145/1322432.1322433. S2CID 207166126. Archived from the original (PDF) on 15 August 2017. Retrieved 28 May 2016. network models [...] lack a good abstraction level: it is difficult to separate the db-model from the actual implementation ^ Kuper, Gabriel M. (1985). The Logical Data Model: A New Approach to Database Logic (PDF) (Ph.D.). Docket STAN-CS-85-1069. Archived (PDF) from the original on June 30, 2016. Retrieved 31 May 2016. ^ "SAP Announces New Capabilities in the Cloud with HANA". 2014-10-22. Retrieved 2016-07-07. ^ Frisendal, Thomas (2017-09-22). "Property Graphs". graphdatamodeling.com. Retrieved 2018-10-23. ^ Das, S; Srinivasan, J; Perry, Matthew; Chong, Eugene; Banerjee, Jay (2014-03-24). "A Tale of Two Graphs: Property Graphs as RDF in Oracle". {{cite journal}}: Cite journal requires |journal= (help) ^ a b Have, Christian Theil; Jensen, Lars Juhl (2013-10-17). "Are graph databases ready for bioinformatics?". Bioinformatics. 29 (24): 3107–3108. doi:10.1093/bioinformatics/btt549. ISSN 1460-2059. PMC 3842757. PMID 24135261. ^ "Resource Description Framework (RDF): Concepts and Abstract Syntax". www.w3.org. Retrieved 2018-10-24. ^ "The Competitive Dynamics of the Consumer Web: Five Graphs Deliver a Sustainable Advantage". www.gartner.com. Retrieved 2018-10-23. ^ a b Codd, E. F. (1970-06-01). "A relational model of data for large shared data banks". Communications of the ACM. 13 (6): 377–387. doi:10.1145/362384.362685. ISSN 0001-0782. S2CID 207549016. ^ "Graph Databases, 2nd Edition". O'Reilly | Safari. Retrieved 2018-10-23. ^ a b c d "From Relational to Graph Databases". Neo4j. ^ a b "Examples where Graph databases shine: Neo4j edition", ZeroTurnaround ^ "AgensGraph". bitnine.net. Retrieved 2025-02-19. ^ "Release AgensGraph v2.14.1 · bitnine-oss/agensgraph". github.com. SKAI Worldwide. 2025-01-16. Retrieved 2025-02-17. ^ "Amazon Neptune Engine version 1.4.0.0 (2024-11-06)". Docs.AWS.Amazon.com. Amazon Web Services. Retrieved 9 November 2024. ^ "In-memory massively parallel distributed graph database purpose-built for analytics". CambridgeSemantics.com. Retrieved 2018-02-20. ^ Rueter, John (15 February 2018). "Cambridge Semantics announces AnzoGraph graph-based analytics support for Amazon Neptune and graph databases". BusinessWire.com. Retrieved 20 February 2018. ^ Zane, Barry (2 November 2016). "Semantic graph databases: a worthy successor to relational databases". DBTA.com. Database Trends and Applications. Retrieved 20 February 2018. ^ "Cambridge Semantics announces AnzoGraph support for Amazon Neptune and graph databases". DBTA.com. Database Trends and Applications. 2018-02-15. Retrieved 2018-03-08. ^ Woodie, Alex (21 June 2016). "Beyond Titan: the evolution of DataStax's new graph database". Datanami.com. Retrieved 9 May 2017. ^ Fireship (2021-06-07). "GUN Decentralized Graph DB in 100 Seconds". YouTube. Retrieved 2024-08-02. ^ Smith, Noah (2019-07-21). "These technologists think the internet is broken. So they're building another one". NBC News. ^ Thakur, Uma; Chichmalkar, Abhishek; Sambhare, Aditya; Chaturvedi, Aman; Khuspare, Chinmay; Tembhe, Nikhil (2022). "Decentralised Chat Application" (PDF). IRJET. ^ "release 1.0.0 · JanusGraph/janusgraph". GitHub.com. 21 October 2023. ^ "JanusGraph storage backends". docs.JanusGraph.org. Archived from the original on 2018-10-02. Retrieved 2018-10-01. ^ "JanusGraph index storages". docs.JanusGraph.org. Archived from the original on 2018-10-02. Retrieved 2018-10-01. ^ "What's new in SQL Server 2017". Docs.Microsoft.com. Microsoft Corp. 19 April 2017. Retrieved 9 May 2017. ^ "Nebula Graph debuts for big data analytics discovery". Datanami.com. 29 June 2020. Retrieved 2 December 2020. ^ "Release Notes: Neo4j 2025-04". Neo4j.com. Neo4j Graph Database Platform. Retrieved 2025-04-30. ^ "Release Notes". Ontotext GraphDB. 9 November 2024. Retrieved 9 November 2024. ^ "Clustering deployment architecture diagrams for Virtuoso". Virtuoso.OpenLinkSW.com. OpenLink Software. Retrieved 9 May 2017. ^ Ewbank, Key. "RedisGraph reaches general availability". I-Programmer.info. ^ "What's new in SAP HANA 2.0 SPS 05". blogs.SAP.com. 2020-06-26. Retrieved 2020-06-26. ^ Rudolf, Michael; Paradies, Marcus; Bornhövd, Christof; Lehner, Wolfgang. The graph story of the SAP HANA database (PDF). Lecture Notes in Informatics. ^ Woodie, Alex (23 October 2015). "The art of analytics, or what the green-haired people can teach us". Datanami.com. Retrieved 9 May 2017. ^ "GitHub Releases". GitHub. Retrieved 2023-07-03. ^ "Release notes : TigerGraph : Docs". Docs.Tig